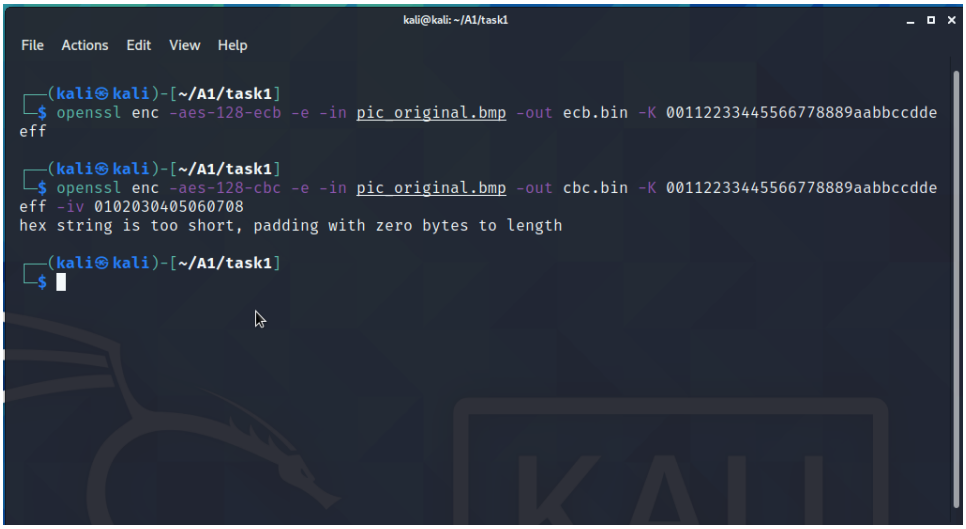
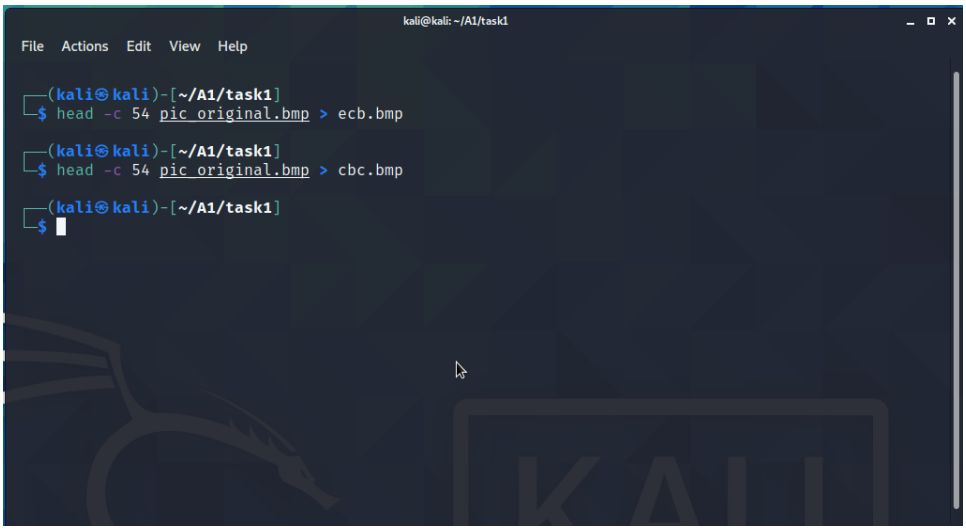
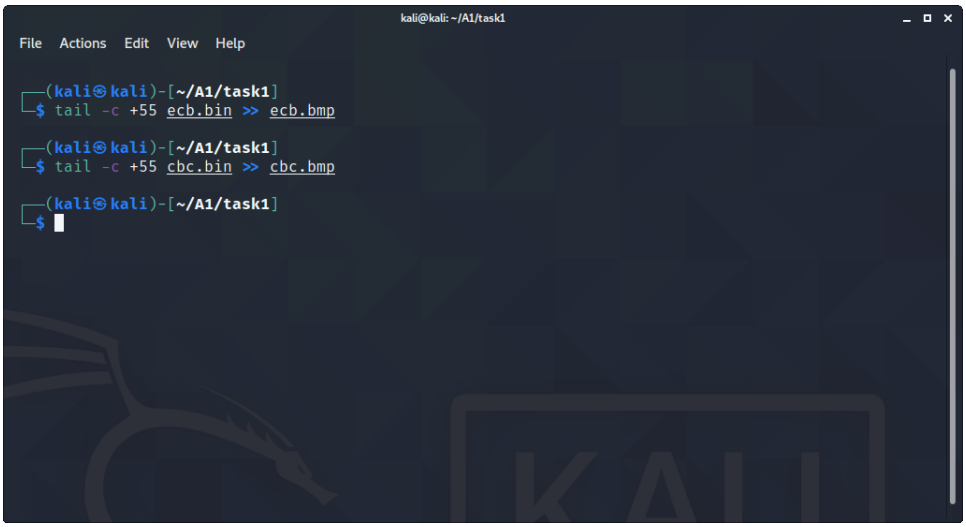
SECRET KEY-TASK 3



- First, we encrypt the file using 2 encrypted method as seen above
- Notice that, while encrypt using cbc, the message is too short compared to the key, that why we need some padding in order to match with the length of the key



- Then we preserve 54 bits header of the file, since they are needed to properly displayed

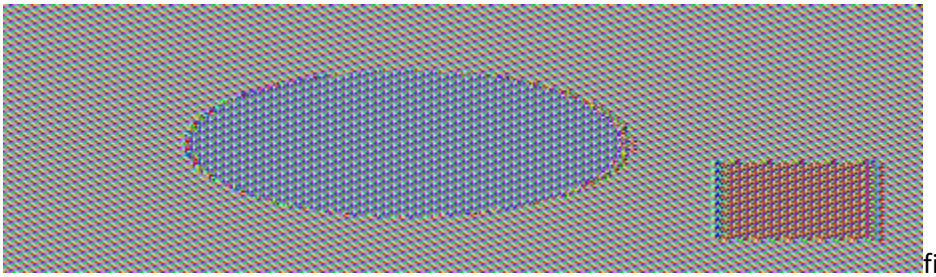


- Finally, we append the encrypted bits from offset 55 to the end of file

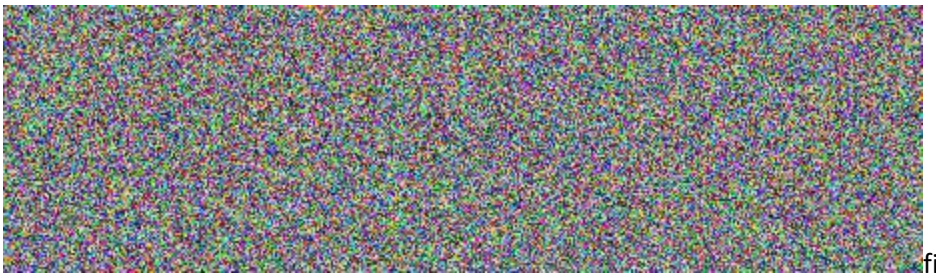

Orignal file

Outcome

- What we observe is the encrypted file using ecb looks similar to the original file.
    - This is because ecb using 1 to 1 mapping encryption methods with single key, The key can then be easily identify


file encrypted using ecb

- Encrypting file using cbc make it impossible to identify the original message
    - This is because cbc using 1 to 1 mapping but the key change with each block of the plaintext

 file encrypted using cbc

SECRET KEY –TASK 7

devel@localhost:~/PrivateKeyTask7

File Edit View Search Terminal Help

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <openssl/evp.h>

unsigned char outbuf[1024];
char word[17];
FILE *dict;
FILE *out;
const unsigned char *plaintext = "This is a top secret";
const unsigned char *ciphertext = "764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2";
const unsigned char *iv = "aabbccddeeff00998877665544332211";

int outlen, tmplen;
void hexToDec(const unsigned char*hexdec,char*result){

    int i =0;
    while (hexdec[i]!='\0') {
        switch (hexdec[i]) {
        case '0':
            strcat(result,"0000");
            break;
        case '1':
            strcat(result,"0001");
            break;
        case '2':
            strcat(result,"0010");
            break;
        case '3':
            strcat(result,"0011");
            break;
        case '4':
            strcat(result,"0100");
            break;
        case '5':
            strcat(result,"0101");
            break;
        case '6':
            strcat(result,"0110");
            break;
        case '7':
            strcat(result,"0111");
            break;
        case '8':
            strcat(result,"1000");
            break;
        case '9':
            strcat(result,"1001");
            break;
        case 'A':
        case 'a':
            strcat(result,"1010");
            break;
        case 'B':
        case 'b':
            strcat(result,"1011");
            break;
        case 'C':
        case 'c':
```

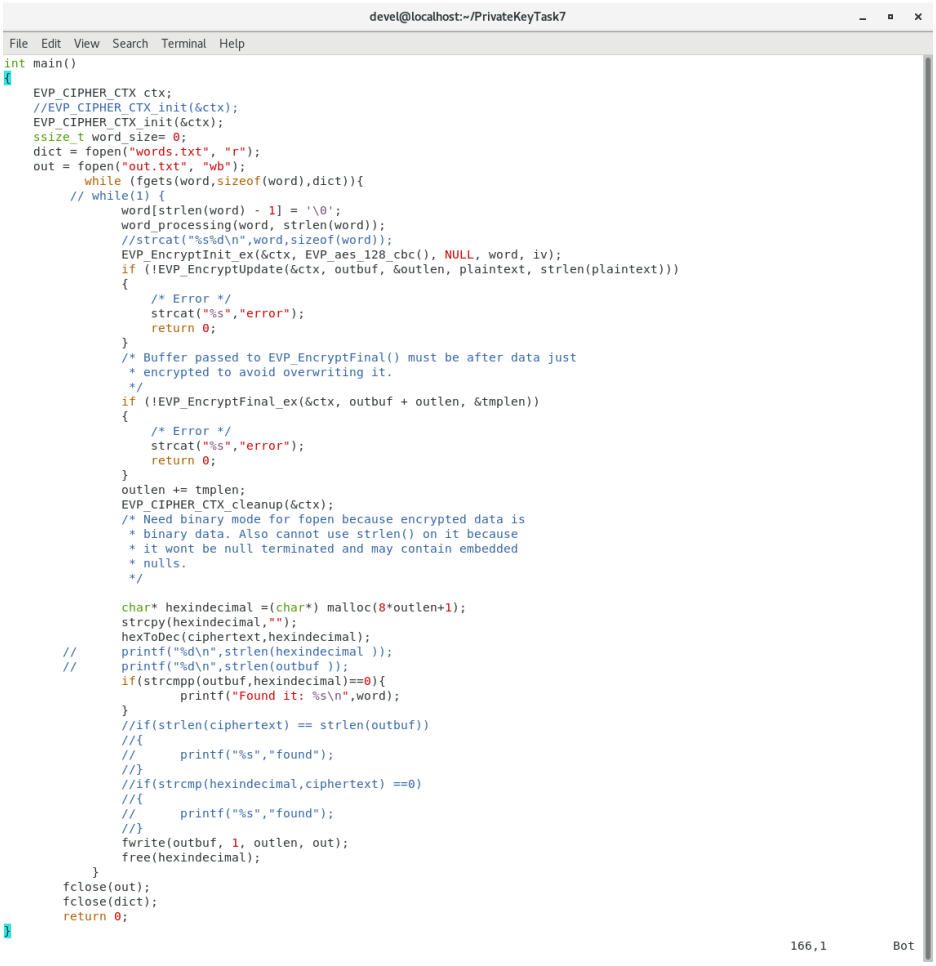
1,9 Top

devel@localhost:~/PrivateKeyTask7

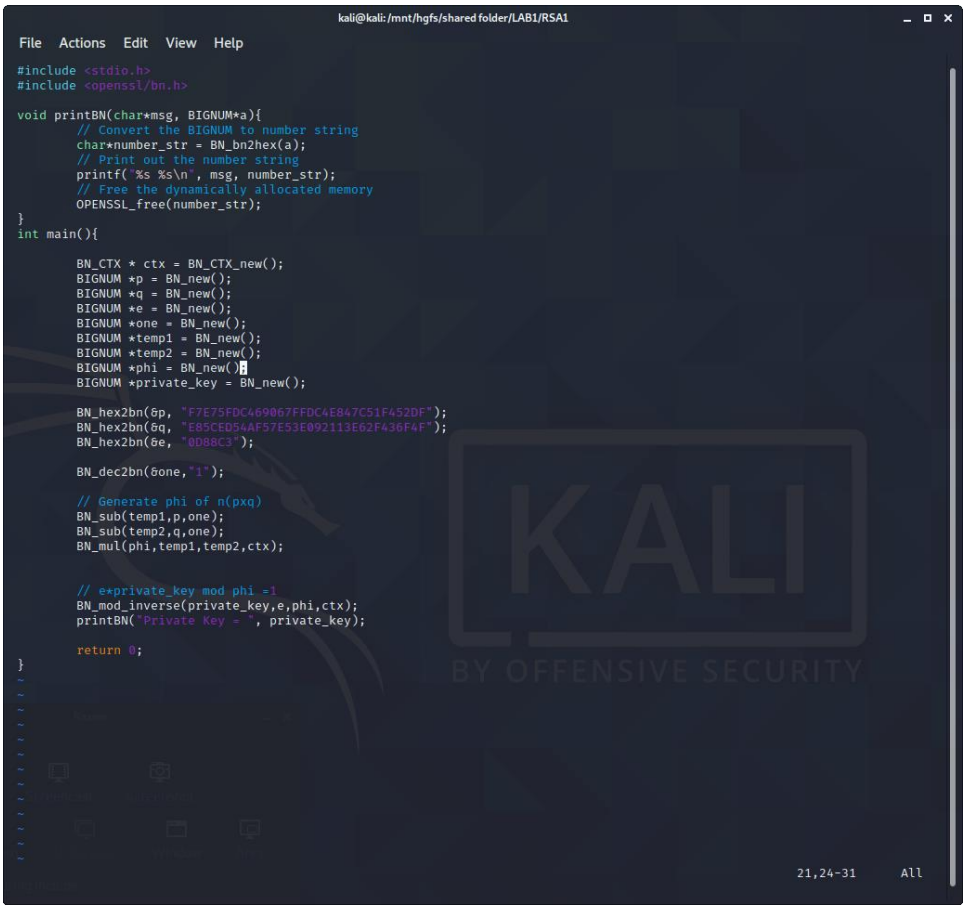
File Edit View Search Terminal Help

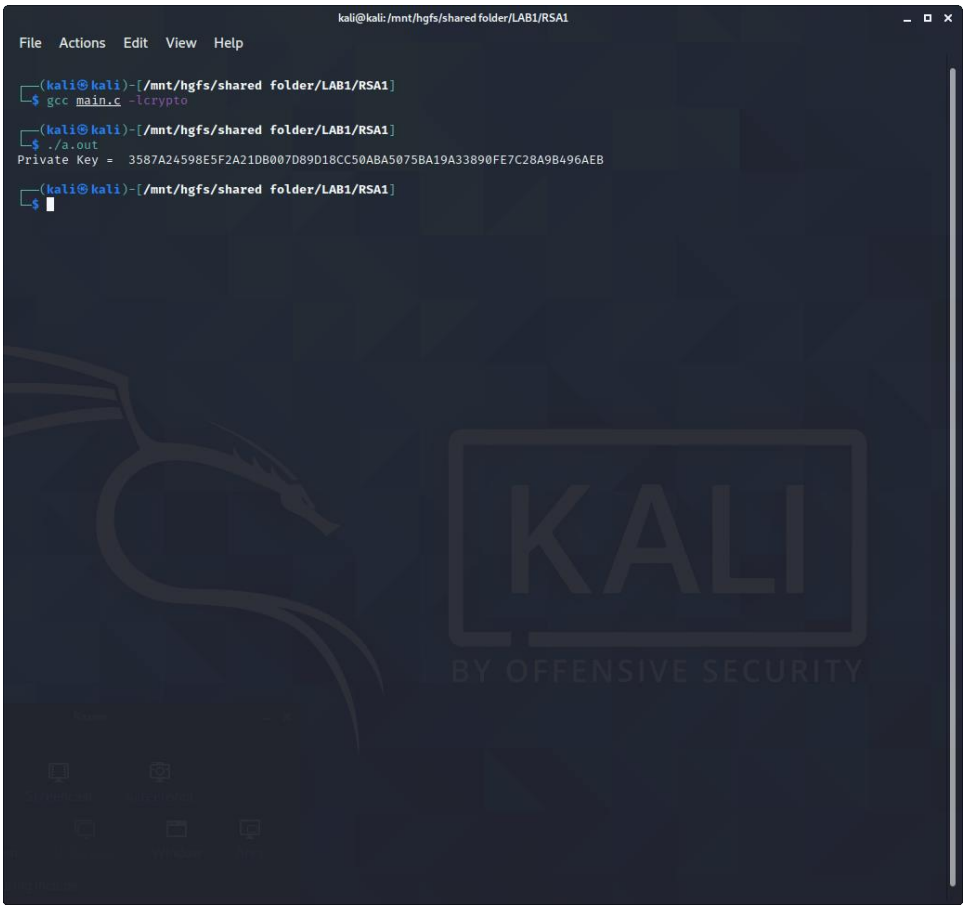
```
        i++;
    }
}
int strcmpp(char const* string1, char const* string2){
        int result=0;
        while(*string1++!='\0'&&*string2++!='\0')
        {
          if((tolower(*string1)-tolower(*string2))!=0){
                result=1;
                return result;
          }
        }
        return result;
}

void word_processing(char *word, int len)
{
    int numWordAdd = 16-len;
    while (numWordAdd--)
    {
        word[len++] = '#';
    }
    word[len] = '\0';
    int i;
    for (i = 0; word[i]; i++)
    {
        word[i] = tolower(word[i]);
    }
}
int main()
{
    EVP_CIPHER_CTX ctx;
    //EVP_CIPHER_CTX_init(&ctx);
    EVP_CIPHER_CTX_init(&ctx);
    ssize_t word_size= 0;
    dict = fopen("words.txt", "r");
    out = fopen("out.txt", "wb");
          while (fgets(word,sizeof(word),dict)){
         // while(1) {
                word[strlen(word) - 1] = '\0';
                word_processing(word, strlen(word));
                //strcat("%s%d\n",word,sizeof(word));
                EVP_EncryptInit_ex(&ctx, EVP_aes_128_cbc(), NULL, word, iv);
                if (!EVP_EncryptUpdate(&ctx, outbuf, &outlen, plaintext, strlen(plaintext)))
                {
                    /* Error */
                    strcat("%s","error");
                    return 0;
                }
                /* Buffer passed to EVP_EncryptFinal() must be after data just
                 * encrypted to avoid overwriting it.
                 */
                if (!EVP_EncryptFinal_ex(&ctx, outbuf + outlen, &tmplen))
                {
                    /* Error */
                    strcat("%s","error");
                    return 0;
                }
                outlen += tmplen;
                EVP_CIPHER_CTX_cleanup(&ctx);
```

78,9 72%


devel@localhost:~/PrivateKeyTask7
File Edit View Search Terminal Help
int main()
{
EVP_CIPHER_CTX ctx;
//EVP_CIPHER_CTX_init(&ctx);
EVP_CIPHER_CTX_init(&ctx);
ssize_t word_size= 0;
dict = fopen("words.txt", "r");
out = fopen("out.txt", "wb");
while (fgets(word,sizeof(word),dict)){
// while(1) {
word[strlen(word) - 1] = '\0';
word_processing(word, strlen(word));
//strcat("%s%d\n",word,sizeof(word));
EVP_EncryptInit_ex(&ctx, EVP_aes_128_cbc(), NULL, word, iv);
if (!EVP_EncryptUpdate(&ctx, outbuf, &outlen, plaintext, strlen(plaintext)))
{
/* Error */
strcat("%s","error");
return 0;
}
/* Buffer passed to EVP_EncryptFinal() must be after data just
* encrypted to avoid overwriting it.
*/
if (!EVP_EncryptFinal_ex(&ctx, outbuf + outlen, &tmplen))
{
/* Error */
strcat("%s","error");
return 0;
}
outlen += tmplen;
EVP_CIPHER_CTX_cleanup(&ctx);
/* Need binary mode for fopen because encrypted data is
* binary data. Also cannot use strlen() on it because
* it wont be null terminated and may contain embedded
* nulls.
*/
char* hexindecimal =(char*) malloc(8*outlen+1);
strcpy(hexindecimal,"");
hexToDec(ciphertext,hexindecimal);
// printf("%d\n",strlen(hexindecimal));
// printf("%d\n",strlen(outbuf));
if(strcmpp(outbuf,hexindecimal)==0){
printf("Found it: %s\n",word);
}
//if(strlen(ciphertext) == strlen(outbuf))
//{
// printf("%s","found");
//}
//if(strcmp(hexindecimal,ciphertext) ==0)
//{
// printf("%s","found");
//}
fwrite(outbuf, 1, outlen, out);
free(hexindecimal);
}
fclose(out);
fclose(dict);
return 0;
}
166,1 Bot

RSA-TASK 1


kali@kali: /mnt/hgfs/shared folder/LAB1/RSA1
File Actions Edit View Help
#include <stdio.h>
#include <openssl/bn.h>
void printBN(char*msg, BIGNUM*a){
// Convert the BIGNUM to number string
char*number_str = BN_bn2hex(a);
// Print out the number string
printf("%s %s\n", msg, number_str);
// Free the dynamically allocated memory
OPENSSL_free(number_str);
}
int main(){
BN_CTX * ctx = BN_CTX_new();
BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *e = BN_new();
BIGNUM *one = BN_new();
BIGNUM *temp1 = BN_new();
BIGNUM *temp2 = BN_new();
BIGNUM *phi = BN_new();
BIGNUM *private_key = BN_new();
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one,"1");
// Generate phi of n(pxq)
BN_sub(temp1,p,one);
BN_sub(temp2,q,one);
BN_mul(phi,temp1,temp2,ctx);
// e*private_key mod phi =1
BN_mod_inverse(private_key,e,phi,ctx);
printBN("Private Key = ", private_key);
return 0;
}
21,24-31 All



- Given p,q we can calculate phi of (q*p), by calculating (phi of q) = q –1 and (phi of p) = p-1
- then we know that m^phi =1 mod n

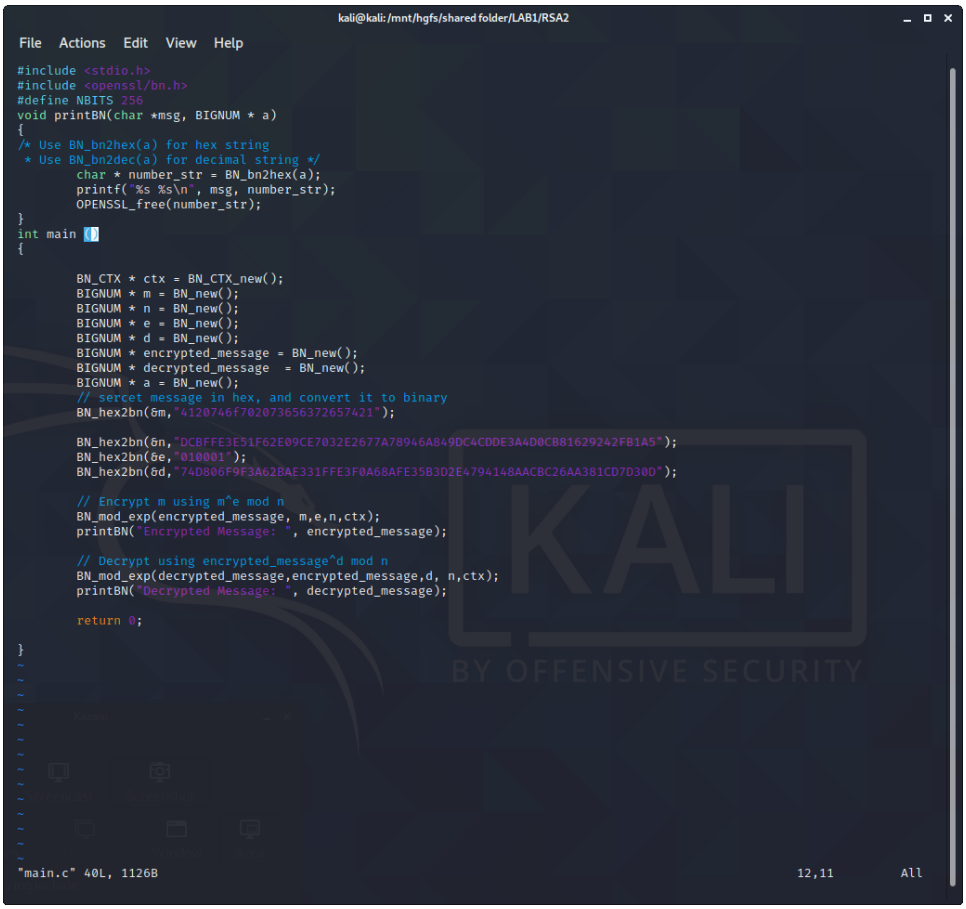
=> m^(phi*random +1) = m mod n

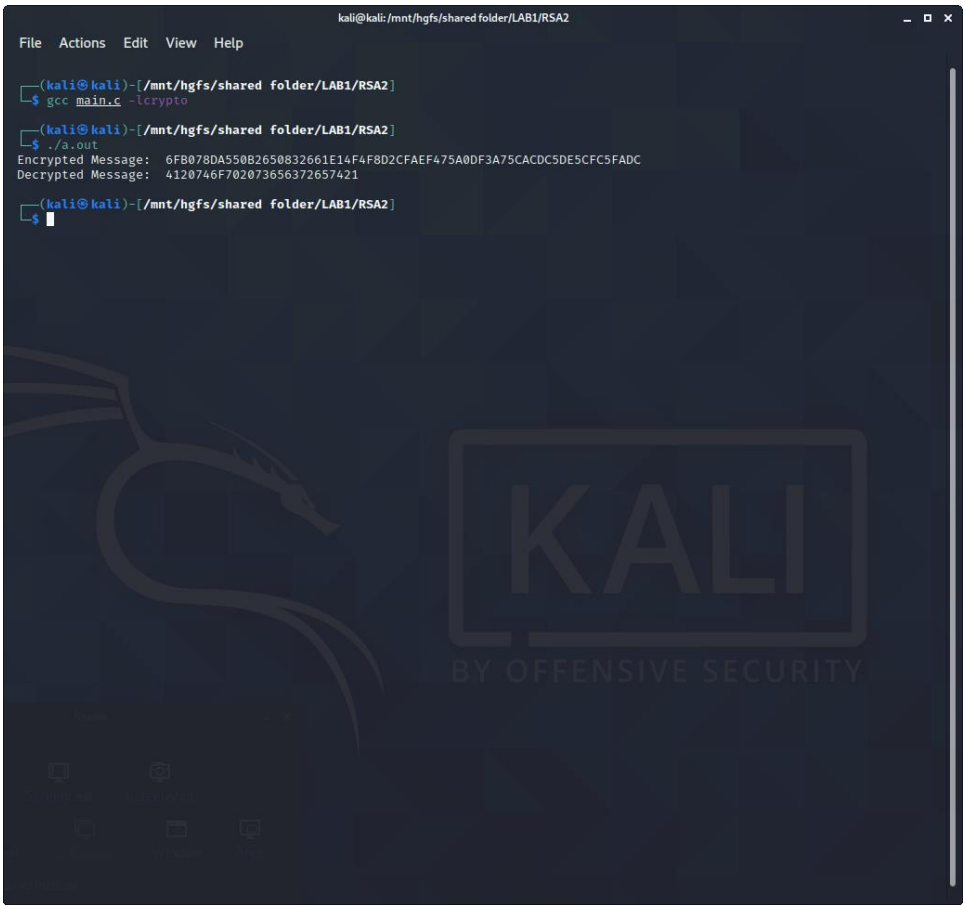
=> phi*random+1 = e*d

=> e*d = 1 mod phi

=> d = inverse modular of e mod phi

RSA-TASK 2


kali@kali: /mnt/hgfs/shared folder/LAB1/RSA2
File Actions Edit View Help
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
 * Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}
int main ()
{
BN_CTX * ctx = BN_CTX_new();
BIGNUM * m = BN_new();
BIGNUM * n = BN_new();
BIGNUM * e = BN_new();
BIGNUM * d = BN_new();
BIGNUM * encrypted_message = BN_new();
BIGNUM * decrypted_message = BN_new();
BIGNUM * a = BN_new();
// sercet message in hex, and convert it to binary
BN_hex2bn(&m,"4120746f702073656372657421");
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e,"010001");
BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
// Encrypt m using m^e mod n
BN_mod_exp(encrypted_message, m,e,n,ctx);
printBN("Encrypted Message: ", encrypted_message);
// Decrypt using encrypted_message^d mod n
BN_mod_exp(decrypted_message,encrypted_message,d, n,ctx);
printBN("Decrypted Message: ", decrypted_message);
return 0;
}
"main.c" 40L, 1126B
12,11
All



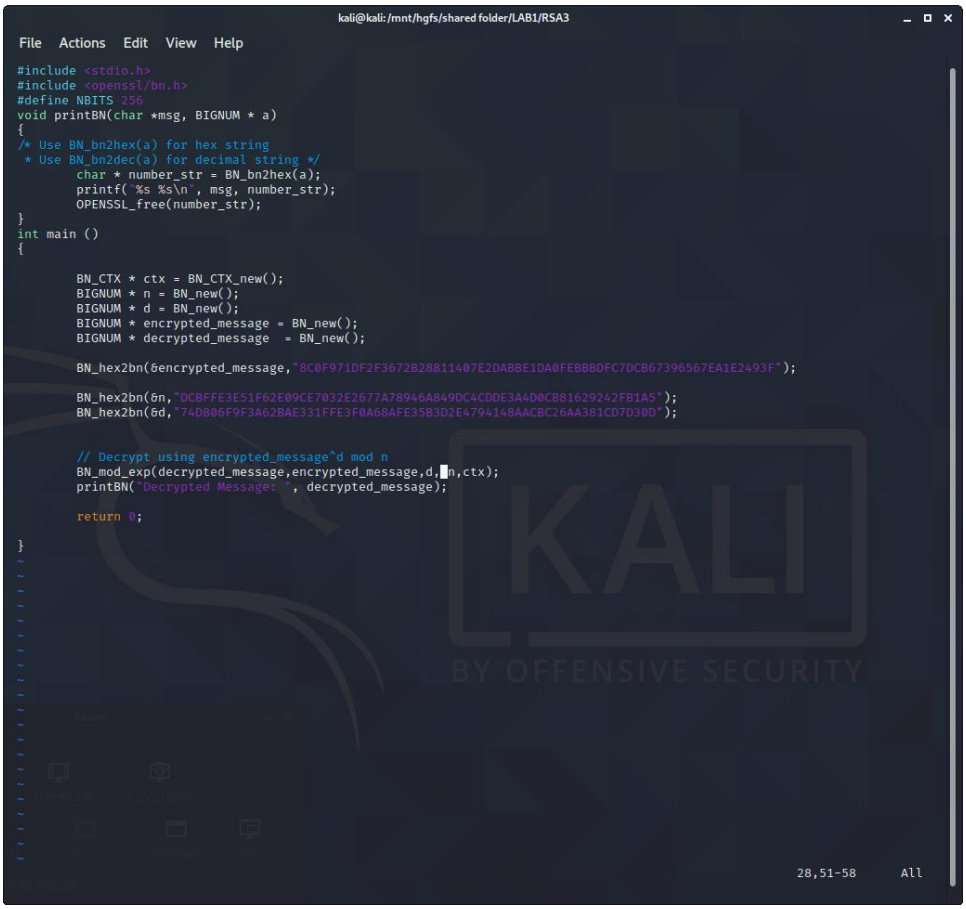
- Convert Ascii message to hex using command

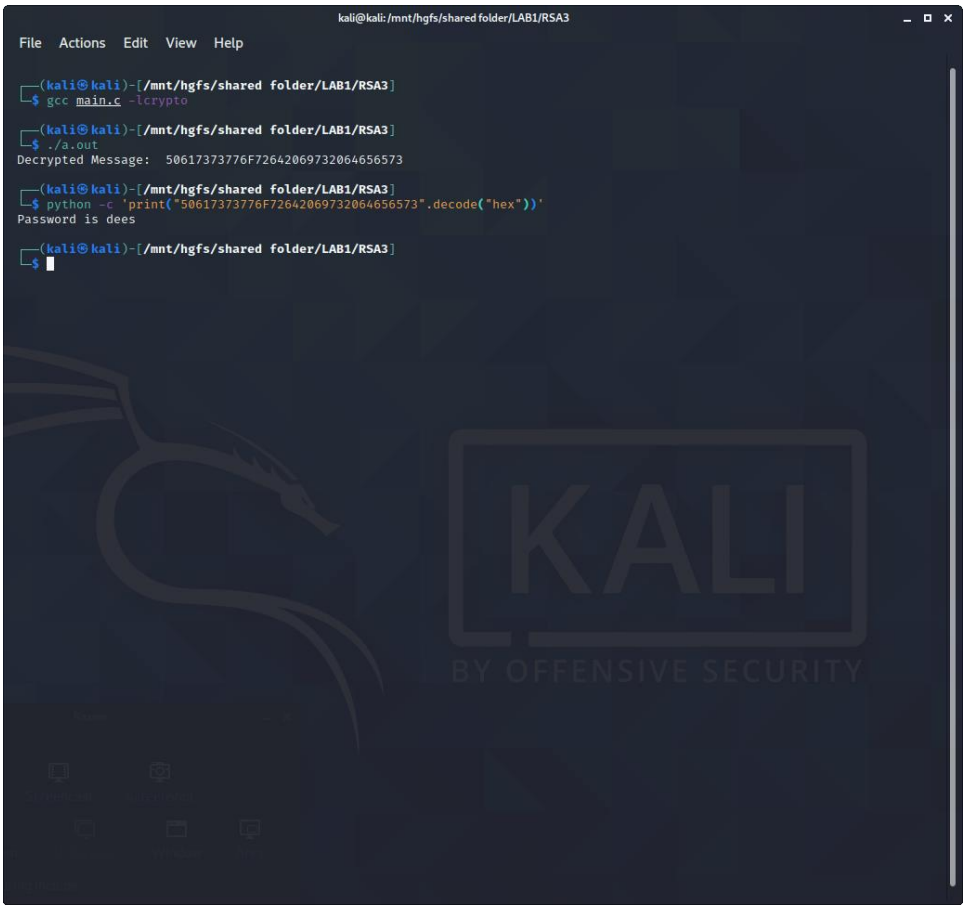
  python -c ’print("A top secret!".encode("hex"))’

  We get message = 4120746f702073656372657421

- Then we encrypt it using message ^ e mod n
- Then we double check by decrypt the ciphertext and compare with original message
- The decryption using encrypted_message ^d mod n

RSA-TASK 3


kali@kali: /mnt/hgfs/shared folder/LAB1/RSA3
File Actions Edit View Help
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char *msg, BIGNUM * a)
{
/* Use BN_bn2hex(a) for hex string
 * Use BN_bn2dec(a) for decimal string */
char * number_str = BN_bn2hex(a);
printf("%s %s\n", msg, number_str);
OPENSSL_free(number_str);
}
int main ()
{
BN_CTX * ctx = BN_CTX_new();
BIGNUM * n = BN_new();
BIGNUM * d = BN_new();
BIGNUM * encrypted_message = BN_new();
BIGNUM * decrypted_message = BN_new();
BN_hex2bn(&encrypted_message,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
// Decrypt using encrypted_message^d mod n
BN_mod_exp(decrypted_message,encrypted_message,d,n,ctx);
printBN("Decrypted Message: ", decrypted_message);
return 0;
}
28,51-58 All



- The decryption using encrypted_message ^d mod n
- Then we get a result in hex value
- Then we convert it back to ascii using command

  python -c ’print(message.encode("hex"))’