# Technical Report

## SDES

For the SDES implementation we decided to use C as the main programming language. This programming language allows us to improve the execution time and the memory use so we can develop a better application in the algorithms for encryption and decryption.

The first part to develop the SDES algorithm we a set of functions that help us to organize the code and implement the algorithm in a better way. We explain them below.

1. **byte.c / byte.h**

In these files we have the an structure data that contains a byte representation as a integers and also the functions to turn from a type to one other. The representation of a byte we called TByte that contains eight integers.

Functions:

1. **INTtoB**: recives an integer and turn it into a TBByte structure
2. **BtoINT**: recives a TBByte strcture and turn it into a integer
3. **UStoB**: Recives an unsigned char data type and turn it into a TByte struct
4. **BtoUS**: Recives a TByte struct and turn it into an unsigned char data type
5. **printTByte**: Recives a char pointer, a TByte data, and an integer if n equals 4 print a TByte with 4 int, if n equals to 8 the TByte size is 10 and print its Hex, dec, and char representation, if n equals 10 only print the bytes of the TByte

|  |  |
| --- | --- |
| #include "byte.h"  TByte INTtoB(int b){  TByte byte;  /\*Conversion\*/  **return** byte;  }  int BtoINT(TByte byte){  int B;  B=((byte.p0)<<9)  +/\*conversion\*/  **return** B;  }  TByte UStoB(unsigned char b){  TByte byte;  /\*Conversion\*/  byte.p0=b%2;  **return** byte;  }  unsigned char BtoUS(TByte byte){  unsigned char B;  B=((byte.p0)<<7)  +((byte.p1)<<6) /\*Conversion\*/  **return** B;  }  */\*Funcion que imprime el TByte\*/*  void printTByte(char \*Tag,TByte B,int n){  **switch**(n){  **case** 4: printf("%s(%d%d%d%d)**\n**",Tag,B.p4,B.p5,B.p6,B.p7);  **break**;  **case** 8:  printf("%s(%d%d%d%d%d%d%d%d)",Tag,B.p0,B.p1,B.p2,B.p3,B.p4,B.p5,B.p6,B.p7);  #ifdef CHARSHOW  printf(" Int:(%d) Hex:(%2x) Char:(%c)",BtoUS(B),BtoUS(B),BtoUS(B));  #endif  putchar('\n');  **break**;  **case** 10:  printf("%s(%d%d%d%d%d%d%d%d%d%d)**\n**",Tag,B.p0,B.p1,B.p2,B.p3,B.p4,B.p5,B.p6,B.p7,B.p8,B.p9);  **break**;  } | **typedef** **struct**{  int p0;  int p1;  int p2;  int p3;  int p4;  int p5;  int p6;  int p7;  int p8;  int p9;  }TByte;  TByte INTtoB(int b);  int BtoINT(TByte byte);  TByte UStoB(unsigned char b);  unsigned char BtoUS(TByte byte);  void printTByte(char \*Tag,TByte B,int n); |

1. **definition.h**

For this file we include definitions that help us to compile the program in a differents modes

DEBUG: Show the differents parts in the SDES cipher

SBOXSHOW: Show the numbers in the SBOX

SBOXINDEX: Show the indexes obtained from the original string

CHARSHOW: To print a data in 8 bits show all the values in format integer, hexa and char

|  |  |
| --- | --- |
| 1. #define PORTADA 2. #define CHARSHOW 3. #define SHOWOPTIONS 4. #define DEBUG 5. #ifdef DEBUG 6. #define SBOXSHOW 7. #ifdef SBOXSHOW 8. #define SBOXINDEX 9. #define SHOWKSCH 10. #endif 11. #endif | To active each one you have to no comment the definition in the file, the opposite to quit the definition.  The definitions inside the others are deactivated with the part up. |

1. **information.c / information.h**

For these files we only put the presentation for the app and also the information about the developers.

|  |  |
| --- | --- |
| void usage(){  printf("USAGE: ./%s <INPUTFILE> <OUTPUTFILE> <10BIT BINKEY> <OPTIONS>**\n**", "MyPGP");  printf("**\t**<10BIT BINKEY> IE. 1000111101**\n**");  printf("**\t**<OPTIONS>**\n**");  printf("**\t** -CTR <CTR:NONCE> IE. CTR:Z**\n**");  printf("**\t** -CBC <CBC:IV:MODE> IE. CBC:F:E**\n**");  printf("**\t** -OFB <OFB:IV:MODE> IE. CTR:Z:e**\n**");  printf("**\t** -CFB <CFB:IV:MODE> IE. CFB:Q:2**\n**");  printf("**\t** -ECB <ECB:0:MODE> IE. ECB:0:D (NOT RECOMENDED)**\n**");  printf("**\t\t** <MODE OPTIONS>**\n\t\t** FOR ENCRIPTION:[1|E|e]**\n\t\t** FOR DECRYPTION:[2|D|d]**\n**");  printf("**\t\t** [IV|NONCE] PARAMETERS SHOULD BE (CHAR)**\n**");  }  void portada(){  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\n**");  printf("\* \***\n**");  printf("\* MMM MMM PPPPPPPPP GGGGGGGGGG PPPPPPPPP \***\n**");  printf("\* M MM MM M PPPPPPPPPP GGGGGGGGGGGG PPPPPPPPPP \***\n**");  printf("\* M MMMMMM M P P PPP G G P P PPP \***\n**");  printf("\* M M MM M M YY YY P P PPP G G P P PPP \***\n**");  printf("\* M M M M YY YY P P PPP G G GGGGGGG P P PPP \***\n**");  printf("\* M M M M YY YY P PPPPPPP G G GGGGG G P PPPPPPP \***\n**");  printf("\* M M M M YY P PPPPPP G G G G P PPPPPP \***\n**");  printf("\* M M M M YY P P G G G G P P \***\n**");  printf("\* M M M M YY P P GGGGGGGGGG G P P \***\n**");  printf("\* MMM MMM YY PPP (PRETTY) GGGGGGGGGG (GOOD) PPP (PROBABILITIES) \***\n**");  printf("\* \***\n**");  printf("\* (to fail at cryptography) \***\n**");  printf("\* \***\n**");  printf("\* BY: \***\n**");  printf("\* DAVID DURAN \***\n**");  printf("\* ALEJANDRO LOPEZ \***\n**");  printf("\* ANGEL LOPEZ \***\n**");  printf("\* JULY 2015 \***\n**");  printf("\* \***\n**");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRESS\*\*\*ANY\*\*\*KEY\*\*\*TO\*\*\*CONTINUE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\n**");  getchar();  system("clear");  } |  |

1. **llave.c / llave.h**

**validaLlave:**

**P10:**

**P8:**

**Keyschedule**

|  |  |
| --- | --- |
| #include "llave.h"  */\*Valida que la llave sea de 10 caracteres y que este en el rango permitido\*/*  int validaLlave(**const** char \*llave)  {  int K10=0,i=0;  **while**(llave[i]!='\0'){  **if**( llave[i]=='0' || llave[i]== '1')  K10 = (K10 << 1) + ((llave[i])-'0');  **else**  K10 = -1;  i++;  }    **return** K10;  }  */\*Permutacion 8bits LLAVE\*/*  int P10(int llave)  {  TByte a,b=INTtoB(llave);    #ifdef SHOWKSCH  printTByte("**\t**KEY:",b,10);  #endif  */\*Permutacion a 10\*/*  a.p0=b.p2;  a.p1=b.p4;  a.p2=b.p1;  a.p3=b.p6;  a.p4=b.p3;  a.p5=b.p9;  a.p6=b.p0;  a.p7=b.p8;  a.p8=b.p7;  a.p9=b.p5;  #ifdef SHOWKSCH  printTByte("**\t**P10:",a,10);  #endif  **return** BtoINT(a);  }  int P8(int llave)  {  TByte a,b=INTtoB(llave);  #ifdef SHOWKSCH  printTByte("**\t\t**RECIEVEDP8:",b,10);  #endif  */\*Permutacion\*/*  a.p2=b.p5;  a.p3=b.p2;  a.p4=b.p6;  a.p5=b.p3;  a.p6=b.p7;  a.p7=b.p4;  a.p8=b.p9;  a.p9=b.p8;  #ifdef SHOWKSCH  printTByte("**\t\t**PERMUTEDP8:",a,10);  #endif    **return** BtoINT(a);  }  Key keyschedule(int key)  {  Key k;  #ifdef SHOWKSCH  printf("KEY SCHEDULE**\n**");  #endif  */\*Introducimos la llave por unica vez a una permutacion P10\*/*  k.k1=P10(key);  */\*LS-1 aplicado a mascaras para realizar corrimientos y sumas sin riesgo a un resultado no deseado\*/*  k.k1= ((k.k1&0x1E0)<<1 | (k.k1&0x200)>>4) | ( (k.k1&0xF)<<1 | (k.k1&0x10)>>4 );  */\*LS-2 aplicado a mascaras para realizar corrimientos y sumas sin riesgo a un resultado no deseado\*/*  k.k2=((k.k1&0xE0)<<2 | (k.k1&0x300)>>3) | ((k.k1&0x7)<<2 | (k.k1&0x18)>>3);  #ifdef SHOWKSCH  printTByte("**\t**LS-1:",INTtoB(k.k1),10);  #endif  */\*Aplicamos una mascara a 8 bits a la P8 para garantizar los resultados en este rango\*/*  k.k1= P8(k.k1)&0xFF;  #ifdef SHOWKSCH  printTByte("**\t**LS-2:",INTtoB(k.k2),10);  #endif  */\*Aplicamos una mascara a 8 bits a la P8 para garantizar los resultados en este rango\*/*  k.k2= P8(k.k2)&0xFF;  #ifdef SHOWKSCH  printTByte("**\t**SUBKEY1:",UStoB((unsigned char)k.k1),8);  printTByte("**\t**SUBKEY2:",UStoB((unsigned char)k.k2),8);  putchar('\n');putchar('\n');  #endif  **return** k | CC=gcc  CFLAGS= -c -g3  PFLAGS= -o libraries/  SOURCES= MyPGP.o byte.o sdes.o llave.o  PSOURCES= MyPGP.o libraries/byte.o libraries/sdes.o libraries/llave.o  OBJECTS=**$(**SOURCES:.c=.o**)**  EXECUTABLE= mypgp  all: mypgp  mypgp: **$(**SOURCES**)**  **$(**CC**)** **$(**PSOURCES**)** -o **$(**EXECUTABLE**)**  MyPGP.o: MyPGP.c  **$(**CC**)** **$(**CFLAGS**)** MyPGP.c  byte.o: libraries/byte.c  **$(**CC**)** **$(**CFLAGS**)** libraries/byte.c **$(**PFLAGS**)**byte.o  llave.o: libraries/llave.c  **$(**CC**)** **$(**CFLAGS**)** libraries/llave.c **$(**PFLAGS**)**llave.o  OpMode.o: libraries/OpMode.c  **$(**CC**)** **$(**CFLAGS**)** libraries/OpMode.c **$(**PFLAGS**)**OpMode.o  sdes.o: libraries/sdes.c  **$(**CC**)** **$(**CFLAGS**)** libraries/sdes.c **$(**PFLAGS**)**sdes.o  clean:  rm libraries/\*o **$(**EXECUTABLE**)** |

1. **OpMode.c / OpMode.h**

**Functions:**

unsigned char CTR(unsigned char nonce, unsigned char counter, unsigned char m, Key K): For this function the program encrypt the plaintext with the CTR mode

unsigned char CBC(unsigned char iv, unsigned char m, unsigned char mode, Key K): For this function the program encrypt the plaintext with the CTR mode

unsigned char ECB(unsigned char m, unsigned char mode, Key K): For this function the program encrypt the plaintext with the CTR mode

unsigned char OFB(unsigned char iv, unsigned char mode, Key K): For this function the program encrypt the plaintext with the CTR mode

unsigned char CFB(unsigned char iv, unsigned char m, unsigned char mode, Key K): For this function the program encrypt the plaintext with the CTR mode

void CIPHERFILE(**const** char \*Source, **const** char \*Target, **const** char \*Mode ,Key key): For this function the program encrypt the plaintext with the CTR mode

|  |
| --- |
| #include "OpMode.h"  unsigned char CTR(unsigned char nonce, unsigned char counter, unsigned char m, Key K){  unsigned char noncounter= (nonce & 0xC0)|(counter & 0x3F);  **return** m ^ SDES(noncounter,K.k1,K.k2);  }  unsigned char CBC(unsigned char iv, unsigned char m, unsigned char mode, Key K){  **if**(mode=='E')  **return** SDES(m^iv,K.k1,K.k2);  **else**  **return** SDES(m,K.k2,K.k1)^iv;  }  unsigned char ECB(unsigned char m, unsigned char mode, Key K){  **if**(mode=='E')  **return** SDES(m,K.k1,K.k2);  **else**  **return** SDES(m,K.k2,K.k1);  }  unsigned char OFB(unsigned char iv, unsigned char mode, Key K){  **return** SDES(iv,K.k1,K.k2);  }  unsigned char CFB(unsigned char iv, unsigned char m, unsigned char mode, Key K){  **return** SDES(iv,K.k1,K.k2)^m;  }  void CIPHERFILE(**const** char \*Source, **const** char \*Target, **const** char \*Mode ,Key key){  FILE \*source, \*target;  char ch;  unsigned char c,i;  unsigned char NONCE=0,MODE=0,IV=0,OpMODE=0;  */\*Llenamos las variables de Opcion basados en el vector de opcion \*OPT \*/*  OpMODE= Mode[0] ^ Mode[1] ^ Mode[2];  **if**(OpMODE==69)  NONCE = Mode[4];  **else**  IV = Mode[4];  **if**( Mode[6]=='1' || Mode[6]=='E' || Mode[6]=='e')  MODE='E';  **else**  MODE='D';  */\*Mostramos las opciones previamente procesadas si se encuentra la directiva activada\*/*  #ifdef SHOWOPTIONS  printf("OpMODE: %c%c%c CODE:%d**\n**",Mode[0],Mode[1],Mode[2],OpMODE);  **if**(OpMODE==69)  printf("NONCE: %d**\n**", NONCE);  **else**  printf("IV: %d**\n**", IV);  printf("MODE: %c**\n**", MODE);  #endif  */\*Procesamos los archivos de entrada y Salida\*/*  source = fopen(Source, "r");    **if**( source == NULL ){  printf("PRESS ANY KEY TO CONTINUE...**\n**");  exit(EXIT\_FAILURE);  }    target = fopen(Target, "w");  **if**( target == NULL ){  fclose(source);  printf("PRESS ANY KEY TO CONTINUE...**\n**");  exit(EXIT\_FAILURE);  }  i=0;  **while**((ch = fgetc(source))!= EOF){  **if** (MODE=='E'){  */\* code \*/*  **if**(OpMODE==69)  c=CTR(NONCE,i,(unsigned char)ch,key);  **if** (OpMODE==68)  c=ECB((unsigned char)ch, MODE, key);  **if** (OpMODE==66){  c=CBC(IV,ch,MODE,key);  IV=c;  }  **if** (OpMODE==71){  IV=CFB(IV,(unsigned char)ch,MODE,key);  c=IV;  }  **if** (OpMODE==75){  IV=OFB(IV,MODE,key);  c=IV^(unsigned char)ch;  }  }**else** **if** (MODE=='D'){  */\* code \*/*  **if**(OpMODE==69)  c=CTR(NONCE,i,(unsigned char)ch,key);  **if** (OpMODE==68)  c=ECB((unsigned char)ch, MODE, key);  **if** (OpMODE==66){  c=CBC(IV,ch,MODE,key);  IV=ch;  }  **if** (OpMODE==71){  c=CFB(IV,(unsigned char)ch,MODE,key);  IV=ch;  }  **if** (OpMODE==75){  IV=OFB(IV,MODE,key);  c=IV^(unsigned char)ch;  }  }  *//printTByte( "RESULT:" ,UStoB(c),8);*  fprintf(target, "%c", (char)c );  }    (MODE=='E')?printf("FILE CYPHERED SUCCESSFULLY....**\n**"):printf("FILE DECYPHERED SUCCESSFULLY....**\n**");  fclose(source);  fclose(target);  } |

1. **sdes.h / sdes.c**

**Functions:**

**extern** **const** unsigned char sbox0[4][4]: Constant extern in other file definiton

**extern** **const** unsigned char sbox1[4][4]: Constant extern in other file definiton

unsigned char IP(unsigned char B): Operates the initial permutation only changing the bits order one by one

unsigned char EP(unsigned char B): Key expansion, takes a key of 4 bits and turn it into an 8 bits key

unsigned char invIP(unsigned char B): Operates the inverse of the initial permutation in the algorithm

unsigned char P4(unsigned char B): also the permutation but now only by 4 bits key

unsigned char fk(unsigned char IP,unsigned char k):p

unsigned char SW(unsigned char B): This function makes a “corrimiento” of 4 bits to simulate the switch operation

unsigned char SDES(unsigned char m, unsigned char k1 ,unsigned char k2): takes as the input the message, the key1 and the key2 ans delect by the definitions in the file ”definition.h” to show one or more parts in the encryption algorithm.

## RSA

For the RSA implementation we decided to use C as the main programming language. This programming language allows us to improve the execution time and the memory use so we can develop a better application in the algorithms for encryption and decryption.

The first part to develop the RSA algorithm we a set of functions that help us to organize the code and implement the algorithm in a better way. We explain them below.

1. **exp.h / exp.c**

|  |  |
| --- | --- |
| #include "exp.h"  int binex(double a){  int e=0,c=1;  **while**(c <= a){  e++;  c = pow(2,e);  }  **return** e-1;  }  int elevar(int a, int n){  int j;  double aux=1;  **for**(j=0;j<n;j++)  aux = aux\*a;  **return** aux;  }  double elevarA(double base, int exponente, int n){  int i;  double res=1;  **for**(i=0;i<exponente;i++){  res = res \* base;  res = fmod(res,n);  }  **return** res;  }  double Power(double a, double base , int n){  double res=1;  int k;  **do**{  k = binex(a);  a= a- elevar(2,k);  *//printf("exponente %d\n ", elevar(2,k) ) ;*  res = res \* elevarA( base, elevar(2,k), n);  *//printf("exponenciacion : %.2f\n", elevarA( base, elevar(2,k), n) );*  }**while**(a > 0);  **return** fmod(res,n);  } | **The function “binex”**  **The function “elevar”** |

1. **generator.c**

|  |
| --- |
| int main(int argc,char \*argv[]){  int c;  int f,x,y;  int p,q,n,phin,e;  int \*primos;  primos = (int \*) malloc( limite );  fromFile(primos , &c);  srand(time(NULL));  p = primos[rn(c)];  q = primos[rn(c)];  printf("p %d q :%d**\n**",p,q);  n = calcN(p,q);  printf("n : %d**\n**",n);  phin = phiN(p, q);  printf("phi (n) = %d**\n**",phin);  **do**{  e = rand();  e = e%phin;  printf(" e : %d**\n**",e);      printf("mcd () = %.0f**\n**",euclides(phin, e));  }**while**( (euclides(phin, e) != 1) || e <= 0 );  *//printf("valor de e : %d\n",e);*  *//e = 7;*  extEuclid((int)e, (int)phin, &x, &y, &f);  printf("x = %d y = %d f = %d**\n**", x, y, f);  printf("RSA public Key : ( %d , %d ) **\n**",e,n);  printf("RSA secret Key : ( %d , %d ) **\n**",mod(x,phin),n);  **return** 0;  } |

1. **rsa.c / rsa.h**

For these files we develop the next set of functions and we have their description below

void extEuclid(int a, int b, int \*x, int \*y, int \*d): This is the implementation of the extended Euclides algorithm

void Inverso(double a, double b, double \*x, double \*y, double \*d);

int ec(int a, int b);

double euclides(double a, double b);

int rn(int mod);

double calcN(int a,int b);

int phiN(int a, int b);

int \* calcularPrimos(unsigned int limite,int \*c): It calculates a cuple

void toFile(int c,int \*primos);

void fromFile(int \*primos,int \*c);

void usageCrypt(void);

int mod(int k, int n);

1. rsacrypt.c
2. 5
3. 6
4. 7