Version 1.1

eRIC_AdvancedEncryptionStandard(AES):

eRIC AES module performs encryption and decryption of 128-bit data with 128-bit keys according to the advanced encryption standard(AES)(FIPS PUB197) in hardware.

Refer eRIC_eROS_Developers_Manual_x.x on wards for complete list of AES definitions. Refer SLAU259E document by Texas Instruments to use core registers.

To achieve AES there are three function and two variables.

The following are two variables:

eRIC_AES_Key[]; This is 17 bytes variable. This variable needs loading before setting AES key or changing AES key. The first byte is the mode (Encryption/Decryption) that needs to be loaded. When eRIC_AES_Key[0] is 0,it is in encryption mode and when it is 1 it sets in Decryption mode. The rest of the 16 bytes is the key for AES. The key is used to encrypt or decrypt data.

eRIC_AES_Data[]; This is a 16 Bytes variable. This is only 16 bytes since AES is only 128-bit data. So to encrypt /decrypt data, this variable needs loading with the 16 bytes of data. And after calling AES Run function, the AES encryption/decryption works in background and encrypted/decrypted data is available in this variable. eRIC_AES_Data[] is modified whenever eRIC_AES_Setkey() is used.

The following are the functions needed to achieve AES:

eRIC_AES_ChangeKey(); This is used to change Key or set Key for the first time. eRIC_AES_Key[] variable needs loading first before calling this function. Depending on first byte of eRIC_AES_Key[], encryption(0) or decryption(1) mode is set. When this function is called after loading mode and 16 Bytes key in eRIC_AES_Key[], the key is encrypted with discrete unique key and encrypted key is stored in discrete location. So on reset, the encrypted key is pulled from location to do any AES which is explained further. eRIC_AES_SetKey() is also called at the end of this function to set key.

eRIC_AES_SetKey(); This function pulls the encrypted key which is stored in discrete location and decrypts the key using the same unique discrete key to get the original key. Depending on the mode (encryption or decryption), the key is set for AES. Whenever the mode is changed from either encryption to decryption or vice versa, this function needs to be called to set key again for desired AES mode. There is no need to call eRIC_AES_ChangeKey(), if the same key is used because eRIC_AES_Setkey() function will pull the encrypted key from discrete locations. When eRIC_AES_ChangeKey() is called, eRIC_AES_Setkey() function is also called within that function to set key.
eRIC_AES_Data[] is modified whenever eRIC_AES_Setkey() is used.

eRIC_AES_Run(); The actual AES is performed after calling this function depending on the mode chosen before in eRIC_AES_SetKey() or eRIC_AES_ChangeKey().

To encrypt the data using this function, the key should be loaded and set using above functions and data to be encrypted should also be loaded in eRIC_AES_Data[]. After calling this function, the encrypted data is stored in the same eRIC_AES_Data[]. AES can only be done in 16 bytes packet.

Similarly to decrypt data the key should be loaded and set using above functions and data to be decrypted should also be loaded in eRIC_AES_Data[].After calling this function, the decrypted data is stored in the same eRIC_AES_Data[]. AES can only be done in 16 bytes packet.

Note: The most important part of sending and receiving AES data is based on byte count of the packet. To send AES encrypted data, the no of bytes to be sent has to be loaded first in `eRIC_RadioTx_Buffer[]` buffer. So the total byte count of the packet to send over-air would be increased by 1.

Say for example to send 16 bytes of data, load with `eRIC_RadioTx_Buffer[]` buffer with 16 .Then load `eRIC_RadioTx_Buffer[]` buffer with the 16 bytes of data. Perform AES on 16 bytes of data. Then before sending the packet over-air increment the buffer count by 1 i.e `eRIC_RadioTx_BuffCount = 16+1;`

And for example to send 30 bytes of data, first load `eRIC_RadioTx_Buffer[]` buffer with 30. Then fill `eRIC_RadioTx_Buffer[]` buffer with 30 bytes of data. Since AES is based on multiples of 16 bytes of packet, the total bytes it needs would be 32(16*2). So to complete the packet load extra two bytes in `eRIC_RadioTx_Buffer[]` buffer with 0s. Perform AES on 32 bytes of data. Then send the packet with 33 as byte count i.e `eRIC_RadioTx_BuffCount = 30+1+2;`

Similarly to receive AES data, based on byte count of the packet read all over-air data. Then perform AES on received data and discard the unwanted bytes based on first byte of the packet. Say for example when 33 bytes of above example packet is sent. Then read all 33 bytes. Leave the first byte(The first byte in this instance would be 30) and perform AES on rest of the bytes i.e 32. Since the first byte read is 30, discard 2 byte from decrypted data of 32 bytes.

Code Example:

```
1) #include<cc430f5137.h>
2) #include"eRIC.h"
3) volatile const char Data[] = {'e','R','I','C',' ','A','E','S',' '
   ,'E','X','A','M','P','L','E'}; //16 bytes in totdal
4) volatile char Send_Encrypy_Decrypt;
5) int main(void)
6) {
7)       eRIC_WDT_Stop();
```

```
8)        eRIC_GlobalInterruptDisable();
9)        eROS_Initialise(434000000);
10)       eRIC_Power = 12;
11)       eRIC_RadioUpdate();
12)       Pin17_SetAsInput();
13)       Pin18_SetAsInput();
14)       Pin19_SetAsInput();
15)       Pin16_SetAsOutput();
16)       Send_Encrypy_Decrypt = 0;
17)       volatile unsigned char i =0;
18)       for(i=0;i<16;i++)
19)             eRIC_AES_Key[i+1] = i;    //Key is 0,1,2,3...15 16 bytes in
   total
20)       eRIC_AES_Key[0] = 0; //Encrypt mode
21)       eRIC_AES_ChangeKey();// eRIC_AES_SetKey() is called within this
   anyway in background
22)       eRIC_GlobalInterruptEnable(); //Global interrupts enabled
23)       while(1)
24)       {
25)             if(Pin17_Read() && Pin18_Read())    //Send encrypted data when
   Pin17 and 18 are held high
26)             {
27)                   while(Pin17_Read()||Pin18_Read());
28)                   Send_Encrypy_Decrypt = 1;//Send encrypted data over air
29)             }
30)             if(Pin17_Read() && Pin19_Read())   //Send decrypted data when
   Pin18 and 19 are held high
31)             {
32)                   while(Pin17_Read()||Pin19_Read());
33)                   Send_Encrypy_Decrypt = 2;//Send decrypted data over air
34)             }
35)             if(Send_Encrypy_Decrypt)
36)             {
37)                   //Encryption
38)                   eRIC_AES_Key[0] = 0; //Encrypt mode
39)                   eRIC_AES_SetKey();
40)                   for(i=0;i<16;i++)
41)                         eRIC_AES_Data[i] = Data[i];    //load 16 bytes of
   data
42)                   eRIC_AES_Run();         //Data is encrypted
43)                   eRIC_RadioTx_Buffer[0] = 16;  //V1.1 fill the buffer
   with byte count first
44)                   for(i=0;i<16;i++)
45)                         eRIC_RadioTx_Buffer[i+1] = eRIC_AES_Data[i];
   //load encrypted data into radio tx buffer
46)                   if(Send_Encrypy_Decrypt == 1)
47)                   {
```

```
48)                         Pin16_SetHigh();
49)                         eRIC_RadioTx_BuffCount = 17;
50)                         eRIC_RfSenddata();
51)                   }
52)                   //Decryption
53)                   eRIC_AES_Key[0] = 1; //Decryption mode
54)                   eRIC_AES_SetKey(); //No eRIC_AES_ChangeKey() is needed
   as same key is used
55)                   for(i=0;i<16;i++)
56)                         eRIC_AES_Data[i] = eRIC_RadioTx_Buffer[i+1];
   //load encrypted bytes to decrypt
57)                   eRIC_AES_Run();         //Data is decrypted, as when
   decryption is done, decrypted data is already loaded in eRIC_AES_Data[]
58)                   if(Send_Encrypy_Decrypt == 2)
59)                   {
60)                         Pin16_SetLow();
61)                         eRIC_RadioTx_Buffer[0] = 16;//V1.1 fill the
   buffer with byte count first
62)                         for(i=0;i<16;i++)
63)                               eRIC_RadioTx_Buffer[i+1] =
   eRIC_AES_Data[i];    //load decrypted data in tp radio tx buffer
64)                         eRIC_RadioTx_BuffCount = 17;
65)                         eRIC_RfSenddata();
66)                   }
67)                   Send_Encrypy_Decrypt = 0;
68)             }
69)       }
70)}
71)
72)void eRIC_RfDataReceivedInterrupt()   // Add code here to deal with
   available received data..THis is triggered when interrupt is enabled and a
   apcket is received
73){
74)}
```

The above code shows AES example:
Line1 and line2 includes cc430F5137 and eRIC.h which is must for any program code. Main starts at line5

Watch dog timer is stopped and Global interrupts are disabled in Line7 and 8. Radio is initialised with 434MHz and 12dbm power at Line 9- 11. AES key is loaded with 1-16 numbers in Line 19. AES encryption mode is set and key is changed in Line 20-21. When Pin17 and Pin18 are high, sending encrypted data over air is chosen in Line 25-28. Otherwise if Pin17 and Pin19 are held high, sending original(which is a result of decrypting encrypted data)data over air is chosen in Line 30-33.16 Bytes of data is encrypted in Line 36-42.Encrypted data is loaded into Radio tx buffer in Line44-45 as eRIC_AES_Data variable is changed when key is set for

decryption. Encrypted data is sent over air when Pin17 and Pin18 holding high condition was satisfied before in Line47-50.
Key is set for AES decryption mode in Line53 -54. Previously encrypted data which was stored in Radio Tx buffer was loaded in to eRIC_AES_Data as it will be changed while setting key for decryption in Line 53-54. Decryption is performed in Line 57 and decrypted data is sent over air when Pin17 and 19 setting high condition was satisfied before in 60-65. The program loops continuously from Line 23-69.
eRIC_RfDataReceivedInterrupt() is copied from eRIC.c which is removed and pasted in main at line 72-74. This has no effect in this example as there is no receiver enabled. But this code is needed for compiler to compile without error or un-remove this same code in eRIC.c.

eRIC_AdvancedEncryptionStandard(AES): USING COMMANDS:

There are two commands for AES. One to set AES in different modes and another to get what mode of AES is set.

1) **ER_CMD#A1xyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy:** where 'x' can be 0-3 and y can be 32 character encryption key.

   When x = 0: AES is disabled

   When x = 1: AES is enabled with both encryption and decryption

   When x = 2: AES is enabled with only encryption

   When x = 3: AES is enabled with only decryption

   For example to set AES as both to encrypt and decrypt data with key ABCDEFGHIJKLMNOP(16 bytes Ascii), the command should be ER_CMD#A1141424344454647484 94A4B4C4D4E4F50 . As ascii A-P is hex 0x41-0x50.

   This command needs to be followed by ACK.

2) **ER_CMD#A1?:** This command returns the current AES setting. It returns with 0-3 where

   0: AES is disabled

   1: AES is enabled with both encryption and decryption

   2: AES is enabled with only encryption

   3: AES is enabled with only decryption

   This command also needs to be followed by ACK.