Xifratge d'arxius

Francisco Miquel San Lorenzo
ETIS

Consultor: Antoni Martinez Ballesté

10/01/2006

Dedicatòria

A Pia, la meva dona, i als
meus fills Francesc i Núria

Aquesta memòria descriu els fonaments teòrics i la funcionalitat d'una aplicació per a xifrar arxius i directoris utilitzant la norma PKCS#5 dels laboratoris RSA, a més d'una modificació de la norma (algorisme TripelDES) per a aconseguir xifres més fortes.

Índex de contingut

Índex de figures

1. Especificació del problema.

No hi ha dia que no apareguin als mitjans de comunicació noticies sobre vulnerabilitats dels sistemes operatius, virus, troians... Fins a quin punt està segura la informació que tenim al nostre ordinador? Amb el desplegament de la xarxa Internet, a més dels innumerables avantatges que ens proporciona, té el inconvenient de minvar la seguretat i la privacitat de les nostres dades.

Per a solucionar, o al menys minimitzar aquest problema hem d'utilitzar tècniques criptogràfiques en les nostres comunicacions, i en el emmagatzemament de les nostres informacions critiques. Aquest treball va encaminat a resoldre el problema del emmagatzemament de dades sensibles, i ho resoldrem creant una aplicació per a xifrar i desxifrar fitxers i directoris del nostre ordinador, utilitzant tècniques de xifratge amb clau compartida.

2. Fonaments.

La part criptogràfica de l'aplicació es divideix en dos parts, la primera consisteix a obtenir una clau a partir d'un text introduït per l'usuari, i la segona a xifrar les dades utilitzant la clau calculada.

Dins de la criptografia moderna ens trobem en una gran quantitat de possibilitats per a implementar l'aplicació, però per a no perdre'ns i fel l'aplicació el més compatible possible anem a seguir la norma PKCS#5 versió 1.5 dels laboratoris RSA.

La norma PKCS#5 Descriu un mètode per a xifrar una cadena de text amb una clau secreta derivada d'una frase de pas. Fa servir MD2 o MD5 per a produir una clau a partir d'una frase de pas. Aquesta clau s'utilitza per a xifrar amb DES (en mode CBC) el missatge en qüestió

2.1.Algorisme de resum de missatge MD5

Una funció hash és una funció que fa correspondre a un missatge m de mida variable una representació H(m) de mida fixa.

MD5 (acrònim de *Message-Digest Algorithm 5*) és un algorisme de reducció criptogràfic de 128 bits molt utilitzat en una gran varietat d'aplicacions de seguretat, i per a comprovar la integritat d'arxius.

2.1.1.Història.

MD5 és un dels algorismes de reducció criptogràfica dissenyats pel professor Ronald Rivest del MIT (*Massachussets Institute of Technology*). Després de que un anàlisis analític li va indicar que el MD4 era insegur, es va decidir a programar l'MD5 per a substituir-lo en l'any 1991.

En 1996 Hans Dobbertin va anunciar una col·lisió, la qual cosa, encara que no era un atac contra la funció de hash del MD5, va fer que molts criptògrafs començaren a recomanar el canvi cap a altres algorismes.

2.1.2.Algorisme

MD5 processa un missatge de longitud variable produint una sortida de longitud constant de 128 bits.

El missatge original es completa fins que la seva longitud sigui múltiple de 512 (encara que ja ho sigui). Per a completar el missatge s'afegeix un únic bit a 1, i després s'afegeixen 0 fins que la longitud sigui 64 bits menor que el següent múltiple de 512. Finalment se completa amb la longitud original del missatge codificada com un enter de 64 bits.

L'algorisme principal de l'MD5 utilitza un registre d'estat de 128 bits dividit en quatre paraules de 32 bits anomenades A, B, C i D, inicialitzades amb uns valors constants (A: 01234567, B: 89abcdef, C: fedcba98, D: 76543210

El missatge d'entrada es divideix en blocs de 512 bits, i cadascun dels blocs, consecutivament, s'utilitzen com a entrada de l'algorisme, modificant el contingut del registre d'estat.

El procés de cada bloc del missatge consisteix en quatre etapes similars, anomenades rondes. Cada ronda es composa de 16 operacions basades en una funció no linear F, diverses sumes(⊞), una rotació de bits a l'esquerra ($<<<_s$), tal i com s'indica a la figura 1.

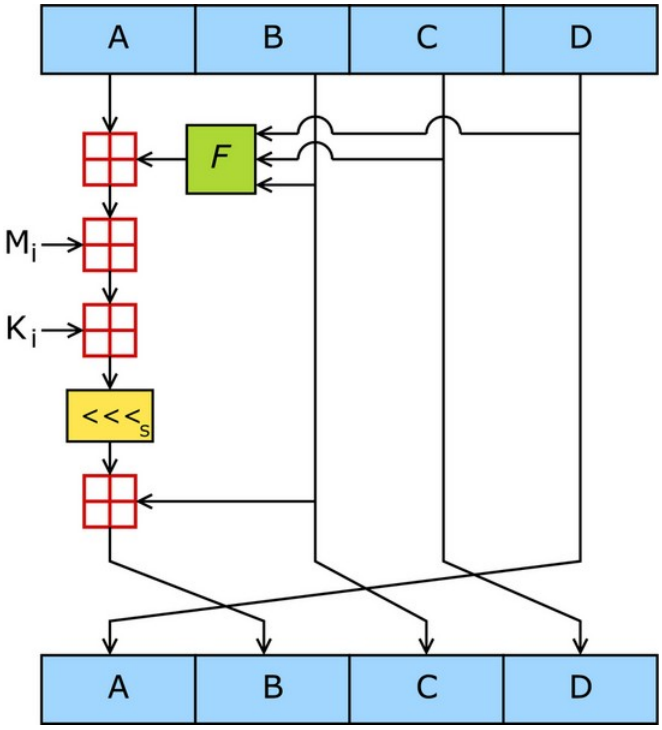


Figura 1: Operació bàsica MD5

Hi ha quatre funcions F diferents, una per a cada ronda:

$F(X,Y,Z)=(X\wedge Y)\vee(\neg X\wedge Z)$
$G(X,Y,Z)=(X\wedge Z)\vee(Y\wedge\neg Z)$
$H(X,Y,Z)=X\oplus Y\oplus Z$
$I(X,Y,Z)=Y\oplus(X\wedge\neg Z)$

M_i és cadascuna de les 16 paraules de 32 bits en que dividim cada bloc de 512 bits del missatge. Utilitza cadascuna de les 16 paraules en cadascuna de les 16 operacions en cada ronda.

K_i és cadascuna de les posicions d'una taula de 64 elements construïda amb la funció sinus de la següent manera: K[i] = floor(abs(sin(i)) × 2^32). Utilitza un element diferent en cadascuna de les 64 operacions de cada grup de 4 rondes.

$<<<_s$ és un desplaçament de s bits a l'esquerra, on el valor de s varia en cada operació, i es un valor prefixat de la següent manera:

Primera ronda (7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22)
Segona ronda (5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,)
Tercera ronda (4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23)
Quarta ronda (6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21)

La sortida del algorisme és el valor del registre d'estat després de processar tots els blocs de 512 bits del missatge.

2.2.Mode de xifratge de bloc CBC

Els xifratges de bloc són útils per a xifrar informacions curtes, com ara identificadors, contrasenyes, claus..., o sigui, aquells missatges en que la llargada del text a xifrar no sobrepassa la llargada d'un bloc.

Per una altra banda, la utilització del xifratge de bloc mitjançant la xifra de cada bloc de manera separada (mode ECB) sobre missatges llargs i especialment en texts que repeteixen determinats patrons, es desaconsella totalment. De tots el possibles modes de xifratge de bloc, la norma PKCS#5 especifica quin utilitzar, el CBC (Cipher Block Chaining).

En el mode CBC, abans de xifrar cada bloc, s'aplica una funció XOR entre el bloc de text pla i el bloc previ ja xifrat. D'aquesta manera, es crea una dependència entre cada bloc xifrat i tots els blocs precedents. Per a xifrar el primer bloc, necessitem un bloc inicial aleatori, el qual no cal que sigui secret, d'aquesta manera podem obtenir xifres diferents per al mateix text i la mateixa clau, només canviat el valor del vector.

2.2.1.Xifrat

La forma de xifrar en CBC és la indicada a la figura següent:

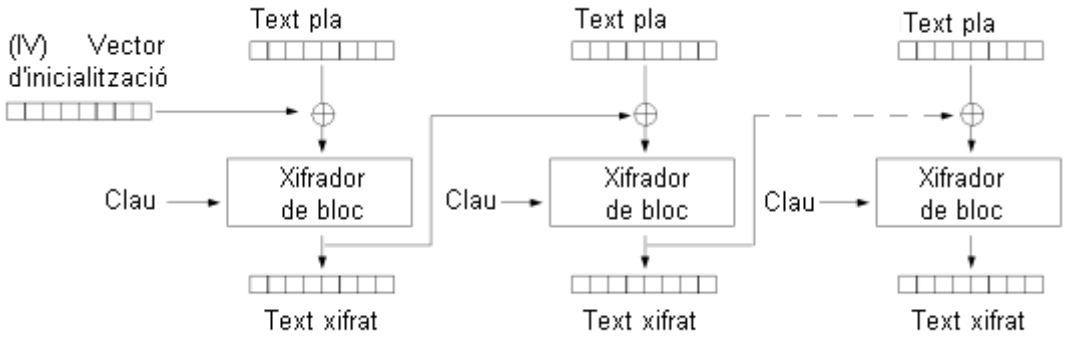


Figura 2: CBC en mode xifrat

Per a xifrar un bloc utilitzarem la següent formula:

$C_i = E_k(M_i \oplus C_{i-1})$
$C_0 = IV$

on K és la clau, E una funció de xifratge i $M_1,...M_n$ son els blocs de text en clar a xifrar.

2.2.2.Desxifrat

La forma de xifrar en CBC és la indicada a la figura següent:

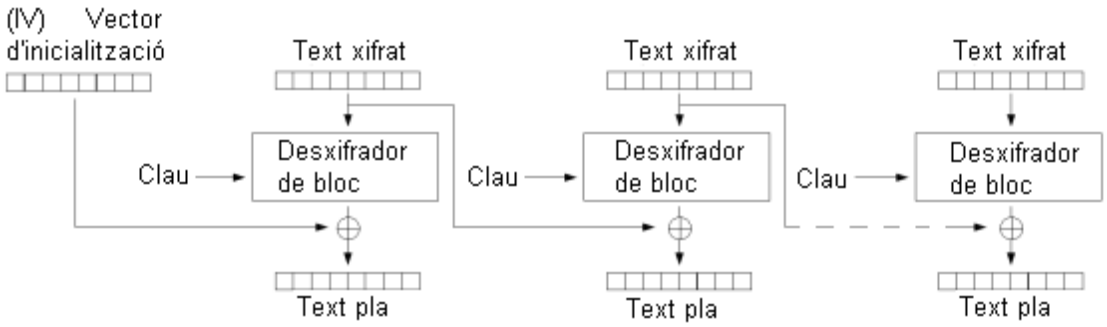


Figura 3: CBC en mode desxifrat

Per a desxifrar un bloc utilitzarem la següent formula:

$M_i = D_k(C_i) \oplus C_{i-1}$
$C_0 = IV$

on K és la clau, D una funció de desxifratge i $C_1,...C_n$ son els blocs de text xifrat.

2.3.Algorismes de xifratge i desxifratge DES i TripleDES.

DES (Data Encryption Standard) és un algorisme de xifrat triat com a Estàndard de Processament d'Informació Federal als Estats Units en 1976, i l'ús del qual s'ha propagat per tot el món. L'algorisme va ser controvertit al principi, amb alguns elements de disseny classificats, una longitud de clau relativament curta, i les contínues sospites sobre l'existència d'alguna porta del darrera per a la NSA (National Security Agency). Posteriorment DES va ser sotmès a un intens anàlisi acadèmic i va motivar el concepte modern del xifrat per blocs i el seu criptoanálisis.

Avui en dia, DES es considera insegur per a moltes aplicacions, degut principalment a que la grandària de clau de 56 bits és curta (les claus de DES s'han trencat en menys de 24 hores). Existeixen també resultats analítics que demostren debilitats teòriques en el seu xifrat, encara que són pràcticament inviables. En la

pràctica, es creu que l'algoritme és segur en la seva variant de Triple DES, encara que existesquin atacs teòrics.

2.3.1.La Història de DES

Els orígens de DES es remunten a principis dels 70. En 1972, després d'acabar un estudi sobre les necessitats del govern en matèria de seguretat informàtica, l'autoritat d'estàndards nord-americamericana NBS (National Bureau of Standards) va concloure en la necessitat d'un estàndard a nivell governamental per a xifrar informació confidencial. En conseqüència, el 15 de maig de 1973, després de consultar amb la NSA, el NBS va sol·licitar, sense èxit, propostes per a un algorisme que complís rigorosos criteris de disseny. Una segona petició va ser realitzada el 27 d'agost de 1974, i en aquella ocasió, IBM va presentar un candidat que va ser considerat acceptable, un algorisme desenvolupat durant el període 1973–1974 basat en un altre anterior, l'algoritme Llucifer d'Horst Feistel.

2.3.2.El paper de la NSA en el disseny

El 17 de març de 1975, la proposta de DES va ser publicada en el Registre Federal. Es van sol·licitar comentaris per part del públic, i l'any següent es van obrir dos tallers lliures per a discutir l'estàndard proposat. Va haver-hi algunes crítiques des de certs sectors, incloent als pioners de la criptografia simètrica Martin Hellman i Whitfield Diffie, mencionant la curta longitud de la clau i les misterioses S-caixes com una evidència de la inadequada interferència de la NSA. La sospita era que l'algoritme havia estat debilitat de manera secreta per l'agència d'intel·ligència de manera que, poguessin llegir missatges xifrats fàcilment. Aquestes sospites van ser desmentides per el Comitè d'intel·ligència del Senat dels Estats Units i per alguns dels desenvolupadors de l'algorisme.

2.3.3.L'algorisme com a estàndard

A pesar de la polèmica, DES va ser aprovat com a estàndard federal al novembre de 1976, i publicat el 15 de gener de 1977.i autoritzat per a l'ús no classificat de dades. Posteriorment, va ser confirmat com a estàndard al any 1983, i revisat als anys 1988, 1993, i 1998, en aquesta última revisió es va definir el TripleDES.

El 26 de maig del 2002, DES va ser finalment reemplaçat per AES (Advanced Encryption Estàndard), després d'una competició pública. Avui en

dia, DES continua sent àmpliament utilitzat.

2.3.4.Algorismes de reemplaçament (TripeDES)

Molts dels anteriors usuaris de DES ara utilitzen Triple DES (3DES) que va ser descrit i analitzat en una de les patents de DES; consisteix en l'aplicació de DES tres vegades consecutives amb diferents claus en cadascuna. 3DES ha estat àmpliament reconegut com a segur per ara, encara que és prou lent.

2.3.5.Descripció de l'algorisme

DES és l'algorisme prototip del xifrat de bloc, un algorisme que pren un text en clar d'una longitud fixa (64 bits) i el transforma mitjançant una sèrie de complicades operacions en un altre text xifrat de la mateixa mida. DES utilitza una clau criptogràfica per a modificar la transformació, de manera que el desxifrat només pot ser realitzat per aquells que coneguin la clau concreta utilitzada en el xifrat. La mida de la clau és de 64 bits, encara que en realitat, només se'n utilitzen 56, els vuit bits restants s'utilitzen únicament per a comprovar la paritat, i després són descartats. Per tant, la longitud de clau efectiva en DES és de 56 bits, i així és com se sol especificar.

2.3.5.1.Estructura bàsica

L'estructura bàsica de l'algorisme apareix representada en la figura 4. Hi ha 16 fases idèntiques de procés, denominades rondes i una permutació inicial i final denominades PI i PF, que són funcions inverses entre si (PI "desfà" l'acció de PF, i a l'inrevés). PI i PF no són criptogràficament significatives, però es van incloure presumptament per a facilitar la càrrega i descàrrega de blocs sobre el

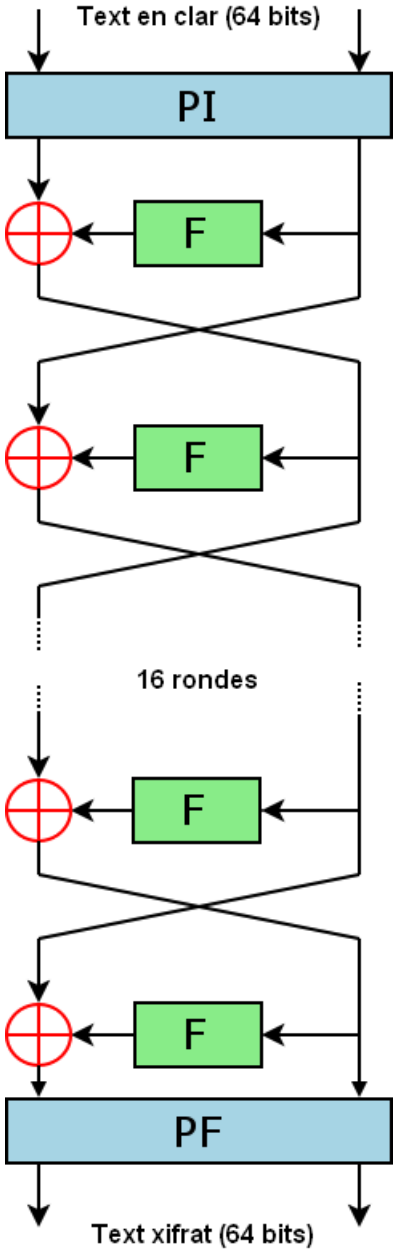


Figura 4: Estructura bàsica DES

maquinari dels anys 70. Abans de cada ronda, el bloc és dividit en dos meitats de 32 bits i processades alternativament. Aquest entrecreuament es coneix com a esquema Feistel.

L'estructura de Feistel assegura que el xifrat i el desxifrat siguin processos molt semblants, l'única diferència és que les subclaus s'apliquen en orde invers quan desxifrem. La resta de l'algorisme és idèntic, i per tant es simplifica enormement la implementació, en especial sobre maquinari, al no haver-hi necessitat d'algorismes distints per al xifrat i el desxifrat.

El símbol roig ⊕ representa l'operació OR exclusiu (XOR). La funció-F mescla la meitat del bloc amb part de la clau. La sortida de la funció-F es combina llavors amb l'altra meitat del bloc, i els blocs són intercanviats abans de la següent ronda. Després de l'última ronda, les meitats no s'intercanvien; aquesta és una característica de l'estructura de Feistel que fa que el xifrat i el desxifrat siguin processos semblants.

2.3.5.2.La funció (F) de Feistel

La funció-F, representada en la Figura 5, opera sobre mig bloc (32 bits) cada vegada i consta de quatre passos:

1. Expansió: La meitat del bloc de 32 bits s'expandeix a 48 bits mitjançant la permutació d'expansió, anomenada E en el diagrama, duplicant alguns dels bits.

2. Mescla: El resultat es combina amb una subclau utilitzant una

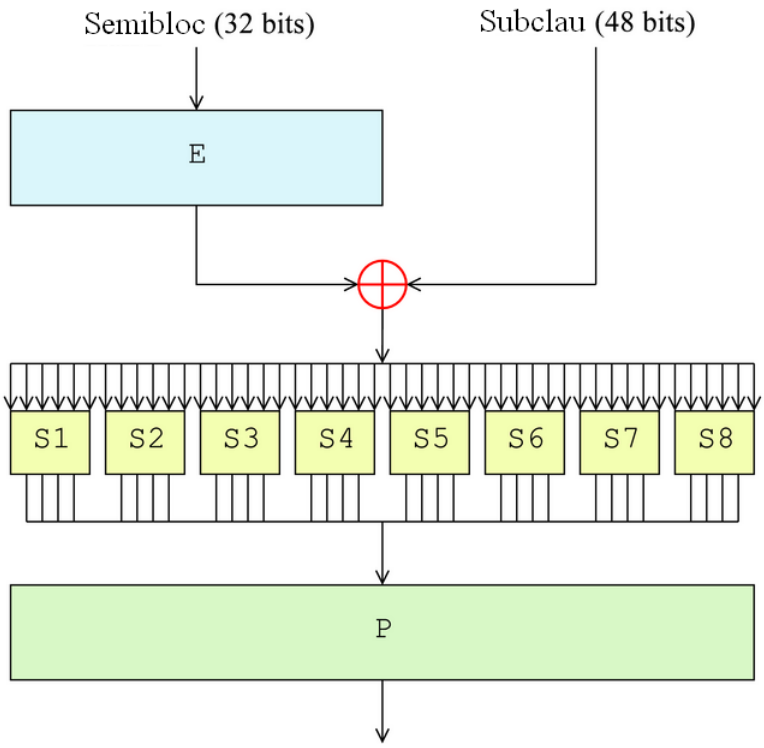


Figura 5: Funció de Feistel

operació XOR. Setze subclaus, una per a cada ronda, es deriven de la clau inicial mitjançant la generació de subclaus descrita més endavant.

3. Substitució: Després de mesclar-lo amb la subclau, el bloc és dividit en vuit trossos de 6 bits abans de ser processats per les S-caixes, o caixes de substitució. Cadascuna de les vuit S-caixes reemplaça els seus sis bits d'entrada amb quatre bits d'eixida, d'acord amb una transformació no lineal, especificada per una taula. Les S-caixes constitueixen el nucli de la seguretat de DES.

4. Permutació: Finalment, les 32 sortides de les S-caixes es reordenen d'acord amb una permutació fixa; la P-caixa.

Alternant la substitució de les S-caixes, i la permutació de bits de la P-caixa i l'expansió-E proporcionen confusió i difusió.

2.3.5.3.Generació de claus

La Figura 6 representa la generació de claus per al xifrat. Primer, se seleccionen 56 bits de la clau dels 64 inicials per mitjà de l'elecció permutada 1 (PC1) Els 56 bits es divideixen llavors en dos meitats de 28 bits; a continuació cada meitat es tracta independentment. En rondes successives, ambdós meitats es desplacen cap a l'esquerra un o dos bits (depenent de cada ronda), i llavors se seleccionen 48 bits de subclau per mitjà de l'elecció permutada 2 (PC2), 24 bits de la meitat esquerra i 24 de la dreta. Els desplaçaments (indicats per "<<<" en el diagrama) impliquen que s'utilitza un conjunt diferent de bits en cada subclau; cada bit

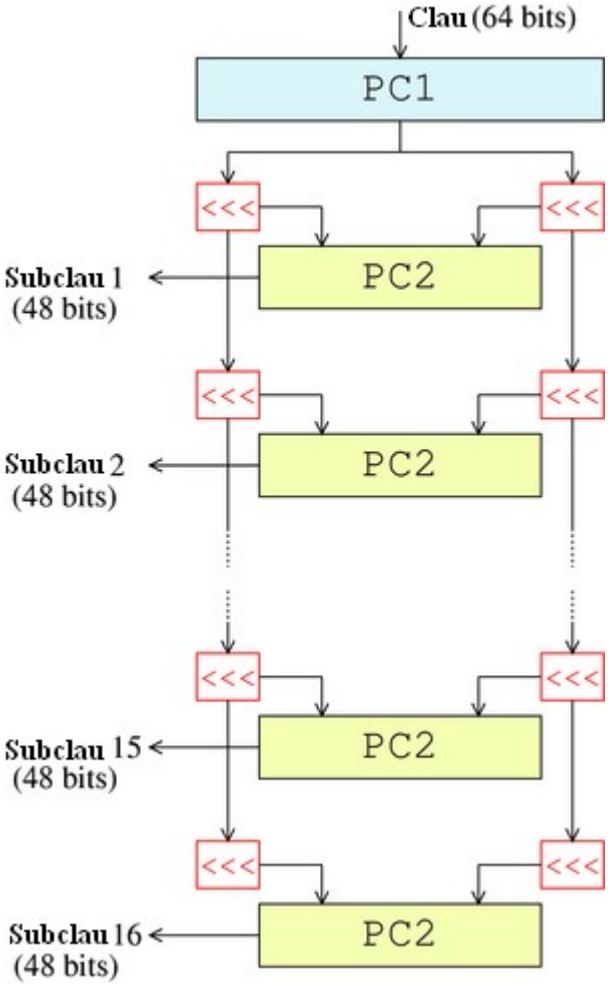


Figura 6: Generador de claus DES

s'utilitza aproximadament en 14 de les 16 subclaus.

La generació de claus per a desxifrat és semblant, s'han de generar les claus en orde invers. Per tant els desplaçaments es fan cap a la dreta, en compte de cap a l'esquerra.

2.3.6.Seguretat

Encara que s'ha publicat més informació sobre el criptoanálisis de DES que de cap altre xifrat de bloc, l'atac més pràctic a hores d'ara continua sent per força bruta. Es coneixen diverses propietats criptoanalítiques menors, i són possibles tres tipus d'atacs teòrics que, encara requerint una complexitat teòrica menor que un atac per força bruta, requereixen una quantitat irreal de texts plans coneguts o triats per a dur-se a terme, i no es tenen en compte en la pràctica.

2.3.6.1.Atac per força bruta.

Per a qualsevol tipus de xifrat, el mètode d'atac mes simple és l'atac per força bruta. La longitud de clau determina el nombre possible de claus, i per tant la factibilitat de l'atac. En el cas de DES, ja en els seus començaments es van plantejar qüestions sobre la seva longitud de clau, inclòs abans de ser adoptat com a estàndard, i va ser la seva reduïda grandària de clau, més que el criptoanálisis teòric, el que va provocar la necessitat reemplaçar-lo.

La vulnerabilitat de DES va ser demostrada en la pràctica en 1998 quan l'Electronic Frontier Foundation, un grup dedicat als drets civils en el ciberespai, va construir una màquina a mida per a trencar DES, amb un cost aproximat de 250.000 dòlars. Aquesta màquina va trencar una clau per força bruta en una busca que va durar poc més de 2 dies.

2.3.6.2.Atacs més ràpids que la força bruta

Hi ha tres atacs coneguts que poden trencar les setze rondes completes de DES amb menys complexitat que un atac per força bruta: el criptoanálisis diferencial, el criptoanálisis lineal i l'atac de Davies. De totes les maneres, aquests atacs són només teòrics i no és possible portar-los a la pràctica.

3. Mètode implementat.

El mètode utilitzat per a la implementació de l'aplicació ha estat part de la norma PKCS#5, i a més una petita variant sobre aquesta, la utilització de l'algorisme TripleDES.

La versió de la norma PKCS#5 utilitzada ha estat la 1.5, encara que ja s'ha publicat la versió 2.0 i que es recomana la utilització d'aquesta última, però el llenguatge utilitzat (Java) té implementades classes que corresponen a la versió 1.5.

El estàndard PKCS#5 descriu un mètode per a xifrar una seqüència d'octets amb una clau secreta derivada d'una contrasenya. El resultat del mètode és una cadena d'octets. Encara que aquest estàndard es pot utilitzar per a xifrar cadenes arbitraries d'octets, la seva aplicació principal, prevista en el àmbit de la criptografia de clau pública, és la de xifrar claus privades per a transferir-les d'un sistema informàtic a un altre segons es descriu en la norma PKCS#8.

Aquest estàndard defineix dos algorismes de xifrat “MD2 amb DES-CBC” i “MD5 amb DES-CBC”. Aquests algorismes utilitzen el xifrador de clau compartida DES, amb el mode de xifratge de bloc CBC, on la clau secreta es deriva d'una contrasenya mitjançant els algorismes de hashing MD2 o MD5.

3.1. Descripció general

Seleccionarem en privat una cadena d'octets *P* per a utilitzar-la com a “contrasenya”. La contrasenya és una cadena arbitraria d'octets, i no cal que sigui el que normalment s'entén com una paraula imprimible. La cadena pot tenir una mida de 0 o més octets. També seleccionarem una seqüencia *S* de vuit octets anomenada salt, i un enter positiu *c* el qual utilitzarem com a comptador d'iteracions. Cada missatge xifrat pot tenir valors diferents de contrasenya, salt i contador d'iteracions.

Els processos de xifrat i desxifrat utilitzen, tots dos, la contrasenya, el valor de salt i el comptador d'iteracions com a clau de l'algorisme de xifrat. Ambdós processos transformen una cadena d'octets en un altra cadena d'octets, i son inversos l'un de l'altre si utilitzen la mateixa clau.

Notes:

a) Hem de tenir en compte algunes condicions de seguretat al elegir la

contrasenya, a fi de dificultar atacs estàndard com per exemple atacs de diccionari.

b) Es recomana, en interès de la interoperabilitat, que quan els missatges xifrats segons aquest estàndard hagin de ser transferits d'un sistema informàtic a un altre, la contrasenya estigui constituïda per caràcters ASCII imprimibles (valors del 32 al 126 en decimals inclosos).

c) El comptador d'iteracions proporciona un mètode per a incrementar el temps necessari per a obtenir una clau DES a partir de la contrasenya, d'aquesta manera es fan més difícils els atacs de diccionari.

d) La selecció d'un valor de salt independent de la contrasenya limita la eficàcia de atacs de diccionari dirigits col·lectivament contra molts missatges xifrats segons aquest estàndard, per exemple, contra una base de dades de claus privades xifrades.

e) Una manera convenient de seleccionar el valor de *S* és obtenir-lo dels vuit primers octets del hashing de la cadena resultant de concatenar la clau amb el missatge (P || M). Aquest mètode funciona molt bé quan el missatge *M* és arbitrari, (com per exemple una clau privada), ja que el valor de *S* no revela ninguna informació significativa sobre *M* o *P*. Aquest mètode no es recomanable si se sap que el missatge pertany a un grup de missatges petits (per exemple "si" o "no")

f) El comptador d'iteracions i el mètode per a seleccionar el valor de salt es poden estandarditzar en aplicacions particulars.

3.2.Procés de xifrat

El procés de xifrat consta de tres passos: generació de la clau DES, formatatge del bloc de xifratge, i el xifrat DES. L'entrada del procés serà una cadena de octets *M*, el missatge; una cadena de octets *P*, la contrasenya; una cadena de d'octets *S*, el valor de salt; i un enter *c*, el comptador d'iteracions. La sortida del procés serà una cadena d'octets *C*, el text xifrat.

3.2.1.Generació de la clau DES

La clau *K* i el vector d'inicialització *IV* que posteriorment utilitzarà DES es calculen a partir del la contrasenya *P*, el valor de salt *S* i un comptador de iteracions *c*. El procés de generació es divideix en tres fases:

a) Apliquem l'algorisme de hashing elegit (MD2 o MD5) a la cadena formada per la concatenació de la contrasenya i el salt (P || S), i al resultat li tornem a aplicar l'algorisme de hashig, i així successivament tantes vegades com ens indiqui el comptador *c*.

T_1 = Hash (P || S) ,
T_2 = Hash (T_1) ,

T_c = Hash (T_{c-1}) ,

b) El bit menys significatiu de cadascun dels vuit primers octets del resultat es canviaran per a donar al byte paritat imparella, segons els requisits del DES. Aquests vuit octets resultants seran la clau *K*.

c) Els últims vuit octets del resultat del pas a) serà el vector d'inicialització *IV*.

3.2.2.Formatatge del bloc de xifratge

La concatenació del missatge *M* i d'una cadena de farciment *PS* produiran una cadena d'octets EB a la qual anomenarem bloc de xifratge.

EB = M || PS

La cadena de farciment *PS* consistirà en 8-(||M|| mod 8) octets idèntics el valor dels quals serà 8-(||M|| mod 8) on ||M|| és la longitud del missatge M (aquesta operació fa que la longitud del bloc de xifratge sigui múltiple de vuit octets). En altres paraules, el bloc de xifrat *EB* satisfà una de les declaracions següents:

EB = M || “01” -- si ||M|| mod 8 = 7 ;
EB = M || “02 02” -- si ||M|| mod 8 = 6 ;
.
.
.
EB = M || “08 08 08 08 08 08 08 08” -- si ||M|| mod 8 = 0 .

El bloc de xifratge es pot analitzar inequívocament ja que tots els blocs de xifratge acaben amb una cadena de farciment i cap cadena de farciment no és un sufix d'un altra.

3.2.3.Xifrat DES

El bloc de xifratge es xifrarà utilitzant l'algorisme DES en el mode d'encadenament de bloc de xifra CBC, amb la clau K i el vector que inicialització IV. El resultat del xifrat serà una cadena d'octets C, el text xifrat (la longitud del text xifrat és múltiple de vuit octets)

3.3.Procés de desxifrat

El procés de desxifrat consta de tres passos: generació de la clau DES, desxifrat DES, i la separació del bloc de xifratge. L'entrada del procés serà una cadena d'octets *C*, el text xifrat; una cadena d'octets *P*, la contrasenya; una cadena d'octets *S*, el valor de salt; i un enter *c*, el comptador d'iteracions. La sortida del procés serà una cadena d'octets M, el missatge.

3.3.1.Generació de la clau DES

La clau *K* i el vector d'inicialització *IV* que posteriorment utilitzarà DES es calculen a partir del la contrasenya *P*, el valor de salt *S* i el comptador de iteracions *c*, tal i com s'ha descrit per al procés de xifrat.

3.3.2.Desxifrat DES

El text xifrat C es desxifrarà utilitzant l'algorisme DES en el mode d'encadenament de bloc de xifra CBC, amb la clau K i el vector que inicialització IV. El resultat del desxifrat serà una cadena d'octets EB, el bloc de xifratge.

3.3.3.Separació del bloc de xifratge

El bloc de xifratge EB es separarà en dues parts: una cadena d'octets M, el missatge; i una cadena d'octets PS, la cadena de farciment, seguint el procediment invers del procés de formatatge del bloc de xifratge vist anteriorment.

4. Aspectes concrets de la implementació.

L'aplicació esta implementada en JAVA i consta de dos classes, TFCXifrador i Xifrador. La primera crea l'entorn gràfic i inicia un dels dos modes de funcionament, i la segona s'encarrega de xifrar o desxifrar els arxius o directoris.

4.1. Plantejaments previs.

Prèviament a la implementació de l'aplicació hem tingut en compte una sèrie de problemes:

a) Nom del fitxer xifrat. El nom del fitxer xifrat tindrà l'extensió *.xif, i es deixarà al mateix directori on estava el fitxer original. Aquest conveni ens permet fer que l'aplicació seleccioni l'operació a dur a terme (xifrat o desxifrat) en funció de l'extensió del fitxer a tractar.

b) Xifratge de directoris. Per a xifrar directoris crearem un fitxer zip amb tots els fitxers del directori, i després el xifrarem. Per a homogeneïtzar els processos de xifrat i desxifrat, en el cas de un sol fitxer, també crearem un fitxer zip.

c) Comprovació de la clau per a desxifrar. La utilització de fitxers zip ens permet, en el procés de desxifrat, comprovar si la clau introduïda per l'usuari és vàlida o no, ja que si la clau no és vàlida no podrem llegir la capçalera del fitxer zip, evitant d'aquesta manera obtenir un fitxer “desxifrat” inintel·ligible.

d) Modes de funcionament. L'aplicació té dos modes de funcionament: Un mode independent, amb un selector de fitxers i que permet anar fent operacions de xifratge o desxifratge fins que sortim; i un altre mode sense selector de fitxers, per a integrar-lo en l'entorn gràfic del sistema operatiu i que sols permet un xifratge o desxifratge per execució.

4.2. Classe TFCXifrador.

Aquesta classe crea un panell per a la selecció de l'algorisme de xifrat o desxifrat i per a la introducció de la clau. Si el nombre de paràmetres introduïts a la aplicació és 0 (aplicació independent) crea un selector de fitxers, i li afegeix el panell d'introducció de dades. Si el nombre de paràmetres és 1, agafa aquest paràmetre com el nom del fitxer a xifrar o desxifrar i ens visualitza el panell d'introducció de dades.

4.3.Classe Xifrador

Aquesta classe inicialitza el valor de salt i del comptador d'iteracions, calcula la clau, crea i inicialitza el xifrador (chipher) i crida al procés de xifrat o de desxifrat segons l'extensió del fitxer a tractar.

4.3.1.Xifrar

El procés seguit per a xifrar és el següent:

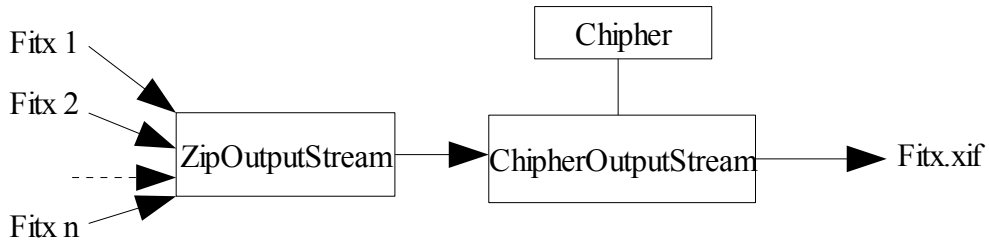


Figura 7: Procés de xifrat

4.3.2.Desxifrar

El procés seguit per a desxifrar és el següent:

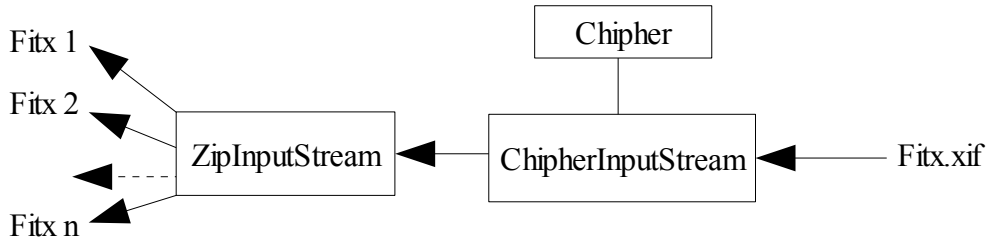


Figura 8: Procés de desxifrat

5. Manual d'instal·lació.

5.1. Requisits previs

Com que aquesta aplicació esta feta en JAVA, necessitarem tenir instal·lada alguna màquina virtual JAVA en la seva versió 1.4 o superior.

Per a poder utilitzar l'algorisme TripleDES, hem de tenir instal·lat el JCE (Java Cryptography Extension) en la seva versió ilimitada ("Unlimited Strength").

5.2. Instal·lació

Per a instal·lar l'aplicació sols hem de descomprimir el fitxer TFCXifrador.zip en qualsevol directori. Per a executar-la, ens situarem al directori creat i executarem la següent comanda:

```
java TFCXifrador
```

5.3. Integració en l'escriptori

5.3.1. Windows

Per a integrar l'aplicació en l'escriptori de windows hem de crear una sèrie de enllaços directes al fitxer Xifrador.bat que es troba al directori que s'ha creat quan hem instal·lat l'aplicació

5.3.1.1. Escriptori.

Premem el botó de la dreta del ratolí sobre l'escriptori i creem un accés directe al fitxer Xifrador.bat.

5.3.1.2. Menú d'inici.

Creem un accés directe al fitxer Xifrador.bat des de la carpeta:
Win 98
C:\Windows\Menu de inicio\Programas
Win XP, 2000, ...
C:\Documents and Settings\>NomUsuari>\Menú Inicio\Programas

5.3.1.3.Enviar a.

Per a que aparegui en el Menú secundari de l'explorador de windows, dins de l'opció “Enviar a”, un enllaç a l'aplicació crearem un accés directe al fitxer Xifrador.bat des de la carpeta:

Win 98

C:\Windows\Send to

Win XP, 2000, ...

C:\Documents and Settings\>NomUsuari>\Send to

5.3.2.Linux-KDE

5.3.2.1.Escriptori.

Premem el botó de la dreta del ratolí sobre l'escriptori i creem un enllaç a aplicació nou posant com a comandament el següent:

cd <directori d'instal·lació>;java TFCXifrador

5.3.2.2.Menú d'inici.

Premem el boto de la dreta del ratoli sobre el Menú d'inici i seleccionem l'editor de Menú. Afegim una aplicació posant com a comandament el següent:

cd <directori d'instal·lació>;java TFCXifrador

5.3.2.3.Acciones.

Per a que aparegui en el Menú secundari de l'explorador del kde (konqueror), dins de l'opció “Acciones”, un enllaç a l'aplicació crearem el fitxer xifrador.desktop dins del directori

~/.kde/share/apps/konqueror/servicemenus.

amb el contingut següent:

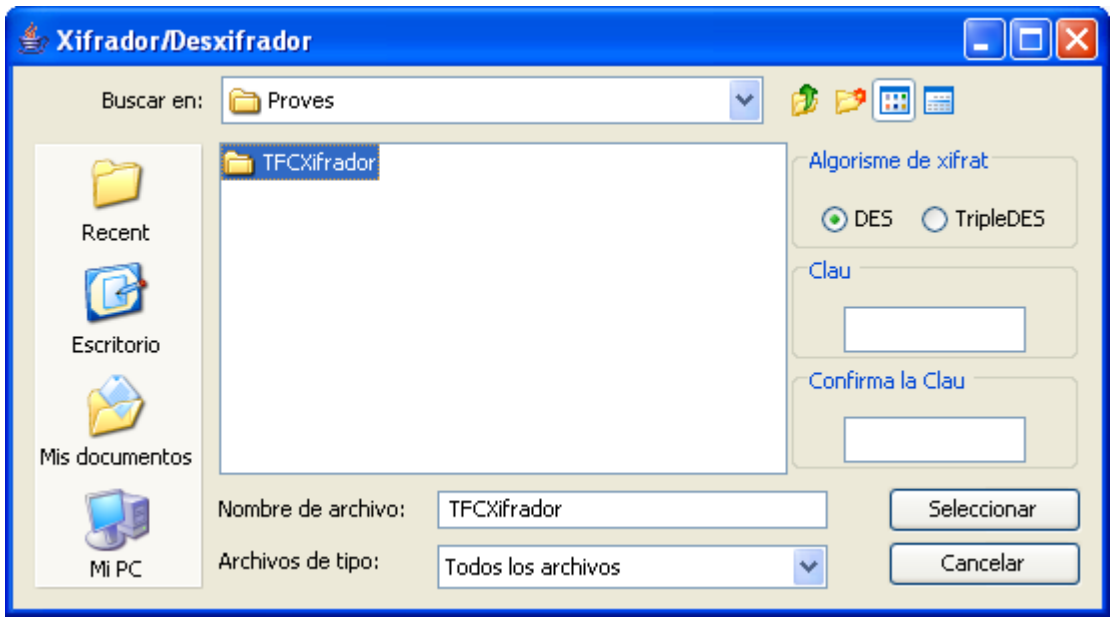
```
[Desktop Entry]
ServiceTypes=all/allfiles,inode/directory
Actions=Xifrador

[Desktop Action Xifrador]
Name[es]=Enviar a Xifrador
Exec=cd <directori d'instal·lació>;java TFCXifrador %f
```

6. Proves.

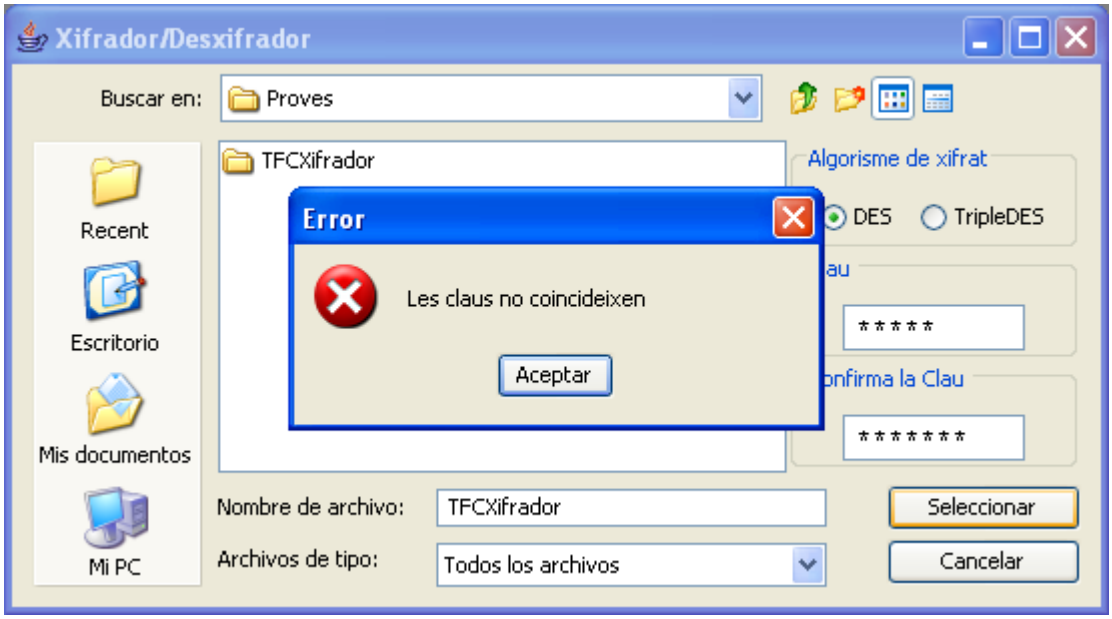
Per a provar el funcionament de l'aplicació, xifrarem una carpeta des del selector de fitxers i la desxifrarem des de l'aplicació integrada a l'explorador.

6.1. Xifrar una carpeta.

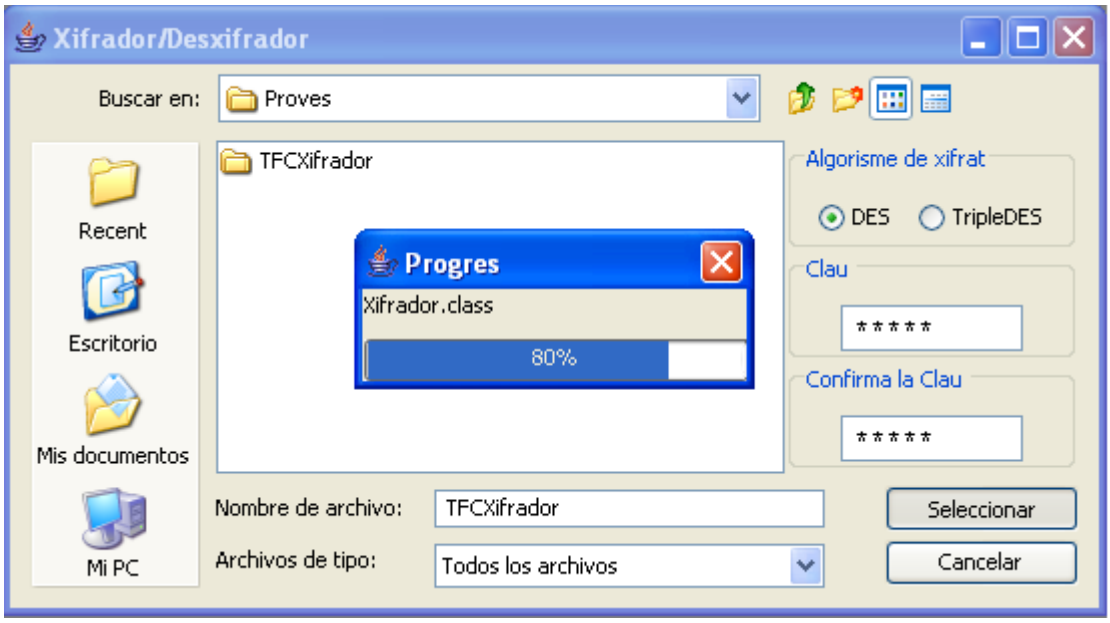
Executem l'aplicació i seleccionem una carpeta.



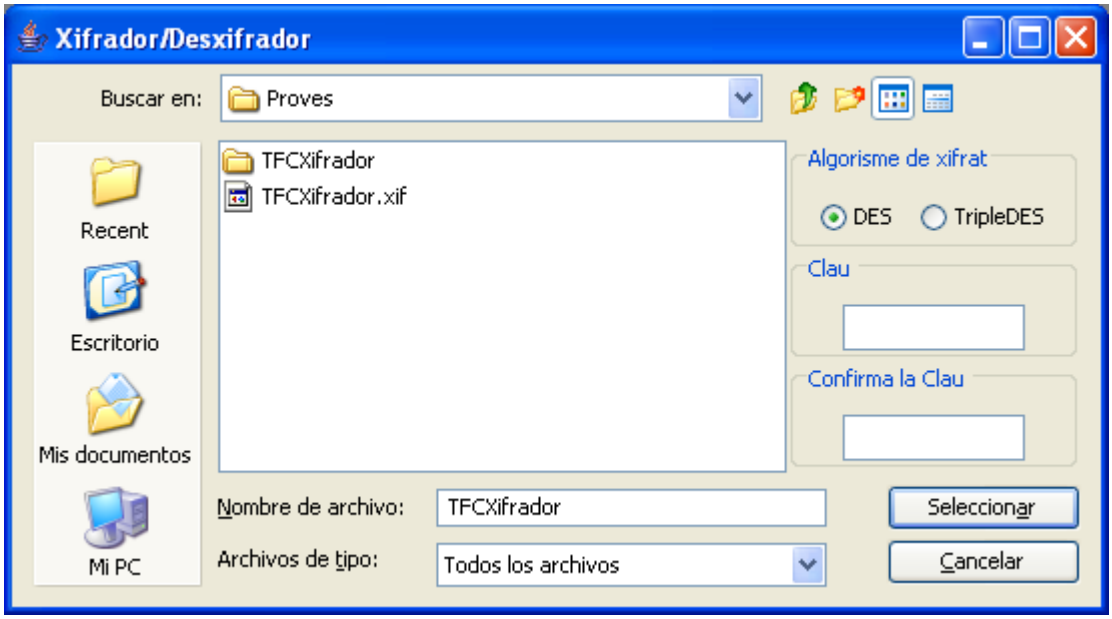
Seleccionem l'algorisme, introduïm la clau i la confirmació de la clau, i premem el boto “Seleccionar”. Si les claus no coincideixen ens dona un missatge de error.



Si les claus coincideixen comença el procés de xifrat:

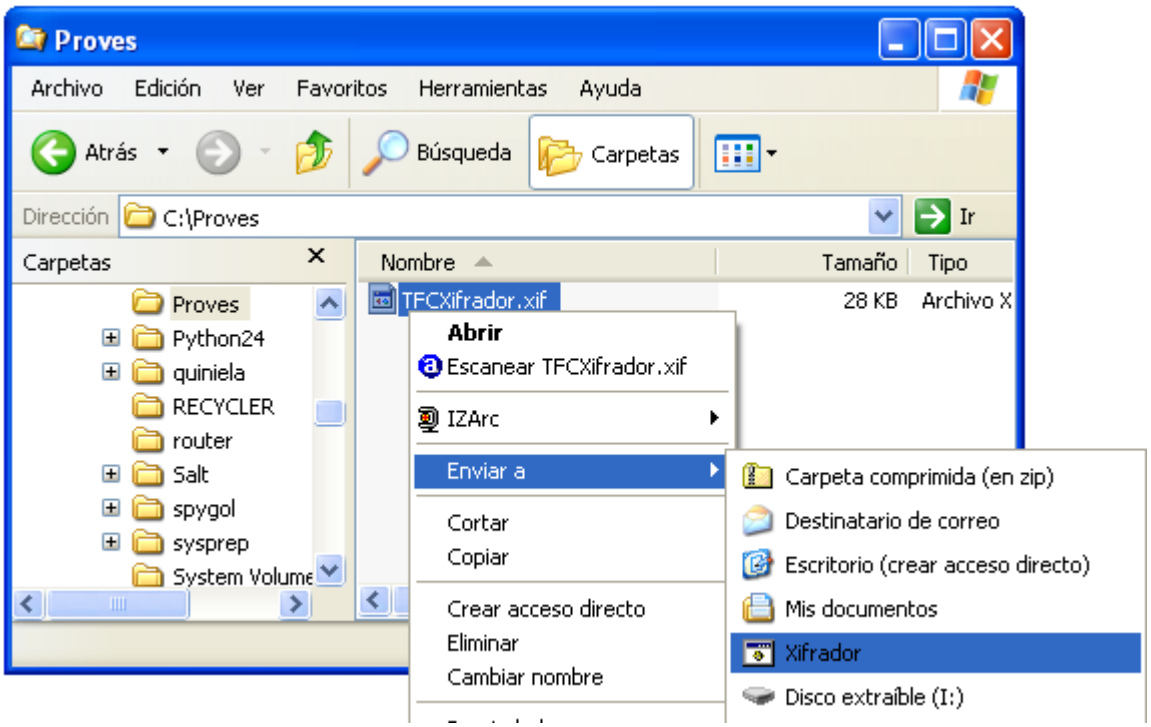


Una vegada acabat el procés apareix el fitxer xifrat al selector de fitxers

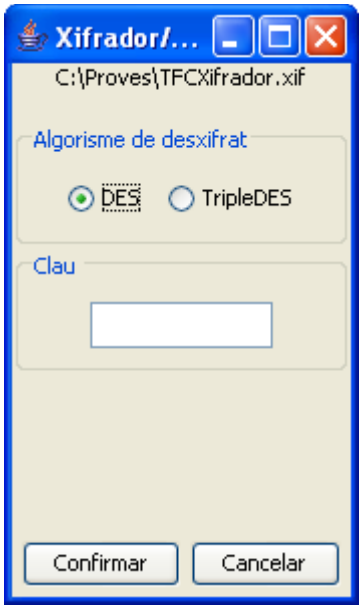


6.2.Desxifrar una carpeta des de l'explorador.

Seleccionem el fitxer a desxifrar des de l'explorador i premem el botó de la dreta del ratolí, i seleccionem enviar a -> xifrador.



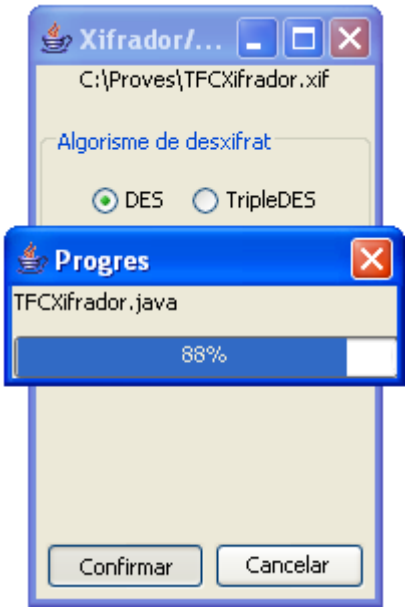
Apareix la pantalla d'introducció de dades de l'aplicació. Com que anem a desxifrar, no apareix el panell de confirmació de la clau.



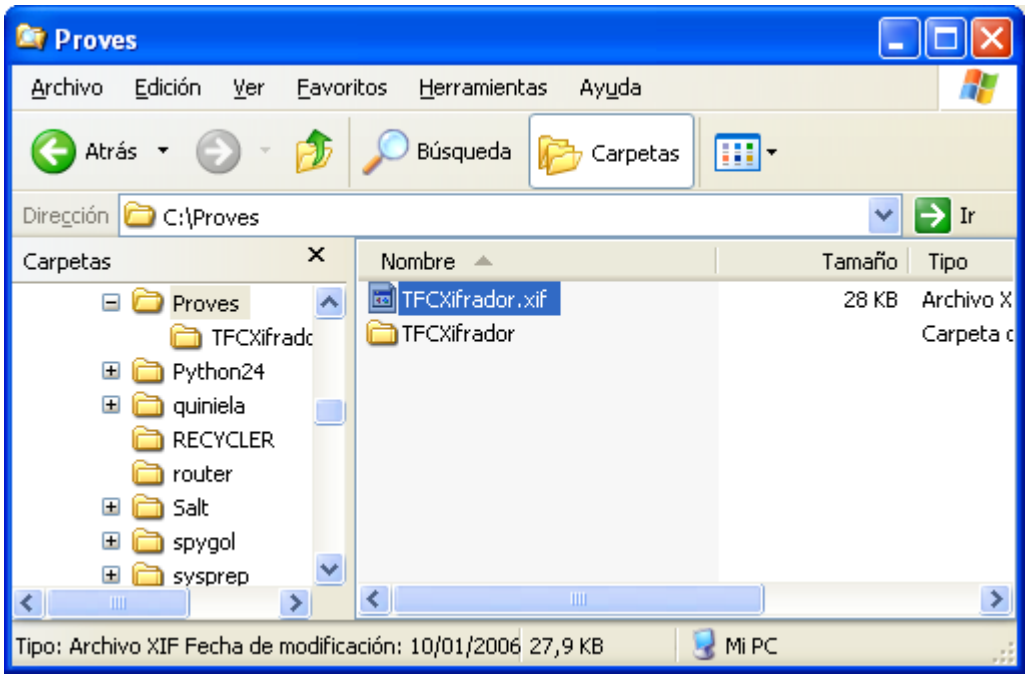
Seleccionem l'algorisme i premem el botó “Confirmar”.

Si l'aplicació no por llegir la capçalera zip, hem introduït mal la clau o l'algorisme, i l'aplicació ens dona un missatge de error

Si l'algorisme seleccionat i la clau introduïda son correctes, comença el procés de desxifrat.



Una vegada acabat el procés, apareix en la finestra del explorador la carpeta desxifrada.



7. Comentaris i conclusions.

Aquesta aplicació, degut a la limitació de temps en el seu desenvolupament, té una sèrie de carències que caldria anar corregint en posteriors versions. Les mes importants son:

a) En el procés de xifrat no esborra els originals. Hauria de preguntar-li al usuari si vol conservar o no els fitxers en text pla una vegada creat el fitxer xifrat.

b) Situa els fitxers desxifrats en el mateix directori absolut on estaven quan es van xifrar. Açò és degut a la forma de afegir els fitxers al fitxer zip. Seria convenient poder desxifrar els fitxers o carpetes en qualssevol lloc, conservant tal sols l'estructura de directoris relativa al directori xifrat.

c) En el procés de xifrat sobreescriu el fitxer *.xif si ja existeix sense preguntar res.

d) En el procés de desxifrat sobreescriu els fitxers existents sense preguntar res. Com que conserva els camins absoluts, sobreescriu els fitxers independentment del lloc des de on es desxifren.

8. Bibliografia i recursos.

Domingo Ferrer J., Herrera Joancomartí J., Rifà Pous H. *Criptografia*. Material didactic de la UOC

Fuster Sabater A., De la Guia Martinez D., Hernández Encinas L., Montoya Vitini F. Muñoz Masqué J. *Tecnicas criptograficas de protección de datos* ra-ma 1997

Norma PKCS#5 v 1.5 dels laboratoris RSA
ftp://ftp.rsasecurity.com/pub/pkcs/doc/pkcs-5.doc

Bruce Eckel *Thinking in Java* e-book http://www.mindview.net/Books/TIJ/

Wikipedia (enciclopedia en línia) http://www.wikipedia.org/

Java examples http://www.java2s.com/

Java Technology http://java.sun.com/

9.Annexos.

9.1.Classe TFCXifrador

```
import java.io.*;
import javax.swing.*;
import javax.swing.border.*;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.*;

/**
* @author Francisco Miquel San Lorenzo
* @version 10/01/2006  Classe principal
* Aquesta és la clase principal del Treball Final de Carrera.
* Implementa un Xifrador/Desxifrador de fitxers i directoris, utilitzant la norma PKCS#5 v 1.5 i a més
* una petita variació sobre la norma, utilitzant l'algorisme TripleDes.
*/
public class TFCXifrador extends JFrame implements PropertyChangeListener,
ActionListener {
   static private TFCXifrador frame;
   static private Xifrador x;
   static private JPanel dades;
    static private TitledBorder algorismeBorde;
   static private JPanel algorismePanel;
    static private JPanel clauPanel;
    static private JPanel confirmaPanel;

   static private JPasswordField clau;
    static private JPasswordField confirma;
    static private ButtonGroup grup;
    static private JRadioButton des;
    static private JRadioButton tripleDes;

   static private JFileChooser seleccionador;
    static private String fitxer;
    /**
  * Constructor de la classe TFCXifrador, Crea un JFrame per al entorn gràfic, i construeix un Panell
  * per a introduir la clau i seleccionar l'algorisme de xifrat/desxifrat.
  */
   public TFCXifrador(){
      super("Xifrador/Desxifrador");    // Cridem al constructor de la classe JFrame
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);    // Per a que acabi el programa al tancar la finestra.
        //  Creem els contenidors dels components
        dades = new JPanel(new GridLayout(0,1));
        algorismePanel = new JPanel();
         clauPanel = new JPanel();
         confirmaPanel = new JPanel();
         algorismeBorde = new TitledBorder("Algorisme de xifrat");
        algorismePanel.setBorder(algorismeBorde);
         // Creem un grup amb dos bottons de radio per a seleccionar l'algorisme i l'afegim a un contenidor
        ButtonGroup grup = new ButtonGroup();
         des = new JRadioButton("DES");
         des.setSelected(true);
         tripleDes = new JRadioButton("TripleDES");
         grup.add(des);
         grup.add(tripleDes);
         algorismePanel.add(des);
         algorismePanel.add(tripleDes);
         // Creem un camp per a la introducció de la clau i l'afegim a un contenidor
         clauPanel.setBorder(new TitledBorder("Clau "));
         clau = new JPasswordField(10);
         clauPanel.add(clau);
         // Creem un camp per a la introducció de la confirmació de la clau i l'afegim a un contenidor
        confirmaPanel.setBorder(new TitledBorder("Confirma la Clau "));
         confirma = new JPasswordField(10);
         confirmaPanel.add(confirma);
         // Afegim els contenidors al panell
        dades.add(algorismePanel);
         dades.add(clauPanel);
         dades.add(confirmaPanel);
     }

     /**
    * El mètode propertyChange implementa el listener PropertyChangeListener i l'utilitzarem per a comprovar
    * si el fitxer seleccionat és un fitxer xifrat (*.xif) o no, i d'aquesta manera ocultar o visualitzar el contenidor
    * de comprovació de clau ( si es tracta d'un fitxer xifrat, l'ocultarem, ja que per a desxifrar no necessitem
    * comprovar la clau introduïda).
    */
     public void propertyChange(PropertyChangeEvent e) {
        String pname = e.getPropertyName();
        if (JFileChooser.SELECTED_FILE_CHANGED_PROPERTY.equals(pname)) {
            // s'ha sel·leccionat un fitxer
            File f = (File) e.getNewValue();
            if ((f != null) && (f.getName().toLowerCase().endsWith(".xif"))){
                confirmaPanel.setVisible(false);
                algorismeBorde.setTitle("Algorisme de desxifrat");
                algorismePanel.paint(algorismePanel.getGraphics());
            }
            else{
```

```
            confirmaPanel.setVisible(true);
            algorismeBorde.setTitle("Algorisme de xifrat");
            algorismePanel.paint(algorismePanel.getGraphics());
        }
    }
}

/**
* El mètode actionPerformed implementa el listener ActionListener i l'utilitzarem per a comprovar
* quin botó s'ha pres, i actuar en conseqüència.
*/
public void actionPerformed(ActionEvent e) {
    File f;
     String clauText, confirmaText;
    if (e.getActionCommand().equals("CancelSelection")){
        // S'ha pres Cancelar, sortim de l'aplicació
        System.exit(0);
    }
    if (e.getActionCommand().equals("ApproveSelection")){
        // S'ha pres  Seleccionar des del selector de fitxers
        f= seleccionador.getSelectedFile();
         clauText = new String(clau.getPassword());
         confirmaText = new String (confirma.getPassword());
         if (f.getName().toLowerCase().endsWith(".xif") || clauText.equals(confirmaText)){
            // Coincideixen la clau i la seva confirmació, o anem a desxifrar el fitxer
            x.run(f, clau.getPassword(), des.getModel().isSelected(), frame );
            seleccionador.rescanCurrentDirectory();
             clau.setText("");
             confirma.setText("");
             des.setSelected(true);
         }
         else {
            // no coincideixen la clau i la seva confirmació
             JOptionPane.showMessageDialog(frame, "Les claus no coincideixen", "Error",
JOptionPane.ERROR_MESSAGE);
         }
     }
    if (e.getActionCommand().equals("Confirmar")){
        // S'ha pres Confirmar des de el panell d'introducció de dades.
        f= new File(fitxer);
         clauText = new String(clau.getPassword());
         confirmaText = new String (confirma.getPassword());
         if (f.getName().toLowerCase().endsWith(".xif") || clauText.equals(confirmaText)){
            // Coincideixen la clau i la seva confirmació, o anem a desxifrar el fitxer
            if (x.run(f, clau.getPassword(), des.getModel().isSelected(), frame ))
                System.exit(0);
        }
         else {
```

```
                // no coincideixen la clau i la seva confirmació
                JOptionPane.showMessageDialog(frame, "Les claus no coincideixen", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }
    /**
    * Mètode principal,
    * Si el nombre d'arguments és 0, crida al selecctor de fitxers.
    * Si el nombre d'arguments és 1, aquest argument serà un fitxer o directori per a xifrar/desxifrar,
    *   per tant, visualitzarem el panell introducció de dades per a xifrar o desxifrar.
    * Si el nombre d'arguments és mes gran que 1, visualizarem un missatge de error.
    */
    public static void main(String[] argument) {
        if (argument.length > 1){
            System.out.println("Nombre de paràmetres incorrecte.\nLa sintaxis és: TFC_Xifrador
[Fitxer o Directori]");
            return;
        }
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) { }

        frame = new TFCXifrador();  // Creem un objecte TFCXifrador
        x = new Xifrador();  // Creem un objecte Xifrador, per a poder xifrar/desxifrar

        if (argument.length == 0){
            // creem un seleccionador de fitxers amb el panell d'introducció de dades.
            seleccionador = new JFileChooser();
            seleccionador.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
            seleccionador.setApproveButtonText("Seleccionar");
            seleccionador.setAccessory(dades);
            seleccionador.addPropertyChangeListener(frame);
            seleccionador.addActionListener(frame);
            frame.getContentPane().add(seleccionador);
            frame.pack();
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
        }
        else {
            // creem un panell amb el panell d'introducció de dades i amb dos botons.
            fitxer = argument[0];
            if (argument[0].toLowerCase().endsWith(".xif")){
                confirmaPanel.setVisible(false);
                algorismeBorde.setTitle("Algorisme de desxifrat");
            }
            JLabel informa = new JLabel(fitxer, SwingConstants.CENTER);
            informa.setVerticalTextPosition(SwingConstants.BOTTOM);
```

```
        JPanel botons = new JPanel();
        JButton ok = new JButton ("Confirmar");
        JButton cancela = new JButton ("Cancelar");
        cancela.setActionCommand("CancelSelection");
        ok.addActionListener(frame);
        cancela.addActionListener(frame);
        botons.add(ok);
        botons.add(cancela);
        JPanel complet = new JPanel(new BorderLayout(0,15));
        complet.add(informa,BorderLayout.NORTH);
        complet.add(dades,BorderLayout.CENTER);
        complet.add(botons,BorderLayout.SOUTH);
        frame.getContentPane().add(complet);
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
  }
}
```

9.2.Classe Xifrador

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.util.zip.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.security.*;
import java.util.zip.CRC32;

/**
* @author Francisco Miquel San Lorenzo
* @version 10/01/2006  Classe principal
* Aquesta és la classe per a xifrar o desxifrar.
*/
public class Xifrador {
   static Cipher c;
    static JDialog dialegBarra;
    static JLabel barraNomFitx;
    static JProgressBar barra;

   /**
  * Mètode per a afegir fitxers al fitxer xifrat
  * Si f és un directori, recorre el seu contingut i per a cada element torna a cridar aquest mètode.
  * Si f és un fitxer, l'afegeix al ZipOutputStream.
```

```
  * @param f, és el fitxer o directori a afegir.
  * @param zos, és el ZipOutputStream on afegirem el fitxer o directori f.
  */
   private static void afigFitxers(File f, ZipOutputStream zos) throws IOException,
FileNotFoundException {
       ZipEntry entrada;
       byte [] buffer = new byte [1000];
       int n;
       if (f.isDirectory()) {
          //Creem un array amb tots els fitxers i subdirectoris del directori actual
          String[] nomsFitxers = f.list();
          if (nomsFitxers != null) {
             //Afegim cada entrada del array recursivament
             for (int i=0; i<nomsFitxers.length; i++)  {
                afigFitxers(new File(f, nomsFitxers[i]),zos);
             }
          }
       }
       else {
          //Afegim el fitxer f al fitxer zip
          CRC32 crc32 = new CRC32();
          FileInputStream fi = new FileInputStream(f);
           while ((n = fi.read(buffer)) > -1)
          {
             crc32.update(buffer, 0, n);
          }
          fi.close();
          FileInputStream in =new FileInputStream(f);
          entrada = new ZipEntry( f.getPath() );  // Creem una nova entrada zip
          entrada.setSize( f.length() );              // Guardem la mida del fitxer
          entrada.setTime( f.lastModified() );       // Guardem la data de modificació
          entrada.setCrc(crc32.getValue());         // Guardem el crc32
          zos.putNextEntry(entrada);            // Guardem la entrada al fitxer zip
          int c,count=0;
          // Fiquem el nom del fitxer a l'etiqueta i inicialitzem la barra de progrés
           barraNomFitx.setText(f.getName());
           dialegBarra.update(dialegBarra.getGraphics());
           barra.setMinimum(0);
           barra.setMaximum((int)(f.length()/1000));
           barra.setValue(0);
          // afegim les dades al fitxer zip (i actualitzem la barra de progrés)
          while ((n = in.read(buffer)) > -1)
          {
             zos.write(buffer, 0, n);
             barra.setValue(count++);
              dialegBarra.update(dialegBarra.getGraphics());
           }
```

```
            in.close();
            zos.closeEntry();
        }
    }

    /**
    * Mètode cridat des de TFCXifrador
    * Crea el quadre de dialeg amb la barra de progrés.
    * Inicialitza el valor de salt i del contador d'iteracions
    * Crea una instancia del xifrador (Chipher) amb l'algorisme i la clau passades com a paràmetres.
    * Inicialitza el xifrador per a xifrar o desxifrar, i crida al mètode per a afegir o extraure arxius.
    * @param f, és el fitxer o directori a afegir.
    * @param clau, és la clau que hem introduït abans.
    * @param des, ens indica l'algorisme a utilitzar (true -> DES, i false -> TripleDES)
    * @param frame, és l'entorn gràfic on visualitzar els missatges.
    * @return vertader, si tot anat be, i fals si hi ha agut algun error (clau o algorisme incorrectes)
    */
    public static boolean run(File f, char[] clau, boolean des, TFCXifrador frame ) {
        String algorisme;
        //creem el quadre de dialeg amb la barra de progrés.
        dialegBarra = new JDialog((JFrame)frame, "Progres", false);
         barraNomFitx = new JLabel(" ");
         barra = new JProgressBar() ;
         barra.setStringPainted(true);
        dialegBarra.getContentPane().setLayout(new BorderLayout(0,10));
         dialegBarra.getContentPane().add(barraNomFitx,BorderLayout.NORTH);
         dialegBarra.getContentPane().add(barra,BorderLayout.CENTER);
         dialegBarra.setSize(200,80);
         dialegBarra.setLocationRelativeTo(frame);
         PBEKeySpec pbeKeySpec;
        PBEParameterSpec pbeParamSpec;
        SecretKeyFactory keyFac;
        // Salt
        byte[] salt = { (byte)0xd4, (byte)0xa3, (byte)0xff, (byte)0x9e,
                        (byte)0x12, (byte)0xc7, (byte)0xd0, (byte)0x84 };
        // contador de Iteracions
        int count = 1000;

        // Creem els parametres PBE
        pbeParamSpec = new PBEParameterSpec(salt, count);
        if (des)
            algorisme = new String("PBEWithMD5AndDES");
        else
             algorisme = new String("PBEWithMD5AndTripleDES");
        try{
             //Convertim la clau en un SecretKey
            pbeKeySpec = new PBEKeySpec(clau);
```

```
            keyFac = SecretKeyFactory.getInstance(algorisme);
            SecretKey pbeKey = keyFac.generateSecret(pbeKeySpec);

           // Creem el PBE Cipher
           c = Cipher.getInstance(algorisme);
           if (f.isFile() && (f.getName().toLowerCase().endsWith(".xif"))){
               //inicialitzem el PBE Cipher per a desxifrar
               c.init(Cipher.DECRYPT_MODE, pbeKey, pbeParamSpec);
               return (desxifra(f));
            }
            else {
               //inicialitzem el PBE Cipher per a xifrar
               c.init(Cipher.ENCRYPT_MODE, pbeKey, pbeParamSpec);
              return(xifra(f));
            }
        } catch (Exception e){
           e.printStackTrace();
           return(false);
        }
    }

    /**
    * Mètode per a xifrar
    * Crea un ZipOutputStream connectat a un CipherOutputStream (amb el xifrador creat abans)
    * i connectat a un FileOutputStream dirigit a un fitxer amb el mateix nom del fitxer o directori
    * a xifrar, i amb l'extensió *.xif
    * @param f, és el fitxer o directori a afegir.
    * @return vertader, si tot anat be, i fals si hi ha agut algun error
    */
    public static boolean xifra(File f) {
        try {
            FileOutputStream fi =new FileOutputStream(f.getPath() + ".xif");
            CipherOutputStream cos = new CipherOutputStream(fi, c);
            CheckedOutputStream csum =new CheckedOutputStream(cos, new Adler32());
            ZipOutputStream out =new ZipOutputStream(new BufferedOutputStream(csum));
           out.setMethod(ZipOutputStream.STORED);  // utilitzem el ZIP sense comprimir
           dialegBarra.show();  // Visualitzem la barra de progrés
            afigFitxers(f,out);    // Afegim els fitxers
           dialegBarra.hide();   // Ocultem la barra de progrés.
            out.close();
           return (true);
        } catch ( Exception e) {
           e.printStackTrace();
           return(false);
        }
    }
```

```
 /**
* Mètode per a desxifrar
* Crea un ZipInputStream connectat a un CipherInputStream (amb el xifrador creat abans)
* i connectat a un FileInputStream connectat al fitxer a desxifrar
* @param f, és el fitxer o directori a afegir.
* @return vertader, si tot anat be, i fals si hi ha agut algun error
*/
 public static boolean desxifra(File f) {
    try {
        FileInputStream fi =new FileInputStream(f);
        CipherInputStream cis = new CipherInputStream(fi,c);
        CheckedInputStream csumi =new CheckedInputStream(cis, new Adler32());
        ZipInputStream in =new ZipInputStream(new BufferedInputStream(csumi));
        ZipEntry ze;
       byte [] buffer = new byte [1000];
       int n;

       if ((ze=in.getNextEntry())==null){
           JOptionPane.showMessageDialog(null, "Clau o Algorisme incorrectes", "Error",
JOptionPane.ERROR_MESSAGE);
          return(false);
       }
       else {
           dialegBarra.show(); // Visualitzem la barra de progrés
           do {
              // Creem un fitxer a partir d'un zipEntry Canviant la barra \ per la barra / per a que el programa
              // sigui multiplataforma
              File fo = new File(ze.getName().replaceAll("\\\\","/"));
              if (fo.getParent() != null ){
                 // Si no existeix el directori on va el fiter, el creem
                 File p=new File(fo.getParent());
                 if (! p.exists()) p.mkdirs();
              }
              FileOutputStream fos =new FileOutputStream(fo);
               int x, count =0;
              // Fiquem el nom del fitxer a l'etiqueta i inicialitzem la barra de progrés
               barraNomFitx.setText(new File(ze.getName()).getName());
               dialegBarra.update(dialegBarra.getGraphics());
               barra.setMinimum(0);
               barra.setMaximum((int)(ze.getSize()/1000));
               barra.setValue(0);
              // Llegim les dades del fitxer zip  i les guardem al fitxer que hem creat
              // i actualitzem la barra de progrés
              while ((n = in.read(buffer)) > -1)
              {
                 fos.write(buffer, 0, n);
                 barra.setValue(count++);
```

```
                    dialegBarra.update(dialegBarra.getGraphics());
                 }
                in.closeEntry();
                fos.close();
            } while((ze = in.getNextEntry()) != null);
            dialegBarra.hide();    // Ocultem la barra de progrés.
          }
          return (true);
      } catch ( Exception e) {
          e.printStackTrace();
          return(false);
      }
   }
}
```