

Introduction to RFID (Radio Frequency Identification)

RFID tags consist of an integrated circuit (IC) attached to an antenna—typically printed or etched conductors on a thin plastic sheet. Data is stored on the IC and transmitted through the antenna. RFID tags are either passive (no battery) or active (self-powered by a battery). Data transmission speed and range depend on the frequency used, antenna size, power output, and interference. Tags can be read-only, read-write, or a combination, in which some data (such as a serial number) is permanently stored, while other memory is left available for later encoding during usage. Information is sent to and read from RFID tags over RF signals. In passive systems, which are the most common, an RFID reader transmits an energy field that ‘wakes up’ the tag and provides power to the chip, enabling it to transmit or store data. Encryption algorithms that ensure security and integrity of the data passing between the tag and reader may protect transmissions.

Applications

RFID provides a quick, flexible, and reliable electronic means to detect, identify, track, and hence manage a variety of items. The technology is well-suited for many operations in all types of industries. Because RFID technology uses radio waves, smart tickets can be read through dirt, paint, and many non-metallic objects. RFID smart tickets feature anti-collision technology, which lets you scan and identify several objects simultaneously, such as totes of supplies. Applications include ticketing (sports, concerts, ski lifts, etc), warehouse/logistics, airline baggage tracking, returnable plastic containers/pallets, library book/video check-out, sensitive document tracking, supply chain management, anti-counterfeiting, and pharmaceuticals.

Smart tickets (RFID Tags)

Smart tickets refer to thermal tickets with embedded ultra-thin RFID tags. Smart tickets are called ‘smart’ because of the flexible capabilities provided by the RFID tag embedded in the ticket. The tag, in most cases, can be programmed and/or updated in the field, so the same ticket can be reused to serve multiple needs and applications. Hence, the ticket is no longer effectively static as is a bar code ticket, but rather is dynamic in its capability when equipped with RFID.

Passive smart ticket RFID systems overcome the limitations of many existing bar code based automatic data capture systems in that they:

- Provide error-free, wireless data transmission that is battery-free and maintenance-free;
- Do not require line-of-site scanners for operation;
- Allow stored data to be altered during sorting or to capture workflow process information; and
- Work effectively even in harsh environments with excessive dirt, dust, moisture, and temperature extremes.

Warning: Static electricity can damage smart tickets. Be sure to ground yourself properly before handling the ticket stock.

Printing and Encoding Smart Tickets

Boca’s Smart ticket (RFID) printers enable users to create smart tickets on demand and encode them with variable information. The printers use ticket stock that incorporates blank RFID integrated circuits sandwiched between the face stock and the adhesive layer. Thus, the ICs may be invisible to the human eye. Smart ticket printers function as traditional printers when creating bar codes, graphics, and human-readable text. However, they also have an RFID encoder embedded inside the printer. Before the ticket is printed, the RFID data is encoded on the tag. Following encoding, the ticket is then fed forward for printing. An error message prints over the ticket if the tag does not read or its data does not verify, voiding the ticket for use. A NAK will be sent back to the Host indicating an error encoding the ticket. The reason for the error can be requested with a special status command (see the RFSN0 command in the programming section). The printer **will not automatically** reprint the ticket. The Host must resend the data in order to reprint the ticket. Therefore, it is recommended that there is bi-directional communication between the Host and Printer. We do not recommend a unidirectional parallel interface on an RFID printer.

The encoding and verification process, which can take milliseconds to seconds depending on the amount of data to be stored in the RFID tag, makes smart ticket throughput somewhat slower than comparable bar code printers. In practice, this slower speed has not been a drawback because smart tickets generally are not used for high-volume, high-throughput applications. The tags used in smart tickets are made from flexible material that does not damage the print head. The integrated circuit may create an uneven surface, which can affect print quality, but this problem is easily avoided by using thicker ticket material or by avoiding printing directly over the IC. Boca's smart ticket printers are capable of printing and encoding smart tickets embedded with 13.56 MHz RFID tags in a single pass.

Supported Tag Types

Boca's RFID printers currently work with six different RFID reader/encoder modules, the M1, M4, Gemini, SL015, SL032 and RT400. The modules differ in the RFID tag types they support. See below.

HF RFID M1 (Obsolete) supports

- **I-Code SLI (SL2) (ISO 15693) – 13.56MHz**
- **MIFARE Ultralight (ISO 14443A) – 13.56MHz**

HF RFID M4 (Replaced by the GEMINI) supports

- **MIFARE Ultralight (ISO 14443A) – 13.56MHz**
- **MIFARE Ultralight C (ISO 14443A) – 13.56MHz**
- **MIFARE 1K (ISO 14443A) – 13.56MHz**
- **MIFARE 4K (ISO 14443A) – 13.56MHz**

HF RFID GEMINI supports

- **MIFARE Ultralight (ISO 14443A) – 13.56MHz**
- **MIFARE 1K (ISO 14443A) – 13.56MHz**
- **MIFARE 4K (ISO 14443A) – 13.56MHz**

HF RFID SL032 supports

- **MIFARE Ultralight (ISO 14443A) – 13.56MHz**
- **MIFARE 1K (ISO 14443A) – 13.56MHz**
- **MIFARE 4K (ISO 14443A) – 13.56MHz**

HF RFID SL015 supports

- **I-Code SLI (SL2) (ISO 15693) – 13.56MHz**
- **I-Code SLIX (ISO 15693) 13.56MHz**

UHF RFID RT400 supports

- **ISO 18000-6C/EPC C1 GEN2 – 840-960MHz**

The RFID printer presently works with the following smart ticket technologies.

Note: Some features may be limited due to reader restrictions or not supported at all. Please refer to each Tag's Official Data Sheet for a more detailed description of the product.

• Philips I-Code (SL2) (ISO 15693) – 13.56MHz

Memory Organization - 32 blocks of 4 bytes each

| Byte number | 0 | 1 | 2 | 3 | Page/block |
|---------------------|----------|----------|----------|-------------------------|------------|
| Data read/write | Data 0 | Data 1 | Data 2 | Data 3 | 0 |
| Data read/write | Data 4 | Data 5 | Data 6 | Data 7 | 1 |
| Data read/write | Data 8 | Data 9 | Data 10 | Data 11 | 2 |
| . | | | | | |
| . | | | | | |
| . | | | | | |
| Data read/write | Data 108 | Data 109 | Data 110 | Data 111 | 27 |
| TID (Serial Number) | 0xE0 | 0x04 | 0x01 | Unique Tag ID (5 bytes) | |

Key features:

- 128 byte EEPROM, organized in 32 blocks of 4 bytes (112 bytes of User memory)
- Unique 8 byte identifier (UID)
- ISO 15693
- Anti-collision
- One-time-programmable user memory

Boca supported features:

- Read Serial Number.
- Read multiple data blocks - up to 16 blocks (64 bytes) at a time.
- Write multiple data blocks - up to 16 blocks (64 bytes) at a time.
- Lock multiple data blocks - up to 16 blocks (64 bytes) at a time.

• Philips MIFARE Ultralight (ISO 14443A) – 13.56MHz

The latest technology development from Philips Semiconductors, MIFARE® Ultralight, is particularly suited for applications requiring low-cost, contactless ticketing solutions. Typical applications include single trip tickets in public transport systems, loyalty cards or event tickets for exhibitions, stadiums and leisure parks.

MIFARE Ultralight Organization - 16 blocks of 4 bytes each

| Byte number | 0 | 1 | 2 | 3 | Page/block |
|-----------------|---------|----------|---------|---------|------------|
| Serial Number | SN0 | SN1 | SN2 | BCC0 | 0 |
| Serial Number | SN3 | SN4 | SN5 | SN6 | 1 |
| Internal/Lock | BCC1 | Internal | Lock0 | Lock1 | 2 |
| OTP | OTP0 | OTP1 | OTP2 | OTP3 | 3 |
| Data read/write | Data 0 | Data 1 | Data 2 | Data 3 | 4 |
| Data read/write | Data 4 | Data 5 | Data 6 | Data 7 | 5 |
| Data read/write | Data 8 | Data 9 | Data 10 | Data 11 | 6 |
| Data read/write | Data 12 | Data 13 | Data 14 | Data 15 | 7 |
| Data read/write | Data 16 | Data 17 | Data 18 | Data 19 | 8 |
| Data read/write | Data 20 | Data 21 | Data 22 | Data 23 | 9 |
| Data read/write | Data 24 | Data 25 | Data 26 | Data 27 | 10 |
| Data read/write | Data 28 | Data 29 | Data 30 | Data 31 | 11 |
| Data read/write | Data 32 | Data 33 | Data 34 | Data 35 | 12 |
| Data read/write | Data 36 | Data 37 | Data 38 | Data 39 | 13 |
| Data read/write | Data 40 | Data 41 | Data 42 | Data 43 | 14 |
| Data read/write | Data 44 | Data 45 | Data 46 | Data 47 | 15 |

| | | |
|-----------------|---------------------------|--|
| Serial Number | block 0, bytes 0-3 | (programmed by IC manufacturer). |
| Serial Number | block 1, bytes 4-7 | (programmed by IC manufacturer). |
| Internal/Lock | block 2, bytes 8-11 | (used to lock bytes). |
| OTP | block 3, bytes 12-15 | (One Time Programming bytes – possible counter). |
| Data read/write | blocks 4 –15, bytes 16-63 | (48 available for user data). |

Key features:

- 64 byte EEPROM, organized in 16 pages of 4 bytes (48 bytes of User memory)
- Unique 7 byte serial number (ISO 14443A, cascade level 2)
- Operating distance up to 10 cm
- 106 k bits/s data transfer rate
- High data integrity – 16 bit CRC, parity per byte, bit coding, bit count check
- 32 bit one-time programmable (OTP) area
- Field programmable ‘Read only’ locking function per page

Boca supported features:

- Read Serial Number.
- Read multiple data blocks - up to 16 blocks (64 bytes) at a time.
- Write multiple data blocks - up to 12 blocks (48 user bytes) at a time. Note: blocks 2 and 3 are special function blocks that can be written to separately.
- Lock multiple data blocks - up to 12 blocks (48 user bytes) at a time.

• Philips MIFARE Ultralight C (ISO 14443A) – 13.56MHz

This is an enhanced version of the MIFARE Ultralight tag above. It contains additional security features and increased user memory.

Key Features:

- 192 byte EEPROM, organized in 48 pages of 4 bytes (144 bytes of User memory)
- Unique 7 byte serial number (ISO 14443A, cascade level 2)
- Operating distance up to 10 cm
- 106 k bits/s data transfer rate
- High data integrity – 16 bit CRC, parity per byte, bit coding, bit count check
- 32 bit one-time programmable (OTP) area
- 3DES Authentication
- Field programmable read-only locking function per page for first 512-bit
- Read-only locking per block for the memory above 512 bit

Boca supported features:

- Read Serial Number.
- Read multiple data blocks - up to 44 blocks (176 bytes) at a time (the last 4 blocks are unreadable).
- Write multiple data blocks - up to 36 blocks (144 user bytes) at a time. Note: blocks 2, 3, 40-47 are special function blocks that can be written to separately.
- Secure tag with 3DES 16 byte key.
- Authenticate Tag using 3DES 16 byte key.
- **Note:** the Boca Write command ‘lock option’ is not supported for the MIFARE Ultralight C tags. They can be locked by writing to Lock Bytes 0,1,2 and 3 as shown below.

Memory organization

192 byte EEPROM, organized in 48 pages of 4 bytes (144 bytes of User memory)

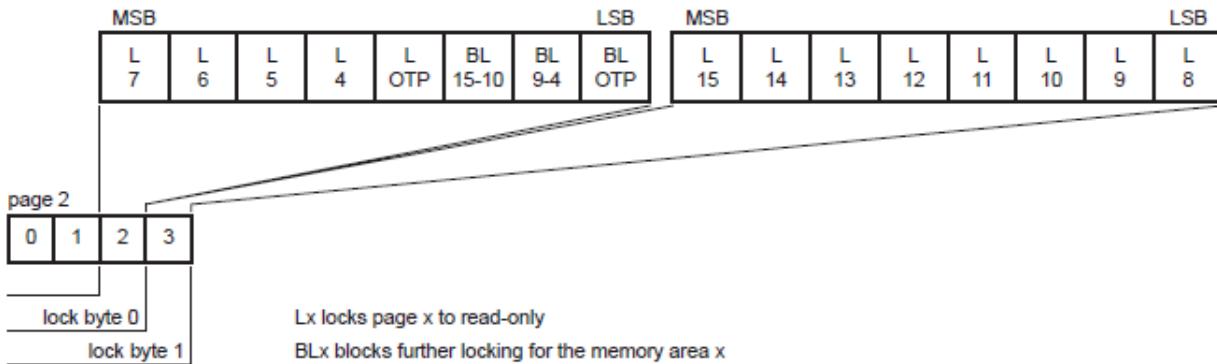
| Page address | | Byte number | | | |
|--------------|------------|------------------------------|----------------|-------------|-------------|
| Decimal | Hex | 0 | 1 | 2 | 3 |
| 0 | 00h | serial number | | | |
| 1 | 01h | serial number | | | |
| 2 | 02h | serial number | internal | lock bytes | lock bytes |
| 3 | 03h | OTP | OTP | OTP | OTP |
| 4 to 39 | 04h to 27h | user memory | user memory | user memory | user memory |
| 40 | 28h | lock bytes | lock bytes | - | - |
| 41 | 29h | 16-bit counter | 16-bit counter | - | - |
| 42 | 2Ah | authentication configuration | | | |
| 43 | 2Bh | authentication configuration | | | |
| 44 to 47 | 2Ch to 2Fh | authentication key | | | |

Lock bytes 0 and 1

The bits of byte 2 and byte 3 of page 02h represent the field programmable permanent read-only locking mechanism. Each page from 03h (OTP) to 0Fh can be individually locked by setting the corresponding locking bit Lx to logic 1 to prevent further write access. After locking, the corresponding page becomes read-only memory. To restrict access to the memory see the authentication section.

The three least significant bits of lock byte 0 are the block-locking bits. Bit 2 deals with pages 0Ah to 0Fh, bit 1 deals with pages 04h to 09h and bit 0 deals with page 03h (OTP). Once the block-locking bits are set, the locking configuration for the corresponding memory area is frozen.

The functionality of the bits inside the lock bytes 0 and 1 are shown below.



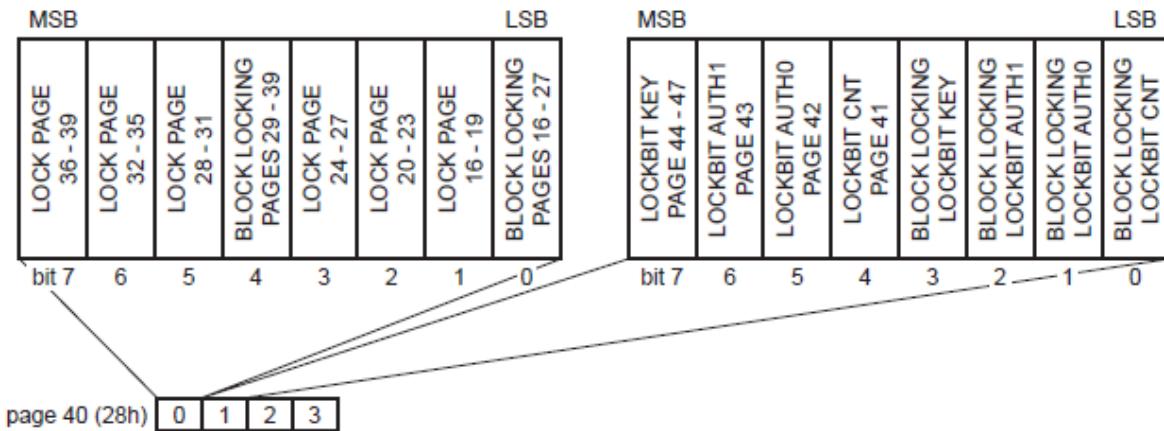
Lock bytes 0 and 1

For example, if BL15-10 is set to logic 1, then bits L15 to L10 (lock byte 1, bit[7:2]) can no longer be changed. A Write command to page 02h, sets the locking and block-locking bits. The data is bit-wise OR'ed and the result then becomes the new content of the lock bytes. This process is irreversible. If a bit is set to logic 1, it cannot be changed back to logic 0. Therefore, before writing the lock bytes, the user has to ensure that the corresponding user memory area and/or configuration bytes to be locked are correctly written.

Lock bytes 2 and 3

To lock the pages starting at page address 10h (16) onwards, the lock bytes 2 and 3 located in page 28h (40) are used. Those two lock bytes cover the memory area of 96 data bytes in pages 10h (16d) to 27h (39d) and the configuration area from page address 28h onwards. The granularity is 4 pages, compared to a single page for the first 512 bits. The functionality of the bits inside the lock bytes 2 and 3 are shown below.

Note: Set all unused bits to 0, when writing to the lock bytes



Lock bytes 2 and 3

Configuration for memory access via 3DES Authentication

The memory access rights are determined by two configuration bytes, AUTH0 and AUTH1, located in pages 2Ah (42) and 2Bh (43). Both configuration bytes are located in Byte 0 of the respective pages.

- AUTH0 defines the hex page address from which the authentication is required. Authentication is required from that page all the way to the last user page 2Fh (47). Valid hex address values are from 03h to 30h.

Setting AUTH0 to 30h effectively disables memory protection.

- AUTH1 determines if write access is restricted or both read and write access is restricted.

Setting AUTH1 to 0h will restrict both read and write access.

Setting AUTH1 to 1h will restrict only write access.

3DES Authentication

The Ultralight C tag can be secured with a 16 byte 3DES key. The key is written to memory pages 44-47 and is unreadable. Once the key is written to the tag, access will be restricted according to the configuration of the AUTH0 and AUTH1 bytes described previously.

Programming of 3DES key to memory

The 16 bytes of the 3DES key are programmed to memory pages 2Ch (44) to 2Fh (47). They are stored internally in little-endian format as two 8 byte keys as shown below. Later the key is sent in big-endian format for authentication. This can get a little confusing. Some examples will help.

| Byte address | | 0h | 1h | 2h | 3h |
|--------------|---------|-----------|-----------|-----------|-----------|
| Page address | | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| 2Ch | Page 44 | Key1 / K0 | Key1 / K1 | Key1 / K2 | Key1 / K3 |
| 2Dh | Page 45 | Key1 / K4 | Key1 / K5 | Key1 / K6 | Key1 / K7 |
| 2Eh | Page 46 | Key2 / K0 | Key2 / K1 | Key2 / K2 | Key2 / K3 |
| 2Fh | Page 47 | Key2 / K4 | Key2 / K5 | Key2 / K6 | Key2 / K7 |

If the desired 3DES key is **000102030405060708090A0B0C0D0E0F**, it is looked at like this:

Key1 = 0001020304050607h and Key2 = 08090A0B0C0D0E0Fh

The command sequence needed for key programming with a Boca Write command is:

<RFW2,44,0>**07060504030201000F0E0D0C0B0A0908** Note: this uses the Boca ASCII format.

Later the tag could be authenticated using the Boca Key and Authenticate commands.

The complete description of these commands is explained later.

The key **000102030405060708090A0B0C0D0E0F** would be stored in memory as shown below.

| Byte address | | 0h | 1h | 2h | 3h |
|--------------|---------|--------|--------|--------|--------|
| Page address | | Byte 0 | Byte 1 | Byte 2 | Byte 3 |
| 2Ch | Page 44 | 07 | 06 | 05 | 04 |
| 2Dh | Page 45 | 03 | 02 | 01 | 00 |
| 2Eh | Page 46 | 0F | 0E | 0D | 0C |
| 2Fh | Page 47 | 0B | 0A | 09 | 08 |

One more 3DES key example:

Key = BREAKMEIFYOUCAN!

Boca Command = <RFW1,44,0>IEMKAERB!NACUOYF **Note:** this uses the Boca Binary format.

Sample AUTH0, AUTH1 and Lock byte write commands

<RFW2,42,0>**10000000** - SET AUTH0 TO REQUIRE AUTHENTICATION FROM PAGE **10h (16)** TILL THE END - PAGE 2Fh (47). **Note:** The value written is the hex page number.

<RFW2,42,0>**22000000** - SET AUTH0 TO REQUIRE AUTHENTICATION FROM PAGE **22h (34)** TILL THE END - PAGE 2Fh (47). **Note:** The value written is the hex page number.

<RFW2,43,0>01000000 - SET AUTH1 TO WRITE RESTRICTIONS

<RFW2,43,0>00000000 - SET AUTH1 TO BOTH READ & WRITE RESTRICTIONS

<RFW2,44,0>07060504030201000F0E0D0C0B0A0908 - SET 3DES KEY TO 000102030405060708090A0B0C0D0E0F

<RFW2,2,0>000000FC - LOCK BLOCKS 10-15 **Note:** first two bytes are 0000 when writing to lock bytes 0 & 1.

<RFW2,40,0>40000000 - LOCK BLOCKS 32-35 **Note:** last two bytes are 0000 when writing to lock bytes 2 & 3.

<RFW2,40,0>FFFF0000 - LOCK ALL BLOCKS FROM 16-39 & 41,42,43 **Note:** last two bytes are 0000 when writing to lock bytes 2 & 3.

• Philips MIFARE 1K (ISO 14443A) – 13.56MHz

The MIFARE-1K classic family is the pioneer and front runner in contactless smart card ICs operating in the 13.56 MHz frequency range with read/write capability. The MIFARE® standard IC, launched in 1995, was the first product which could be fitted into a ISO contactless smart card, and with its slim coil allowed very high volume production.

Key features:

- 1024 byte EEPROM, organized in 16 sectors of 64 bytes (752 bytes of User memory)
- Unique serial number (4 Byte)
- 16 securely separated sectors supporting multi-application
- Each sector consists 4 blocks with a length of 16 Byte
- 2 x 48 bit keys per sector for key hierarchy
- Number of single write operations: 100,000
- Data retention - 10 years

| | | Byte Number within a Block | | | | | | | | | | | | | | | Description | | | |
|--------|-------|----------------------------|---|---|---|-------------|---|---|---|-------|---|----|----|----|----|----|--------------------|--|--|--|
| Sector | Block | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | | |
| | | Key A | | | | Access Bits | | | | Key B | | | | | | | | | | |
| 15 | 3 | | | | | | | | | | | | | | | | Sector Trailer 15 | | | |
| | 2 | | | | | | | | | | | | | | | | Data | | | |
| | 1 | | | | | | | | | | | | | | | | Data | | | |
| | 0 | | | | | | | | | | | | | | | | Data | | | |
| 14 | 3 | | | | | | | | | | | | | | | | Sector Trailer 14 | | | |
| | 2 | | | | | | | | | | | | | | | | Data | | | |
| | 1 | | | | | | | | | | | | | | | | Data | | | |
| | 0 | | | | | | | | | | | | | | | | Data | | | |
| : | : | | | | | | | | | | | | | | | | | | | |
| : | : | | | | | | | | | | | | | | | | | | | |
| : | : | | | | | | | | | | | | | | | | | | | |
| 1 | 3 | | | | | | | | | | | | | | | | Sector Trailer 1 | | | |
| | 2 | | | | | | | | | | | | | | | | Data | | | |
| | 1 | | | | | | | | | | | | | | | | Data | | | |
| | 0 | | | | | | | | | | | | | | | | Data | | | |
| 0 | 3 | | | | | | | | | | | | | | | | Sector Trailer 0 | | | |
| | 2 | | | | | | | | | | | | | | | | Data | | | |
| | 1 | | | | | | | | | | | | | | | | Data | | | |
| | 0 | | | | | | | | | | | | | | | | Manufacturer Block | | | |

| | |
|---------------|--|
| Sector | Memory segment of the MIFARE 1K Card. Each segment consists of 4 blocks and has individual keys and access conditions. |
| Security Key | 6 byte structure assigned to each sector of the card. |
| Transport Key | Key as stored after delivery from the manufacturer.(for example A0A1A2A3A4A5, B0B1B2B3B4B5 or FFFFFFFFFFFF) |
| Block | 16 byte memory segment of the MIFARE 1K card. |
| Value | 4 byte (unsigned long) variable stored in a special format in a block or page. Values are 2s complement numbers that can be negative also. Values are used for cashless payment. Values consume a complete block each using redundancy for integrity checking. |
| Card ID | 4 byte unique serial number. Together with manufacturer code and check byte 16 bytes. Read-only. It is stored in block 0 (sector 0) of each tag. |

Sector 0 / Block 0

Block 0 is read only.

| | | |
|------------------------|---------------------|-----------------------------|
| Serial Number (4 byte) | Check byte (1 byte) | Manufacturer data (11 byte) |
|------------------------|---------------------|-----------------------------|

Block 3, 7, 11, 15 ...

Transport keys are set on delivery:

| | | |
|----------------|-----------------------------|----------------|
| Key A (6 byte) | Access Conditions (4 bytes) | Key B (6 byte) |
|----------------|-----------------------------|----------------|

Key A

A0 A1 A2 A3 A4 A5 (Infineon) or FF FF FF FF FF FF (new Philips cards)

Key B

B0 B1 B2 B3 B4 B5 (Infineon) or FF FF FF FF FF FF (new Philips cards)

Access Conditions

FF 07 80 xx (key A used to read or write, the key A itself is not readable; key B is data only).

Remarks

Enabled keys are always read as 00 00 00 00 00 00

Using key B as data area will cause a security gap, due to the fact that it is necessary to rewrite key A and access conditions each write process. It is not recommended for use as data storage.

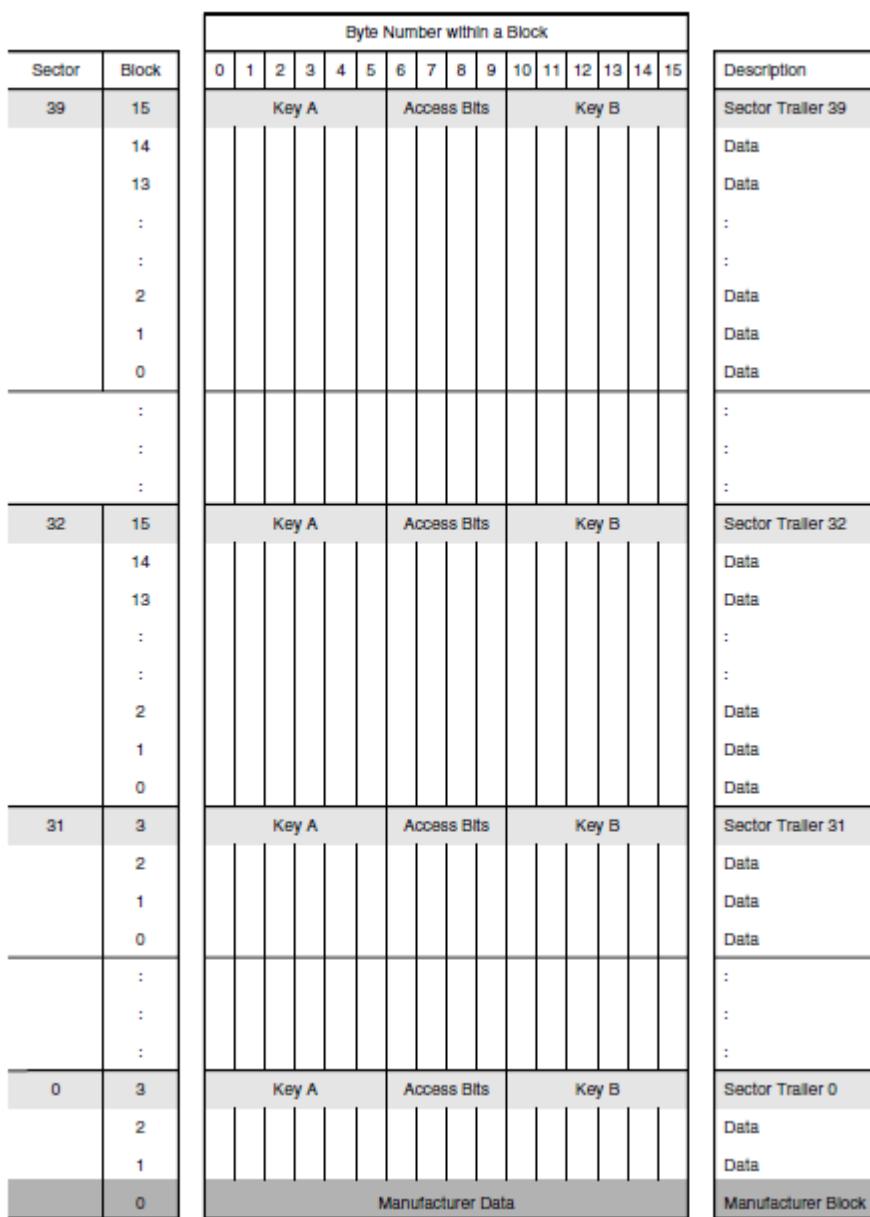
IT IS STRONGLY RECOMMENDED THAT THE KEY CODES AND THE ACCESS BITS STORED ON THE MIFARE CARD ARE NOT CHANGED UNTIL THEIR OPERATION IS FULLY UNDERSTOOD.

Boca supported features:

- Read Serial Number.
- Read single data block - only 1 block (16 bytes) or less at a time. Requests for more are clipped.
- Write single data block - only 1 block (16 bytes) or less at a time. Note: if less, the remainder is padded with nulls. Any writes of more than 1 block of data will result in an error.
- Permanent Key A value can be changed. Default value is Philips Transport Configuration.
- Blocks can't be locked due to a reader limitation at this time.
- Value functions Inc, Dec, Transfer and Restore are not supported by the reader.

- Philips MIFARE 4K (ISO 14443A) – 13.56MHz

MIFARE Classic 4K offers 4096 bytes split into forty sectors, of which 32 are same size as in the 1K with eight more that are quadruple size sectors. The following figure shows the structure of the MIFARE Classic 4K memory.



All of the features and programming information for the MIFARE 4K tag are the same as stated above for the MIFARE 1K. The main difference is the size of memory available for the user - 3440 bytes.

• UHF ISO 18000-6C/EPC C1 GEN2 – 840-960MHz

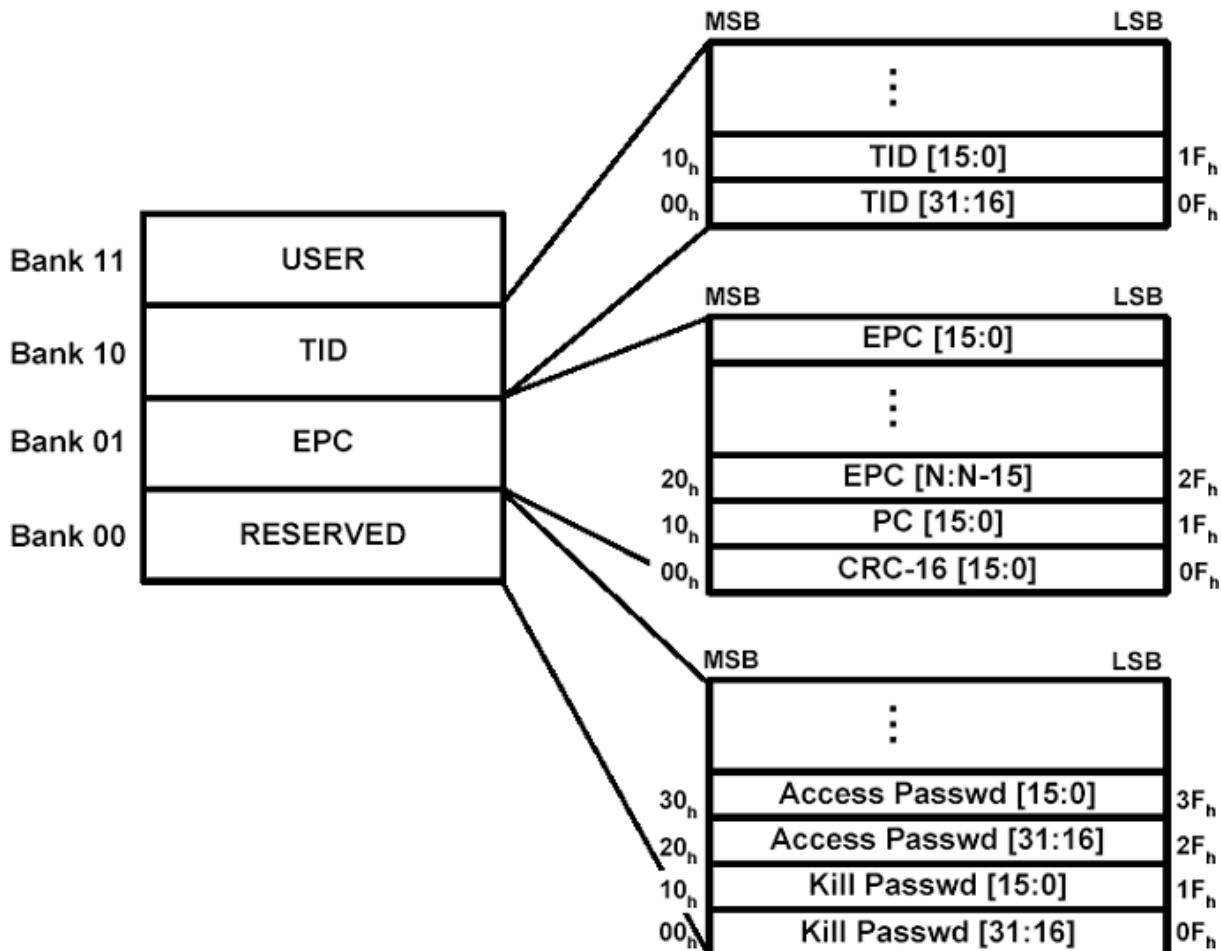
Key features:

- Read distances over 10 meters
- 40-50 year data retention
- 96 bit EPC (UII) Number
- 32-64 bit tag identifier (TID)
- 32 bit kill password
- 32 bit access password
- User memory dependent on manufacturer

Boca supported features:

- Reading Single Block
- Reading Multiple Blocks
- Writing Single Block
- Writing Multiple Blocks
- Writing 12-byte EPC
- Writing to User Memory (not supported on all tags)
- Writing a Tag Password (only on tags that support access passwords)
- Sending a Tag Password
- Setting Lock Functions
- Killing a Tag

The following figure shows the structure of a general EPC Class1 Gen2 UHF tag.



Memory organization of HF vs UHF Class1 Gen2 tags

The structure of the memory in HF (High Frequency) tags is standardized and fixed. There are a known number of blocks in each sector. The structure of the UHF Class1 Gen2 tags can vary depending on manufacturer. Each Gen2 tag can contain up to 4 banks* (0-3) of *different* block lengths. Each block is two bytes (16 bits) in length. Because of the *varying* number of blocks per bank, the format of the '**starting block**' parameter in the standard Boca Read and Write commands (described later in the Programming Tickets section) is slightly different for UHF tags vs HF tags. Normally, the '**starting block**' is defined as the starting block location to read or write data. The parameter is sent as a decimal value. For a UHF tag, the 16 bit '**starting block**' address field defines both the starting bank and the starting block within that bank. The first nibble (4 bits) specifies the memory bank number and the remaining 12-bits specify the block address within the memory bank. Because this parameter is nibble based the value must be sent as a 4 digit hex number. For example, to address block #2 of the EPC memory bank, the address field is x1002. The first nibble, "1" specifies the EPC memory bank (i.e., bank 01), and the next 12-bits specify the block number (x002). Address x3000 would be bank3, block 0. Address x0002 would be for the tag Access Password location – bank 0, block 2. Address x200A would be bank 2, block 10.

***Note**— The user memory bank (bank 3) is not supported on all of the EPC Class1 Gen2 tags.

States

Note – This applies only to tags that support access passwords.

A tag can be in one of two states:

Open (if access password is all zeros, which is the factory default)

Secured (if non-zero access password has been programmed to the tag)

A tag in the open state can be moved to secured by providing a non-zero access password.

The key thing to remember is that *assigning a non-zero access password does not, in itself, prevent anyone with a GEN 2 reader from reading or changing data on the tag*. It only requires that any future users must provide the access password in order to change the lock state and is simply one step in effectively locking tag memory. Further protection can be added by locking and/or making the password read protected.

Access Password

You can use the Boca Write command to address x0002 to assign a non-zero password to a tag. The 32-bit tag password is stored in Blocks 2 and 3 of the reserved memory bank (bank 0). Block 2 holds the higher 16-bits of the password, and Block 3 holds the lower 16-bits. **Note:** the 32 bit value must be sent as 8 hexadecimal characters (0-F).

Example commands:

| | |
|-----------------------|---|
| <RFW2,0002,0>DEADBEEF | – stores 32 bit password (DEADBEEF) in bank 0, block 2. |
| <RFW2,0002,0>12345678 | – stores 32 bit password (12345678) in bank 0, block 2. |
| <RFW2,0002,0>1234ABCD | – stores 32 bit password (1234ABCD) in bank 0, block 2. |

After you write a non-zero password to location x0002, the reader requires that you send the password before any future Secure State transactions *with that tag*. Whether an operation is secure is determined by the Lock Functionality explained in the next section.

The Boca Password Command format is <RFTP#####> where the #'s represent the 32-bit (8 hex digit) password. Use this command to send the password before you execute a secure command for that tag. This remains a requirement until you change the access password or reset the password value to zero. Once the Password command has been sent, the printer will automatically send it before each secured read or write command. Therefore, you only need to send the command once, after modifying address x0002, unless an error occurs. Usually after an error, an <RFC> (clear error) command is issued, which resets this auto password function. Therefore, you must re-issue a Password command after an <RFC> to avoid further errors. See the end of the Lock Section for more detailed command examples.

Lock Functionality

Class 1 Gen 2 tags let you use the lock functionality to set read permissions, write permissions, and passwords for tag memory. The following tables list the different values for Locking/Password protecting different sections of the tag memory for EPC Class1 Gen2 tags.

Each memory bank can be in one of four lock states:

1. Unlocked
2. Perma-unlocked (can never be locked)
3. Locked
4. Perma-locked (can never be unlocked)

Note: there is no way to read back the Lock Status of a UHF tag.

The lock contains a 20-bit payload defined as follows:

- The first 10 payload bits are Mask bits. A Tag interprets these bit values as follows:
- Mask = 0: Ignore the associated Action field and retain the current lock setting.
- Mask = 1: Implement the associated Action field, and overwrite the current lock setting.

Lock-Command Payload

| | | | | | | | | | | | | | | | | | | | |
|----------------|------------------|---------------|---|---------------|---|----------------|---|------------------|---|--------------------|----|-----------------|----|-----------------|----|----|------------------|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| ⋮ Kill Mask | ⋮ Access Mask | ⋮ EPC Mask | ⋮ | ⋮ TID Mask | ⋮ | ⋮ User Mask | ⋮ | ⋮ Kill Action | ⋮ | ⋮ Access Action | ⋮ | ⋮ EPC Action | ⋮ | ⋮ TID Action | ⋮ | ⋮ | ⋮ User Action | ⋮ | |

Masks and Associated Action Fields

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

The last 10 payload bits are Action bits. A Tag interprets these bit values as follows:

- Action = 0: De-assert lock for the associated memory location.
- Action = 1: assert lock or permalock for the associated memory location.

| Pwd write | Permalock | Description |
|--------------------|-----------|---|
| 0 | 0 | Associated memory bank is writable from either the open or secured states. |
| 0 | 1 | Associated memory bank is permanently writable from either the open or secured states and can never be locked. |
| 1 | 0 | Associated memory bank is writable from the secured state only. |
| 1 | 1 | Associated memory bank is not writable from any state. |
| Pwd read/ write | Permalock | Description |
| 0 | 0 | Associated memory bank is readable and writable from either the open or secured states. |
| 0 | 1 | Associated memory bank is permanently readable and writable from either the open or secured states and can never be locked. |
| 1 | 0 | Associated memory bank is readable and writable from the secured state only. |
| 1 | 1 | Associated memory bank is not readable or writable from any state. |

Only reserved memory bank (access and kill passwords) can be both WRITE and READ locked - all others (EPC, TID, and User) can be WRITE-locked only. **Typically the Tag Identification (TID) memory bank is perma-locked at the factory.**

Use the Boca Lock Command to send the 20-bit Lock payload to the tag. This command will automatically store the payload in the correct memory location on the tag.

The Lock command format is <RFTL#> where the # is the hex value for the 20-bit payload. Examples are the easiest way to understand this command.

IT IS STRONGLY RECOMMENDED THAT THE LOCK COMMAND AND THE ACCESS BITS STORED ON THE UHF TAG ARE NOT CHANGED UNTIL THEIR OPERATION IS FULLY UNDERSTOOD.

To set the user memory bank for secured writing you need to have a 10 bit mask value of 00 00 00 00 10 (which sets bit 8 in the mask for writing) and a 10 bit action value of 00 00 00 00 10 (which sets bit 18 for pwd write). Therefore the value sent in the command is 0000 0000 1000 0000 0010 binary which is 00802 or 802 hex. So the command would be <RFTL802>. This would mean that a password is needed in order to write data to the user memory. So, the full command sequence for storing a non-zero password on a tag, setting the Lock Payload for secured user memory writing, and sending the password before writing the word BOCA to user memory is as follows:

| | |
|-----------------------|---|
| <RFW2,0002,0>12345678 | - stores non-zero password 12345678 bank 0 block 2 |
| <RFTP12345678> | - sends 32-bit password 12345678 to printer |
| <RFTL802> | - send Lock Payload of 802 which sets user memory for secure writes |
| <RFW1,3000,0>BOCA | - writes the word BOCA to user memory bank 3 block 0 |

Note: if the tag is not password protected then writing the word BOCA to user memory is as simple as:

| | |
|-------------------|--|
| <RFW1,3000,0>BOCA | - writes the word BOCA to user memory bank 3 block 0 |
|-------------------|--|

More examples:

To permalock the EPC you need a 10 bit mask value of 00 00 11 00 00 (which sets bits 4 & 5 in the mask for writing) and a 10 bit action value of 00 00 11 00 00 (which sets bits 14 & 15 for pwd write & permalock). This translates to a value of 0000 1100 0000 0011 0000 or 0C030 or C030 hex. So the command sequence on a new tag (password 00000000) would be as follows:

| | |
|-----------------------|--|
| <RFW2,0002,0>11AA22BB | - stores non-zero password 11AA22BB bank 0 block 2 |
| <RFTP11AA22BB> | - sends 32-bit password 11AA22BB to printer |
| <RFTLC030> | - send Lock Payload of C030 which permalocks the EPC |

To set the user memory bank for secured writing and to make the **access password unreadable without knowing the password** you need to have a 10 bit mask value of 00 11 00 00 10 (which sets bits 2,3,8 in the mask for writing) and a 10 bit action value of 00 10 00 00 10 (which sets bits 12,18 for pwd read/write and pwd write). Therefore the value sent in the command is 0011 0000 1000 1000 0010 binary which is 30882 hex. So the command would be <RFTL30882>. This would mean that a password is needed in order to write data to the user memory and to be able to read or change the password. So, the full command sequence for storing a non-zero password on a tag, setting the Lock Payload for secured user memory writing and password reading/writing, and sending the password before writing the word BOCA to User memory is as follows:

| | |
|-----------------------|---|
| <RFW2,0002,0>11223344 | - stores non-zero password 11223344 bank 0 block 2 |
| <RFTP11223344> | - sends 32-bit password 11223344 to printer |
| <RFTL30882> | - send Lock Payload of 30882 which sets secures user and password |
| memory | |
| <RFW1,3000,0>TEST | - writes the word TEST to user memory bank 3 block 0 |

If you wanted to change the access password to a new one, for example from 12345678 to 12ABCDEF you need to send the old password first and use the following sequence.

| | |
|-----------------------|--|
| <RFTP12345678> | - sends old 32-bit password 12345678 to printer |
| <RFW2,0002,0>12ABCDEF | - stores new non-zero password 12ABCDEF bank 0 block 2 |
| <RFTP12ABCDEF> | - sends 32-bit password 12ABCDEF to printer |

Killing a Tag

Killing a tag makes it unusable. It can't be read or written. You must write a Kill password to bank 0 block 0 and then issue a Boca Kill Tag command in order to kill a tag. The Format of the Kill Tag command is <RFTK#####> where the #'s represent the 32-bit (8 hex digit) kill password. For example, to kill a tag with the password DEADDEAD use the following sequence:

| | |
|-----------------------|--|
| <RFW2,0000,0>DEADDEAD | - stores non-zero password DEADDEAD bank 0 block 0 |
| <RFTKDEADDEAD> | - sends 32-bit kill password DEADDEAD to printer – tag is now killed |

Programming Tickets

RFID extensions have been added to the FGL programming language allowing users to program the tag data with the same ease as defining the data for a bar code ticket or label. RFID commands can be used in conjunction with any standard FGL programming command. The RFID commands can be used to read the unique ID number of each ticket, read/write tag data and send that data to the Host.

RFID tags are based on an EEPROM technology that requires time to be programmed. As mentioned before, you may notice a slight printing delay between tickets. This time is necessary to ensure consistent reliability. Occasionally, a ticket may need to be written and verified more than once. This retry period will increase the delay between tickets. The number of retries can be set using an RFID command. Another factor that can be changed by the user, is the period of time that the printer will wait for a response from the RFID encoder. This will vary among different stocks and RFID technologies. You should only increase the timeout amount if ‘timeout error’ void tickets are being printed.

FGL RFID Programming Extensions:

READ RF CARD UNIQUE TAG ID (SERIAL NUMBER) COMMAND - <RFSN 'format', 'send option'>

The printer returns the TID (serial number) for the selected tag. The number of bytes returned depends on the type of tag selected.

Each I-Code (SL2) ticket contains a unique 8 byte serial number starting with E00401. This number can be read from the ticket and either printed or sent to the Host.

Each MIFARE Ultralight and Ultralight C ticket contains a unique 7 byte serial number. This number can be read from the ticket and either printed or sent to the Host.

Each MIFARE 1K/4K ticket contains a unique 4 byte serial number. This number can be read from the ticket and either printed or sent to the Host.

Each ISO 18000-6C ticket contains a unique 12 byte serial number (EPC). This number can be read from the ticket and either printed or sent to the Host.

The ‘format’ field values are as follows:

1 - *Binary* format (1 byte/value) Data is used as is. For example, the character ‘1’ (31h) is stored or sent as 31h. Use this format for programming or reading printable Text characters like names and addresses.

2 – *ASCII* format (2 bytes/value) Data is represented as the Hex value of each nibble. For example, a data value of 1 (01h) is stored or sent as 30h (ASCII character ‘0’) and 31h (ASCII character ‘1’). Use this format for programming or reading non-printable characters like serial numbers or data.

Note the difference between the character ‘1’ (31h) and the byte value of 1 (01h). You use Binary format to send a character ‘1’ and ASCII format to send a byte value of 1.

The ‘send options’ are as follows:

- 0 – send data to ticket
- 1 – send data to Host
- 2 – send data to ticket and Host

Example:

The <RC10,10><F2><RFSN2,2> command would be used to read the ticket’s serial number in ASCII format and print it in font2 at position 10,10 on the ticket as well as sending it to the Host. For example, if the 7 byte serial number was comprised of values 04h,0Ch,65h,D1h,10h,00h,40h the printer would print 040C65D1100040 on the ticket and send it to the Host as 30h,34h,30h,43h,36h,35h,44h,31h,31h,30h,30h,30h,34h,30h.

REQUEST RF CARD STATUS COMMAND - <RFSN0> (Note: this is a zero).

This command can be sent by the Host after receiving a NAK (15H) to determine the cause of the error. The printer sends back a single byte ASCII status indicating the status. A list of the status codes and errors are shown at the end. Looking at the list you will see that different errors can have the same code. For example, the printer will return a generic 'C' code for all command errors but displays the specific error message on the LCD display.

Once a NAK has been sent the ticket is considered 'Void'. This Void state remains in effect until the ticket is printed, removed or an <RFC> command is sent (see next command). Any remaining RFID operations, even if successful, will not change the Void state. If the ticket is printed, it will have the word 'VOID' printed on it along with the **original** cause for the NAK.

Note: a NAK is sent after **every** failed RFID operation. If multiple operations are attempted it is possible to receive multiple NAK's. If the Request Status command is sent after a string of RFID commands, the printer will only return the status code for the **last** operation. The status code remains in effect until a new RFID operation completes, an <RFC> command is received, or the ticket is either printed or removed. The 'status state' is different than the 'Void' state. It is always the status of the **last** RFID operation performed.

CLEAR ALL RFID ERRORS - <RFC>

As mentioned above, once an error has occurred the ticket will be in a 'void' state. You will no longer be able to print a valid ticket using that tag. If for some reason you want to try another RFID operation using the same tag you must clear the error state flags. Sending the <RFC> command will clear all the error states and return the printer to the normal state. **Note:** if a tag had been authenticated previously, it will have to be re-authenticated after issuing this command. If a password has been sent to the tag, it will have to be resent.

READ RF CARD COMMAND - <RFR 'format', 'starting block', 'number of bytes to read', 'send option'>

The '*format*' field uses the same values as explained in the RFSN command (1,2).

The '*starting block*' number is the starting block to begin reading. The number can range from 0-27 (I-Code2), 0-15 (MIFARE Ultralight), 0-43 (MIFARE Ultralight C), 0-63 (MIFARE 1K), 0-255 (MIFARE 4K). **Note:** these are decimal values. For UHF Gen2 tags this parameter is defined differently as explained in the *Memory organization of HF vs UHF Class1 Gen2 tags section*.

The '*number of bytes to read*' is limited to 64 when using the M1 reader. If you need to read more than 64 bytes, you must issue extra read commands. The M4, Gemini and SL032 readers allow reading the full tag except when using MIFARE 1K/4K tags. These tags can only be read up to 16 bytes at a time. **Note:** these are decimal values.

The '*send options*' are the same as before:

- 0 – send data to ticket
- 1 – send data to Host
- 2 – send data to ticket and Host

Examples: The <RC10,10><F2><RFR1,4,12,0> command would be used to read 12 bytes starting at block 4 and print it in Binary format in font2 at position 10,10 on the ticket. If the 12 bytes starting at block 4 were comprised of values 42h,4Fh,43h,41h,20h,53h,59h,53h,54h,45h,4Dh,53h the printer would print BOCA SYSTEMS on the ticket.

If the word 'TEST' was stored at page 5 then the <RFR1,5,4,1> command would cause the printer to send back the following to the Host – TEST (54h,45h,53h,54h). The <RFR2,5,4,1> would send back 2 bytes for each byte read 35h,34h,34h,35h,35h,33h,35h,34h.

WRITE RF CARD COMMAND - <RFW ‘format’, ‘starting block’, ‘lock option’, ‘optional byte count field’>data bytes(CR-carriage return) or command delimiter ends the data stream unless optional byte count field is used.

- On an I-Code2 card, blocks 0-27 (112 bytes) are available for user data.
- On the MIFARE Ultralight card, blocks 4-15 (48 bytes) are available for user data (additional configuration blocks can be written).
- On the MIFARE Ultralight C card, blocks 4-39 (144 bytes) are available for user data (additional configuration blocks can be written).
- On the MIFARE 1K, blocks 0-63 (752 bytes) are available for user data (some user bytes unavailable).
- On the MIFARE 4K, blocks 0-255 (3440 bytes) are available for user data (some user bytes unavailable).

Note: these are decimal values.

Important: Data must be written in multiples of block size. If not, the remaining bytes will be programmed to zero on the card. UHF Gen2 tag block size is 2 bytes. MIFARE 1K and 4K must be written to in one single block size (16 bytes) or less.

The ‘format’ field uses the same values as explained in the RFSN command (1,2).

The ‘starting block’ number is the starting block to begin writing. The number can range from 0-27 (I-Code2), 2-15 (MIFARE Ultralight), 2-43 (MIFARE Ultralight C), 0-63 (MIFARE 1K), 0-255 (MIFARE 4K).

Note: these are decimal values. For UHF Gen2 tags this parameter is defined differently as explained in the *Memory organization of HF vs UHF Class1 Gen2 tags section*.

The ‘lock option’ values are as follows: 0 – Do not lock data
1 – Lock data*

Important Note: locking the data with the ‘lock option’ prevents further write access to that data and is irreversible. The data can still be read by the Host. All data is locked on a block wide (4 byte) basis. Make sure you have written all the data you want to a block before locking it.

***Note:** this ‘lock option’ feature does not work with MIFARE Ultralight C, MIFARE 1K or 4K tags. They can be locked by writing directly to their appropriate Lock Bytes. This bit has no meaning with UHF C1 Gen2 tags and should be set to 0.

The *data bytes* are the numbers or characters you want to store on the RF card. The data stream must be terminated by a carriage return or a Boca command starting delimiter character (<) unless an optional byte count is sent. **Note:** The data bytes are in binary (Hex) or ASCII format. The letter T would be sent as either ‘T’ or ‘54’ not as a ‘84’ which is the decimal value for character T.

Examples: The <RFW1,4,0>BOCA SYSTEMS(CR) command would be used to program 12 bytes starting at block 4. The 12 bytes would be comprised of values 42h,4Fh,43h,41h,20h,53h,59h,53h,54h,45h,4Dh,53h. The data is not locked.

The following two commands perform the same function using the optional byte count field. The <RFW1,8,0,4>TEST command would be used to program 4 bytes starting at block 8. The 4 bytes would be comprised of values 54h,45h,53h,54h. The data is not locked.

The <RFW2,8,0,4>54455354 command would be used to program 4 bytes starting at block 8. The 4 bytes would be comprised of values 54h,45h,53h,54h. The data is not locked.

The <RFW2,8,1>01020322(CR) command would be used to program 4 bytes starting at block 8. The 4 bytes would be comprised of values 01h,02h,03h,22h (Note it is using 2 byte ASCII format). The data would be locked in block 8.

The <RFW1,10,0>test<RFR1,10,4,1> command string writes the word ‘test’ at page 10 and reads it back to the Host. **Note:** the next command delimiter ends the data stream.

The difference between Binary mode and ASCII mode can be a little confusing. Another example may be helpful.

Binary format example - the <RFW1,4,0>12345678(CR) command would be used to program 8 bytes starting at block 4. The 8 bytes would be comprised of values of 31h,32h,33h,34h,35h,36h,37h,38h. The data is not locked.

If you sent the same command using ASCII format <RFW2,4,0>12345678(CR) it would write just 4 data bytes of 12h,34h,56h and 78h.

Special OTP (One-Time Programming) Block –

Block 3 (4 bytes) on the MIFARE Ultralight and Ultralight C card is a bit-wise modified, one-time only programming area that can be used for a 32 bit counter. The region is initially programmed to '0'. Each bit can be individually programmed to a '1' but once set, the process is irreversible. Each time you write data to this block it is bit-wise 'or-ed' with the current contents of the OTP bytes. You can also lock the entire block, thereby preventing any more accesses to this region.

The <RFW2,3,0>000000FF(CR) command would be used to program the 4 bytes of the OTP block with a counter value of 255 (FFh). The count would not be locked. Note this is using 2 byte ASCII format.

TEMPORARY RF KEY COMMAND - <RFK00,#,#,#,#,#,> Note: those are zeros after the RFK.
RFK00 will temporarily change key A. **RFK01** will change key B. Numbers # are hex values.

Note: This command format is for MIFARE 1K or 4K tags.

This command specifies what security key value should be used when authenticating a MIFARE 1K or 4K tag. All MIFARE 1K or 4K tags must be authorized before performing a read or write operation. This is handled automatically by the printer using the permanent flash value it has stored for key A.. Normally, a tag contains two keys (A and B) per sector that can be used for authentication. Since key B is usually readable, the printer initially uses key A. If the temporary Key command is sent, the printer will use that value for authentication.

The key functions are controlled by the access conditions set for each sector. Note: The key values and access condition bits for each sector can be changed by writing directly to the appropriate bytes in the sector trailer. Please refer to the MIFARE 1K or 4K specification for a more detailed description.

If key B is desired for authentication, then the access bits must be changed accordingly and the proper RFK01 command sent to the printer.

Warning: Writing incorrect information to these blocks can permanently disable the sector concerned.

MIFARE 1K/4K tags are shipped with a factory default key A value called the Transport Configuration. The printer uses the Philips Transport Key A value (FF,FF,FF,FF,FF,FF). If there are errors using the tag, try changing the key A value to another typical value like A0,A1,A2,A3,A4,A5.

Ex: change key A value to Infineon Transport Configuration before reading 16 bytes from block 1.
<RFK00,A0,A1,A2,A3,A4,A5><RFR1,1,16,1>

Ex: change key B value to B0,B1,B2,B3,B4,B5 before reading 16 bytes from block 2.
<RFK01,B0,B1,B2,B3,B4,B5><RFR1,2,16,1>

Alternate format of the TEMPORARY RF KEY COMMAND - <RFK#,>
Note: there are 16 bytes in the command.

Note: This command format is for sending the MIFARE Ultralight C 3DES key needed for authentication. The #'s must be hex values.

If an Ultralight C tag has been secured with a 3DES key it must be authorized before using depending on the settings of the AUTH0 and AUTH1 bytes. The key must be sent down and then a authenticate command (see next command) must be sent to access the card. If the same key is used for more than one tag it is not necessary to send the key command more than once as long as the printer has remained power on. The key is stored in RAM so it will be lost if the printer is turned off.

AUTHENTICATE RF TAG COMMAND - <RFA>

Note: This command is for authenticating the MIFARE Ultralight C using the 3DES key sent previously.

This command only needs to be sent once per tag unless an error occurred and the <RFC> command was issued. In that case, it must be sent again.

Example: if you want to read a series of tags that have been secured from reading and writing with a 3DES key of **000102030405060708090A0B0C0D0E0F** send the following commands.

<RFC><RFK00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F> Note: <RFC> is not required but good practice to start a series of RF commands. The key command is only sent once unless a new key is needed.

<RFA><RFR2,20,4,1> - authenticates tag before reading 4 bytes from block (page) 20 (14h). Data is sent back to the Host.

<RFR2,30,8,1> - tag is still authenticated from previous command. This reads 8 bytes from block (page) 30 (1Eh). Data is sent back to the Host.

Another Key Example: Assume tag secured with 3DES key of **BREAKMEIFYOUCAN!**
Send the following commands.

<RFC><RFK42,52,45,41,4B,4D,45,49,46,59,4F,55,43,41,4E,21> - Sends Key
<RFA><RFW1,8,0,4>Test - authenticates tag before writing 'Test' to block (page) 8.

SPECIAL UHF CLASS1 GEN2 COMMANDS:

LOCK RF TAG COMMAND - <RFTL#>

UHF C1 Gen2 tags banks can be access restricted using a special Lock command. This command sends a 20-bit Lock Payload value to the tag to restrict access as explained in the '*Lock Functionality*' section above. This command will automatically store the payload in the correct memory location on the tag. The # is the hex value for the 20-bit payload.

Only the reserved memory bank (access and kill passwords) can be both WRITE and READ locked - all others (EPC, TID, and User) can be WRITE-locked only. **Typically, the Tag Identification (TID) memory bank is perma-locked at the factory.**

IT IS STRONGLY RECOMMENDED THAT THE LOCK COMMAND AND THE ACCESS BITS STORED ON THE UHF TAG ARE NOT CHANGED UNTIL THEIR OPERATION IS FULLY UNDERSTOOD.

PASSWORD RF TAG COMMAND - <RFTP#####>

Once a tag has been secured with an access password, that password must be sent to the printer before any other secured operations. Once the password is sent, the printer will automatically perform this operation before any secure transactions. Therefore, it only needs to be sent once. The #'s represent the 32-bit (8 hex digit) kill password. The values can range from 0 - F.

Ex. <RFTP1234DEAD> - sends 32-bit access password 1234DEAD to printer

Note: the tag access password must be set in tag memory location x0002 before this command will work (explained in the '*Access Password*' section above).

KILL RF TAG COMMAND - <RFTK#####>

Killing a tag makes it unusable. It can no longer be read or written. The #'s represent the 32-bit (8 hex digit) kill password. The values can range from 0 - F.

Ex. <RFTK1234DEAD> - sends 32-bit kill password 1234DEAD to printer

Note: the tag kill password must be set in tag memory location x0000 before this command will work (explained in the '*Killing a Tag*' section above).

PERMANENT COMMANDS:

PERMANENT RF KEY COMMAND - <rfk00,#,#,#,#,#,>

This command is the same as the uppercase <RFK> command except it is permanently stored in flash. This command should only be sent once to configure the printer. If changing key A frequently, use the uppercase command. **Note:** only key A value can be permanently stored using this command.

RFID DISABLED COMMAND - <rfd>

This command disables the RFID mode for the printer and is permanently stored in flash. This can also be set using the 'Factory Menu'.

RFID ENABLED COMMAND - <rfe#>

This command has been modified to now select the type of tag being used as well as enabling the RFID mode for the printer. The command is permanently stored in flash. This can also be set using the 'Factory Menu'. The # value selects the type of tag being used. The values can range as shown below.

Note: Recommended for experienced users only. This feature is set at the factory and generally should not be changed.

- No value sent (original command <rfe>) – MIFARE Ultralight
- 0 – RFID disabled (same as <rfd>)
- 1 – MIFARE Ultralight
- 2 – I-code (SL2)
- 3 – Auto Detect M1*
- 4 – Auto Detect M4*
- 5 – Auto Dual M1*
- 6 – Auto Dual M4*
- 7 – Auto Detect M7 – not used at this time
- 8 – Auto P2 M1*
- 9 – Auto P2 M4*
- 10 – Auto P1 SL032*

* see 'AUTO DETECT OPTION FOR SINGLE OR DUAL PATH PRINTERS' for further explanation.

AUTO-DETECT OPTION FOR SINGLE OR DUAL PATH PRINTERS

The Auto-Detect option will automatically determine the tag type during initialization. Only the supported tag types listed previously for each reader are detected.

Single Path Printer Auto Detect

This mode is selected using the Auto Detect M1, M4 or SL032 RFID Menu option or <rfe#> command.

Dual Path Printer Auto Detect

A Dual Path RFID printer can only be configured with two readers of the same type. You can not mix different reader modules. Each path will have its own reader. These are configured with the Auto Dual RFID Menu options or <rfe#> command.

A Dual RFID printer uses the same RFID commands as a single path (standard) RFID printer. However, you must tell the printer for which path the RFID commands are intended. This is accomplished by sending a path command <P1> or <P2> before issuing any RFID commands. Once a path has been selected, you no longer need to send a path command before the RFID commands. In order to change paths, a new path command must be sent.

Ex: <P1><RFSN2,1> would read the serial number from the tag on Path1.

Ex: <P2><RFSN2,1> would read the serial number from the tag on Path2.

RFID RETRY COMMAND - <rfr#>

This command permanently stores the retry value (#) in flash. The value (#) can range from 1-5. The default value is 2. This is the number of times the printer will retry performing the requested RFID operation before printing a 'Void' ticket.

RFID TIMEOUT COMMAND - <rfto#>

This command permanently stores the timeout value (#) in flash. The value (#) can range from 1-256ms. The default value is 60ms. This is the period of time that the printer will wait for a response from the RFID encoder before timing out. This will vary among different stocks and RFID technologies. You should increase the timeout amount if 'timeout error' void tickets are being printed.

RFID Error Messages

<RFSN0>

STATUS

| <u>CODE - LCD Error Message</u> | <u>-</u> | <u>Explanation</u> |
|---------------------------------|----------|---|
| A - NO ERRORS | - | RFID operation(s) completed OK. |
| C - NON ASCII CHAR | - | Non-ASCII character in request. |
| C - BAD CRC | - | Bad CRC. |
| C - FLAGS DON'T MATCH | - | Flags don't match command or tag type. |
| C - UNKNOWN COMMAND | - | Unknown command. |
| C - BAD START BLK | - | Invalid Starting block. |
| C - BAD NUM BLKS | - | Invalid Number of blocks. |
| C - BAD MSG LEN | - | Invalid Message length. |
| R - READ TAG FAIL | - | Error reading card. |
| S - SELECT TAG FAILED | - | No card or multiple cards in the RFID field of the encoder. |
| T - CARD TIMEOUT | - | Card timeout error. |
| W - WRITE TAG FAIL | - | Error programming card. |
| Z - RFID ENCODER ERR | - | Could not establish communication with RFID encoder during Initialization (may not be connected). |