RFID-RC522

[**Descripción general** 3](#_Toc206515477)

[Etiquetas compatibles y capacidades 3](#_Toc206515478)

[Pinout y cableado 3](#_Toc206515479)

[Alimentación y consideraciones eléctricas 4](#_Toc206515480)

[Software y ejemplos 4](#_Toc206515481)

[Código base de lectura de UID (Arduino): 4](#_Toc206515482)

[Lectura de un bloque MIFARE Classic (ejemplo, sector 1 bloque 4) tras autenticación: 5](#_Toc206515483)

[Buenas prácticas, alcance y resolución de problemas 6](#_Toc206515484)

[**Qué es ISO/IEC 14443A** 7](#_Toc206515485)

[Las 4 partes del estándar 7](#_Toc206515486)

[Diferencias Tipo A vs Tipo B 7](#_Toc206515487)

[Ejemplos de etiquetas ISO/IEC 14443A 7](#_Toc206515488)

[Cómo es la comunicación 7](#_Toc206515489)

[Alcance y limitaciones 8](#_Toc206515490)

[**SPI** 9](#_Toc206515491)

[Arquitectura y señales 9](#_Toc206515492)

[Parámetros eléctricos y temporales clave 9](#_Toc206515493)

[Flujo de datos 9](#_Toc206515494)

[Topologías 10](#_Toc206515495)

[Ventajas técnicas 10](#_Toc206515496)

[Limitaciones 10](#_Toc206515497)

[**Tipos de tags** 11](#_Toc206515498)

[MIFARE Classic 1K / 4K 11](#_Toc206515499)

[MIFARE Ultralight / NTAG 11](#_Toc206515500)

[UID (Unique Identifier) 11](#_Toc206515501)

[**Operaciones** 12](#_Toc206515502)

[Lectura / escritura 12](#_Toc206515503)

[Autenticación por sector (MIFARE Classic) 12](#_Toc206515504)

[**Limitaciones** 13](#_Toc206515505)

[Frecuencia y estándar 13](#_Toc206515506)

[Seguridad de MIFARE Classic 13](#_Toc206515507)

[**Resumen visual de diferencias** 13](#_Toc206515508)

[***Esquema de memoria MIFARE Classic*** 14](#_Toc206515509)

[Contenido del sector trailer y por qué está así 14](#_Toc206515510)

[Access Bits: qué son y cómo se interpretan 14](#_Toc206515511)

[Qué permiten los Access Bits en la práctica 15](#_Toc206515512)

[Operaciones y flujo correcto en MIFARE Classic 15](#_Toc206515513)

[**Cómo planificar un sector sin equivocarte** 16](#_Toc206515514)

[***Ejemplo MIFARE CLassic 1k*** 17](#_Toc206515515)

[1. Estructura general 17](#_Toc206515516)

[2. Patrón de un sector 17](#_Toc206515517)

[3. Sector Trailer en detalle 17](#_Toc206515518)

[4. Ejemplo real de sector (hexadecimal) 17](#_Toc206515519)

[5. Ejemplo de permisos (Access Bits) 18](#_Toc206515520)

[6. Ejemplo de uso práctico 18](#_Toc206515521)

[7. Resumen visual (1K) 18](#_Toc206515522)

[***Cómo afectan los Access Bits a lectura y escritura*** 19](#_Toc206515523)

[Puntos clave: 19](#_Toc206515524)

[Ejemplos de impacto 19](#_Toc206515525)

[Cómo modificar los Access Bits en una tarjeta 19](#_Toc206515526)

# ***Descripción general***

El RC522 es un módulo lector/escritor RFID de 13.56 MHz basado en el chip MFRC522. Trabaja con etiquetas ISO/IEC 14443A (como MIFARE Classic y Ultralight) y se comunica principalmente por SPI, aunque el chip soporta también I2C y UART. Es popular por su bajo coste, tamaño compacto y buen soporte en Arduino. Su alcance típico es corto (2–5 cm), ideal para control de acceso, fichajes y pequeños sistemas de identificación.

## Etiquetas compatibles y capacidades

* **Tipo de tags**:
  + **MIFARE Classic 1K/4K**: memoria sectorizada con claves por sector (Key A/Key B).
  + **MIFARE Ultralight/NTAG**: menor memoria, orientadas a NDEF/simple data.
  + **UID**: tags con UID de 4 o 7 bytes (anticolisión y selección por cascada).
* **Operaciones**:
  + **Lectura/escritura** en bloques/ páginas según el tipo de tag.
  + **Autenticación** por sector (MIFARE Classic) usando Key A/B.
* **Limitaciones**:
  + **Solo 13.56 MHz, ISO14443A**. No sirve para 125 kHz (EM4100/EM4200) ni ISO15693.
  + **Seguridad de MIFARE Classic**: el cifrado Crypto-1 está roto; no almacenes datos sensibles sin capa adicional.

## Pinout y cableado

* **Tensión de trabajo**: 3.3 V (alimentación del módulo).
* **Interfaz típica**: SPI en Arduino/MCU.
* **Conexión recomendada** (Arduino UNO y ESP32 como referencia):

| **Pin módulo** | **Función** | **Arduino UNO** | **ESP32 (ejemplo)** |
| --- | --- | --- | --- |
| VCC | 3.3 V alimentación | 3.3 V | 3V3 |
| GND | Tierra | GND | GND |
| RST | Reset | D9 | GPIO 22 |
| SDA/SS | Chip Select (SPI) | D10 | GPIO 5 |
| SCK | Reloj SPI | D13 | GPIO 18 |
| MOSI | SPI MOSI | D11 | GPIO 23 |
| MISO | SPI MISO | D12 | GPIO 19 |
| IRQ | Interrupción | No usado | Opcional |

Consejo: mantén los cables cortos, evita pasar cerca de motores o cables de potencia, y separa la antena del módulo de superficies metálicas.

## Alimentación y consideraciones eléctricas

* **3.3 V estable**: el RC522 consume ~15–30 mA en operación; usa 3.3 V con buena regulación.
* **Niveles lógicos**: el chip es de 3.3 V. Muchos montajes con Arduino UNO (5 V) funcionan “directos” por tolerancias y resistencias en placa, pero lo seguro es:
  + **Hacia el RC522 (SS/SCK/MOSI/RST)**: usar conversión de nivel a 3.3 V.
  + **Desde el RC522 (MISO)**: sale a 3.3 V, lo lee bien un 5 V como HIGH.
* **IRQ y ahorro**: el pin IRQ permite despertar al micro o al lector para ahorrar energía en baterías.
* **Interferencias**: el alcance cae si hay metal o ruido EMI cerca; reubica la antena o añade ferrita/aislamiento.

## Software y ejemplos

* **Librería recomendada**: “MFRC522” de Miguel Balboa (Arduino/PlatformIO: lib\_deps = miguelbalboa/MFRC522).
* **Ejemplos útiles**:
  + **DumpInfo**: listar UID, tipo de tag y contenido (para pruebas).
  + **Read/Write**: leer/escribir bloques (MIFARE Classic requiere autenticación por sector).
  + **Clave por defecto**: muchos tags vienen con Key A = FF FF FF FF FF FF; cámbiala cuanto antes.

## Código base de lectura de UID (Arduino):

cpp

#include <SPI.h>

#include <MFRC522.h>

#define SS\_PIN 10

#define RST\_PIN 9

MFRC522 rfid(SS\_PIN, RST\_PIN);

void setup() {

Serial.begin(115200);

SPI.begin();

rfid.PCD\_Init();

Serial.println("Acerque una tarjeta...");

}

void loop() {

if (!rfid.PICC\_IsNewCardPresent() || !rfid.PICC\_ReadCardSerial()) return;

Serial.print("UID: ");

for (byte i = 0; i < rfid.uid.size; i++) {

if (rfid.uid.uidByte[i] < 0x10) Serial.print("0");

Serial.print(rfid.uid.uidByte[i], HEX);

Serial.print(i + 1 < rfid.uid.size ? ":" : "\n");

}

rfid.PICC\_HaltA();

rfid.PCD\_StopCrypto1();

}

## Lectura de un bloque MIFARE Classic (ejemplo, sector 1 bloque 4) tras autenticación:

cpp

#include <MFRC522.h>

MFRC522::MIFARE\_Key key;

byte block = 4;

byte buffer[18];

byte size = sizeof(buffer);

void initKey() {

// Key A por defecto

for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

}

void tryReadBlock() {

if (!rfid.PICC\_IsNewCardPresent() || !rfid.PICC\_ReadCardSerial()) return;

initKey();

// Autenticación con Key A en el bloque

if (rfid.PCD\_Authenticate(MFRC522::PICC\_CMD\_MF\_AUTH\_KEY\_A, block, &key, &(rfid.uid)) != MFRC522::STATUS\_OK) {

Serial.println("Fallo autenticando");

return;

}

if (rfid.MIFARE\_Read(block, buffer, &size) == MFRC522::STATUS\_OK) {

Serial.print("Bloque "); Serial.print(block); Serial.print(": ");

for (byte i = 0; i < 16; i++) {

if (buffer[i] < 0x10) Serial.print("0");

Serial.print(buffer[i], HEX); Serial.print(" ");

}

Serial.println();

} else {

Serial.println("Fallo leyendo bloque");

}

rfid.PICC\_HaltA();

rfid.PCD\_StopCrypto1();

}

Tip: no escribas en el “trailer block” del sector si no sabes exactamente lo que haces; podrías bloquear el sector.

## Buenas prácticas, alcance y resolución de problemas

* **Seguridad**:
  + **Cambia claves** de los sectores (Key A/B) y documenta dónde las usas.
  + **No guardes secretos** en MIFARE Classic; usa un backend seguro y trata el tag como un identificador.
  + **Evita “magic cards”** salvo entornos de laboratorio; no son para producción.
* **Alcance y rendimiento**:
  + **Distancia**: 2–5 cm típico; tarjetas suelen leerse mejor que llaveros pequeños.
  + **Orientación**: alinea la antena del tag con la bobina del lector.
  + **Entorno**: metal y otros lectores cercanos degradan el acoplamiento; reubica o apantalla.
* **Problemas comunes**:
  + **Sin detección**: revisa VCC=3.3 V, GND común, pines SPI correctos del hardware, y SS exclusivo.
  + **Lectura inestable**: cables largos, 5 V directos al chip, mala fuente; usa niveladores y cables cortos.
  + **Autenticación falla**: clave incorrecta o escribiste mal el “trailer block”; prueba DumpInfo y reconfigura claves.
  + **Conflicto SPI**: si compartes SPI con otros dispositivos, baja la velocidad SPI o asegura SS altos en inactivos.

Si quieres, lo aterrizamos ya mismo en tu repo: defino el pinout exacto que usas, te preparo el platformio.ini, un ejemplo de lectura/escritura con manejo de errores, y un README con diagramas. ¿Vas a trabajar con Arduino UNO, MEGA, ESP32 o ESP8266?

# ***Qué es ISO/IEC 14443A***

Es un estándar internacional que define cómo se comunican las tarjetas o llaveros de proximidad con un lector usando radiofrecuencia a 13,56 MHz. Lo publican la ISO y la IEC, y se utiliza en transporte público, control de accesos, pagos sin contacto y NFC.

## Las 4 partes del estándar

1. **Características físicas**
   * Tamaño y forma (normalmente como tarjeta de crédito, pero también puede ser llavero o etiqueta).
   * Resistencia mecánica, temperatura de trabajo y tolerancia a campos eléctricos y magnéticos.
2. **Interfaz de señal y potencia RF**
   * Describe cómo el lector envía energía y datos a la tarjeta.
   * En Tipo A, la señal se modula con ASK al 100 % (variar la intensidad del campo para transmitir bits).
   * La tarjeta responde usando codificación Manchester (cambios de señal que representan 0 y 1).
3. **Inicialización y anticollision**
   * Procedimiento para detectar una o varias tarjetas y seleccionar una sin interferencias.
   * El lector pregunta y las tarjetas responden en turnos hasta que queda una seleccionada.
4. **Protocolo de transmisión (T=CL)**
   * Define cómo se envían los datos una vez que la tarjeta está seleccionada.
   * Incluye bloques de datos, tiempos de espera y métodos para encadenar mensajes.

## Diferencias Tipo A vs Tipo B

* Tipo A: ASK 100 %, codificación Manchester, usado en MIFARE y NFC.
* Tipo B: Modulación diferente (BPSK o NRZ), ASK al 10 %.
* Ambos usan el mismo protocolo de alto nivel (T=CL).

## Ejemplos de etiquetas ISO/IEC 14443A

* MIFARE Classic 1K / 4K – Memoria dividida en sectores con claves A/B.
* MIFARE Ultralight / NTAG – Menos memoria, ideales para NFC y datos simples.
* Tarjetas UID – Solo identificador único, sin memoria de usuario.

## Cómo es la comunicación

1. El lector crea un campo electromagnético a 13,56 MHz.
2. La tarjeta obtiene energía de ese campo por inducción, sin batería.
3. El lector varía la intensidad del campo (ASK) para enviar bits.
4. La tarjeta cambia la carga de su antena para “reflejar” datos al lector.
5. Se aplica el proceso de anticollision para manejar múltiples tarjetas.
6. Se intercambian datos (lectura/escritura o envío del UID).
7. El lector finaliza la comunicación antes de pasar a otra tarjeta.

## Alcance y limitaciones

* Distancia típica: 2–5 cm.
* Velocidad: ~106 kbps (puede ser mayor en algunos chips).
* Seguridad: El estándar no define cifrado; depende del chip (por ejemplo, MIFARE Classic usa Crypto-1, actualmente vulnerable).
* Compatibilidad: Solo con tags ISO14443A; no funciona con 125 kHz ni ISO15693.

# ***SPI***

## Arquitectura y señales

SPI es un bus **síncrono** y **full‑dúplex** que sigue una arquitectura **maestro–esclavo**. El maestro controla:

* **SCLK** (*Serial Clock*): frecuencia y polaridad del reloj.
* **MOSI** (*Master Out, Slave In*): datos del maestro al esclavo.
* **MISO** (*Master In, Slave Out*): datos del esclavo al maestro.
* **SS/CS** (*Slave Select / Chip Select*): línea activa en bajo para habilitar un esclavo concreto.

Cada esclavo debe poner su salida MISO en alta impedancia cuando no está seleccionado para evitar conflictos en el bus.

## Parámetros eléctricos y temporales clave

* **Frecuencia de reloj**: depende de las especificaciones del periférico; el RC522 soporta hasta 10 MHz en SPI.
* **Polaridad (CPOL)** y **fase (CPHA)**: definen en qué flanco del reloj se muestrean y cambian los datos. Hay cuatro modos posibles:
  + Modo 0: CPOL=0, CPHA=0
  + Modo 1: CPOL=0, CPHA=1
  + Modo 2: CPOL=1, CPHA=0
  + Modo 3: CPOL=1, CPHA=1 El RC522 usa normalmente **modo 0**.
* **Nivel lógico**: típicamente 3,3 V o 5 V; si maestro y esclavo trabajan a tensiones distintas, se requiere adaptación de nivel.
* **Setup time / Hold time**: tiempo mínimo que los datos deben estar estables antes y después del flanco de muestreo.

## Flujo de datos

Internamente, maestro y esclavo usan **registros de desplazamiento** de igual longitud (normalmente 8 bits). En cada ciclo de reloj:

1. El maestro desplaza un bit por MOSI hacia el esclavo.
2. Simultáneamente, el esclavo desplaza un bit por MISO hacia el maestro.
3. Tras 8 ciclos, ambos han intercambiado un byte completo.

Esto forma un **buffer circular inter‑chip**: lo que entra por un extremo sale por el otro en tiempo real.

## Topologías

* **Multiesclavo con líneas SS independientes**: cada esclavo tiene su propio pin de selección.
* **Daisy chain**: la salida de un esclavo alimenta la entrada del siguiente; todos comparten reloj y selección, pero los datos se encadenan.

## Ventajas técnicas

* **Baja latencia**: no hay direccionamiento ni confirmaciones obligatorias.
* **Ancho de banda alto**: limitado solo por la frecuencia de reloj y la integridad de señal.
* **Flexibilidad de palabra**: no está restringido a 8 bits; se pueden enviar 16, 24 o más bits por transacción.

## Limitaciones

* **Distancia corta**: pensado para comunicación en placa o entre placas cercanas (pocos decímetros).
* **Sin control de errores integrado**: si se requiere, debe implementarse a nivel de aplicación.
* **Más pines**: comparado con I²C, consume más líneas de E/S.

# ***Tipos de tags***

## MIFARE Classic 1K / 4K

* **Memoria**:
  + **1K** → 1024 bytes organizados en 16 sectores × 4 bloques × 16 bytes por bloque.
  + **4K** → 4096 bytes organizados en 40 sectores (32 sectores de 4 bloques y 8 sectores de 16 bloques).
* **Sector Trailer**: el último bloque de cada sector guarda:
  + **Key A** (6 bytes)
  + **Bits de acceso** (3 bytes + 1 byte de control)
  + **Key B** (6 bytes, opcional o usado como datos)
* **Protección**: cada sector puede tener reglas distintas para lectura/escritura según se autentique con Key A o Key B.
* **Uso típico**: control de accesos, transporte público, monederos electrónicos.

## MIFARE Ultralight / NTAG

* **Memoria**: mucho menor que Classic; organizada en **páginas** de 4 bytes.
  + Ejemplo: NTAG213 → 144 bytes de usuario; NTAG216 → 888 bytes.
* **Orientación**: diseñadas para almacenar datos simples o mensajes NFC en formato **NDEF** (URLs, texto, vCard, etc.).
* **Protección**: no usan Crypto-1; pueden tener contraseñas simples o bits de bloqueo permanentes.
* **Uso típico**: etiquetas NFC para marketing, tickets de un solo uso, identificación rápida.

## UID (Unique Identifier)

* **Longitud**: 4 bytes (Single Size UID) o 7 bytes (Double Size UID) según ISO14443A.
* **Anticolisión**:
  + Si hay varias tarjetas en el campo, el lector ejecuta un protocolo de **anticolisión** para identificar una a la vez.
  + En UIDs largos, se usa **selección por cascada**: el lector obtiene la primera parte del UID, detecta un “Cascade Tag” (0x88) y sigue pidiendo la siguiente parte hasta completarlo.
* **Función**: identificar de forma única la tarjeta, aunque en algunos modelos el UID puede no ser realmente único (NUID).

# ***Operaciones***

## Lectura / escritura

* **MIFARE Classic**:
  + Se trabaja por **bloques** de 16 bytes.
  + Antes de acceder a un bloque, hay que autenticar con la clave del sector correspondiente.
* **Ultralight / NTAG**:
  + Se trabaja por **páginas** de 4 bytes.
  + No requieren autenticación compleja, salvo que se configure contraseña.

## Autenticación por sector (MIFARE Classic)

* **Proceso**:
  1. El lector envía comando de autenticación (0x60 para Key A, 0x61 para Key B) junto con el número de bloque.
  2. Se usa un reto–respuesta cifrado con **Crypto-1** para verificar que lector y tarjeta comparten la misma clave.
  3. Si es correcto, se habilitan las operaciones permitidas según los bits de acceso.
* **Key A / Key B**:
  1. Pueden dar permisos diferentes (por ejemplo, Key A solo lectura, Key B lectura/escritura).
  2. Se almacenan en el sector trailer y deben cambiarse desde el valor por defecto FF FF FF FF FF FF para evitar accesos no autorizados.

# ***Limitaciones***

## Frecuencia y estándar

* Solo funcionan a **13,56 MHz** bajo **ISO/IEC 14443A**.
* No son compatibles con:
  + **125 kHz** (tags como EM4100/EM4200, que usan otro estándar y tecnología de baja frecuencia).
  + **ISO15693** (etiquetas de alta frecuencia pero de largo alcance, hasta ~1,5 m).

## Seguridad de MIFARE Classic

* Usa el cifrado propietario **Crypto-1** (stream cipher).
* Vulnerabilidades:
  + Algoritmo roto desde 2008; ataques prácticos permiten clonar o leer datos en segundos.
  + Investigaciones recientes han encontrado incluso **backdoors** que permiten leer todo el contenido sin conocer las claves.
* **Recomendación**: no almacenar datos sensibles sin una capa adicional de cifrado (AES, etc.) o usar tags más seguros (MIFARE DESFire, NTAG con autenticación).

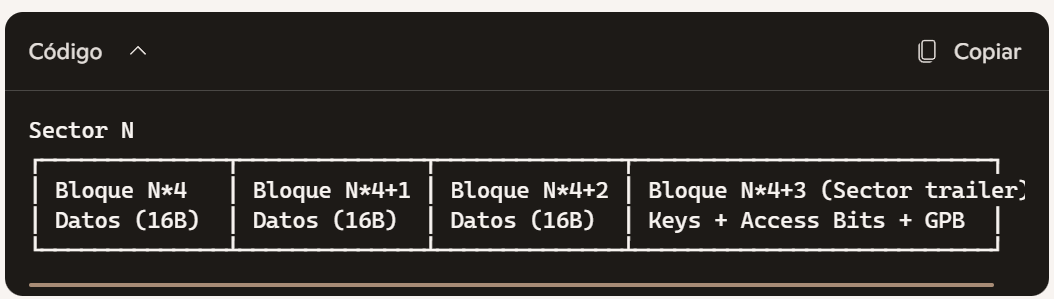
## Resumen visual de diferencias

| **Tipo de tag** | **Memoria** | **Organización** | **Seguridad principal** | **Uso típico** |
| --- | --- | --- | --- | --- |
| MIFARE Classic 1K | 1024 bytes | Sectores/bloques | Crypto-1 (roto) | Accesos, transporte |
| MIFARE Classic 4K | 4096 bytes | Sectores/bloques | Crypto-1 (roto) | Monederos, control |
| MIFARE Ultralight | 48–128 bytes | Páginas | Contraseña opcional | Tickets, NFC simple |
| NTAG213/215/216 | 144–888 bytes | Páginas | Contraseña opcional | NFC, marketing |
| UID 4/7 bytes | N/A | N/A | Ninguna (solo ID) | Identificación |

# ***Esquema de memoria MIFARE Classic***

* **Tamaño de bloque**: 16 bytes.
* **Sector trailer**: el último bloque de cada sector. Contiene claves y permisos.
* **MIFARE Classic 1K**:
  + 16 sectores × 4 bloques = 64 bloques.
  + En cada sector: bloques 0–2 datos, bloque 3 sector trailer.
* **MIFARE Classic 4K**:
  + Sectores 0–31: como 1K (4 bloques c/u).
  + Sectores 32–39: 16 bloques por sector (los 15 primeros de datos, el último sector trailer).

Diagrama ASCII (1K, patrón por sector):

Nota importante: En el sector 0, el bloque 0 es “manufacturer block” (incluye UID y BCC) y no es escribible.

## Contenido del sector trailer y por qué está así

* **Bytes 0–5: Key A (6 bytes)** Motivo: clave de acceso para el sector. Nunca es legible como datos en condiciones normales.
* **Bytes 6–8: Access Bits (3 bytes)** Motivo: codifican los permisos de cada bloque del sector y del propio trailer, con redundancia para detectar errores.
* **Byte 9: GPB (General Purpose Byte)** Motivo: byte de propósito general del sector trailer; su uso depende de la configuración de permisos.
* **Bytes 10–15: Key B (6 bytes)** Motivo: segunda clave del sector. Puede usarse como clave o, en ciertas configuraciones, tratarse como datos legibles.

Suma total: 6 + 3 + 1 + 6 = 16 bytes ajustados exactamente al tamaño del bloque.

## Access Bits: qué son y cómo se interpretan

* **Qué representan**: para cada bloque del sector (3 de datos + 1 trailer) hay un trío de bits lógicos: **C1, C2, C3**.
  + Esos tres bits eligen una fila de una tabla de permisos: quién puede leer, escribir, incrementar/decrementar (en bloques de valor) o modificar el sector trailer.
* **Redundancia y comprobación**: los 12 bits lógicos (4 bloques × 3 bits) se guardan con redundancia e inversión en los bytes 6, 7 y 8.
  + Cada bit lógico se almacena dos veces: una “normal” y otra “invertida”.
  + Si no coinciden (normal ≠ not(invertido)), el sector queda con permisos inválidos/bloqueado.
* **Idea práctica**: en vez de memorizar el mapa bit a bit, piensa así:
  + Para cada bloque i ∈ {0,1,2,3} obtienes un trío (C1i, C2i, C3i).
  + Ese trío indexa la política de acceso de ese bloque.
  + Existe un trío especial para el trailer (i=3) que controla lectura/escritura de claves y acceso a los bits de permiso.

Tip: muchas librerías (p. ej., MFRC522 en Arduino) ofrecen utilidades o ejemplos para calcular y verificar estos bytes. Aun así, siempre comprueba la coherencia normal/invertido antes de grabar el trailer para evitar “brickear” el sector.

## Qué permiten los Access Bits en la práctica

* **Bloques de datos**:
  + Pueden quedar “abiertos” (leer y escribir con Key A o Key B).
  + “Solo lectura” (leer con A/B; escribir prohibido).
  + “Lectura con A; escritura con B” (típico cuando separas roles).
  + “Bloques de valor” (formato especial que permite operaciones atómicas de incrementar, decrementar, restaurar, transferir). Estas operaciones solo funcionan si el bloque está en el modo “valor”.
* **Sector trailer**:
  + Permite combinar: quién puede leer los bits de acceso, quién puede escribirlos, y si **Key B** se trata como una clave (no legible) o como “datos” legibles (desaconsejado para seguridad).
  + Configuraciones seguras típicas: Key A nunca legible; Key B no legible; los bits de acceso solo reconfigurables con la clave adecuada (normalmente Key B).

Regla de oro: primero planifica la política de cada sector, calcula los bytes de acceso y escribe el trailer en un solo paso con la clave correcta. Cambiar los bits de acceso en el orden o con valores equivocados puede dejarte sin acceso.

## Operaciones y flujo correcto en MIFARE Classic

* **Autenticación por sector**:
  + Selecciona el bloque objetivo (dentro del sector).
  + Autentica con 0x60 (Key A) o 0x61 (Key B) contra ese bloque. Si pasa, “entras” con esa clave a todo el sector bajo los permisos definidos.
* **Lectura/escritura**:
  + Datos: en bloques de 16 bytes (respetando permisos).
  + Trailer: escribe de una sola vez Key A, Access Bits, GPB y Key B para asegurar coherencia.
* **Bloques de valor**:
  + Tienen un formato redundante interno (valor y su complemento, dirección y su complemento).
  + Las operaciones incrementar/decrementar/transfer/restore solo funcionan si el bloque está en ese modo.

**Cómo planificar un sector sin equivocarte**

1. Define tu política por cada bloque:
   * Datos críticos: lectura con A, escritura solo con B.
   * Datos públicos: lectura A/B, escritura con B.
   * Inmutables: solo lectura (nadie escribe).
2. Define la política del trailer:
   * Key A no legible; Key B no legible; bits de acceso reconfigurables solo con B.
3. Calcula los Access Bits:
   * Obtén los tríos (C1,C2,C3) por bloque según la tabla de permisos oficial.
   * Construye los bytes 6–8 con su redundancia invertida y verifica coherencia.
4. Programa el trailer en un solo write:
   * Key A (bytes 0–5) + Access Bits (6–8) + GPB (9) + Key B (10–15).
5. Prueba:
   * Autentica con A y verifica lecturas/escrituras esperadas.
   * Autentica con B y verifica las operaciones restringidas.

Si quieres, dime la política exacta que buscas (qué quieres permitir en cada bloque y cómo de protegido quieres el trailer) y te devuelvo:

* Los tríos (C1,C2,C3) por bloque.
* Los bytes 6–8 ya calculados y verificados.
* Un ejemplo de comandos de lectura/escritura y autenticación para probarlo.

# ***Ejemplo MIFARE CLassic 1k***

## 1. Estructura general

**MIFARE Classic 1K**

* 16 sectores.
* Cada sector tiene **4 bloques** de 16 bytes.
* Total: 16 × 4 × 16 bytes = 1024 bytes.

**MIFARE Classic 4K**

* 40 sectores.
* Sectores 0–31: 4 bloques cada uno.
* Sectores 32–39: 16 bloques cada uno.
* Total: 4096 bytes.

## 2. Patrón de un sector

En **cualquier sector**, el último bloque es **Sector Trailer** (contiene las claves y permisos). Ejemplo para un sector de 4 bloques:

Código

Bloque 0 → Datos (16 bytes)

Bloque 1 → Datos (16 bytes)

Bloque 2 → Datos (16 bytes)

Bloque 3 → Sector Trailer (Key A, Access Bits, GPB, Key B)

En sectores grandes (16 bloques), el patrón es igual: el último bloque es el trailer.

## 3. Sector Trailer en detalle

* **Bytes 0–5**: Key A (6 bytes) → clave de autenticación.
* **Bytes 6–8**: Access Bits (3 bytes) → permisos de lectura/escritura.
* **Byte 9**: GPB (General Purpose Byte) → uso general o configuración.
* **Bytes 10–15**: Key B (6 bytes) → segunda clave o datos.

## 4. Ejemplo real de sector (hexadecimal)

Supongamos que leemos el **sector 1** y obtenemos:

Código

Bloque 4: 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10

Bloque 5: 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20

Bloque 6: 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30

Bloque 7: FF FF FF FF FF FF FF 07 80 69 00 00 00 00 00 00

Interpretación del bloque 7 (Sector Trailer):

* **Key A**: FF FF FF FF FF FF (clave por defecto, insegura).
* **Access Bits**: FF 07 80 → definen permisos para los bloques 4, 5, 6 y el trailer.
* **GPB**: 69 → valor configurable.
* **Key B**: 00 00 00 00 00 00 (aquí no configurada).

## 5. Ejemplo de permisos (Access Bits)

Supongamos que para un bloque de datos tienes el trío de bits **C1=0, C2=0, C3=0**:

* Lectura y escritura permitidas con Key A o Key B.

Si el trío es **C1=1, C2=0, C3=0**:

* Lectura con Key A o Key B, escritura solo con Key B.

Para el trailer, un trío como **C1=0, C2=0, C3=0**:

* Key A no legible, Key B no legible, Access Bits modificables con Key B.

## 6. Ejemplo de uso práctico

**Caso: Control de acceso**

* Sector 1:
  + Bloques 4 y 5 → datos del usuario (nombre, ID).
  + Bloque 6 → fecha de alta y estado.
  + Trailer (bloque 7) → Key A para lectura en tornos, Key B para escritura desde el sistema central.

**Caso: Monedero electrónico**

* Sector 2:
  + Bloque 8 → valor actual (formato “value block” con redundancia).
  + Bloque 9 → copia de seguridad del valor.
  + Bloque 10 → contador de recargas.
  + Trailer (bloque 11) → claves para operaciones de débito/crédito.

## 7. Resumen visual (1K)

Código

Sector 0: [UID+Fabricante] [Datos] [Datos] [Trailer]

Sector 1: [Datos] [Datos] [Datos] [Trailer]

...

Sector 15: [Datos] [Datos] [Datos] [Trailer]

# ***Cómo afectan los Access Bits a lectura y escritura***

En MIFARE Classic, **cada patrón de C1, C2 y C3** determina:

* **Quién puede leer** (Key A, Key B o ninguno)
* **Quién puede escribir** (Key A, Key B o nadie)
* En bloques tipo *value*, quién puede **incrementar/decrementar**
* En el **Sector Trailer**, quién puede leer/escribir claves y Access Bits

## Puntos clave:

1. **Restricción total**: si configuras un patrón que prohíbe escritura y lectura a ambas claves, el bloque queda “bloqueado para siempre” (irrecuperable sin ataque físico).
2. **Solo lectura**: ideal para datos críticos como UID lógico, configuración fija o valores de fábrica.
3. **Lectura con A/B y escritura con B**: patrón más común para datos dinámicos, porque permite que varias aplicaciones lean pero sólo una clave (B) pueda actualizar.
4. **Value Blocks**: para saldos o contadores, los Access Bits determinan quién puede incrementar, decrementar o transferir valores.

## Ejemplos de impacto

| **C1** | **C2** | **C3** | **Uso típico** | **Lectura** | **Escritura** |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | Datos generales editables | A o B | Solo B |
| 0 | 1 | 0 | Solo lectura | A o B | Nadie |
| 1 | 0 | 0 | Bloque valor, solo B actualiza | A o B | Solo B |
| 1 | 1 | 1 | Bloque bloqueado | Nadie | Nadie |

## Cómo modificar los Access Bits en una tarjeta

1. **Autenticación** Antes de acceder al Sector Trailer, debes autenticarte con **Key A** o **Key B** que tenga permiso para escribirlo.
2. **Lectura de Sector Trailer**
   * Lee los bytes 6, 7, 8 (Access Bits) y 9 (control) para conocer la configuración actual.
   * Recuerda que están **invertidos y en orden cruzado**.
3. **Cálculo de nuevo patrón**
   * Decide permisos para cada bloque.
   * Convierte C1, C2, C3 en bits reales siguiendo la tabla de NXP.
   * Calcula los bytes con bits invertidos y control redundante.
4. **Escritura de Sector Trailer**
   * Inserta: **Key A (6 bytes)** + **3 bytes Access Bits** + **1 byte de control** + **Key B (6 bytes)**.
   * Graba todo de una vez en el bloque trailer.
5. **Verificación**
   * Vuelve a leer el Sector Trailer para confirmar que el patrón quedó correctamente.
   * Prueba lectura/escritura en los bloques con ambas claves para validar permisos.