Application Report
SLOA213–December 2014

MIFARE DESFire EV1 AES Authentication With TRF7970A

Ralph Jacobi and Josh Wyatt

Safety and Security (S2) NFC/RFID Applications

ABSTRACT

MIFARE® DESFire® EV1 is an ISO14443A RFID transponder and an NFC Type 4A Tag Platform that is used in many NFC and RFID security applications due to the ability it has to operate in the clear or operate as a secure transponder using any one of three different types of encryption: Single DES, Triple DES, or AES.

In general, AES is viewed as the most secure level of encryption of the methods listed above, and therefore TI has developed a software library for demonstrating the use of AES encryption in RFID applications.

Using this library along with an MSP430G2553 LaunchPad™ Development Kit combined with a DLP-7970ABP BoosterPack™ Plug-in Module, TI can now provide a demonstration of a cost-effective reader system solution for AES authentication applications that use MIFARE DESFire EV1 tags. Common application examples are digital door locks, access control, closed- or open-loop authentication systems, and prepayment or micropayment systems.

Sample code described in this document can be downloaded from http://www.ti.com/lit/zip/sloa213.

LaunchPad, BoosterPack are trademarks of Texas Instruments.
MIFARE, DESFire are registered trademarks of NXP Semiconductors.
All other trademarks are the property of their respective owners.

Contents

List of Figures

List of Tables

1 Introduction

The MIFARE DESFire EV1 (MFDFEV1) tags are ISO14443A transponders / NFC Type 4A Tag Platforms. They can function with three different modes of encryption: Single DES (DES), Triple DES (3DES), and Advanced Encryption Standard (AES). The 3DES method can use one, two, or three keys as well (3DES, 2K3DES, and 3KDES respectively).

Since AES has is viewed to have superior safety and security due to its encryption method, this report will cover how to implement an AES authentication process for MFDFEV1 tags. The provided firmware will not support the authentication of a MFDFEV1 tag that is not functioning in AES mode. Any such tags would need to be personalized and placed into AES mode to be used for this specific design.

2 Near Field Communication/Radio Frequency Identification (NFC/RFID)

To communicate over the air with any MIFARE DESFire EV1 tags, an NFC or RFID transceiver module is required. All NFC/RFID communication from the reader to the tag occurs at a frequency of 13.56 MHz.

The MFDFEV1 tags follow the ISO14443A standard, which specifies how to go through the tag detection, anticollision, and selection processes for such tags. Figure 1 shows the basic flow of the example firmware.

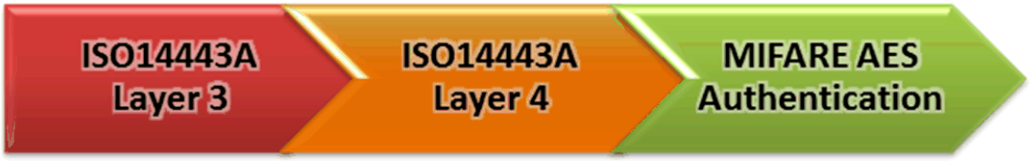

Figure 1. Communication Structure

The tag detection and anticollision processes (layer 3) are handled according to the ISO/IEC 14443-3 standard, and the selection process (layer 4) is handled according to the ISO/IEC 14443-4 standard. When the selection process is completed, the data is exchanged between the NFC Reader using the frame format defined in the ISO/IEC 14443-4 standard, using commands from ISO/IEC 7816-4 or the DESFire APDU frame format.

2.1 TRF7970A Reader Mode

The TRF7970A can be used as an NFC Reader for bitrates of 106 kbps (*fc*/128), 212 kbps (*fc*/64), 424 kbps (*fc*/32), and 848 kbps (*fc*/16). When the transceiver is in default mode [which is ISO mode (see the TRF7970A data sheet Section 5.9.6 Direct Mode for more information)] only the decoded data is available to the MCU, and it can be accessed through the FIFO.

Because the TRF7970A supports higher bitrates as a Reader, and the MFDFEV1 tags all support those higher bitrates as well, it is important to understand that as the bitrate speeds increase, the read range for the tag decreases. To maximize the range at which a MFDFEV1 tag can be read, it is recommended to leave the bitrate at 106 kbps, which is done in this application. If the use of a higher bitrate is desired, then it must be set through the use of the PPS command during the selection process.

2.2 Tag Detection/Polling

To poll for any Type 4A tag platforms either a REQA (also called SENS_REQ) or a WUPA (also called ALL_REQ) must be issued. When a reply (ATQA or SENS_RES) is received, the anticollision process begins.

2.3 Anticollision

The anticollision process serves two main purposes. One is for the Reader to place a single tag into an activated state if multiple tags are placed in the vicinity, and the second is to acquire the Unique Identifier (UID) and Select Acknowledge (SAK) of that tag. There are three cascade levels used for the ISO14443A anticollision process in order to resolve any collisions and determine the UID of the tag. After the SAK response is received, the tag is in an activated state but has not been fully selected yet. See Figure 2 for the flow diagram of the anticollision process.

2.4 Selection

The selection process is what finalizes the selection of a tag that has been placed in an activated state. This is done by issuing a Request for Answer to Select (RATS) command. If a reply (ATS) is received, then the tag is now fully selected and ready to exchange data with the Reader module. After the tag is in the selected state, a PPS command can be issued to change the data rate of the communication between the reader and the tag.

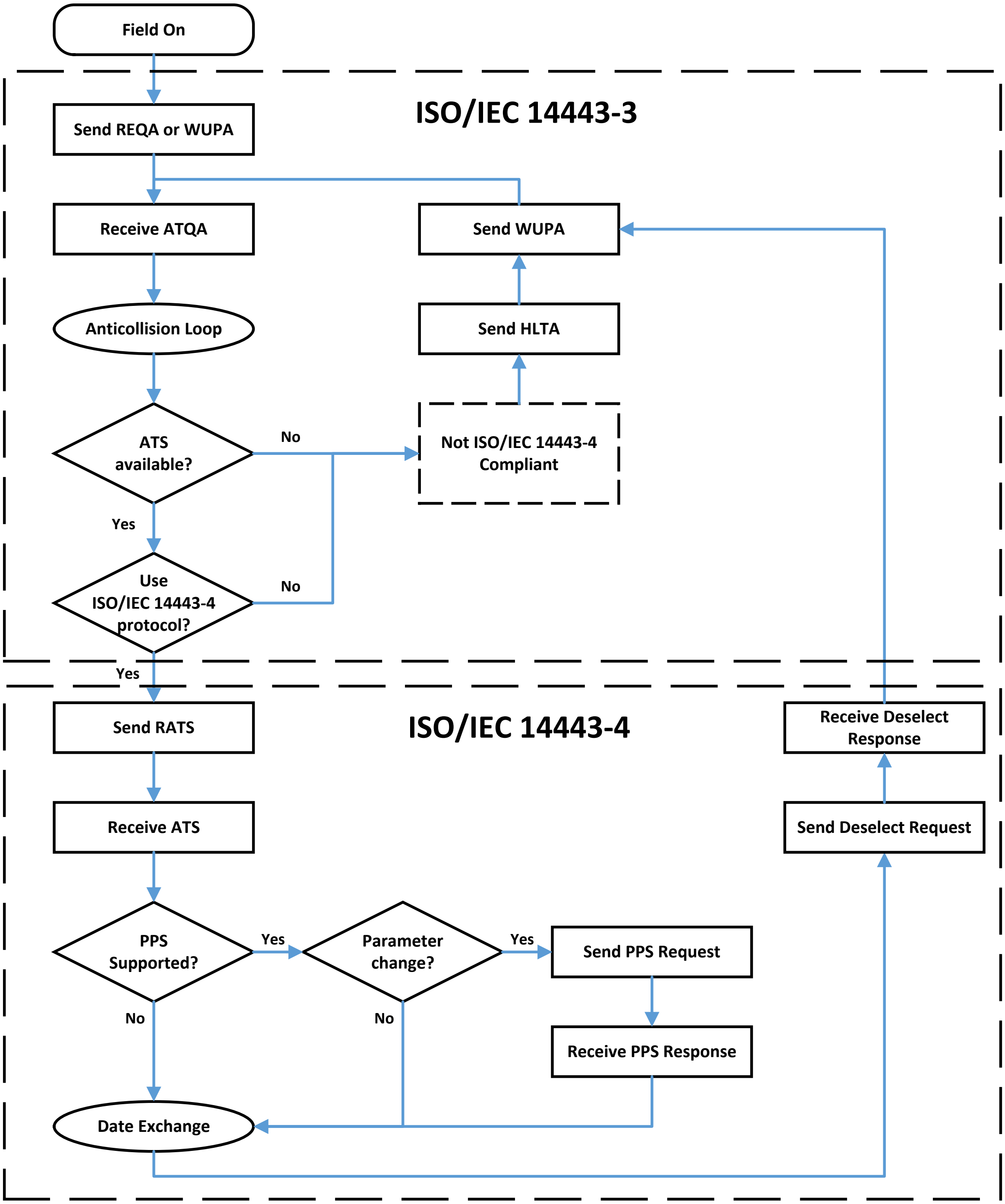

Figure 2. ISO 14443 Type A Anticollision

2.5 Data Exchange

After the anticollision and selection processes are completed the data exchange layer is entered. For MIFARE MFDFEV1 tags, commands can be delivered based on either the DESFire APDU frame format, or the ISO/IEC 7816-4 frame format. The format of the first APDU received determines the subsequent frame formatting. For the provided example firmware, the DESFire APDU frame format is used and will be the only format covered in detail.

2.5.1 Frame Format for DESFire APDUs

The frame format for DESFire APDUs is based on only the ISO 14443-4 specifications for block formats. This is the format used by the example firmware, and seen in Figure 3.

- PCB – Protocol Control Byte, this byte is used to transfer format information about each PDU block.
- CID – Card Identifier field, this byte is used to identify specific tags. It contains a 4 bit CID value as well as information on the signal strength between the reader and the tag.
- NAD – Node Address field, the example firmware does not support the use of NAD.
- DESFire Command Code – This is discussed in the next section.
- Data Bytes – This field contains all of the Data Bytes for the command.

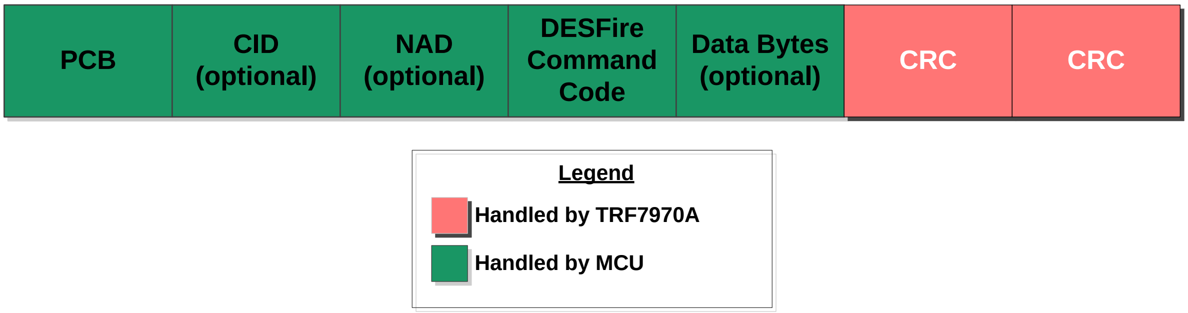

Figure 3. Frame Format for Data Exchange Commands Using DESFire APDUs

2.5.2 DESFire Commands

There are a number of command codes used by MFDFEV1 tags in order to modify them, but as they are application dependent only the AES Authenticate command has been implemented in the example firmware. Table 1 contains a short summary of other useful DESFire commands that may be implemented on an application specific basis. The actual command codes are provided within the MFDFEV1 specifications.

Table 1. DESFire Commands

Command Function	Summary
AES Authenticate	Used to begin AES Authentication process
Change Key Setting	Used to change the master key settings
Change Key	Used to change the stored keys and the encryption mode that is used
Get Application IDs	Used to fetch the Application ID's for all applications within the tag
Select Application	Used to select a specific application
Get File IDs	Used to fetch the File ID's for all files in the currently selected application.
Read Data	Used to read data from a specific file in the tag
Write Data	Used to write data to a specific file in the tag

3 AES Authentication

The AES authentication process is a multiple step sequence in which the NFC/RFID Reader and the MFDFEV1 tag exchange encrypted data to verify that they share the same key. During this process, a session key will be created which is used for certain commands such as the Change Key command. Figure 4 shows a summary of the AES authentication process.

MFDFEV1 tags can have multiple keys stored within them. One of the keys stored is known as the master key, and there are some DESFire commands which require that the master key is used for the authentication process.

3.1 Authentication Process

1. The Reader will issue the AES authentication command along with a key number. This will tell the MFDFEV1 tag which AES key to use. A key number of 0x00 is used to indicate the master key.
2. The tag will select the AES key indicted by the command, generate a 16 byte Random Number B (RndB), and encrypt RndB with the selected AES key. It will then reply to the command by transmitting the encrypted data packet.
3. The Reader will receive the reply and go through the following process:
   (a) Decrypt the reply with the AES key, which gives the Reader/Write the RndB that was generated by the tag.
   (b) Generate a 16 byte Random Number A (RndA), or use a pre-generated RndA.
   (c) Rotate RndB to the left by 8 bits (1 byte), which gets RndB'.
   (d) Concatenates RndA and RndB' together to create a new 32 byte value.
   (e) Encrypt the resulting 32 byte value with the AES key.
   (f) Transmit the resulting packet to the MFDFEV1 tag.
4. The tag will receive the command and go through the following process:
   (a) Decrypt the command with the AES key.
   (b) Split the 32 byte value to get the separate 16 byte values for RndA and RndB'.
   (c) Generate RndB' by rotating the RndB the tag generated to the left by 8 bits.
   (d) Compare the received RndB' to the generated RndB'.

       If they match, then the packet was received correctly and the tag now has the RndA that was generated by the Reader.
   (e) Rotate RndA to the left by 8 bits (1 byte), which gets RndA'.
   (f) Encrypt the 16 byte RndA' value with the AES key.
   (g) Transmit the resulting packet back to the Reader.
5. The Reader will receive the reply and go through the following process:
   (a) Decrypt the reply with the AES key, acquiring the RndA' value.
   (b) Generate RndA' by rotating the RndA that was generated to the left by 8 bits.
   (c) Compared the received RndA' to the generated RndA'. If they match, then the authentication process is considered to be successful.
   (d) At this time, a 16 byte AES session key is generated using the bytes of RndA, RndA', RndB, and RndB':

$$\boxed{\textbf{AES Session Key} = \textbf{RndA}_{(\text{byte } 0\text{-}3)} + \textbf{RndB}_{(\text{byte } 0\text{-}3)} + \textbf{RndA}_{(\text{byte } 12\text{-}15)} + \textbf{RndB}_{(\text{byte } 12\text{-}15)}}$$

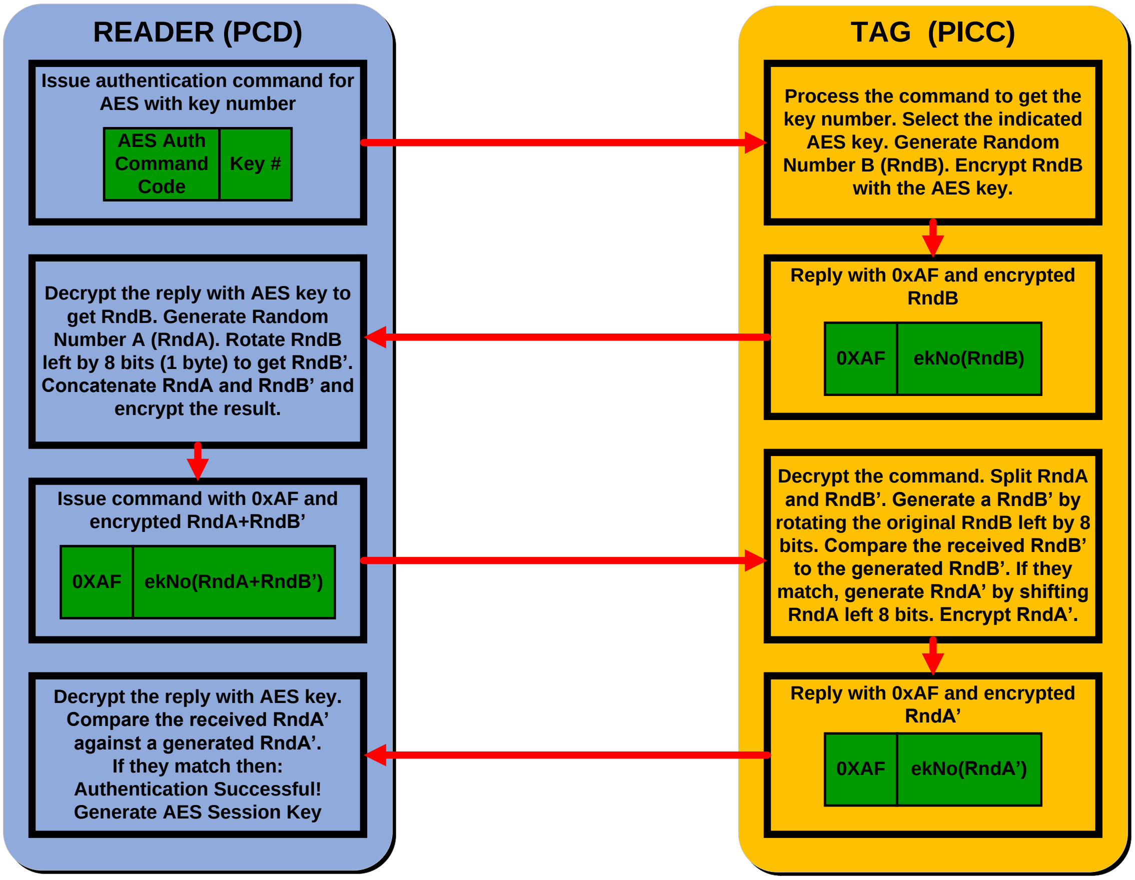
READER (PCD)
Issue authentication command for AES with key number
AES Auth Command Code
Key #
Decrypt the reply with AES key to get RndB. Generate Random Number A (RndA). Rotate RndB left by 8 bits (1 byte) to get RndB'. Concatenate RndA and RndB' and encrypt the result.
Issue command with 0xAF and encrypted RndA+RndB'
0XAF
ekNo(RndA+RndB')
Decrypt the reply with AES key. Compare the received RndA' against a generated RndA'. If they match then: Authentication Successful! Generate AES Session Key
TAG (PICC)
Process the command to get the key number. Select the indicated AES key. Generate Random Number B (RndB). Encrypt RndB with the AES key.
Reply with 0xAF and encrypted RndB
0XAF
ekNo(RndB)
Decrypt the command. Split RndA and RndB'. Generate a RndB' by rotating the original RndB left by 8 bits. Compare the received RndB' to the generated RndB'. If they match, generate RndA' by shifting RndA left 8 bits. Encrypt RndA'.
Reply with 0xAF and encrypted RndA'
0XAF
ekNo(RndA')

Figure 4. AES Authentication Communication Sequence

3.2 AES Code Benchmarks

The AES encryption and decryption processes are being performed within software for this application. However, when AES encryption is done through a hardware implementation, the speed typically increases by at least a tenfold. Since the MSP430G2553 microcontroller does not have the hardware required for an AES hardware implementation on it natively, the software process illustrated here does sacrifice some speed but it is a very cost effective solution.

In order to provide a clear picture of the resources that the hardware implementation uses, benchmarks have been run to determine the speed and size constraints of the AES software that is included with the firmware example. The results of these benchmarks are included in Table 2.

Table 2. AES Code Benchmarks

AES (ENC/DEC Function)	Optimization	
	Speed	Size
Memory: Data	34 bytes	34 bytes
Memory: Constants	0.55 KB	0.55 KB
Memory: Code	1 KB	0.83 KB
Clock Cycles (in kilocycles)	7.9	12.3

4 Hardware Description

MSP- EXP430G2

The MSP-EXP430G2 LaunchPad is an easy-to-use flash programmer and debugging tool for the MSP430G2xx Value Line microcontrollers. It features everything you need to start developing on an MSP430 microcontroller device. It has on-board emulation for programming and debugging and features a 14/20-pin DIP socket, on-board buttons and LEDs & BoosterPack-compatible pinouts that support a wide range of plug-in modules for added functionality such as wireless, displays, and more.

DLP-7970ABP

The third party provider DLP Design NFC/RFID BoosterPack (DLP-7970ABP) is an add-on board designed to fit all of TI's MCU LaunchPads. This BoosterPack allows the software application developer to get familiar with the functionalities of TRF7970A Multi-Protocol Fully Integrated 13.56-MHz NFC/HF RFID IC on their Texas Instruments Embedded microcontroller platform of choice without having to worry about developing the RF section.

The TRF7970A device is an integrated analog front end and data-framing device for a 13.56-MHz RFID and Near Field Communication (NFC) system. Built-in programming options make the device suitable for a wide range of applications for proximity and vicinity identification systems.

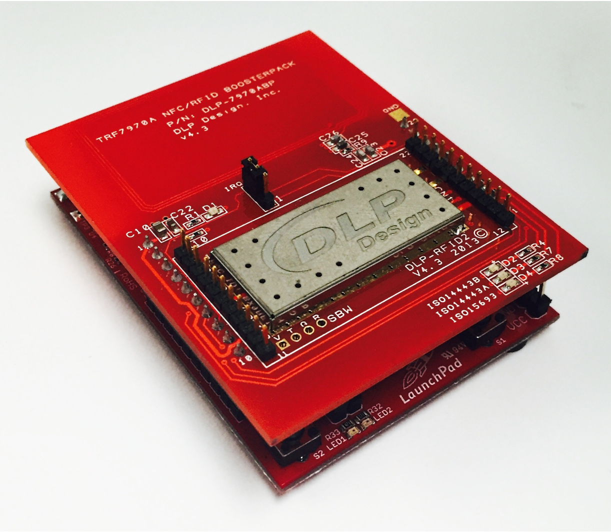

Figure 5. MSP430G2553 LaunchPad and DLP-7970ABP BoosterPack

5 MIFARE DESFire EV1 AES Authentication Firmware

The software that is provided is structured as shown by the flow chart in Figure 1. The Nfc_runAesAuth Application Programming Interfaces (API) is used to run through anticollision, ISO 14443 layer 4 commands, and then the AES authentication sequence. The entire flow of the software, including the initialization process, can be seen in Figure 6.

Due to memory size constraints with the hardware, the provided firmware does not support the storage of multiple AES keys in RAM on the G2553. However, although RAM is limited, it is possible to store many different keys within Flash memory. Therefore, in order to use multiple custom AES keys the Nfc_setAesKey API is provided as a method to dynamically modify the contents of the key array that is used in the AES authentication process.

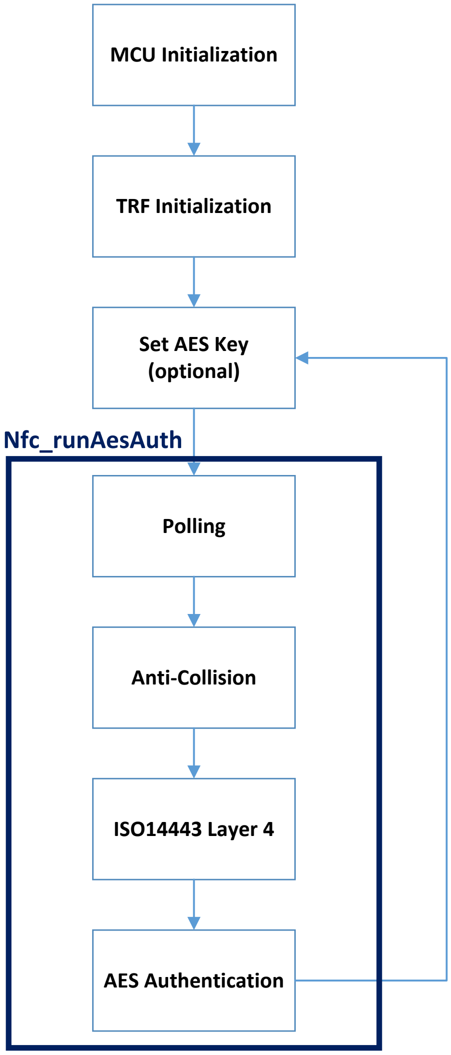

Figure 6. Software Flowchart for TI's AES Authentication Example Code

5.1 Available Application Programming Interfaces (APIs)

This section is an overview of the APIs that are provided to communicate with MIFARE DESFire EV1 tags, including the function that runs the AES authentication process.

5.1.1 Nfc_runAesAuth

This function is configured to run through the whole AES authentication sequence from tag detection all the way through authentication. It should be called within the while(1) loop in the main application.

Prototype:

```
void Nfc_runAesAuth(void)
```

Parameters:

None

Return:

None

Description:

This function is setup to call all of the APIs that are needed to go through an entire AES authentication process for MFDFEV1 tags. It will start with the initial polling commands and go all the way through to the AES authentication sequence. It follows the flow diagram that is shown in Figure 6.

5.1.2 Nfc_setAesKey

This function is used to overwrite the current AES key used for authentication.

Prototype:

```
void Nfc_setAesKey(const u08_t * pui8AESKey, u08_t ui8AESKeyLength)
```

Parameters:

pui8AESKey must be a pointer to an array which contains the new AES key that will be used for the authentication sequence.

ui8AESKeyLength is the length of the new AES key.

Return:

None

Description:

This function allows the user to overwrite the current AES key and replace it with a new one. This should be used to dynamically change which AES key is used for authentication without having multiple arrays declared in RAM, especially when using microcontrollers which have limited RAM resources.

The AES key should always be 16 bytes long. In order to prevent an AES key that is not 16 bytes to be written in, the length is always passed through and verified.

5.1.3 Iso14443aAnticollision

This function handles polling for Type 4A tags, and the subsequent anticollision sequence once a tag has been detected.

Prototype:

```
void Iso14443aAnticollision(u08_t ui8Command)
```

Parameters:

ui8Command is the polling command to transmit.

Return:

None

Description:

This function issues the polling commands for the detection of ISO14443A transponders / NFC Type 4A Tag Platforms. If a reply is received, then it will call the anticollision process to begin the selection process.

When calling the API the user must provide which polling command will be issued:

- REQA (also called SENS_REQ), pre-defined as 0x26.
- WUPA (also called ALL_REQ), pre-defined as 0x52.

The API call for the anticollision function should therefore be one of the following:

- Iso14443aAnticollision(REQA);
- Iso14443aAnticollision(WUPA);

If a tag replies to the polling command, the function will then call the Iso14443aLoop to handle the anticollision process.

5.1.4 Iso14443aLoop

This function handles the anticollision loop for Type 4A tags.

Prototype:

```
void Iso14443aLoop(u08_t ui8CascadeLevel, u08_t ui8NVB, u08_t * pui8UID)
```

Parameters:

ui8CascadeLevel is the cascade level of the anticollision sequence that shall be executed by the Iso14443aLoop function.

ui8NVB is the Number of Valid Bits value that shall be sent out by each Select command.

pui8UId is a pointer to an array which contains the UID of the tag.

Return:

None

Description:

This function will check the current cascade level and issue the Select command based on the cascade level specified in the function call. If any collisions occur, typically due to multiple tags are detected by the reader, then the function will proceed to go through an anticollision sequence to resolve the collisions. During the anticollision sequence, the NVB is used to track how many bytes and bits have been received by the reader.

This function will call itself recursively when it goes through anticollision sequences, up to three additional times (one for each cascade layer).

5.1.5 Iso14443aLayer4

This function is used to handle the ISO 14443-4 activation sequence.

Prototype:

```
u08_t Iso14443aLayer4(void)
```

Parameters:

None

Return:

STATUS_SUCCESS or *STATUS_FAIL*

Description:

This function will transmit the layer 4 (ISO 14443-4) commands that are required before the tag can be authenticated. This API should be called after the anticollision process is complete in order to enter data exchange.

The function will first transmit a RATS command. If the received ATS response properly matches with the expected response based on the MIFARE DESFire EV1 tag specifications, then a PPS request will be transmitted next. If the received PPS response is correct, then STATUS_SUCCESS will be returned. If either of the responses were erroneous, then STATUS_FAIL will be returned.

Because the function will return either STATUS_SUCCESS or STATUS_FAIL to indicate whether or not the layer 4 commands were successful, it is recommended to verify that the returned value is STATUS_SUCCESS before proceeding to the AES authentication process.

5.1.6 Iso14443aPollingCommand

This function is used to transmit ISO14443A polling commands.

Prototype:

```
void Iso14443aPollingCommand(u08_t ui8Command)
```

Parameters:

ui8Command is the byte of the ISO14443A command being transmitted.

Return:

None

Description:

This function will transmit the polling command that is specified in the input parameter. The polling commands that should be issued for Type 4A tags are:

- REQA (also called SENS_REQ), pre-defined as 0x26.
- WUPA (also called ALL_REQ), pre-defined as 0x52.

5.1.7 Iso14443aSelectCommand

This function is used to transmit the ISO14443A Select command.

Prototype:

```
void Iso14443aSelectCommand(u08_t ui8Select, u08_t ui8NVB, u08_t * pui8UID)
```

Parameters:

ui8Select is the Select command byte that shall be transmitted.

ui8NVB is the Number of Valid Bits value that shall be sent out by each Select command.

pui8UId is a pointer to an array which contains the UID of the tag.

Return:

STATUS_SUCCESS or *STATUS_FAIL*

Description:

This function will transmit the Select command based on the inputs given to it, and it will determine whether or not to transmit with CRC based on the value of the ui8NVB provided in the function call.

5.1.8 Iso14443aHalt

This function is used to transmit the ISO14443A halt command.

Prototype:

```
void Iso14443aHalt(void)
```

Parameters:

None

Return:

None

Description:

This function will transmit the halt command for ISO14443A transponders / NFC Type 4A Tag Platforms as specified in the ISO/IEC 14443-3 specifications.

5.1.9 Aes_authenticate

This function handles the AES authentication process for MIFARE DESFire EV1 tags.

Prototype:

```
u08_t Aes_authenticate(u08_t * pui8SessionKey, u08_t * pui8RndA, u08_t * pui8Key)
```

Parameters:

pui8SessionKey must be a pointer to an array that can store the resulting 16 byte session key of the authentication sequence.

pui8RndA must be a pointer to an array that contains a 16 byte random number used for the authentication sequence.

pui8Key must be a pointer to an array which contains the 16 byte key that is being used for the authentication sequence.

Return:

STATUS_SUCCESS or *STATUS_FAIL*

Description:

This function handles the sequence of transmissions and receptions during the AES authentication process. If the authentication process is successful, then it will return a STATUS_SUCCESS along with the session key for that sequence. The session key is required for certain commands, including the Change Key command. The random number that is provided should be generated by a random number generator (RNG), but due to memory constraints it is a static hard coded value in the example software that is being provided.

5.2 Example Application With Required Initializations

In addition to the listed APIs, there is a specific set of commands that must be executed to initialize the MCU and the TRF7970A prior to any application. In the following example, the MCU is initialized by setting the MSP430G2553 main clock frequency to 8 MHz. Then the TRF7970A is initialized by setting the SPI Slave Select high, then the TRF's Enable pin high, and then waiting 5 milliseconds before going through the initialization functions. Lastly, the global interrupts are enabled so that the IRQ interrupt service routine becomes functional. After these initializations have been completed, any application-based initializations can be run. See Example 1 for a summary of the initializations.

Example 1. Required Initialization Sequence

```
#include "mcu.h"
#include "nfc.h"
// Initialize global variables
u08_t buf[127];     // Transmit/Receive Buffer
u08_t i_reg = 0x01;     // Interrupt Register
u08_t irq_flag = 0x00;     // Flag for the IRQ Status Register
s08_t rxtx_state = 1;     // Used for transmit receive byte count

void main (void)
{
     // Stop the Watchdog timer
     WDTCTL = WDTPW + WDTHOLD;

     // Select DCO to be 8 MHz
     McuOscSel(0x00);
     McuDelayMillisecond(10);

     // Set the SPI SS high
     SLAVE_SELECT_PORT_SET;
     SLAVE_SELECT_HIGH;

     // Set TRF Enable Pin high
     TRF_ENABLE_SET;
     TRF_ENABLE;

     // Wait until TRF system clock started
     McuDelayMillisecond(5);

     // Initialize SPI settings for communication with TRF
     Trf797xCommunicationSetup();

     // Set up TRF initial settings
     Trf797xInitialSettings();

     // Enable global interrupts
     __bis_SR_register(GIE);

     // End of Initialization

     // Infinite loop for application
     while (1)
     {
          // Runs Authentication sequence
          Nfc_runAesAuth();
     }
}
```

5.3 Memory Size Benchmarks

This section contains a summary of the code sizes for the AES authentication example firmware that is included. The code sizes that are listed below were achieved by applying level three Code Composer Studio optimizations, for both speed and size, to the firmware.

The results show that there is a significant amount of Flash memory space available (Table 3 and Table 4) for user-specific applications that can supplement the AES authentication portion of the firmware. However, the RAM space (Table 3 and Table 5) is fairly limited in the MSP430G2553 (maximum size: 512 bytes) and 128 bytes of it is used for the buffer that handles the transmitted and received NFC/RFID communication data.

Table 3. Overall Memory Use

Memory Type	Optimized for Size (bytes)	Optimized for Speed (bytes)
Flash (Max: 16KB)	6074	8454
RAM (Max: 512B)	408	408

Table 4. Flash Memory Use

Memory Sector	Optimized for Size (bytes)	Optimized for Speed (bytes)
Code (.text)	5420	7800
Variables (.const)	554	554
Initializations (.cinit)	100	100

Table 5. RAM Memory Use

Memory Sector	Optimized for Size (bytes)	Optimized for Speed (bytes)
Variables (.bss)	228	228
Stack (.stack)	180	180

6 AES Authentication Demonstration

The example software is configured for use with the MSP430G2553 Value Line LaunchPad along with the DLP-7970A BoosterPack. The connections between the MSP430G2553 and the TRF7970ABP that are utilized by the firmware example are shown in Table 6.

Table 6. TRF7970ABP + MSP430G2553 LaunchPad Hardware Connection

TRF7970ABP Pins	MSP-EXP430G2553LP Pins
MOSI	P1.7
MISO	P1.6
CLK	P1.5
Slave Select	P2.1
EN1	P2.2
IRQ	P2.0
ISO 14443B LED	P2.3
ISO14443A LED	P2.4
ISO15693 LED	P2.5

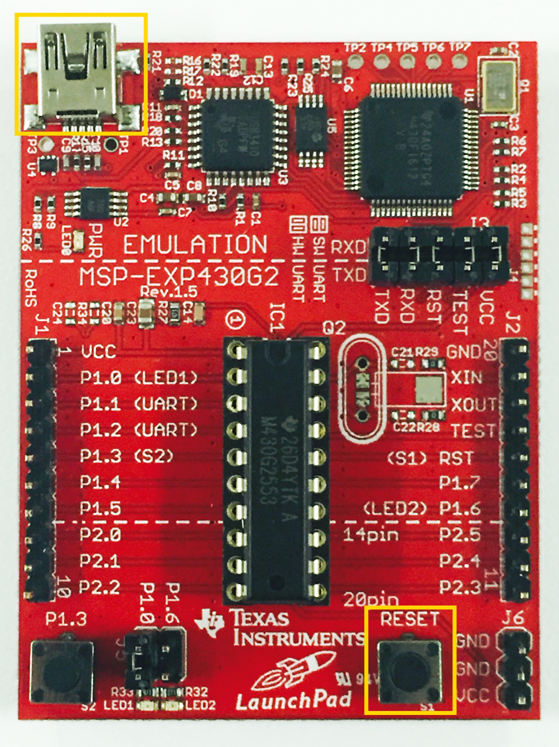

Figure 7. MSP430G2553 LaunchPad

6.1 AES Key Setup

The authentication of a MFDFEV1 tag requires that the proper AES key be used. In order to setup the AES key that will be used by the Reader for the authentication process, it is necessary to modify the main file of the AES authentication project. Since there are limited RAM resources on the MSP430G2553, it is advised to place all AES keys within Flash memory to save some RAM space. To do this, declare a 16 byte array as a global variable with the 'const' keyword and initialize it upon declaration with the values for the AES key (see Example 2). The 'static' keyword can also be used to keep the array localized to the file it has been declared in.

Example 2. Declaring AES Key Arrays in main.c

```
const static u08_t CustomKey1[16] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

const static u08_t CustomKey2[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66,
0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};

const static u08_t CustomKey3[16] = {0x79, 0x70, 0x25, 0x53, 0x79, 0x70, 0x25,
0x53, 0x79, 0x70, 0x25, 0x53, 0x79, 0x70, 0x25, 0x53};
```

When the AES keys are loaded into Flash memory, it is possible to change which key is used for authentication by the application. To do this, use the Nfc_setAesKey API to load the correct AES key first, and then use the Nfc_runAesAuth API to run the authentication process. The Nfc_runAesAuth API will use the newest key that was loaded, or a default key (all bytes = 0x00) if no AES key has been set. The example code shown in Example 3 is structured to demonstrate an application that will attempt to authenticate using three different AES keys by cycling between them prior to each Nfc_runAesAuth function call. This example is one possible method that can be implemented to avoid using up the limited RAM resources while still authenticating MFDFEV1 tags with a variety of preprogrammed AES keys.

Example 3. Cycling Between AES Keys in a while(1) Loop

```
while(1)
{
     // Set AES Key to a custom key stored in Flash Memory
     Nfc_setAesKey(CustomKey1,16);
     // Runs Authentication sequence
     Nfc_runAesAuth();

     // Repeat for the remaining keys
     Nfc_setAesKey(CustomKey2,16);
     Nfc_runAesAuth();

     Nfc_setAesKey(CustomKey3,16);
     Nfc_runAesAuth();
}
```

6.2 Demonstration

To demonstrate the AES authentication firmware, follow these steps:

1. Connect the DLP-7970ABP to the MSP430G2553 LaunchPad.
2. Put a jumper in Position 2 on the DLP-7970ABP board (see Figure 8).
3. Connect a USB cable to the onboard USB Emulator (top left corner, see Figure 7).
4. Open Code Composer Studios V6.0.1.
5. Import the AES authentication project (download the firmware from http://www.ti.com/lit/zip/sloa213).
6. Go to main.c and add arrays for the AES key(s) for each tag that will be authenticated.
7. Modify the while loop as needed to use Nfc_setAesKey and Nfc_runAesAuth to run through the AES authentication process for each key that has been loaded.
8. Click the "Debug" button to load the firmware.
9. After the code had finished downloading, click Stop.
10. Reset the board by either power cycling it or pressing the reset button.

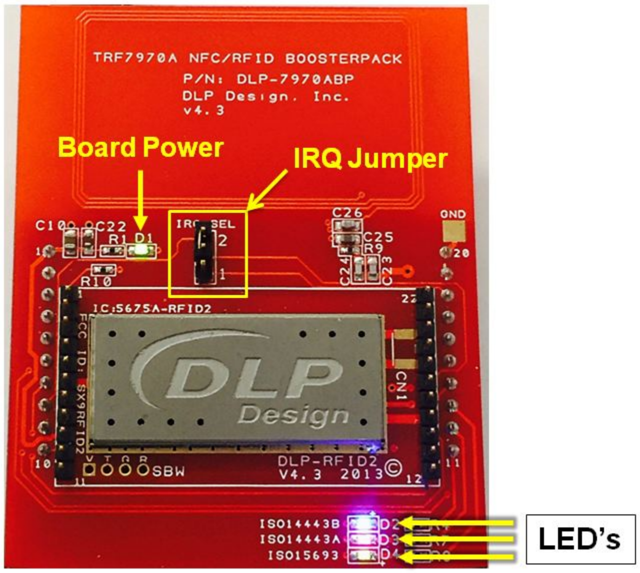

Figure 8. Hardware Configuration

11. Now present a MIFARE DESFire EV1 tag that has been configured for AES mode. The bottom 3 LEDs will light up to give the user feedback about which portions of the code have been successful. They represent the following:
    - ISO 14443B LED (Blue) – Received a reply from REQA/WUPA
    - ISO14443A LED (Red) – Successful ISO14443A Anticollision
    - ISO15693 LED (Green) – Successful AES Authentication

If one of the keys that was loaded into the firmware and used by the authentication process matches with the key contained inside of the MFDFEV1 tag, then the green LED will be lit to signify a successful authentication process.

7 Conclusion

The combination of the MSP430G2553 Value Line microcontroller and the TRF7970A NFC transceiver can be used along with the included firmware to provide a cost-effective solution for authenticating MIFARE DESFire EV1 tags in AES mode. The MSP430G2553 also provides enough Flash memory for additional user applications, as well as the storage of many different AES keys, allowing for more robust and customized applications.

8 References

1. *TRF7970A Multi-Protocol Fully Integrated 13.56-MHz RFID and NFC Transceiver IC* (SLOS743)
2. *MSP430G2x53, MSP430G2x13 Mixed-Signal Microcontrollers* (SLAS735)
3. *C Implementation of Cryptographic Algorithms* (SLAA547)
4. *ISO/IEC 7816-4:2005(E)* (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54550)
5. *ISO/IEC 14443-3:2009(E)* (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=50942)
6. *ISO/IEC 14443-4:2008(E)* (http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=50648)
7. *MF3ICD81 MIFARE DESFire EV1* (http://www.nxp.com/)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265