AES ATTACK REPORT

Name : Soumyadeep Choudhury
NetId - sxc180056
Subject - Hardware Security

Problem Statement :-

- To find the Round Key of the Round 10 using DFA attack on the output of the Round 9 in the AES key generation algorithm.

Challenges :-

- How to run and compile the AES code and the makefile with the test_code file which will generate an executable to take input plaintext and obtain encrypted text using phex.
- I understand the number of rounds required for 128-bit, 192-bit and 256-bit encryption and which mode to choose for the encryption depending on its key structure.
- How to use the state variable among the private and public functions inside the aes.c code to obtain the K10 round key for the AES encryption.

Method :-

- Firstly, obtain the ciphertext to make sure the code is encrypting properly.
- Second, flip the bit in a single state value by converting it into binary and flipping the last bit. (Assuming hardware implementation as software)
- Perform the experiment in ECB mode with 128 -bit AES encryption.
- Perform the subbyte for both original M9 and M9 (error) for each 16 sub set of the state.
- XOR the subbytes to check the error in which subbyte-state if the XOR value is non-zero.
- Iterate over the guessed M9 values and perform the same above to get the correct M9.
- The equations and the changes made in codes - aes.c, makefile and check_test.c are below -

3 Bit-fault attack

In this section, by using a DFA attack where a fault occurs on only one bit of the temporary cipher result at the beginning of the Final Round, we show how to obtain the entire last round key, i.e. the AES key for an AES-128. For more information about this fault model, the reader can refer to [18].

For the sake of simplicity, we describe the attack on an AES using a 128-bit key.

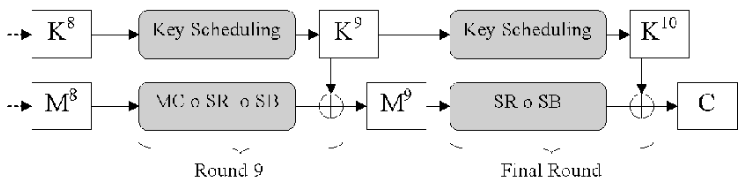


Fig. 2. The last rounds of an AES-128.

By definition, we have

$$C = ShiftRows(SubBytes(M^9)) \oplus K^{10} \qquad (1)$$

By definition, we have

$$C = ShiftRows(SubBytes(M^9)) \oplus K^{10} \tag{1}$$

Let us denote by $SubByte(M_j^i)$ the result of the substitution table applied on the byte M_j^i and by $ShiftRow(j)$ the position of the j^{th} byte of a temporary result after applying the $ShiftRows$ transformation.
So, we have from (1)

$$C_{ShiftRow(i)} = SubByte(M_i^9) \oplus K_{ShiftRow(i)}^{10}, \quad \forall i \in \{0, ..., 15\} \tag{2}$$

If we induce a fault e_j on one bit of the j^{th} byte of the temporary cipher result M^9 just before the Final Round, we obtain a faulty ciphertext D where:

$$D_{ShiftRow(j)} = SubByte(M_j^9 \oplus e_j) \oplus K_{ShiftRow(j)}^{10} \tag{3}$$

and for all $i \in \{0, ..., 15\} \backslash \{j\}$, we have:

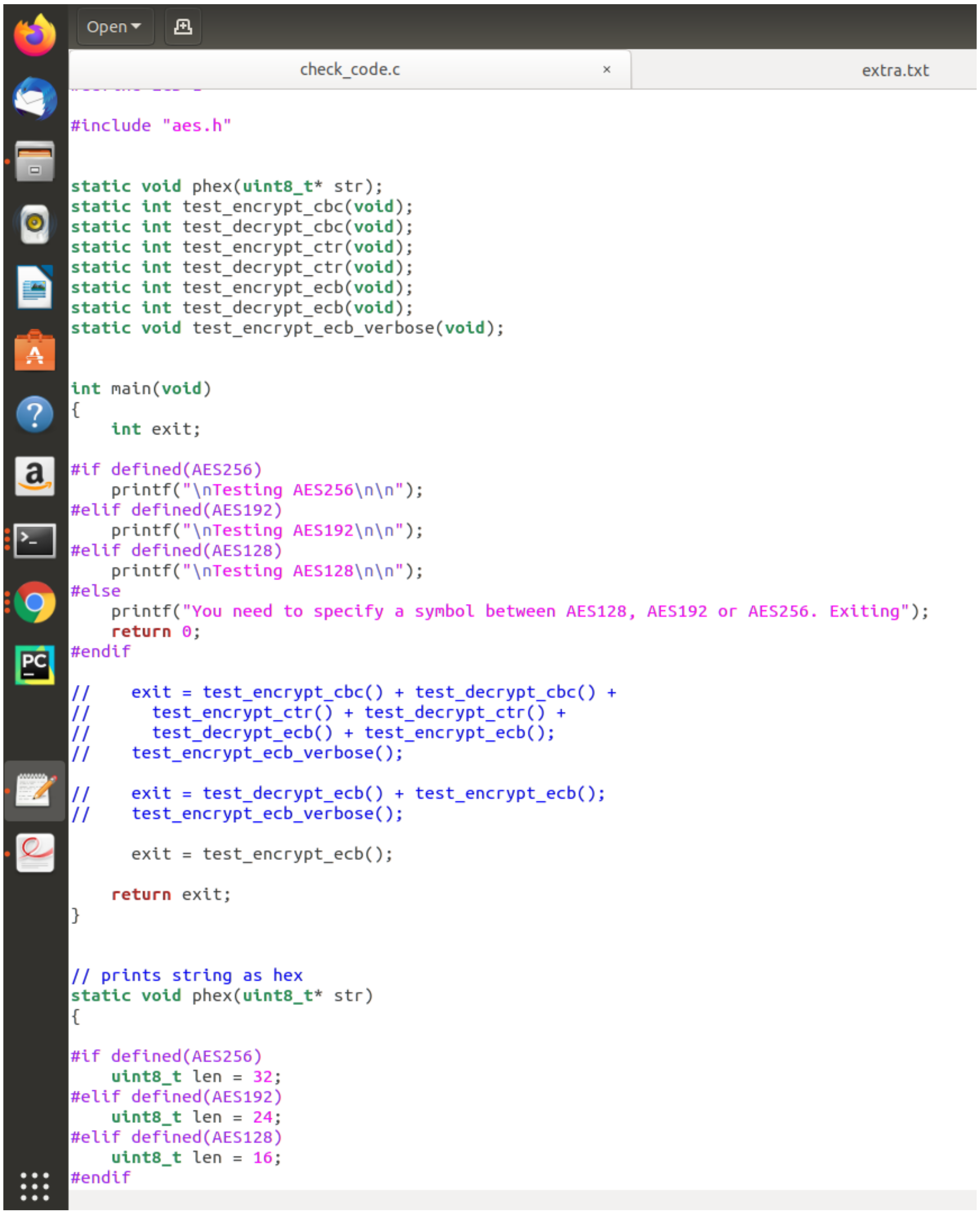
$$D_{ShiftRow(i)} = SubByte(M_i^9) \oplus K_{ShiftRow(i)}^{10} \tag{4}$$

So, if there is no induced fault on the i^{th} byte of M^9, we obtain from (2) and (4)

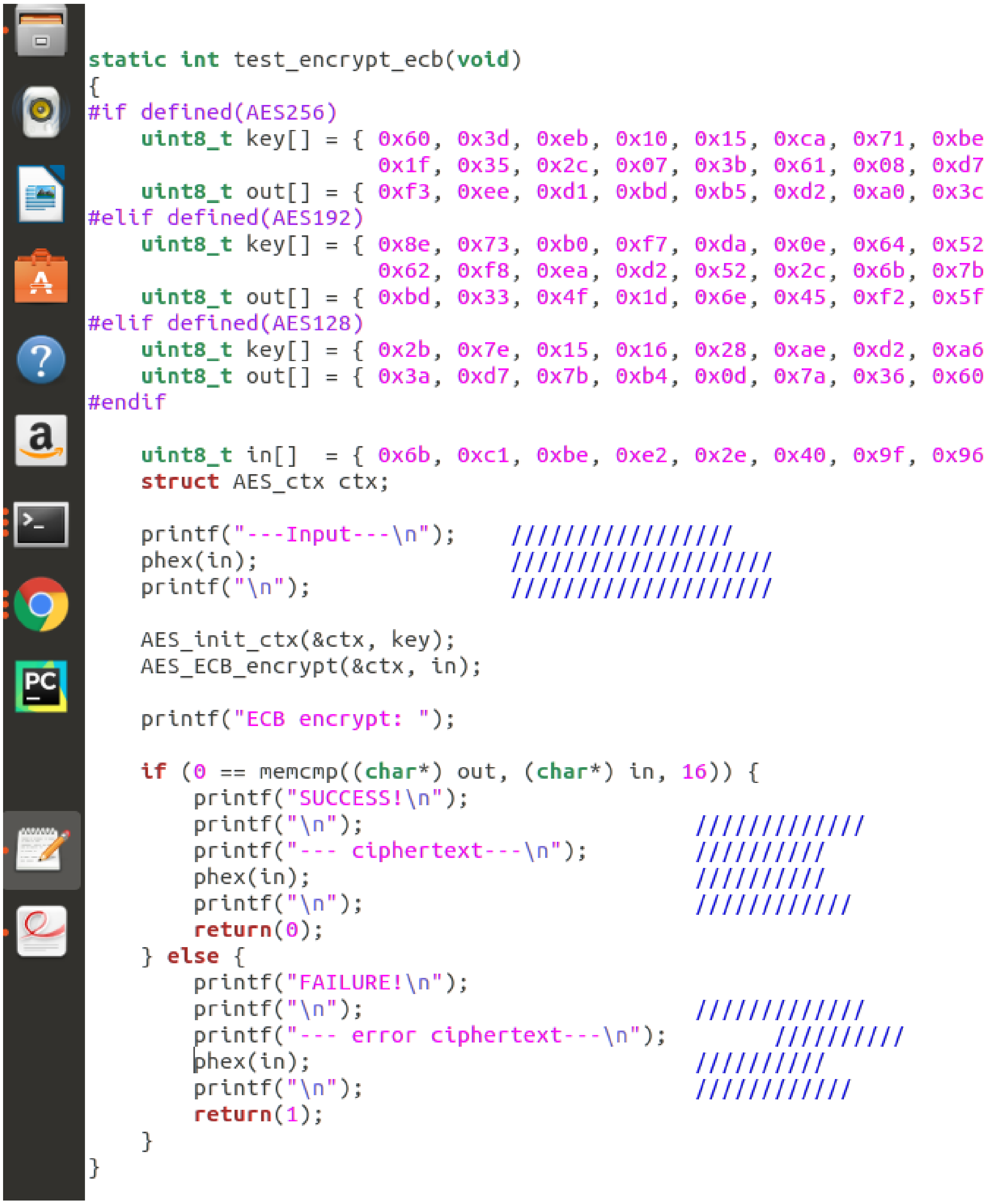
$$C_{ShiftRow(i)} \oplus D_{ShiftRow(i)} = 0 \tag{5}$$

and if there is an induced fault on M_j^9, we have from (2) and (3)

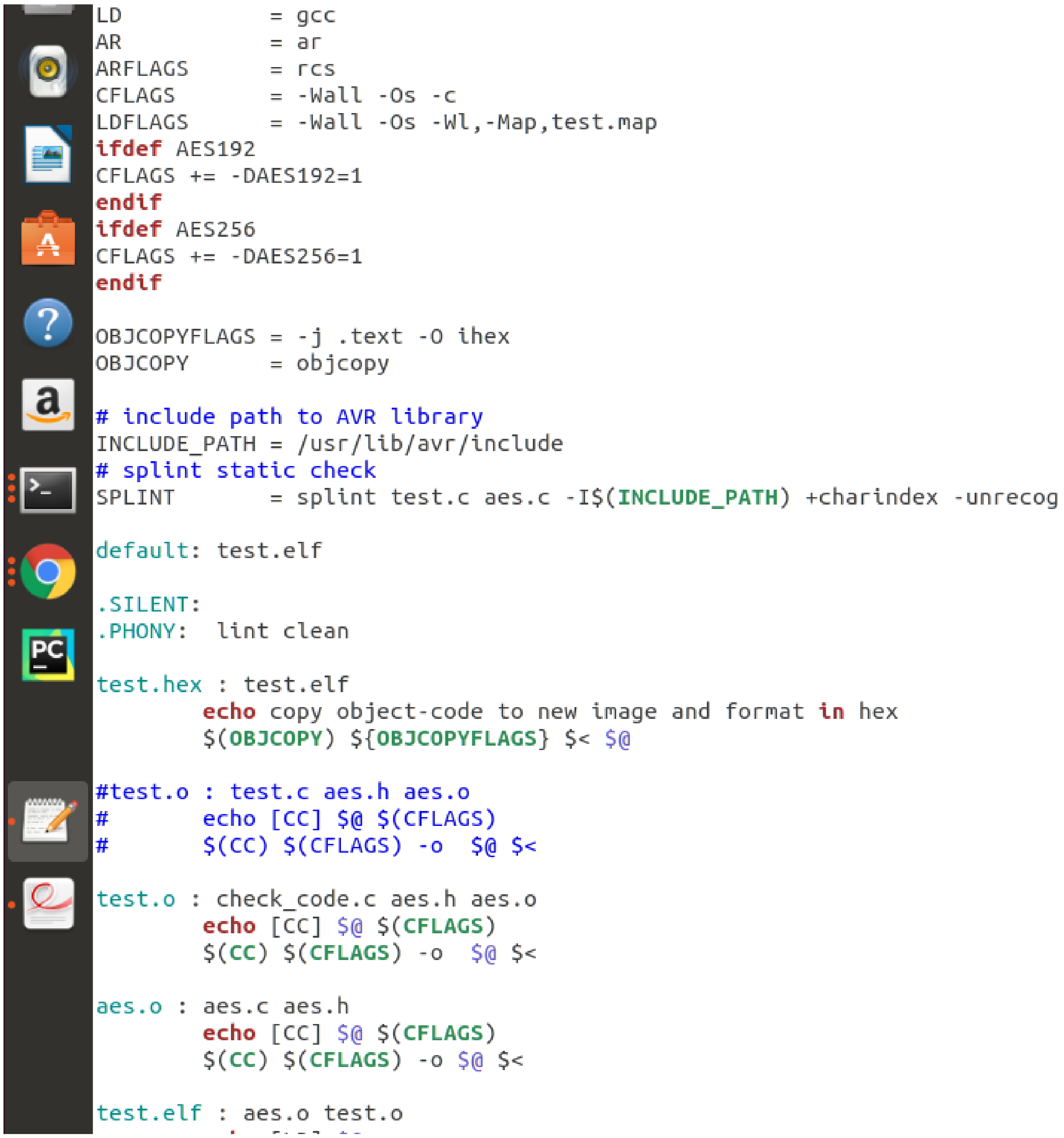
$$C_{ShiftRow(j)} \oplus D_{ShiftRow(j)} = SubByte(M_j^9) \oplus SubByte(M_j^9 \oplus e_j) \tag{6}$$



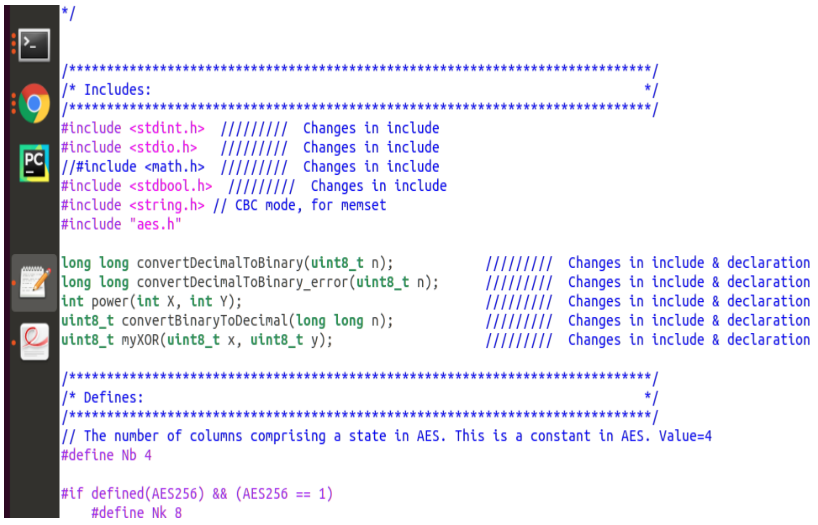
```c
#include "aes.h"


static void phex(uint8_t* str);
static int test_encrypt_cbc(void);
static int test_decrypt_cbc(void);
static int test_encrypt_ctr(void);
static int test_decrypt_ctr(void);
static int test_encrypt_ecb(void);
static int test_decrypt_ecb(void);
static void test_encrypt_ecb_verbose(void);


int main(void)
{
    int exit;

#if defined(AES256)
    printf("\nTesting AES256\n\n");
#elif defined(AES192)
    printf("\nTesting AES192\n\n");
#elif defined(AES128)
    printf("\nTesting AES128\n\n");
#else
    printf("You need to specify a symbol between AES128, AES192 or AES256. Exiting");
    return 0;
#endif

//    exit = test_encrypt_cbc() + test_decrypt_cbc() +
//      test_encrypt_ctr() + test_decrypt_ctr() +
//      test_decrypt_ecb() + test_encrypt_ecb();
//    test_encrypt_ecb_verbose();

//    exit = test_decrypt_ecb() + test_encrypt_ecb();
//    test_encrypt_ecb_verbose();

      exit = test_encrypt_ecb();

    return exit;
}


// prints string as hex
static void phex(uint8_t* str)
{

#if defined(AES256)
    uint8_t len = 32;
#elif defined(AES192)
    uint8_t len = 24;
#elif defined(AES128)
    uint8_t len = 16;
#endif
```



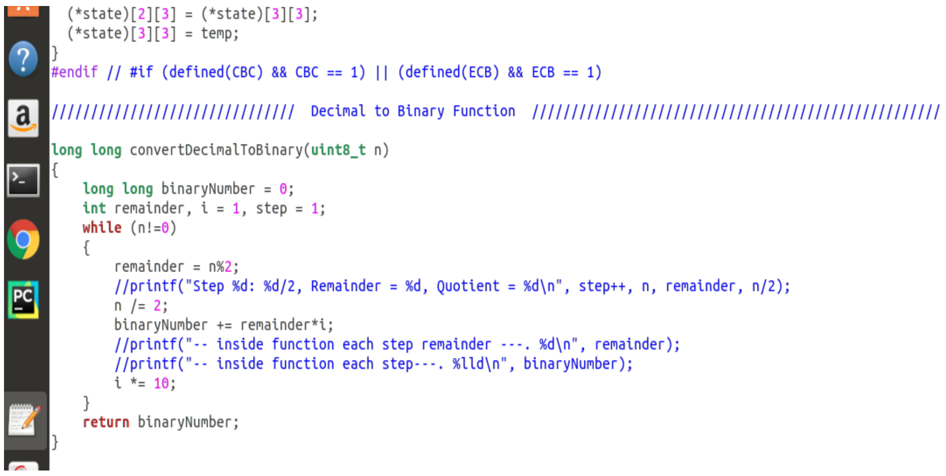
```c
static int test_encrypt_ecb(void)
{
#if defined(AES256)
    uint8_t key[] = { 0x60, 0x3d, 0xeb, 0x10, 0x15, 0xca, 0x71, 0xbe
                      0x1f, 0x35, 0x2c, 0x07, 0x3b, 0x61, 0x08, 0xd7
    uint8_t out[] = { 0xf3, 0xee, 0xd1, 0xbd, 0xb5, 0xd2, 0xa0, 0x3c
#elif defined(AES192)
    uint8_t key[] = { 0x8e, 0x73, 0xb0, 0xf7, 0xda, 0x0e, 0x64, 0x52
                      0x62, 0xf8, 0xea, 0xd2, 0x52, 0x2c, 0x6b, 0x7b
    uint8_t out[] = { 0xbd, 0x33, 0x4f, 0x1d, 0x6e, 0x45, 0xf2, 0x5f
#elif defined(AES128)
    uint8_t key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6
    uint8_t out[] = { 0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60
#endif

    uint8_t in[]  = { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96
    struct AES_ctx ctx;

    printf("---Input---\n");    /////////////////
    phex(in);                   ////////////////////
    printf("\n");               ////////////////////

    AES_init_ctx(&ctx, key);
    AES_ECB_encrypt(&ctx, in);

    printf("ECB encrypt: ");

    if (0 == memcmp((char*) out, (char*) in, 16)) {
        printf("SUCCESS!\n");
        printf("\n");                         /////////////
        printf("--- ciphertext---\n");        //////////
        phex(in);                             //////////
        printf("\n");                         ////////////
        return(0);
    } else {
        printf("FAILURE!\n");
        printf("\n");                         /////////////
        printf("--- error ciphertext---\n");        //////////
        phex(in);                             //////////
        printf("\n");                         ////////////
        return(1);
    }
}
```



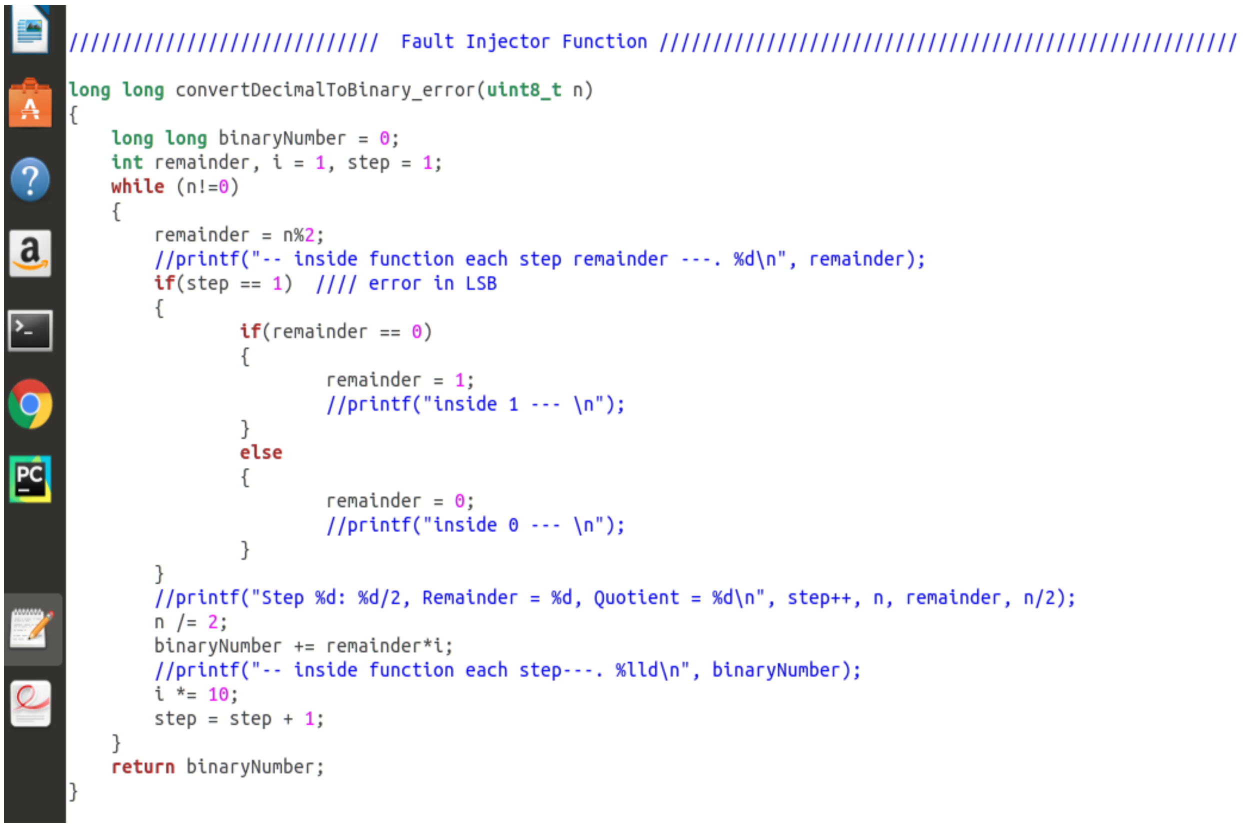
```
LD             = gcc
AR             = ar
ARFLAGS        = rcs
CFLAGS         = -Wall -Os -c
LDFLAGS        = -Wall -Os -Wl,-Map,test.map
ifdef AES192
CFLAGS += -DAES192=1
endif
ifdef AES256
CFLAGS += -DAES256=1
endif

OBJCOPYFLAGS = -j .text -O ihex
OBJCOPY        = objcopy

# include path to AVR library
INCLUDE_PATH = /usr/lib/avr/include
# splint static check
SPLINT         = splint test.c aes.c -I$(INCLUDE_PATH) +charindex -unrecog

default: test.elf

.SILENT:
.PHONY:  lint clean

test.hex : test.elf
	echo copy object-code to new image and format in hex
	$(OBJCOPY) ${OBJCOPYFLAGS} $< $@

#test.o : test.c aes.h aes.o
#	echo [CC] $@ $(CFLAGS)
#	$(CC) $(CFLAGS) -o  $@ $<

test.o : check_code.c aes.h aes.o
	echo [CC] $@ $(CFLAGS)
	$(CC) $(CFLAGS) -o  $@ $<

aes.o : aes.c aes.h
	echo [CC] $@ $(CFLAGS)
	$(CC) $(CFLAGS) -o $@ $<

test.elf : aes.o test.o
```



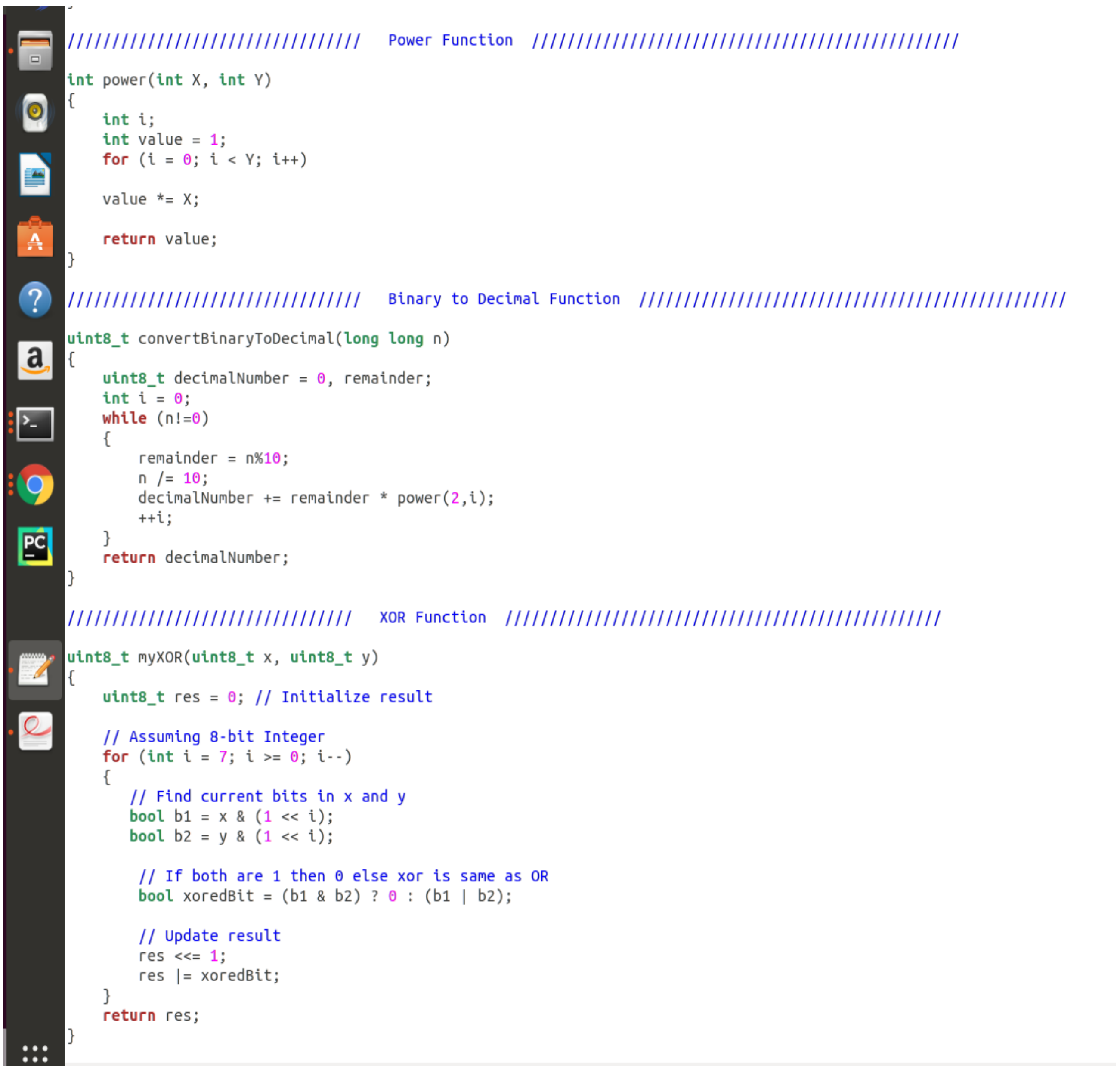
```c
*/


/*****************************************************************************/
/* Includes:                                                                 */
/*****************************************************************************/
#include <stdint.h>  /////////  Changes in include
#include <stdio.h>   /////////  Changes in include
//#include <math.h>  /////////  Changes in include
#include <stdbool.h>  /////////  Changes in include
#include <string.h> // CBC mode, for memset
#include "aes.h"

long long convertDecimalToBinary(uint8_t n);          /////////  Changes in include & declaration
long long convertDecimalToBinary_error(uint8_t n);    /////////  Changes in include & declaration
int power(int X, int Y);                              /////////  Changes in include & declaration
uint8_t convertBinaryToDecimal(long long n);          /////////  Changes in include & declaration
uint8_t myXOR(uint8_t x, uint8_t y);                  /////////  Changes in include & declaration

/*****************************************************************************/
/* Defines:                                                                  */
/*****************************************************************************/
// The number of columns comprising a state in AES. This is a constant in AES. Value=4
#define Nb 4

#if defined(AES256) && (AES256 == 1)
    #define Nk 8
```

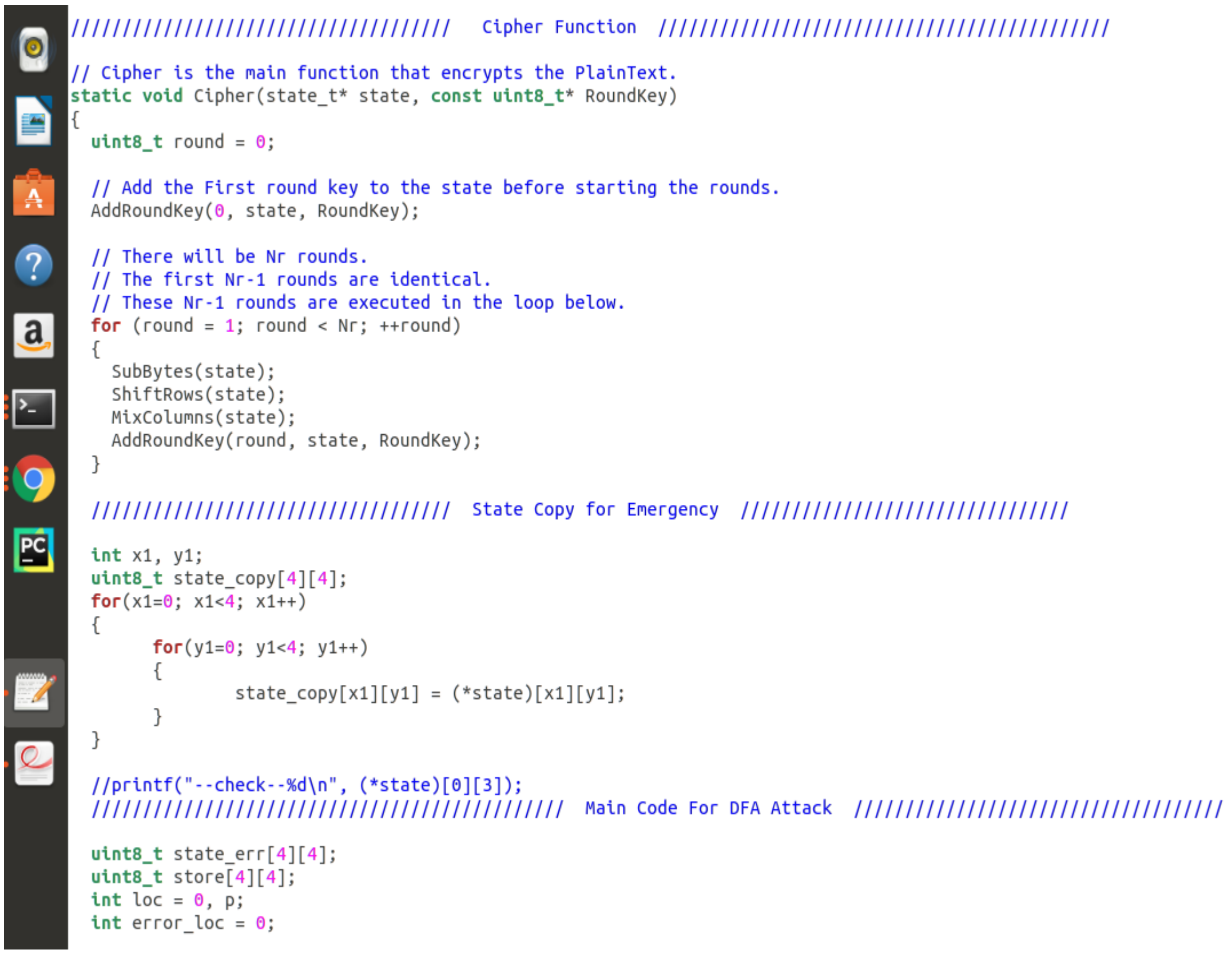


```
  (*state)[2][3] = (*state)[3][3];
  (*state)[3][3] = temp;
}
#endif // #if (defined(CBC) && CBC == 1) || (defined(ECB) && ECB == 1)

//////////////////////////////  Decimal to Binary Function  ///////////////////////////////////////////////////

long long convertDecimalToBinary(uint8_t n)
{
    long long binaryNumber = 0;
    int remainder, i = 1, step = 1;
    while (n!=0)
    {
        remainder = n%2;
        //printf("Step %d: %d/2, Remainder = %d, Quotient = %d\n", step++, n, remainder, n/2);
        n /= 2;
        binaryNumber += remainder*i;
        //printf("-- inside function each step remainder ---. %d\n", remainder);
        //printf("-- inside function each step---. %lld\n", binaryNumber);
        i *= 10;
    }
    return binaryNumber;
}
```

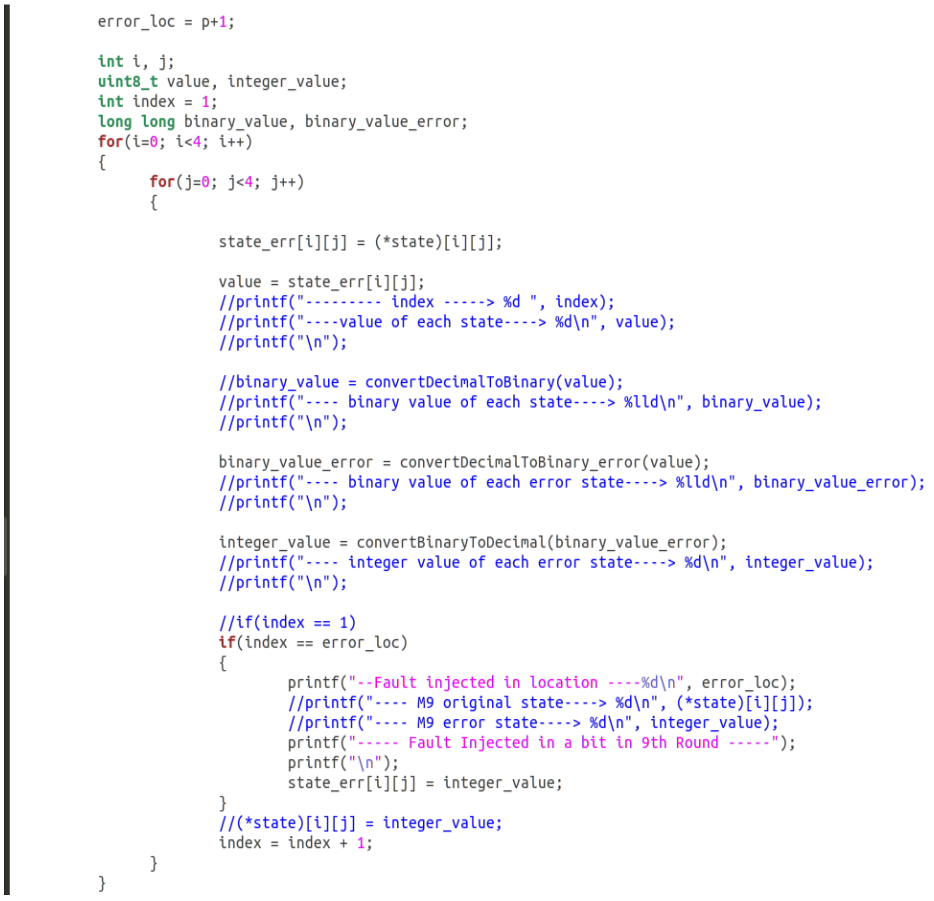


```
//////////////////////////////  Fault Injector Function ///////////////////////////////////////////////////////

long long convertDecimalToBinary_error(uint8_t n)
{
    long long binaryNumber = 0;
    int remainder, i = 1, step = 1;
    while (n!=0)
    {
        remainder = n%2;
        //printf("-- inside function each step remainder ---. %d\n", remainder);
        if(step == 1)  //// error in LSB
        {
                if(remainder == 0)
                {
                        remainder = 1;
                        //printf("inside 1 --- \n");
                }
                else
                {
                        remainder = 0;
                        //printf("inside 0 --- \n");
                }
        }
        //printf("Step %d: %d/2, Remainder = %d, Quotient = %d\n", step++, n, remainder, n/2);
        n /= 2;
        binaryNumber += remainder*i;
        //printf("-- inside function each step---. %lld\n", binaryNumber);
        i *= 10;
        step = step + 1;
    }
    return binaryNumber;
}
```

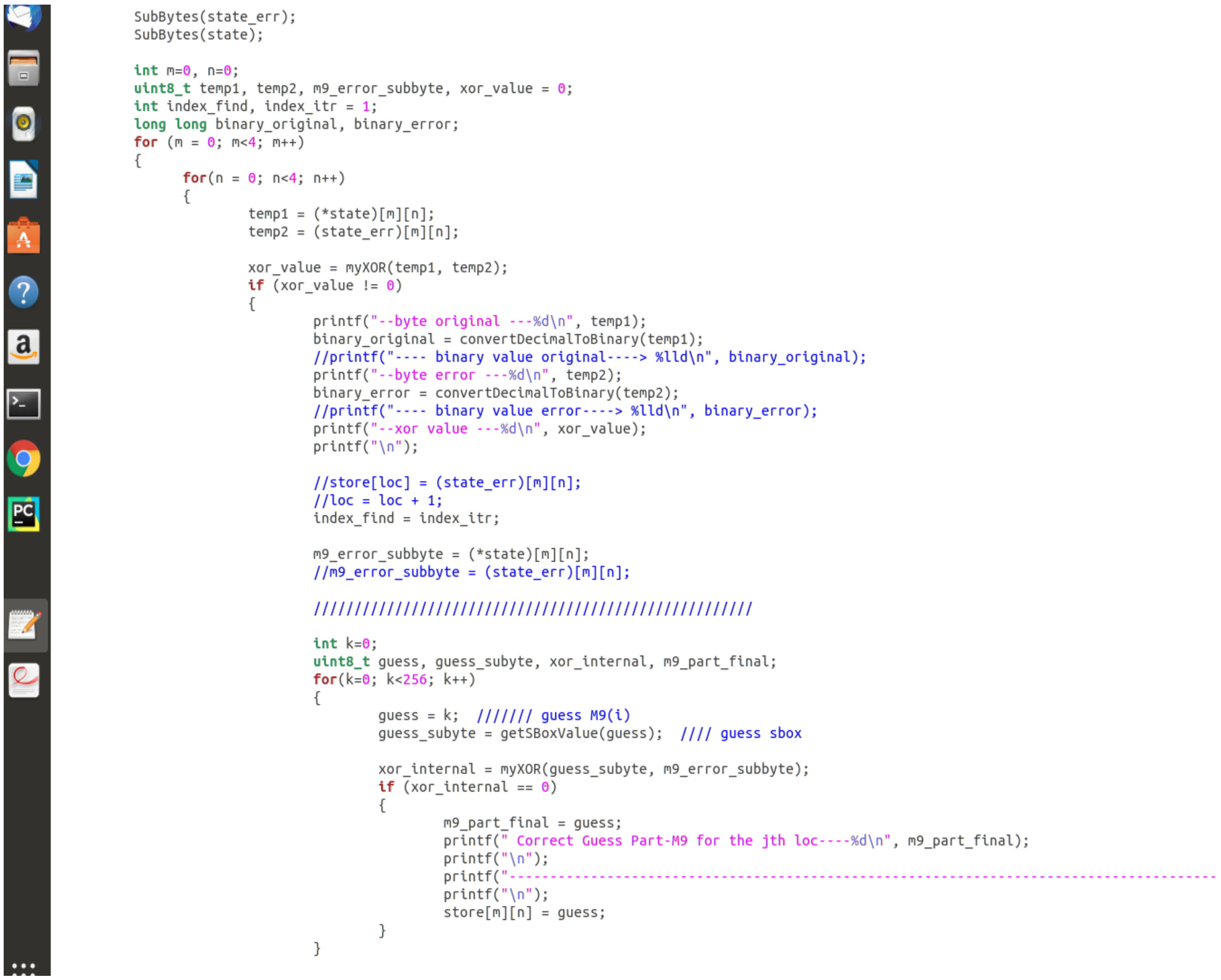

```
}
////////////////////////////////   Power Function  ///////////////////////////////////////////////

int power(int X, int Y)
{
    int i;
    int value = 1;
    for (i = 0; i < Y; i++)

    value *= X;

    return value;
}

////////////////////////////////   Binary to Decimal Function  ///////////////////////////////////////////////

uint8_t convertBinaryToDecimal(long long n)
{
    uint8_t decimalNumber = 0, remainder;
    int i = 0;
    while (n!=0)
    {
        remainder = n%10;
        n /= 10;
        decimalNumber += remainder * power(2,i);
        ++i;
    }
    return decimalNumber;
}

////////////////////////////////   XOR Function  ///////////////////////////////////////////////

uint8_t myXOR(uint8_t x, uint8_t y)
{
    uint8_t res = 0; // Initialize result

    // Assuming 8-bit Integer
    for (int i = 7; i >= 0; i--)
    {
       // Find current bits in x and y
       bool b1 = x & (1 << i);
       bool b2 = y & (1 << i);

        // If both are 1 then 0 else xor is same as OR
        bool xoredBit = (b1 & b2) ? 0 : (b1 | b2);

        // Update result
        res <<= 1;
        res |= xoredBit;
    }
    return res;
}
```



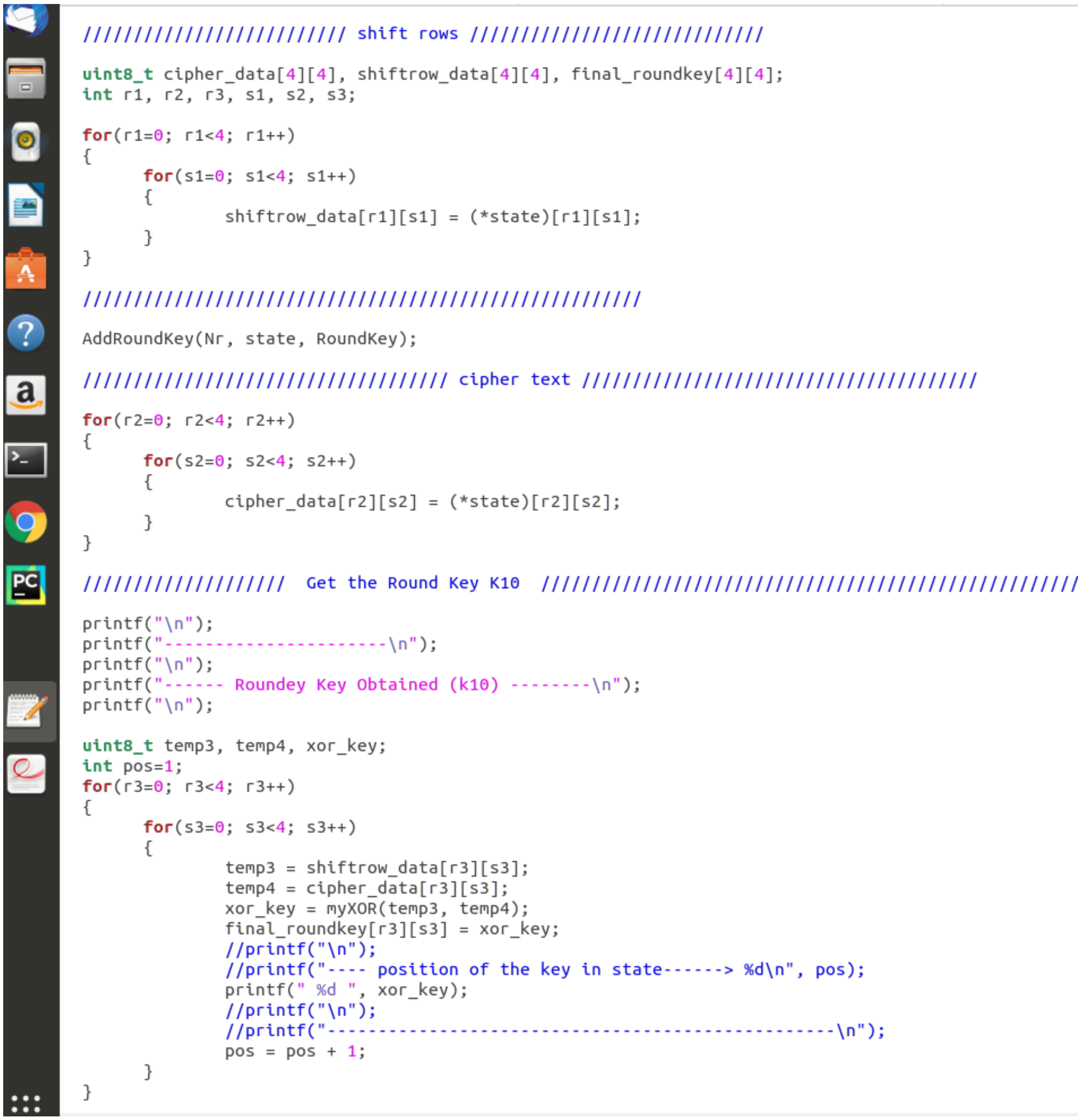
```
/////////////////////////////////////   Cipher Function  ////////////////////////////////////////////

// Cipher is the main function that encrypts the PlainText.
static void Cipher(state_t* state, const uint8_t* RoundKey)
{
  uint8_t round = 0;

  // Add the First round key to the state before starting the rounds.
  AddRoundKey(0, state, RoundKey);

  // There will be Nr rounds.
  // The first Nr-1 rounds are identical.
  // These Nr-1 rounds are executed in the loop below.
  for (round = 1; round < Nr; ++round)
  {
    SubBytes(state);
    ShiftRows(state);
    MixColumns(state);
    AddRoundKey(round, state, RoundKey);
  }

  ///////////////////////////////////  State Copy for Emergency  ///////////////////////////////

  int x1, y1;
  uint8_t state_copy[4][4];
  for(x1=0; x1<4; x1++)
  {
        for(y1=0; y1<4; y1++)
        {
                state_copy[x1][y1] = (*state)[x1][y1];
        }
  }

  //printf("--check--%d\n", (*state)[0][3]);
  //////////////////////////////////////////////  Main Code For DFA Attack  ///////////////////////////////////

  uint8_t state_err[4][4];
  uint8_t store[4][4];
  int loc = 0, p;
  int error_loc = 0;
```



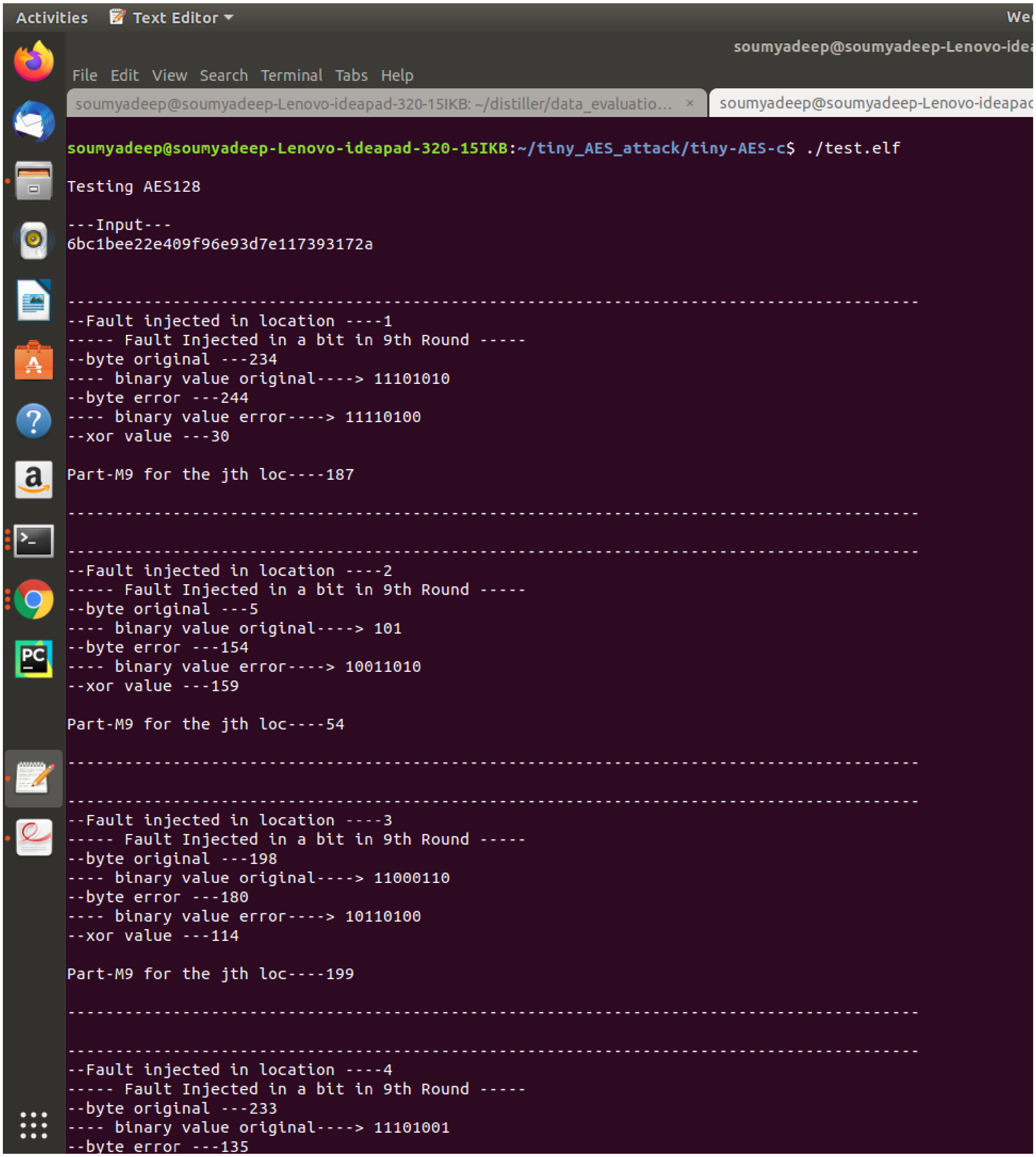
```
  error_loc = p+1;

  int i, j;
  uint8_t value, integer_value;
  int index = 1;
  long long binary_value, binary_value_error;
  for(i=0; i<4; i++)
  {
        for(j=0; j<4; j++)
        {

                state_err[i][j] = (*state)[i][j];

                value = state_err[i][j];
                //printf("--------- index -----> %d ", index);
                //printf("----value of each state----> %d\n", value);
                //printf("\n");

                //binary_value = convertDecimalToBinary(value);
                //printf("---- binary value of each state----> %lld\n", binary_value);
                //printf("\n");

                binary_value_error = convertDecimalToBinary_error(value);
                //printf("---- binary value of each error state----> %lld\n", binary_value_error);
                //printf("\n");

                integer_value = convertBinaryToDecimal(binary_value_error);
                //printf("---- integer value of each error state----> %d\n", integer_value);
                //printf("\n");

                //if(index == 1)
                if(index == error_loc)
                {
                        printf("--Fault injected in location ----%d\n", error_loc);
                        //printf("---- M9 original state----> %d\n", (*state)[i][j]);
                        //printf("---- M9 error state----> %d\n", integer_value);
                        printf("----- Fault Injected in a bit in 9th Round -----");
                        printf("\n");
                        state_err[i][j] = integer_value;
                }
                //(*state)[i][j] = integer_value;
                index = index + 1;
        }
  }
```



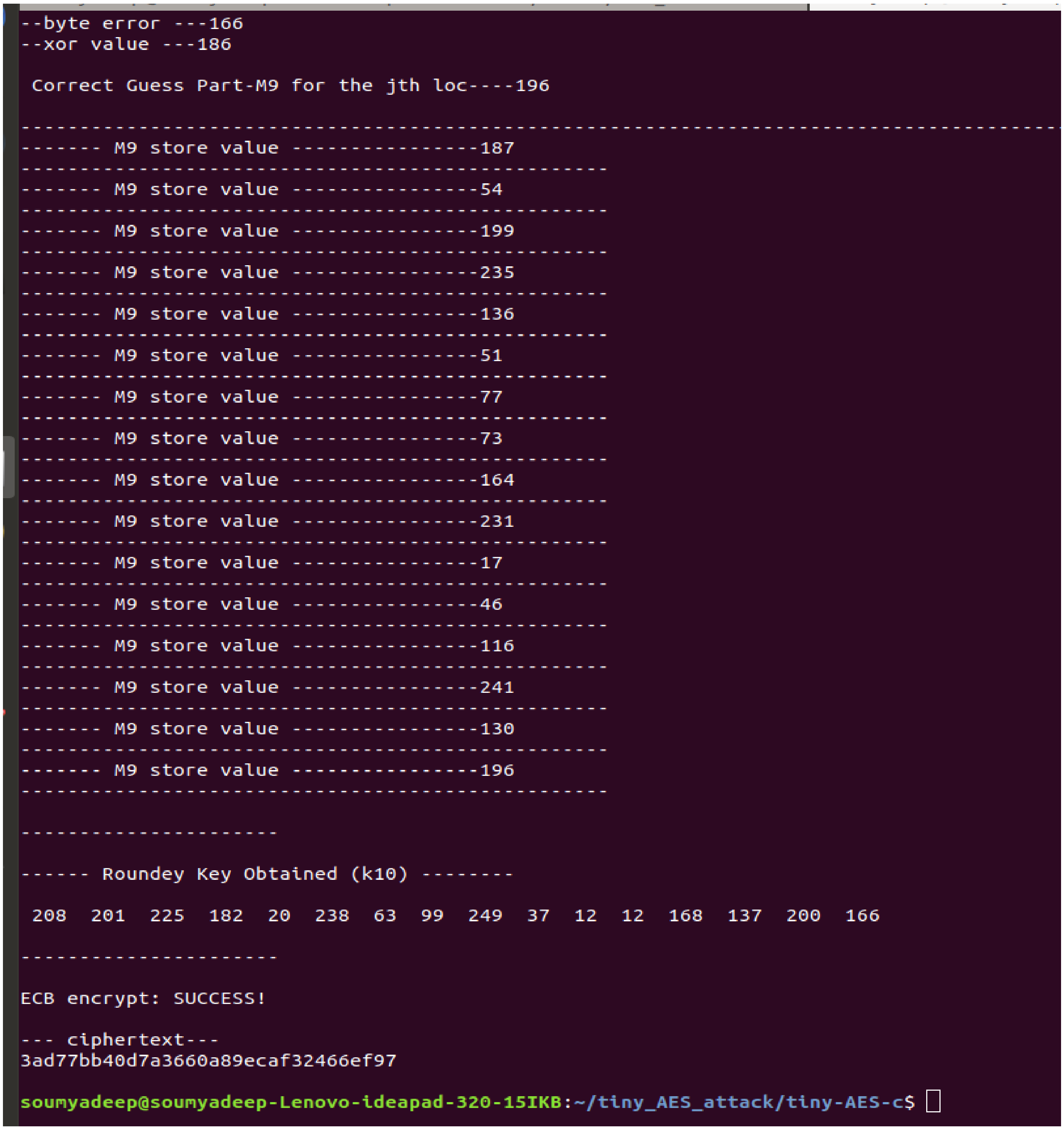
```
SubBytes(state_err);
SubBytes(state);

int m=0, n=0;
uint8_t temp1, temp2, m9_error_subbyte, xor_value = 0;
int index_find, index_itr = 1;
long long binary_original, binary_error;
for (m = 0; m<4; m++)
{
      for(n = 0; n<4; n++)
      {
              temp1 = (*state)[m][n];
              temp2 = (state_err)[m][n];

              xor_value = myXOR(temp1, temp2);
              if (xor_value != 0)
              {
                      printf("--byte original ---%d\n", temp1);
                      binary_original = convertDecimalToBinary(temp1);
                      //printf("---- binary value original----> %lld\n", binary_original);
                      printf("--byte error ---%d\n", temp2);
                      binary_error = convertDecimalToBinary(temp2);
                      //printf("---- binary value error----> %lld\n", binary_error);
                      printf("--xor value ---%d\n", xor_value);
                      printf("\n");

                      //store[loc] = (state_err)[m][n];
                      //loc = loc + 1;
                      index_find = index_itr;

                      m9_error_subbyte = (*state)[m][n];
                      //m9_error_subbyte = (state_err)[m][n];

                      //////////////////////////////////////////////////////

                      int k=0;
                      uint8_t guess, guess_subyte, xor_internal, m9_part_final;
                      for(k=0; k<256; k++)
                      {
                              guess = k;  /////// guess M9(i)
                              guess_subyte = getSBoxValue(guess);  //// guess sbox

                              xor_internal = myXOR(guess_subyte, m9_error_subbyte);
                              if (xor_internal == 0)
                              {
                                      m9_part_final = guess;
                                      printf(" Correct Guess Part-M9 for the jth loc----%d\n", m9_part_final);
                                      printf("\n");
                                      printf("---------------------------------------------------------------------------------------
                                      printf("\n");
                                      store[m][n] = guess;
                              }
                      }
```


```
////////////////////////// shift rows ////////////////////////////

uint8_t cipher_data[4][4], shiftrow_data[4][4], final_roundkey[4][4];
int r1, r2, r3, s1, s2, s3;

for(r1=0; r1<4; r1++)
{
      for(s1=0; s1<4; s1++)
      {
              shiftrow_data[r1][s1] = (*state)[r1][s1];
      }
}

///////////////////////////////////////////////////////

AddRoundKey(Nr, state, RoundKey);

//////////////////////////////////// cipher text ///////////////////////////////////////

for(r2=0; r2<4; r2++)
{
      for(s2=0; s2<4; s2++)
      {
              cipher_data[r2][s2] = (*state)[r2][s2];
      }
}

////////////////////  Get the Round Key K10  /////////////////////////////////////////////////////

printf("\n");
printf("----------------------\n");
printf("\n");
printf("------ Roundey Key Obtained (k10) --------\n");
printf("\n");

uint8_t temp3, temp4, xor_key;
int pos=1;
for(r3=0; r3<4; r3++)
{
      for(s3=0; s3<4; s3++)
      {
              temp3 = shiftrow_data[r3][s3];
              temp4 = cipher_data[r3][s3];
              xor_key = myXOR(temp3, temp4);
              final_roundkey[r3][s3] = xor_key;
              //printf("\n");
              //printf("---- position of the key in state------> %d\n", pos);
              printf(" %d ", xor_key);
              //printf("\n");
              //printf("--------------------------------------------------\n");
              pos = pos + 1;
      }
}
```

Results :- The Results of the AES K10 Round key and the M9 values are shown below -


Activities
Text Editor
soumyadeep@soumyadeep-Lenovo-idea
File Edit View Search Terminal Tabs Help
soumyadeep@soumyadeep-Lenovo-ideapad-320-15IKB: ~/distiller/data_evaluatio...
soumyadeep@soumyadeep-Lenovo-ideapac

```
soumyadeep@soumyadeep-Lenovo-ideapad-320-15IKB:~/tiny_AES_attack/tiny-AES-c$ ./test.elf

Testing AES128

---Input---
6bc1bee22e409f96e93d7e117393172a


-----------------------------------------------------------------------------------------
--Fault injected in location ----1
----- Fault Injected in a bit in 9th Round -----
--byte original ---234
---- binary value original----> 11101010
--byte error ---244
---- binary value error----> 11110100
--xor value ---30

Part-M9 for the jth loc----187

-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
--Fault injected in location ----2
----- Fault Injected in a bit in 9th Round -----
--byte original ---5
---- binary value original----> 101
--byte error ---154
---- binary value error----> 10011010
--xor value ---159

Part-M9 for the jth loc----54

-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
--Fault injected in location ----3
----- Fault Injected in a bit in 9th Round -----
--byte original ---198
---- binary value original----> 11000110
--byte error ---180
---- binary value error----> 10110100
--xor value ---114

Part-M9 for the jth loc----199

-----------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------
--Fault injected in location ----4
----- Fault Injected in a bit in 9th Round -----
--byte original ---233
---- binary value original----> 11101001
--byte error ---135
```


```
--byte error ---166
--xor value ---186

 Correct Guess Part-M9 for the jth loc----196

------------------------------------------------------------------------------------------
------- M9 store value ----------------187
--------------------------------------------------
------- M9 store value ----------------54
--------------------------------------------------
------- M9 store value ----------------199
--------------------------------------------------
------- M9 store value ----------------235
--------------------------------------------------
------- M9 store value ----------------136
--------------------------------------------------
------- M9 store value ----------------51
--------------------------------------------------
------- M9 store value ----------------77
--------------------------------------------------
------- M9 store value ----------------73
--------------------------------------------------
------- M9 store value ----------------164
--------------------------------------------------
------- M9 store value ----------------231
--------------------------------------------------
------- M9 store value ----------------17
--------------------------------------------------
------- M9 store value ----------------46
--------------------------------------------------
------- M9 store value ----------------116
--------------------------------------------------
------- M9 store value ----------------241
--------------------------------------------------
------- M9 store value ----------------130
--------------------------------------------------
------- M9 store value ----------------196
--------------------------------------------------

---------------------

------ Roundey Key Obtained (k10) --------

 208  201  225  182  20  238  63  99  249  37  12  12  168  137  200  166

---------------------

ECB encrypt: SUCCESS!

--- ciphertext---
3ad77bb40d7a3660a89ecaf32466ef97

soumyadeep@soumyadeep-Lenovo-ideapad-320-15IKB:~/tiny_AES_attack/tiny-AES-c$ 
```

Information to Run the Code :- Code is added with the report. Run it directly by these commands.

- cd AES_code_directorey/
- rm -r test.map test.o aes.o test.elf
- make (Please ignore the warnings)
- ./test.elf

-- END --