Treball Fi de Carrera
Seguretat Informàtica
Xifratge d'arxius

Albert Daudé i Costa
ETIG / ETIS

Antoni Martínez Ballesté

19.09.2005

Aquest treball es el fruit d'innumerables hores de son i d'un munt de vivències que sense el teu recols no hagués pogut superar, que menys que dedicar-te'l.

'm.v.

Resum

Aquest treball de final de carrera inclou el desenvolupament d'un programari que permet xifrar arxius utilitzant un algorisme de flux. El desenvolupament del programari ha estat dividit en tres parts, el generador de números aleatoris, l'aplicació de l'algorisme de xifrat on s'han aplicat les tècniques adients de criptografia i la interfície gràfica per l'usuari.

L'aplicatiu permet xifrar des d'un fitxer, a un directori sencer, el mecanisme es aplicar un xifratge de flux basat en un generador de números aleatoris de tipus determinista. Aquest últim s'inicialitza mitjançant la clau del propi usuari. El mateix programari permet tornar a desxifrar els fitxers i validar la congruència i consistència dels mateixos.

Per a la realització de qualsevol de les operacions permeses es demana a l'usuari la introducció de la corresponent paraula clau, evidentment en el cas del xifrat aquesta paraula es lliure, però per el desxifrat i el procés de validació, es necessari que la clau coincideixi amb la clau que va ser utilitzada en el procés de xifrat.

Per raons de seguretat, no s'ha introduït cap 'backdoor' ni recurs que permeti recuperar el text en clar en cas d'oblit de la clau. Malgrat la complexitat de les operacions que es realitzen en el processos de xifrat i desxifrat (i com no també en el de validació), no és massa difícil incloure en el codi un algorisme de recuperació de la clau mitjançant una 'master-key' o un sistema de preguntes, però s'ha entès que això podria vulnerar la integritat del sistema de xifrat, i que per tant quedava fora de l'interès del projecte inicial.

Pel mateix motiu, totes les parts del projecte s'han recolzat, en la mesura del possible, sobre solucions i formats criptogràfics reconeguts.

Índex

1. Introducció

Aquest projecte de final de carrera s'ha realitzat íntegrament en plataforma Java release v1.2. Tret de les llibreries pròpies de la plataforma Java, tot el codi de l'aplicatiu s'ha desenvolupat íntegrament per aquest projecte en concret.

L'objectiu del projecte és l'elaboració d'un programari que permeti obtenir fitxers xifrats mitjançant un criptosistema simètric de flux, on la clau de xifratge-desxifratge ha de dependre de la contrasenya de l'usuari.

Per al procés de xifrat, s'utilitza una clau d'encriptació basada en el **teorema fonamental de Shannon per al secret perfecte**, (es un dels teoremes de la criptografia moderna), on la clau de xifratge es tant llarga com el text que es pretén xifrar (xifra de Vernam). Amb aquesta finalitat, l'aplicatiu utilitza un generador de números pseudo-aleatoris de tipus determinista.

El desenvolupament del projecte s'ha estructurat en tres parts:

- Generació de l'aplicatiu, que inclou les següents etapes:
  - Desenvolupament del motor del criptosistema de xifrat – desxifrat.
  - Desenvolupament de la gestió de la contrasenya d'usuari.
  - Desenvolupament del generador de números pseudo-aleatoris.
  - Desenvolupament de la interfície gràfica per l'usuari.

- Elaboració de la memòria del treball.
- Elaboració de la presentació del treball.

Fruit de la subdivisió de les etapes de la generació de l'aplicatiu a dalt esmentades, s'ha obtingut un programari altament estructurat, fins el punt en cada classe inclou una funcionalitat específica, lligada amb el mateix nom de la classe (p.e. TFCXifrador, integra el criptosistema per a xifrar el text en clar). Tot i que, la funcionalitat de cadascuna de les classes, s'especifica en els següents apartats, en síntesi podem concloure que el motor de l'aplicatiu es pot desglossar en quatre parts ben diferenciades, el criptosistema de xifrat-desxifrat, la gestió de la contrasenya d'usuari, el generador de números pseudo-aleatoris i la interfície gràfica d'usuari.

2. Disseny del programa.

2.1. Característiques de l'aplicatiu del projecte.

Com ja s'ha dit en la introducció, el resultat de l'aplicatiu es un codi altament estructurat envers la funcionalitat. Per aquest motiu s'ha separat el codi amb funcionalitats diferents, en classes independents, d'aquesta manera, tot el codi que gestiona la clau d'usuari ó el que s'encarrega d'obtenir una signatura així com el que integra el generador de números pseudo-aleatoris es troben en classes diferents.

Com a resultat d'aquesta divisió del codi de l'aplicatiu, resulta molt senzill modificar una funcionalitat en concret, per exemple, si es volgués incloure un criptosistema diferent, seria tant senzill com canviar la crida de la classe actual (TFCXifrador), per la nova classe.

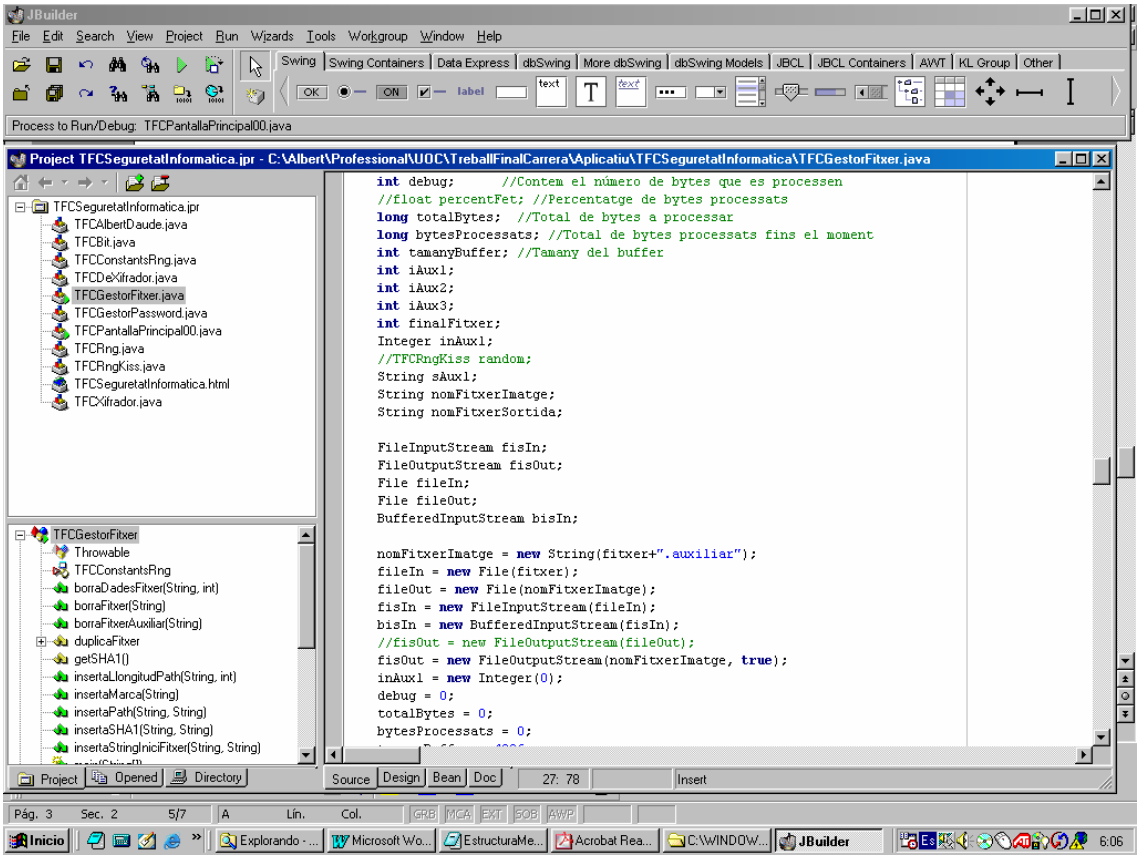
2.2. Classes que integren l'aplicatiu del projecte.

A continuació s'enumeren alfabèticament i s'especifiquen les funcionalitat de les classes que s'integren en l'aplicatiu del projecte:

- TFCAlbertDaude.java, aquesta classe inclou el que podríem anomenar els procediments a executar, es a dir, s'encarrega de cridar a la resta de les classes en funció del tipus de selecció realitzada per l'usuari. Jeràrquicament es troba a dalt de tot de l'arbre d'execució i des de el punt de vista de un desenvolupament en tres capes, la podríem classificar com una capa de segon nivell.
- TFCBit.java, aquesta classe inclou tots els mètodes i funcions necessaris per a realitzar operacions a nivell de bit amb la plataforma Java. En propers apartats ja s'explicarà el perquè ha estat necessari generar aquesta classe.
- TFCConstantsRng.java, aquesta es una classe parametritzada, com a tal, no inclou mètodes ni funcions. Defineix totes les constants que s'utilitzen en la resta de classes que s'inclouen en el projecte, p.e. la ubicació de la signatura dintre del fitxer de xifrat.
- TFCDeXifrador.java, com el seu nom indica, aquesta classe implementa la part de l'algorisme de xifratge simètric de flux que s'encarrega de realitzar la conversió del xifrat a text en clar.
- TFCGestorFitxer.java, inclou tots els mètodes que s'encarrega de manipular fitxers. Concretament en alguns dels processos implementats es necessari generar fitxers temporals i aquesta classe s'encarrega del seu tractament.
- TFCGestorPassword.java, aquesta classe s'encarrega d'una de les particularitats del projecte, fer de pont entre la contrasenya introduïda per l'usuari i el generador de números aleatoris que determinarà el xifrat. Es a dir, a partir de la contrasenya d'usuari, aquesta classe genera el conjunt de claus (llavors) que inicialitzaran el generador de números pseudo-aleatoris i per tant, que determinarà la seqüència dels valors de la clau de xifrat.

- TFCPantallaPrincipal00.java, tal i com el seu nom indica, aquesta classe inclou el codi de la interfície gràfica d'usuari (GUI). Sota la perspectiva d'un desenvolupament en tres capes aquesta classe s'ubicaria en la primera capa.
- TFCRngKiss.java, en aquesta classe s'inclou el codi (mètodes i funcions) del generador de números pseudo-aleatoris. Concretament s'ha implementat un dels millors generadors segons la comunitat científica on un dels seus màxims representats en George Marsaglia que es qui el va desenvolupar. En els pròxims apartats s'inclou un anàlisi del comportament d'aquest generador.
- TFCXifrador.java, de la mateixa manera que la classe TFCDeXifrador.java, aquesta classe implementa la part de l'algorisme de xifratge simètric de flux que s'encarrega de realitzar la conversió de text en clar a xifrat.

A continuació s'inclou una 'foto' del projecte on es poden visualitzar totes aquestes classes.



2.3. Estructura del programa.

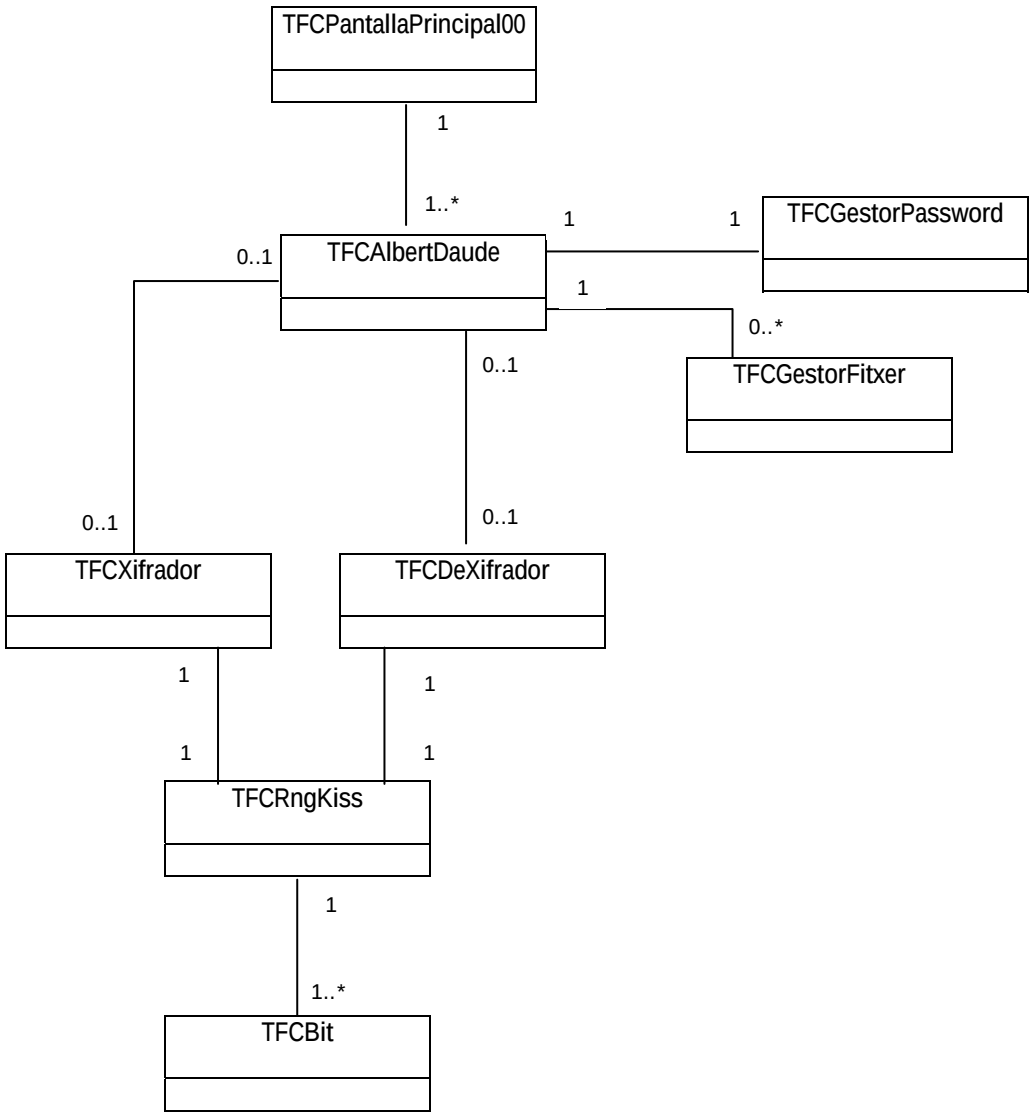
Tot i que no es vol omplir aquest document amb 'mil' gràfics, i sent conscients de que es demana explícitament evitar al màxim la inclusió de diagrames en aquest tipus de redactats, s'ha cregut oportú incloure un conjunt mínim d'aquests, que permeten, sota el nostre punt de vista, entendre més ràpidament la composició i l'execució del programari desenvolupat. Per aquesta raó s'inclouen només un diagrama UML (el diagrama estàtic de classes), i un diagrama

de flux que per raons de comprensió s'ha fragmentat en 4 parts. Repetim que aquesta inclusió es mínima i entenem que necessària, no podem imaginar el desenvolupament d'un projecte que no estigui recolzat mínimament per aquests diagrames.

2.3.1. Diagrama UML.

2.3.1.1. Diagrama UML estàtic.

A continuació s'adjunta el diagrama UML estàtic on es reflecteixen les relacions entre classes. En el mateix s'han suprimit intencionadament la descripció de les funcions i variables de cadascuna de les classes per evitar una sobrecàrrega d'informació.

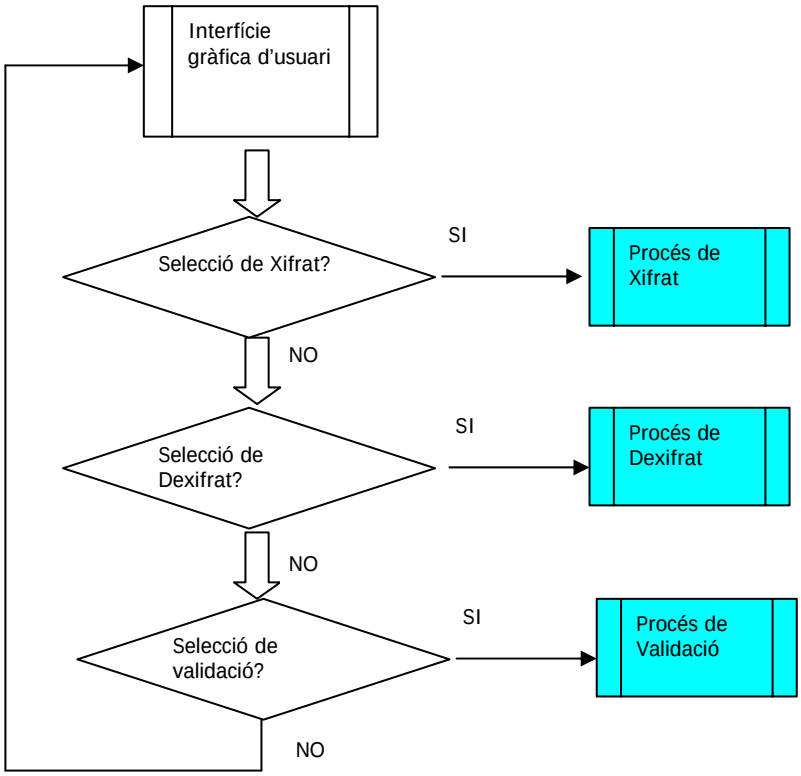


2.3.2. Diagrames de flux.

En els propers quadres s'inclou el diagrama de flux de l'aplicació, donat que el procés de xifrat es lleugerament diferent al procés de desxifrat, s'especifiquen tots dos processos per separat.

2.3.2.1. Diagrama de flux del menú principal.

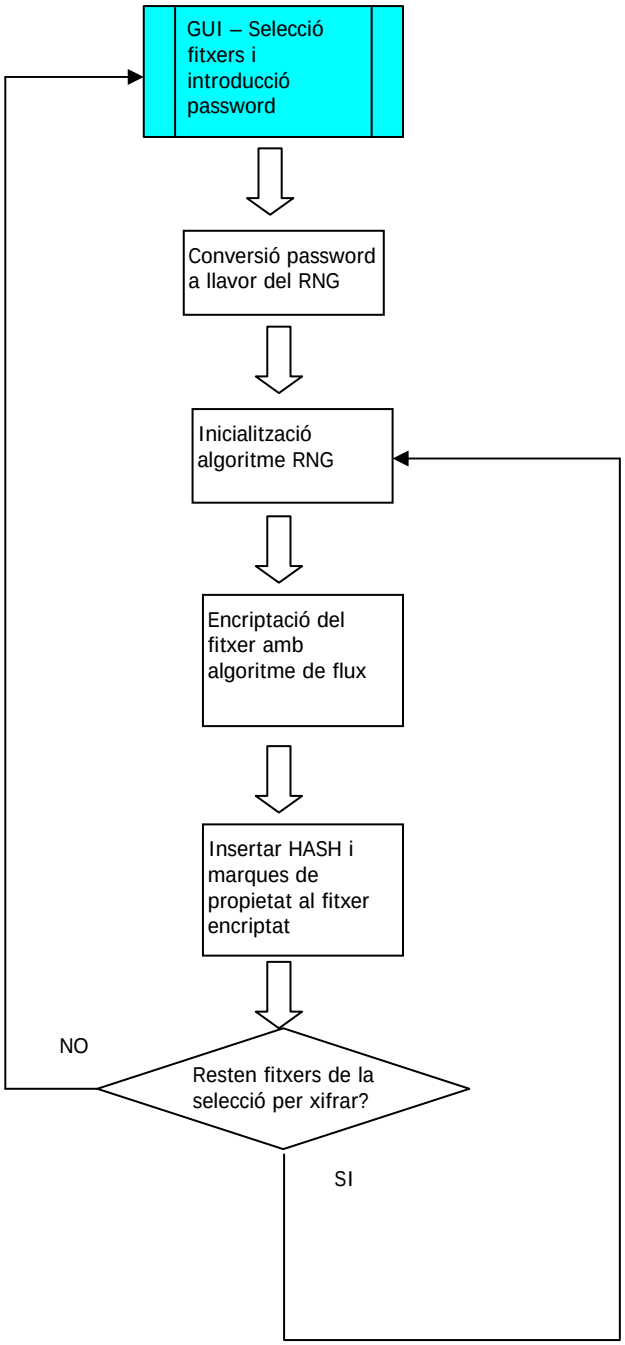
Aquest diagrama representa la pantalla principal de l'aplicatiu, i des d'on l'usuari pot accedir a les diferents opcions del programa.



2.3.2.2. Diagrama de flux del procés de xifrat.

En aquest diagrama es descriu de manera molt esquemàtica quin es l'arbre d'execució del procés de xifrat d'un fitxer.

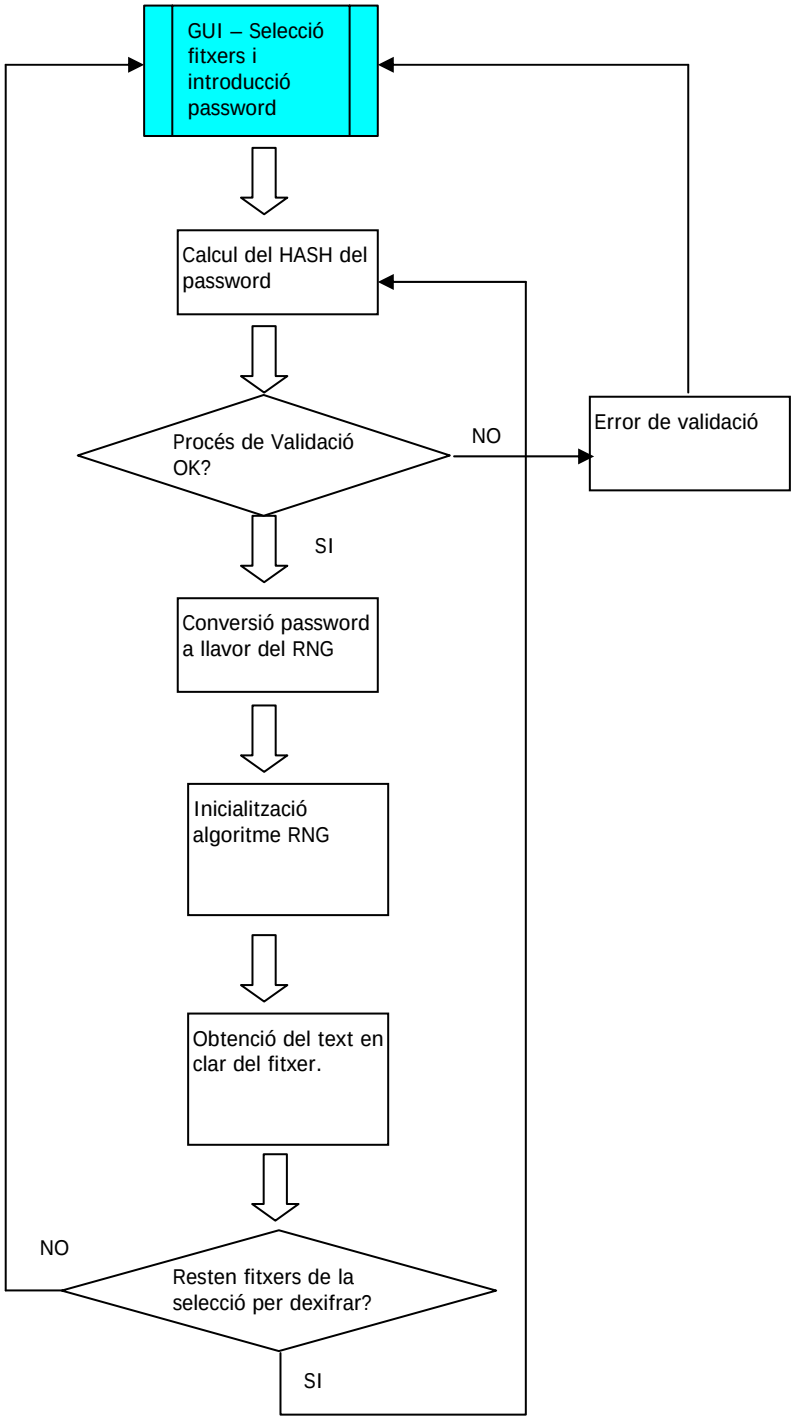
Procés encriptació del fitxer



2.3.2.3. Diagrama de flux del procés de desxifrat.

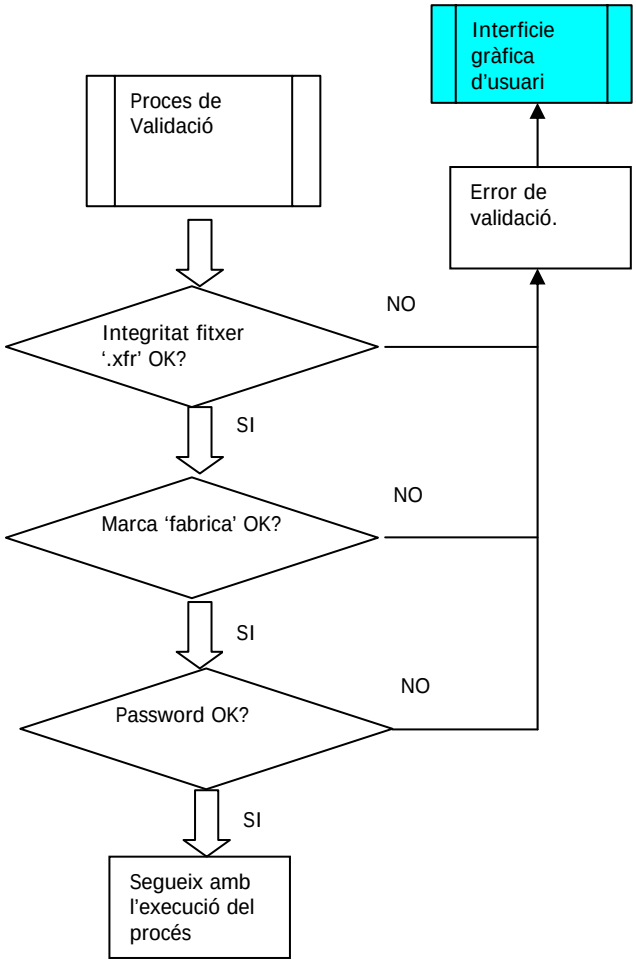
En aquest diagrama es descriu de manera molt esquemàtica quin es l'arbre d'execució del procés de desxifrat d'un fitxer, la diferència fonamental entre aquest procés i el procés de xifrat es el procés de validació.

Procés desencriptació del fitxer



2.3.2.3.1. Detall del procés de validació.

El procés de validació es un dels pilars on es recolza el procediment de desxifrat. Tot i que aquesta funcionalitat no està explícitament inclosa en la descripció del projecte. En el nostre programa, aquest procés té un protagonisme molt accentuat, el motiu es que entenem necessari autenticar el fitxer i verificar la seva integritat, es a dir, validar que un fitxer conserva les característiques originals que el fan nostre i no ha estat alterat ni manipulat. Bàsicament la idea consisteix en aplicar el procés de desxifrat, únicament si el fitxer a desencriptar es realment un fitxer xifrat i signat pel nostre programari. La raó de fons es reforçar la seguretat del sistema de xifrat i evitar atacs dirigits a obtenir informació parcial de com opera al sistema, malgrat que des del punt de vista de la criptografia un criptoatacant sempre coneix l'algorisme de xifrat, entenem que sempre es millor evitar de donar informació al respecte.



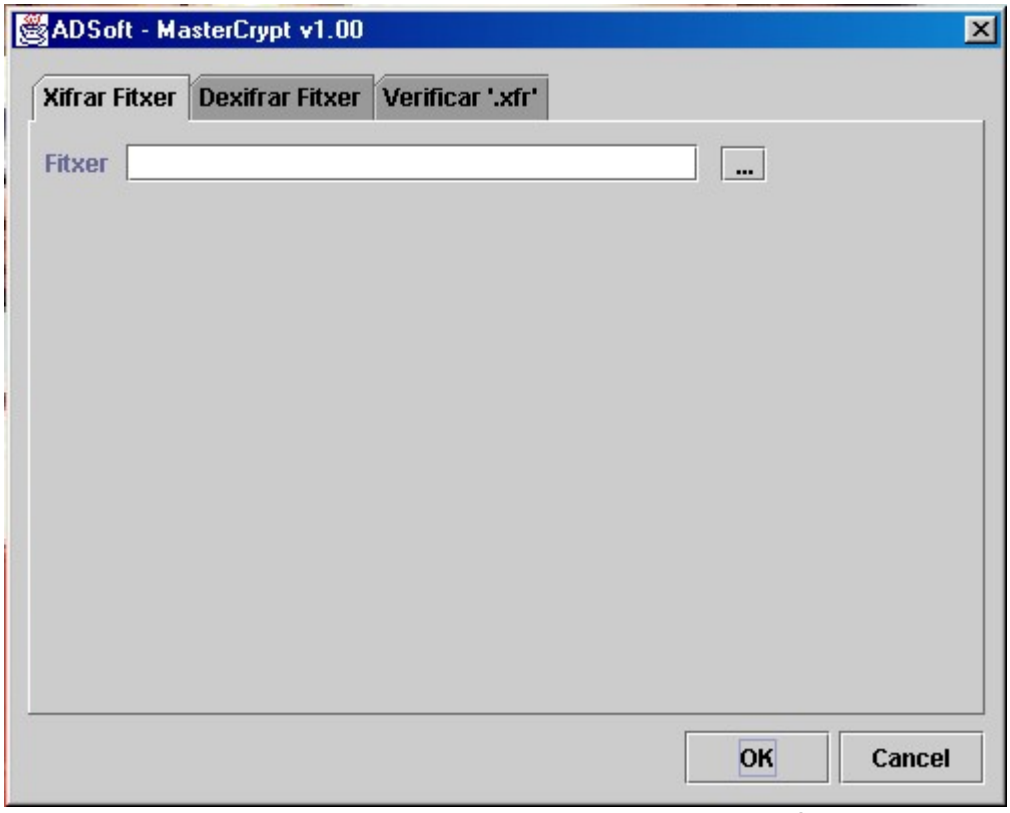
2.4. Descripció de la implementació del programa.

Després d'haver fet una introducció esquemàtica del projecte, resta fer una descripció més detallada del funcionament així com dels raonaments i fonaments sobre els que s'ha desenvolupat, es a dir, un descripció del com i del perquè.

2.4.1. Detall del funcionament del programa.

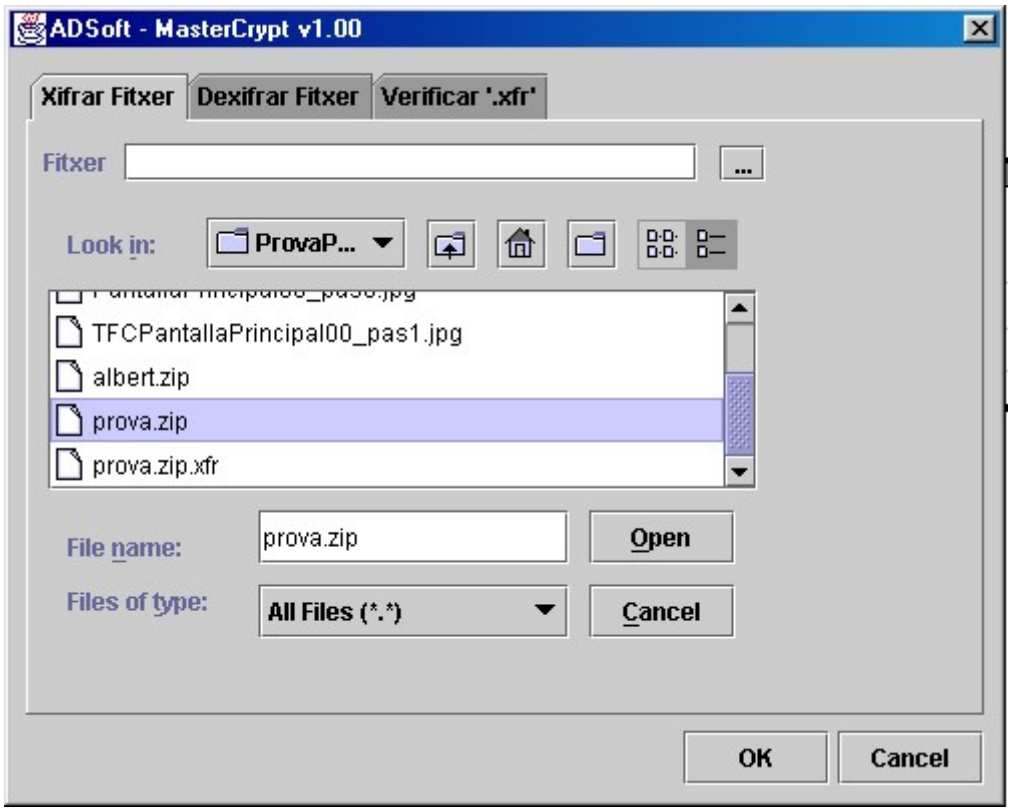
Tal i com s'ha mencionat en la introducció, aquest projecte tracta d'oferir una solució al xifrat de fitxers. La solució que nosaltres proposem s'inicia en una pantalla principal, tal i com es dedueix en els diagrames dels apartats anteriors, des d'on l'usuari pot seleccionar les opcions disponibles:

- Xifrat de fitxers
- Desxifrat de fitxers
- Validació de fitxers

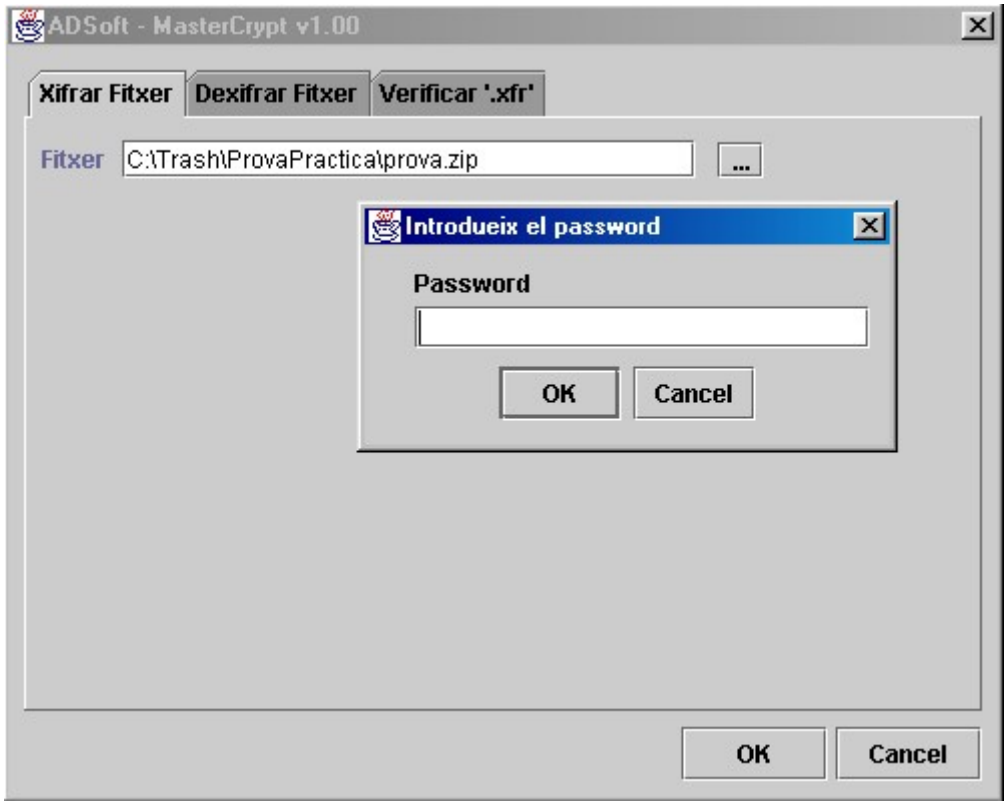


Tal i com es pot apreciar en la captura de pantalla, les opcions són accessibles mitjançant les pestanyes de la pantalla principal.

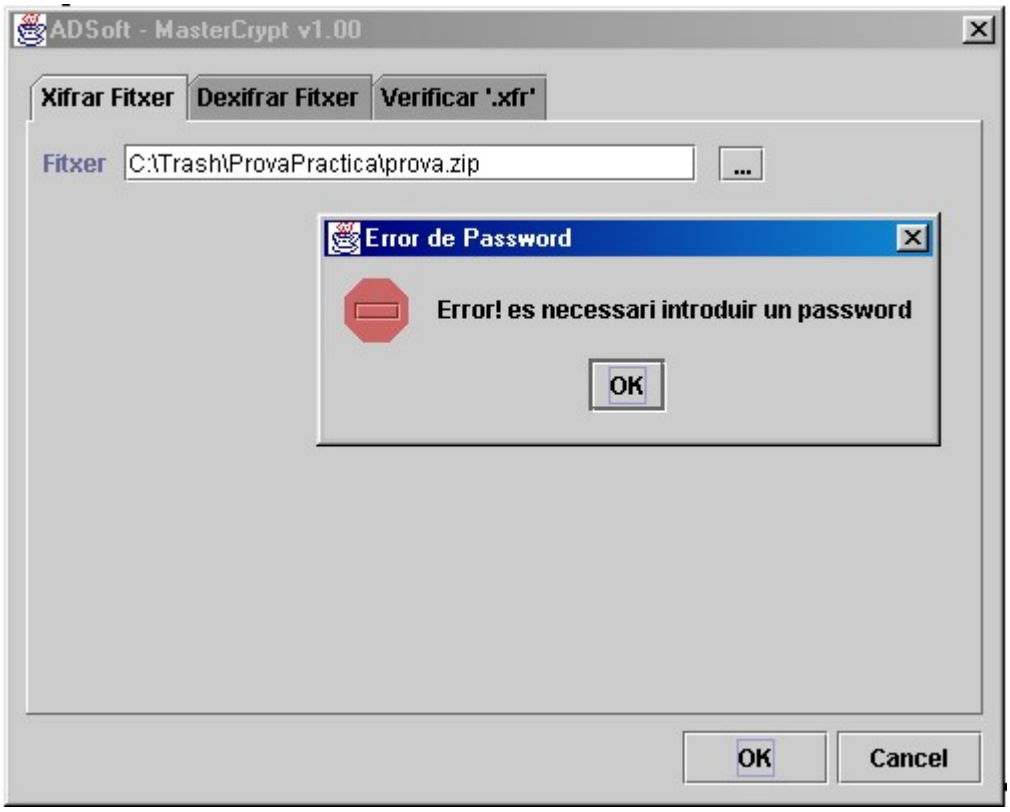
Dintre de cada opció l'usuari pot realitzar la selecció dels fitxers que li interessa xifrar, desxifrar o simplement validar, donant directament el nom del fitxer, o activant el botó al costat del camp de text, que permet desplegar un selector de fitxers tal i com es mostra en la imatge següent .



Un cop feta la selecció del fitxer, el procés continuarà un cop activem el botó OK, que en aquest cas es comú per a totes les opcions, activant-se una finestra en la que es demana la contrasenya a l'usuari.



Si l'usuari no introdueix una contrasenya vàlida (es considera una contrasenya no vàlida un password de longitud 0), el programa generarà un error i cancel·larà les accions fetes fins el moment, excepte la selecció del fitxer que es manté mentre l'usuari no la canviï.



Si pel contrari la contrasenya es considera adient, s'inicia el procés corresponent, que dependrà de la pestanya seleccionada per l'usuari.

3. Aspectes concrets de la implementació.

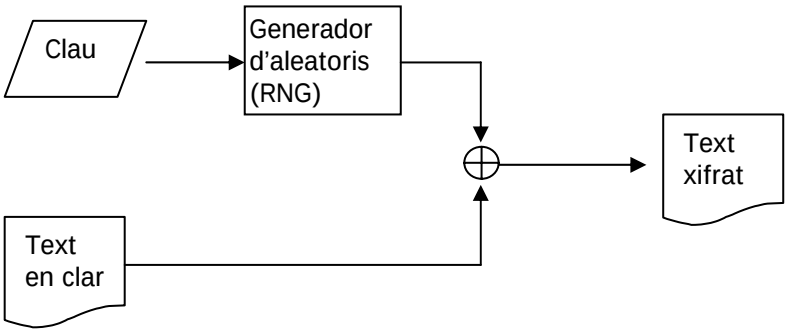
3.1. Fonaments de l'interior del programa (el 'com').

Centrant-nos en el nucli del nostre programa, la solució proposada per el xifrat i el desxifrat es basa en la implementació d'un algorisme simètric de flux, aquests algorismes s'inclouen en el grup d'algorismes de clau compartida. Recordem que aquests criptosistemes funcionen sumant a cada bit del text en clar un bit d'una seqüència aleatòria, també anomenada seqüència de xifratge. Les característiques principals d'aquest tipus d'algorismes són:

- L'operació a nivell de bit ha de ser reversible.
- La seqüència pseudo-aleatòria es la responsable directa de la qualitat del xifrat.

La clau compartida en aquest tipus de criptosistemes actua com a llavor de l'esmentada seqüència aleatòria.

El resultat de l'operació es un nou bit que representa el text xifrat. En el següent gràfic es reflecteix l'operació d'una forma esquemàtica.



3.1.1. Fonaments criptogràfics relacionats amb el projecte (el 'perquè').

Fent una mica d'història, en l'any 1949 C.E. Shannon va proporcionar uns fonaments teòrics a la criptografia, entre aquests, trobem el teorema de la fita fonamental de Shannon per al secret perfecte.

Que un criptosistema es consideri perfectament secret es important perquè indica una màxima protecció contra un criptoanàlisi, i per tant, es sinònim de robustessa en front d'atacs i per tant més segur.

Sense entrar en profunditat, el teorema de la fita fonamental de Shannon per al secret perfecte, ens porta a la conclusió que en un criptosistema perfectament secret, la clau secreta ha de ser com a mínim tant llarga com el text en clar que es pretén xifrar. Fins l'actualitat, l'única clau coneguda que compleix amb aquest requisit es la coneguda com a xifra de Vernam.

La xifra de Vernam consisteix en sumar una clau 'K' al text en clar que es vol xifrar 'T' i el resultat de l'operació es el text xifrat 'X'. On:

- K, T i X prenen valors exclusivament binaris {0,1}.
- La clau es aleatòria i només es fa servir una vegada, es a dir, cada bit de text en clar 'T' es xifra amb un nou bit de clau aleatori 'K' de rang {0,1}.

Tota aquesta argumentació ens porta a pensar que si som capaços de simular la xifra de Vernam, obtindrem un criptosistema prou robust i per tant maximitzarem la seguretat del nostre xifrat. I això es el que nosaltres hem reproduït en el nostre programa.

3.2. Descripció de l'algorisme implementat de xifrat – desxifrat en el projecte.

En el nostre programari hem implementat un critposistema format per un generador de números pseudo-aleatoris determinista que té com a funció simular la xifra de Vernam, es a dir, s'encarrega de generar una seqüència pseudo-aleatòria, on cada nou valor de la seqüència s'utilitza una única vegada i per un únic valor del text en clar, en aquest cas però el rang està comprés entre 0 i 255, donat que el xifratge es realitza sobre un byte.

Donat que l'algorisme implementat es determinista, es a dir, el valor precedent determina el següent valor de la seqüència, podrem obtenir sempre exactament la mateixa seqüència de valors amb una mateix valor d'inicialització (llavor). Aquest detall que pot semblar trivial, es importantíssim, doncs sense aquesta característica no podríem tornar a obtenir el text en clar a partir del xifrat.

Tornant al nostre algorisme de xifrat – desxifrat, obtenim el byte xifrat fent una operació XOR (or exclusiva) entre el següent byte del text en clar i el següent byte obtingut del nostre generador de números pseudo-aleatoris. Malgrat parlem de combinar bytes, l'algorisme realitza l'operació XOR a nivell de bit, això si ho fa de byte en byte.

Al ser l'operació XOR, una funció reversible, ens assegurem que podrem obtenir el text en clar a partir del text xifrat. De fet, si xifrem dues vegades amb la mateixa llavor obtenim de nou el text en clar.

A continuació s'inclou una transformació idèntica a la que es realitza en l'algorisme implementat:

	Valor del byte	Representació binària							
Text en clar	75	0	1	0	0	1	0	1	1
Seqüència aleatòria	156	1	0	0	1	1	1	0	0
Operació XOR	**215**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**

Si tornem a aplicar el mateix valor de la seqüència sobre el resultat, obtindrem el byte original.

	Valor del byte	Representació binària							
Text en clar	215	1	1	0	1	0	1	1	1
Seqüència aleatòria	156	1	0	0	1	1	1	0	0
Operació XOR	**75**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**

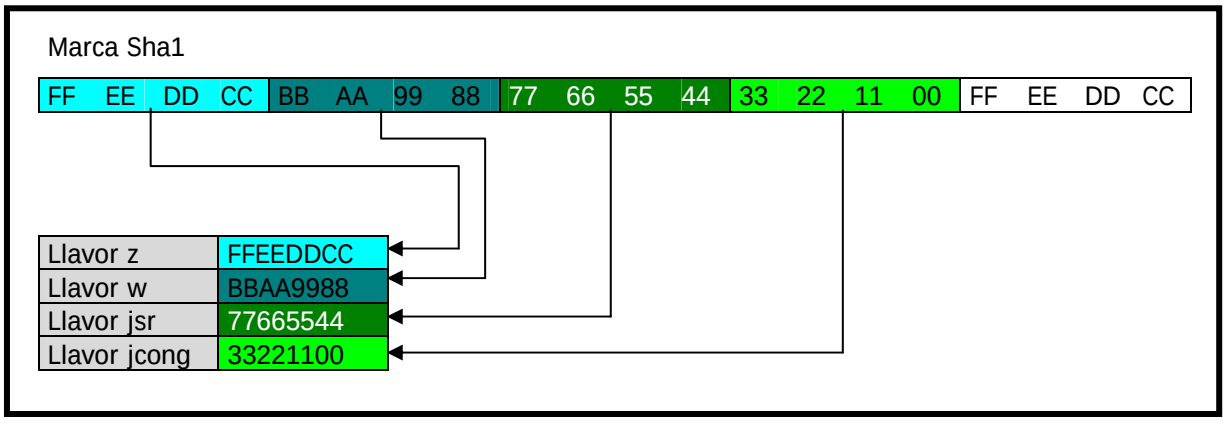
En aquest exemple es pot comprovar com veritablement la funció XOR es reversible.

3.3. Tractament de la contrasenya d'usuari.

Després d'haver vist com funciona el nucli del nostre programari i els fonaments criptogràfics sobre els que ens basem per a realitzar el xifrat, el següent punt, es la descripció del tractament que fa la nostra aplicació amb la contrasenya d'usuari.

En els apartats anteriors ja hem comentat la importància d'utilitzar un generador de números aleatoris determinista, si som capaços d'inicialitzar-lo amb un mateix valor obtindrem sempre la mateixa seqüència. No tractarem en aquest apartat les característiques del generador de números pseudo-aleatoris que s'ha implementat en el nostre projecte, això es quelcom que explicarem en un proper apartat, però si estem obligats a apuntar que una de les particularitats d'aquest generador es que no utilitza una llavor, n'utilitza quatre.

Un cop l'usuari ha donat la seva contrasenya, el nostre programa hi aplica una funció resum, concretament un SHA1 que genera una marca de 160 bits, es a dir 20 bytes o el que es el mateix 40 caràcters hexadecimals. Aquesta signatura es fragmenta i s'agafen els 4 primers bytes per inicialitzar la primera llavor, els següents 4 bytes per inicialitzar la segona llavor, els tercers 4 bytes per a la tercera llavor i finalment els següents 4 bytes per a inicialitzar la darrera llavor, tal i com s'indica en el següent esquema.



El resultat es que obtenim 4 llavors de 32 bits que es manipulen amb variables de tipus 'integer' (en java els 'integer' són de 32 bits). Donat que aquest procés es fix, es possible regenerar les llavors a partir de la mateixa contrasenya, i la entropia de les llavors queda completament lligada al comportament de la funció resum utilitzada.

3.3.1. Fonaments relacionats amb l'ús d'un SHA1 (el 'perquè').

En aquest apartat introduïm el raonament del perquè obtenir una empremta digital SHA1 sobre la contrasenya introduïda per l'usuari. Comencem doncs per una breu introducció sobre els algorismes que calculen funcions resum també anomenades funcions HASH.

Una bona funció HASH ha d'ajustar-se a un seguit de característiques. En primer lloc, la discreció, es a dir a partir del resultat del hash 'H', ha de ser difícil o gairebé impossible obtenir el missatge que l'ha generat. Per altra banda, s'ha d'exigir que qualsevol alteració en el text original 'T' que genera una empremta 'M', per petita que sigui, generi un canvi significatiu en empremta 'M'. En un algorisme ideal, el fet de canviar un sol bit en el en el text original 'T' hauria de generar una empremta 'M' on almenys la meitat de bits hagin canviat.

Addicionalment, es pretén que empremta sigui única, es a dir, que dos empremtes diferents identifiquin a dos texts diferents. Malgrat això no es possible en la realitat, ja que el número empremtes està limitat pel número de bits de la mateixa $2^{bitsEmprempta}$, aquest es un dels motius pels que els algorismes generen empremtes de longituds grans. Un altre motiu tant o més important que el primer es la robustessa enfront d'atacs, es a dir, evitar que sigui factible trobar un altre text que al aplicar l'algorisme HASH 'H', permeti obtenir la mateixa empremta:

$H(T) \neq H(T')$

aquesta possibilitat es coneix com 'RESISTÈNCIA DÈBIL A LES COL.LISIONS', però encara hi ha una condició més restrictiva coneguda com 'RESITÈNCIA A LES COL.LISIONS', per il·lustrar la diferència entre les dues condicions, s'utilitza el anomenat 'atac de l'aniversari'. Imaginem-nos que una persona entra en una sala plena de gent, i es planteja quina quantitat de persones hi ha d'haver per que amb una probabilitat del 50% hi hagi una altra persona que celebri l'aniversari el mateix dia, l'equació d'aquest problema es:

P = 1-(364/365)^n

i per a P>50% es dóna quan n≥253, i correspondria al primer cas plantejat. En canvi si el plantejament es quantes persones hi ha d'haver a la mateixa sala perquè amb la mateixa probabilitat del 50% ni hagi dues qualsevol que celebrin l'aniversari el mateix dia, l'equació es planteja com:

P = 1- [365!/(365^n*(365-1)!)]

on trobem que per a P>50% n>22, es a dir, 11,5 vegades menys que en el cas anterior, aquest es l'altre motiu pel que els algorismes HASH calculen empremtes de molta longitud.

L'algorisme HASH més utilitzat actualment es el SHA-1 (Secure Hash Standart), que va ser desenvolupat per l'Agencia de Seguretat Nacional (NSA) dels EUA, i proposat com standart en 1991 pel National Institute of Standarts and Technology (NIST) dels EUA.

Aquest es el motiu pel qual s'ha decidit implementar-lo en el procés de codificació de la contrasenya, les característiques del SHA1 ofereixen una alta resistència a les col·lisions, i es adequat per a evitar que s'arribi a obtenir dues contrasenyes diferents que podessin generar el mateix SHA1.

Addicionalment, el fet d'utilitzar una empremta 'estandaritza' la contrasenya, es a dir, no importa el tipus de cadena alfanumèrica que l'usuari introdueixi per a protegir el seu fitxer, doncs sempre es transformarà a una signatura de 160 bits, pot semblar irrellevant, però no ho es pas, si ens imaginem per un moment de quina manera podríem arribar a fer, per aconseguir uns valors d'inicialització de la seqüència aleatòria sense saber a priori el tamany de la contrasenya, en seguida ens adonaríem de que seria un problema, no tant sols per a trobar un mecanisme raonable de codificació, sinó també per a assegurar que la llavor de la seqüència pseudo-aleatòria no s'inicialitza sempre amb valors semblants.

3.4. Procés de generació del fitxer '.xfr'.

Fins aquest moment em vist com es realitza el procés de xifrat i com s'inicialitza la seqüència mitjançant la contrasenya de l'usuari. En aquest apartat veurem l'estructura del fitxer '.xfr', que conté tant el text xifrat com un seguit de camps, que seran els responsables de donar coherència al procés de desxifrat.

3.4.1. Estructura del fitxer '.xfr'.

El nostre programa, quan crea el fitxer encriptat (extensió '.xfr'), genera una sèrie de camps que s'inserien al principi del xifrat, i que seran utilitzats posteriorment per al procés de validació i obtenció del text en clar original.

En el següent esquema s'especifica l'ubicació de les diferents marques i funcions resum que s'utilitzen en el procés de validació que comentarem en el proper apartat.

Model de capçalera '.xfr'					
SHA1 Fitxer*	SHA1 password	Id.fitxer '.xfr'	long	path fitxer	Fitxer xifrat

<-----40 bytes-----><-----40 bytes-----><-----40 bytes------><2by><-yy bytes-><--xx bytes-->

*Es considera part del fitxer els camps afegits durant el procés de xifratge, amb excepció d'ell mateix.

Els camps tenen la següent funcionalitat:

- SHA1 Fitxer, representa el SHA1 de tot el fitxer xifrat, incloent-hi com a part del fitxer i per tant del càlcul, els camps addicionals que no formen part del fitxer original.

.- SHA1 Password, representa el SHA1 del SHA1 de la clau introduïda per l'usuari (es calcula el SHA1 de la cadena alfanumèrica que representa el SHA1 de la contrasenya, per evitar la vulnerabilitat que suposaria insertar la primera en l'estructura del fitxer).

.- Id. fitxer, es empremta que permet determinar a l'aplicatiu que es tracta veritablement d'un fitxer xifrat per ell. A efectes pràctics actua com una signatura digital.

.- long, representa la longitud en bytes que ocupa en el següent camp el path complert del fitxer.

.- path fitxer, indica el path+nom del fitxer original sense xifrar, es de mida variable, per aquest motiu s'utilitza el camp anterior.

.- Fitxer xifrat, conté el codi xifrat del fitxer en clar.

3.5. Procés de validació i autenticació del fitxer xifrat.

Fins ara em vist dos dels processos principals del nostre programari, el procés de fragmentar la contrasenya per obtenir l'inici de la seqüència aleatòria que s'utilitza en el xifrat, i el procés del xifrat pròpiament dit que combina el text en clar amb la seqüència aleatòria per a obtenir el text xifrat. Resta doncs per veure el darrer procés del nucli del nostre programari, el procés de validació.

3.5.1. Descripció del procés de validació i autenticació del fitxer xifrat.

Com ja s'ha dit, el procés de validació es el pilar, des del punt de vista de la seguretat, sobre el que es recolza la nostra aplicació, tot i que no es una funcionalitat estrictament inclosa en el projecte.

Aquest procés s'ha dividit en tres parts:

- Validació de la contrasenya d'usuari.
- Autenticació del fitxer.
- Validació d'integritat del fitxer encriptat.

En els següents apartats es descriu amb detall el funcionament de cadascun d'aquest subprocessos, i començarem pel procés de validació de la contrasenya.

3.5.2. Subprocés de validació de la contrasenya d'usuari.

En el procés de xifrat, la contrasenya es trivial, es a dir, tret de la restricció de que no es permet una contrasenya de longitud 0, qualsevol altra es considera vàlida, ara be, queda una qüestió per a resoldre... com s'encarrega el nostre programa de verificar que la contrasenya introduïda per a desencriptar coincideix amb la que es va utilitzar en el procés de xifrat?. En aquest apartat aclarirem aquesta qüestió i descriurem els recursos utilitzats per a solventar aquesta situació.

Com ja s'ha dit, s'utilitza un SHA1 per a identificar empremta de la contrasenya d'usuari, doncs bé, es aquesta empremta es la que s'utilitza per a validar que la contrasenya es correcta.

El procés en realitat es molt senzill, es calcula el SHA1, i es guarda el resultat, un cop el procés de xifrat acaba el seu treball, es torna a calcular el SHA1 sobre la cadena alfanumèrica que representa el SHA1 que s'obté de la contrasenya, i s'incrusta aquesta última signatura en el fitxer resultant que conté el codi xifrat, d'aquesta manera, quan s'intenta desxifrar i per tant s'introdueix la contrasenya, es compara el SHA1 del SHA1 de la contrasenya, amb el SHA1 incrustat en el fitxer '.xfr', si coincideix es valida la clau sinó es dona un missatge d'error i no permet desxifrar el fitxer.

3.5.3. Subprocés d'autenticació del fitxer xifrat.

Em cregut necessari introduir un procés d'autenticació com a part del procés de validació. El motiu, es ajustar-nos al màxim possible al model de sobre digital signat que s'explica més endavant. Concretament em emulat la clau pública del signant d'aquest format, mitjançant el que nosaltres anomenem 'marca de fàbrica', tot i que aquesta intervé en el procés de xifrat.

En aquesta part del procés de validació, el que es fa es extreure del fitxer xifrat '.xfr' la 'marca de fàbrica' i es compara amb l'esperada, si el resultat de la comparació es que no hi ha diferència, es dóna com a valida aquesta part i es continua amb el procés de validació, en cas contrari, es rebutja el fitxer entenent que no es autèntic o ha estat alterat.

3.5.4. Subprocés de comprovació d'integritat del fitxer xifrat.

El darrer pas del procés de validació consisteix en comprovar la integritat del fitxer. Al finalitzar el procés de xifratge del text en clar, s'adjunta empremta de la contrasenya, la marca de fàbrica, i altres camps també creats durant el procés de xifrat i que s'explicaran més endavant. Tota aquesta informació forma un paquet, que a nivell d'integritat es tracta com un únic bloc. Sobre aquest bloc es calcula un SHA1 el resultat del mateix s'adjunta al principi del fitxer.

Així doncs aquesta part consisteix en calcular el SHA1 d'una part del fitxer '.xfr', concretament, abans de fer el càlcul de empremta s'elimina del fitxer el camp corresponent al SHA1 inserit en el procés de generació del fitxer '.xfr'. Un cop finalitzat el càlcul de empremta, es compara amb el camp que em extret. Com sempre, si en aquesta comparació no hi ha diferències, la validació es dóna per correcta i s'inicia el procés de desxifrat, si no es així s'avorta el procés de validació, entenent que la integritat del fitxer ha estat violada i es retorna l'error corresponent.

3.5.5. Fonaments del procés de validació i autenticació del fitxer xifrat (el 'perquè').

A mode de resum, direm que criptogràficament parlant hi han descrits uns standarts, que normalitzen els formats de les dades utilitzades en criptografia de clau pública. De fet, no es pot afirmar que siguin estàndards, ja que es van desenvolupar per una única empresa amb la col·laboració d'altres com (Sun, Apple, Microsoft, etc.), però tot i aquesta consideració, aquestes normes han estat adoptades com a estàndarts formals per protocols de xifrat, SSL, S/MIME, etc.

Fent una mica d'història, aquestes normes conegudes com normes PKCS (també anomenades public-key cryptography standarts), van ser publicades per primer cop en 1991 i des d'aleshores, desenvolupadors d'arreu els han utilitzat com a referència i per tant, han estat implementades àmpliament.

En l'actualitat, hi ha deu normes PKCS, descrites amb la nomenclatura PKCS#x, on x indica el número de norma. Com el nostre objectiu no es entrar en detall en la descripció del conjunt de normes, sinó exposar el perquè s'ha utilitzat uns recursos concrets alhora de generar el procés de validació, passem directament a descriure les normes que hem pres com a referència, la PKCS#5 i la PKCS#7.

La norma PKCS#5, descriu un mètode per a xifrar una cadena de text amb una clau secreta derivada d'una frase de pas. El seu objectiu primari és permetre la transmissió xifrada de claus privades entre ordinadors, encara que pot ser utilitzada per a xifrar missatges. 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.

La norma PKCS#7, que s'encarrega de definir una sintaxi general per als missatges que inclouen millores criptogràfiques, com signatures digitals o xifratge. Aquesta norma defineix diferents tipus d'objectes, però ens centrarem en el 'signedEnvelopedData', que es el format que nosaltres hem pres com a referència per a desenvolupar el procés de validació. L'objecte SignedEnvelopedData consisteix en un sobre autenticat, és a dir, la informació està xifrada amb

una clau simètrica de sessió i aquesta clau està, al seu torn, xifrada amb les claus públiques dels destinataris, i, a més, està signada. Les dades poden estar signades per un nombre qualsevol de signants en paral·lel i, alhora, tot el contingut pot estar xifrat per un o més destinataris.

En el nostre programari em pres com a referència del PKCS#5, la forma d'obtenir una clau a partir de la contrasenya, i dels PKCS#7 em agafat el format d'objecte del sobre digital signat. En tots dos casos s'han introduït variacions per que l'objectiu es recolzar-nos en un estàndard, i per altra banda, el plantejament del projecte no s'ajusta exactament a les definicions d'aquestes normes, com exemple nosaltres apliquem un criptosistema de flux, mentre que el PKCS#5 implica utilitzar un triple DES.

3.6. Altres processos – productes inclosos en l'aplicatiu.

En els apartats anteriors hem vist amb detall els processos principals que suporten la funcionalitat de la nostra aplicació. No obstant, aquests es recolzen en altres subprocessos que tot i que per si mateixos no suporten una funcionalitat específica del programari, si són imprescindibles per que els processos de nivell superior presentin una funcionalitat adequada.

3.6.1. Generador de números pseudo-aleatoris (PRNG).

Probablement aquest es l'apartat més important de l'aplicatiu, si pensem, tal i com s'ha exposat en apartats anteriors, que una de les principals fites que es pretén solventar es el de la seguretat del xifrat. De fet, no serveix per gaire implementar esquemes de xifrat i validació robustos, si el motor que s'encarrega de recolzar tot el sistema no té un comportament adient, es a dir, imaginem-nos per un moment que el generador de números pseudo-aleatoris no té un comportament que podem considerar 'aleatori', segur que un bon criptoanàlisi permetrà amb poc temps trencar el xifrat, i més quan el xifrat utilitzat es un algorisme de flux.

Per tot això, creiem que es important el tractar aquest tema com un apartat més i independent de la resta. Així doncs, veurem uns quants aspectes relacionats amb les característiques del generador de números pseudo-aleatoris, i començarem amb l'estudi del comportament del mateix.

3.6.1.1. Comportament del generador de números pseudo-aleatoris (PRNG).

Sense intenció d'aprofundir massa en la qüestió de decidir si un generador de números pseudo-aleatoris, es pot considerar aleatori, o dit en altres paraules si presenta un comportament que ens permeti pensar que realment es aleatori, direm que nosaltres considerarem un generador vàlid quan:

- Presenti un cicle complert, es a dir un generador de 32 bits, ha de ser capaç de generar els 2^{32}-1 valors possibles (tots menys el 0).
- Tots els valors siguin equiprobables.
- No es pugui predir el següent valor.

Com hem dit aquests serien uns requisits mínims a poder decidir si podem acceptar que un generador d'aleatoris realment se'l pot considerar com a tal. No obstant, hi ha molta literatura al respecte i en l'actualitat podríem dir que el màxim representant a nivell internacional en l'estudi d'aquesta matèria es en George Marsaglia professor d'estadística de la Universitat de l'estat de Florida. Ell mateix es el dissenyador del generador 'KISS' (Keep It Simple Stupid), generador que nosaltres utilitzem en el nostre programari.

3.6.1.2. Avaluació del Comportament del PRNG KISS.

A continuació presentem els resultats obtinguts després d'aplicar uns estadístics sobre una mostra de valors recollits des de el propi generador utilitzat en l'aplicatiu.

Tal i com s'ha argumentat en l'apartat anterior, l'avaluació de comportament es basa en els resultats obtinguts de l'aplicació d'una estadístics en concret sobre una mostra suficientment significativa. Tornant als requisits mínims que nosaltres mateixos hem determinat, per a decidir que qualsevol valor es equiprobable aplicarem l'estadístic del Xi-quadrat, que ens compara el valor esperat amb l'obtingut, en aquest sentit nosaltres compararem la freqüència d'aparició de cadascun dels valors de la mostra. Per a determinar que no es pot predir el seu valor aplicarem un run-Test, que ens dirà si existeixen patrons dintre de la mostra avaluada. En tots els casos, s'avaluaran els resultats utilitzant un contrast amb un nivell de confiança del 95% i sota la hipòtesi nul·la de que la mostra té un comportament aleatori.

Abans de començar però, avaluem la mostra mitjançant els estadístics bàsics (Tamany de mostra, mitjana, i rang són els valors que més ens interessen, tot i que el primer i tercer quartil, també ens donen una idea de si la mostra està sesgada o presenta alguna inclinació).

Descriptive Statistics: Kiss

Variable	N	Mean	Median	TrMean	StDev	SE Mean
Kiss	1000000	127,54	128,00	127,55	73,89	0,07

Variable	Minimum	Maximum	Q1	Q3
Kiss	0,00	255,00	64,00	192,00

Com podem veure en el quadre anterior la mostra té un tamany de 1000000 de dades en una variable independent anomenada Kiss (aquesta mostra es podrà reproduir sempre que s'executi

el main de la classe TFCRngKiss), i on el rang està compres entre 0 i 255. El motiu d'avaluar el generador amb aquest rang es perquè d'aquesta manera avaluem que el comportament es adient per la nostra aplicació (recordem que el nostre mòdul de xifrat opera a nivell de byte, es a dir es codifica amb valors aleatoris compresos entre 0 i 255).

A continuació indiquem els resultats obtinguts de l'estadístic chi-quadrat:

	N observado	N esperado	Residual
0	3942	3906,3	35,8
1	3995	3906,3	88,8
2	3942	3906,3	35,8
3	3890	3906,3	-16,3
4	3811	3906,3	-95,3
5	3776	3906,3	-130,3
6	3879	3906,3	-27,3
7	3868	3906,3	-38,3
8	3935	3906,3	28,8
9	3938	3906,3	31,8
10	3927	3906,3	20,8
11	3959	3906,3	52,8
12	3907	3906,3	,8
13	3854	3906,3	-52,3
14	3951	3906,3	44,8
15	3910	3906,3	3,8
16	3902	3906,3	-4,3
17	3828	3906,3	-78,3
18	3909	3906,3	2,8
19	3906	3906,3	-,3
20	3948	3906,3	41,8
21	3882	3906,3	-24,3
22	3931	3906,3	24,8
23	3898	3906,3	-8,3
24	3827	3906,3	-79,3
25	3922	3906,3	15,8
26	3936	3906,3	29,8
27	3897	3906,3	-9,3
28	3873	3906,3	-33,3
29	3912	3906,3	5,8
30	3845	3906,3	-61,3
31	3983	3906,3	76,8
32	3939	3906,3	32,8
33	3808	3906,3	-98,3
34	3812	3906,3	-94,3
35	3937	3906,3	30,8
36	3894	3906,3	-12,3
37	3914	3906,3	7,8
38	3837	3906,3	-69,3
39	3998	3906,3	91,8
40	3926	3906,3	19,8

	N observado	N esperado	Residual
41	3999	3906,3	92,8
42	3973	3906,3	66,8
43	3873	3906,3	-33,3
44	3869	3906,3	-37,3
45	3831	3906,3	-75,3
46	3883	3906,3	-23,3
47	3854	3906,3	-52,3
48	4044	3906,3	137,8
49	3841	3906,3	-65,3
50	3762	3906,3	-144,3
51	3806	3906,3	-100,3
52	4050	3906,3	143,8
53	4015	3906,3	108,8
54	3882	3906,3	-24,3
55	3950	3906,3	43,8
56	3874	3906,3	-32,3
57	4032	3906,3	125,8
58	3872	3906,3	-34,3
59	3881	3906,3	-25,3
60	3935	3906,3	28,8
61	3941	3906,3	34,8
62	3975	3906,3	68,8
63	3870	3906,3	-36,3
64	3842	3906,3	-64,3
65	4081	3906,3	174,8
66	3892	3906,3	-14,3
67	3917	3906,3	10,8
68	3927	3906,3	20,8
69	3855	3906,3	-51,3
70	3834	3906,3	-72,3
71	3820	3906,3	-86,3
72	3940	3906,3	33,8
73	3936	3906,3	29,8
74	3926	3906,3	19,8
75	3946	3906,3	39,8
76	3864	3906,3	-42,3
77	3900	3906,3	-6,3
78	3919	3906,3	12,8
79	3860	3906,3	-46,3
80	3802	3906,3	-104,3
81	3833	3906,3	-73,3
82	3902	3906,3	-4,3
83	3838	3906,3	-68,3
84	3860	3906,3	-46,3
85	3983	3906,3	76,8
86	3845	3906,3	-61,3
87	3880	3906,3	-26,3
88	3916	3906,3	9,8
89	3887	3906,3	-19,3

	N observado	N esperado	Residual
90	3905	3906,3	-1,3
91	3882	3906,3	-24,3
92	3903	3906,3	-3,3
93	3905	3906,3	-1,3
94	3935	3906,3	28,8
95	3877	3906,3	-29,3
96	3928	3906,3	21,8
97	3874	3906,3	-32,3
98	3992	3906,3	85,8
99	3996	3906,3	89,8
100	3915	3906,3	8,8
101	3918	3906,3	11,8
102	3907	3906,3	,8
103	3870	3906,3	-36,3
104	4020	3906,3	113,8
105	3916	3906,3	9,8
106	3855	3906,3	-51,3
107	3867	3906,3	-39,3
108	3835	3906,3	-71,3
109	4032	3906,3	125,8
110	4055	3906,3	148,8
111	4033	3906,3	126,8
112	3915	3906,3	8,8
113	3879	3906,3	-27,3
114	3952	3906,3	45,8
115	3795	3906,3	-111,3
116	3831	3906,3	-75,3
117	3895	3906,3	-11,3
118	3897	3906,3	-9,3
119	3871	3906,3	-35,3
120	3815	3906,3	-91,3
121	3877	3906,3	-29,3
122	3884	3906,3	-22,3
123	3899	3906,3	-7,3
124	3755	3906,3	-151,3
125	3916	3906,3	9,8
126	3880	3906,3	-26,3
127	3871	3906,3	-35,3
128	3941	3906,3	34,8
129	3871	3906,3	-35,3
130	3825	3906,3	-81,3
131	3864	3906,3	-42,3
132	3887	3906,3	-19,3
133	3908	3906,3	1,8
134	3902	3906,3	-4,3
135	3957	3906,3	50,8
136	3898	3906,3	-8,3
137	3898	3906,3	-8,3
138	3931	3906,3	24,8

	N observado	N esperado	Residual
139	3957	3906,3	50,8
140	3837	3906,3	-69,3
141	4021	3906,3	114,8
142	3809	3906,3	-97,3
143	3874	3906,3	-32,3
144	3879	3906,3	-27,3
145	3851	3906,3	-55,3
146	3884	3906,3	-22,3
147	3956	3906,3	49,8
148	3938	3906,3	31,8
149	3919	3906,3	12,8
150	3920	3906,3	13,8
151	3974	3906,3	67,8
152	3927	3906,3	20,8
153	4046	3906,3	139,8
154	4013	3906,3	106,8
155	3898	3906,3	-8,3
156	3881	3906,3	-25,3
157	3948	3906,3	41,8
158	3972	3906,3	65,8
159	3943	3906,3	36,8
160	3863	3906,3	-43,3
161	3775	3906,3	-131,3
162	4034	3906,3	127,8
163	3894	3906,3	-12,3
164	3834	3906,3	-72,3
165	3977	3906,3	70,8
166	4037	3906,3	130,8
167	3883	3906,3	-23,3
168	3912	3906,3	5,8
169	3896	3906,3	-10,3
170	3924	3906,3	17,8
171	3917	3906,3	10,8
172	3892	3906,3	-14,3
173	3857	3906,3	-49,3
174	3924	3906,3	17,8
175	3828	3906,3	-78,3
176	3827	3906,3	-79,3
177	3905	3906,3	-1,3
178	3888	3906,3	-18,3
179	3981	3906,3	74,8
180	3922	3906,3	15,8
181	3817	3906,3	-89,3
182	3904	3906,3	-2,3
183	3866	3906,3	-40,3
184	3853	3906,3	-53,3
185	3922	3906,3	15,8
186	3923	3906,3	16,8
187	3945	3906,3	38,8

	N observado	N esperado	Residual
188	3977	3906,3	70,8
189	3920	3906,3	13,8
190	3940	3906,3	33,8
191	3962	3906,3	55,8
192	3889	3906,3	-17,3
193	3917	3906,3	10,8
194	3946	3906,3	39,8
195	3864	3906,3	-42,3
196	3931	3906,3	24,8
197	3957	3906,3	50,8
198	3908	3906,3	1,8
199	3931	3906,3	24,8
200	3992	3906,3	85,8
201	3966	3906,3	59,8
202	3838	3906,3	-68,3
203	3979	3906,3	72,8
204	3926	3906,3	19,8
205	3925	3906,3	18,8
206	3901	3906,3	-5,3
207	3929	3906,3	22,8
208	3907	3906,3	,8
209	3902	3906,3	-4,3
210	3852	3906,3	-54,3
211	3887	3906,3	-19,3
212	3895	3906,3	-11,3
213	3913	3906,3	6,8
214	4007	3906,3	100,8
215	3886	3906,3	-20,3
216	3845	3906,3	-61,3
217	3898	3906,3	-8,3
218	4014	3906,3	107,8
219	3824	3906,3	-82,3
220	3876	3906,3	-30,3
221	3860	3906,3	-46,3
222	3811	3906,3	-95,3
223	3962	3906,3	55,8
224	3995	3906,3	88,8
225	3902	3906,3	-4,3
226	3834	3906,3	-72,3
227	3920	3906,3	13,8
228	3856	3906,3	-50,3
229	3948	3906,3	41,8
230	3893	3906,3	-13,3
231	3950	3906,3	43,8
232	3923	3906,3	16,8
233	3934	3906,3	27,8
234	3858	3906,3	-48,3
235	3937	3906,3	30,8
236	3903	3906,3	-3,3

	N observado	N esperado	Residual
237	3904	3906,3	-2,3
238	3945	3906,3	38,8
239	3879	3906,3	-27,3
240	3891	3906,3	-15,3
241	3883	3906,3	-23,3
242	3893	3906,3	-13,3
243	3993	3906,3	86,8
244	3896	3906,3	-10,3
245	3944	3906,3	37,8
246	3889	3906,3	-17,3
247	3922	3906,3	15,8
248	3934	3906,3	27,8
249	3939	3906,3	32,8
250	3867	3906,3	-39,3
251	3796	3906,3	-110,3
252	3849	3906,3	-57,3
253	3940	3906,3	33,8
254	3912	3906,3	5,8
255	3888	3906,3	-18,3
Total	1000000		

Resultat Chi-Cuadrat

	V1
Chi-cuadrado(a)	215,839
gl	255
Sig. asintót.	,964

a 0 casillas (,0%) tienen frecuencias esperadas menores que 5. La frecuencia de casilla esperada mínima es 3906,3.

A continuació indiquem els resultats obtinguts de l'estadístic run-test:

Runs Test: Kiss

```
   Kiss

   K =   127,5413


   The observed number of runs =****
   The expected number of runs =5,000E+05
5E+05 Observations above K5E+05 below
            The test is significant at  0,2606
            Cannot reject at alpha = 0,05
```

En ambdós casos, em obtingut un nivell de significació per sobre de 0,05 i per sota de 0,95, això vol dir que no podem rebutjar la hipòtesi nul·la i acceptem que la mostra obtinguda te un comportament que ens porta a pensar que pot ser considerat aleatori.

3.6.1.3. Característiques del KISS.

A continuació descriurem les propietats del generador utilitzat en el nostre programa, així com la seva estructura.

El generador KISS presenta un període aproximat de 2^{123} –1, combina dos generadors de multiplicació amb arrossegament que es gestionen dintre d'un altre generador anomenat MWC amb els tres registres de desplaçament SHR3 i el generador congruencial CONG. que fa servir la suma el XOR. A continuació descrivim les característiques de tots aquests components:

- El generador MWC, concatena dos generadors de multiplicació amb arrossegament de 16 bits, que tenen la següent equació x(n)=36969x(n-1)+carry, i y(n)=18000y(n-1)+carry mod 2^{16} respectivament, el MWC presenta un període de 2^{60}-1.
- El SHR3, es un generador amb tres registres de desplaçament que te la següent forma y(n)=y(n-1)(I+L^{17})(I+R^{13})(I+L^{5}), on les y representen vectors binaris, la L es una matriu binària de 32x32 que desplaça el vector a l'esquerra i R es la seva transposada. Aquest generador un període de 2^{32}-1.
- El CONG, que es un generador congruencial amb equació: x(n)=69069x(n-1)+1234567. Presenta un període 2^{32}-1.
- El LFIB4, es un generador dels anomenats generadors retardats de Finobacci, i presenta la següent forma: x(n)=x(n-256)+x(n-179)+x(n-119)+x(n-55) mod2^{32}. Té un període de 2^{31}*(2^{256}-1), aproximadament 2^{287}.
- El SWB, es un generador de resta amb arrossegament i la seva equació té aquesta forma: x(n)=x(n-222)-x(n-237)- arrossegament mod 2^{32}. On arrossegament valdrà 0 excepte quan hi hagi un 'overflow' al calcular x(n-1), el seu període es molt llarg, concretament 2^{7098}*(2^{480}-1), aproximadament, 2^{7578}.

Aquest generador a diferència dels generadors convencionals s'inicialitza mitjançant 4 llavors, z, w, jsr i jcong. La forma en que nosaltres inicialitzem aquestes llavors ja s'ha vist en l'apartat del tractament de la contrasenya d'usuari.

Per a finalitzar, el nostre programa utilitza una combinació del KISS+SWB, amb aquesta combinació s'arriba a obtenir un període superior de 2^{7700}. Utilitzar un període d'aquest tamany, garanteix que en el procés d'encriptació no es reiniciarà el cicle del RNG, encara que el fitxer sigui molt gran i en conseqüència es més robust a atacs per força bruta.

3.6.1.4. Fonament d'ús del KISS.

Hi ha varies raons que ens permeten justificar el perquè ens hem decantat per l'ús del KISS enlloc d'un altre. Malgrat no es l'únic generador de números pseudo-aleatoris que ofereix un bon comportament des del punt de vista de l'aleatorietat.

Concretament el motiu d'implementar-lo respon al fet de que el seu dissenyador es el propi George Marsaglia, i això en garanteix el comportament. Per altra banda, ofereix un període d'iteracions prou llarg com per a cobrir els objectius del nostre programari, i finalment, presenta una singularitat, i es el fet d'utilitzar 4 valors diferents per inicialitzar-se, això fa que per a la nostra aplicació resulti molt útil, ja que indirectament ens permet reforçar l'entropia pròpia de la funció SHA1 (veure l'apartat de fragmentació de la contrasenya). Com sempre tots els objectius miren cap a reforçar la robustesa del xifrat, i per tant de la protecció dels fitxers encriptats.

3.6.2. Llibreria d'operacions a nivell de bit.

El darrer apartat de l'aplicatiu que entenem cal comentar per la seva rellevància es la creació d'una llibreria pròpia per a manipular bits.

Com a conseqüència de l'ús del Kiss en el nostre programa, s'ha detectat que les funcions que manipulen bits en la plataforma Java utilitzada per el nostre desenvolupament, generava de forma esporàdica problemes relacionats amb 'overflows' no esperats, per exemple inicialment totes les operacions matemàtiques relacionades directament amb el generador de números pseudo-aleatoris es manipulaven amb variables de tipus Long per tal d'evitar els citats desbordaments, la sorpresa era que tot i així es produïen errors inesperats. Després d'analitzar el codi moltíssimes vegades, es va descobrir que en aquest entorn de desenvolupament quan un Long rep un valor de tipus int on el bit de mes a l'esquerra es 1, automàticament omple amb '1' tots els bits que en el tipus Long haurien de rebre valor 0, de tal manera, que si el que es pretenia era fer una suma sobre un int, si aquest presentava la característica que s'ha descrit, es generava el mateix tipus de desbordament que es volia evitar, de fet, el desbordament era pitjor perquè era totalment incomprensible.

Com anècdota, per a detectar-ho va ser necessari 'debugar' pas a pas les funcions que presentaven anomalies i avaluar les variables que retornaven valors estranys a nivell de bit, i comparar strings de 64 bits no es una feina massa agradable!... amb la qual cosa ens podem imaginar que això va representar una cost de temps prou important, i per sobre de la previsió inicial.

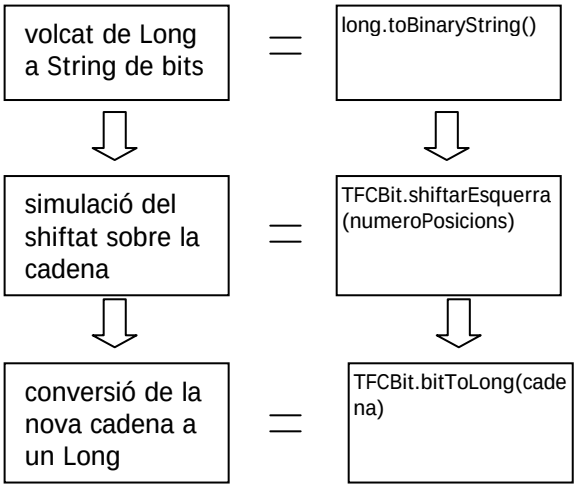
Per altra banda, la plataforma Java no ofereix tantes funcions a nivell de bit com el llenguatge C, originalment, l'algorisme KISS està implementat en C i en forma de macros (sota comandes '#define'), això ha produït que s'hagin tingut que crear totes les funcions de manipulació de bits que no estan presents en aquesta plataforma i que si hi eren presentaven comportaments irregulars com els que s'ha comentat en el paràgraf anterior, es a dir, desbordaments inesperats i per tant, valors incorrectes.

Així doncs, es va decidir crear una nova llibreria (classe) que dones suport a totes les operacions que presentaven defectes, implementant les següents funcions:

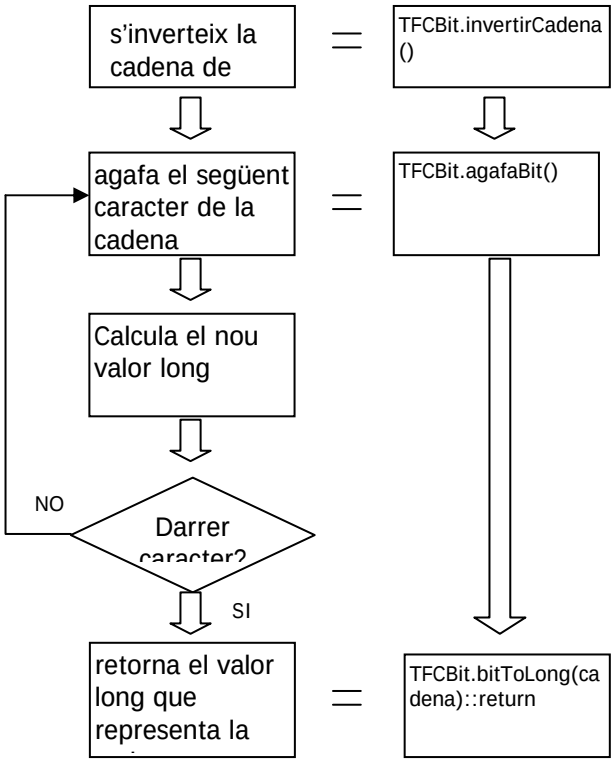
- funcions de conversió de tipus de variables, p.e. de bit a int (bitToInt), bit a long (bitToLong), etc.
- funcions de truncament de variables, p.e. truncar un valor a 8 bits (truncar8), truncar un valor a 32 bits (truncar32), etc.
- funcions de rotacions de bits, p.e. shiftar bits a la dreta (shiftarDreta).
- funcions d'operació a nivell de bit, p.e. XOR (xor).
- funcions de manipulació de bits, p.e. afegir 0 a una variable (afegirZeroEsquerra), invertir un bit concret d'una variable (invertirBitCadena).

Per evitar que en les nostres funcions es produís el mateix efecte observat al utilitzar les variables natives del Java, es va decidir fer sempre una transformació a cadena, es a dir, qualsevol manipulació d'una variable a nivell de bit passa pel següent esquema, com exemple hem utilitzat la transformació que pateix una variable per a ser shiftada:

Long(TFCBit.bitToLong(TFCBit.shiftarEsquerra(lAux1.toBinaryString(lAux1.intValue()),16)));



A més hi ha funcions intermitges que no hem inclòs en aquest esquema, per exemple la funció TFCBitToLong(), inclou subfuncions que es criden tantes vegades com llarga es la cadena que es vol transformar:



Això pot donar una idea del cost computacional que representa, shiftar un long implica recórrer un String de 64 posicions i es una operació que es realitza un munt de vegades per a cada nou valor que es demana del PRNG, i el pitjor, no es l'única, aquest es un dels motius que com veurem més endavant va fer que ens qüestionéssim l'ús d'aquest entorn per a desenvolupar l'aplicatiu.

4. Manual d'instal·lació.

Al tractar-se d'un desenvolupament realitzat íntegrament en Java necessitem d'una màquina virtual de Java per poder utilitzar l'aplicació, per tant, haurem de seguir els passos que es descriuen a continuació, en funció de si disposem d'una màquina virtual Java instal·lada en el nostre maquinari o si no en disposem:

4.1. Manual d'instal·lació. Disposem d'una màquina virtual en el nostre maquinari.

Si el nostre maquinari ja té instal·lada una màquina virtual Java 1.2 o superior, només haurem d'ubicar els .class de tots els mòduls en el directori que desitgem i executar des d'una consola de MS-DOS per a Windows o des d'una shell per a Linux, la comanda 'java TFCPantallaPrincipal00' que es la classe principal del nostre programari. Això hauria de produir que s'obris una interfície d'usuari com s'indica en l'apartat 2.4.1.

4.2. Manual d'instal·lació. Hem d'instal·lar una màquina virtual en el nostre maquinari.

Si el nostre maquinari no té instal·lada una màquina virtual Java 1.2 o superior, o disposem d'una versió massa antiga com per exemple jview, haurem d'actualitzar-la donat que el nostre programari utilitza les classes Java Swing per activar la interfície gràfica.

Per a la instal·lació de la màquina virtual java, recomanem entrar en la següent web i descarregar-se la màquina virtual adient segons el nostre S.O. *http://www.java.com/es/download/help/5000010500.xml#download* . En aquesta mateixa web, trobarem les instruccions instal·lació, posteriorment només haurem de seguir els passos indicats en l'apartat anterior. Recordem que JRE (Java Runtime Environment) és l'entorn mínim per a executar programes Java 2. Inclou la JVM (màquina virtual Java) i la API. Està inclosa en el J2SE encara que es pot descarregar i instal·lar per separat. Si només volem executar programes Java i no els volem compilar el JRE es suficient

5. Joc de Proves.

En aquest apartat veurem quines proves s'han realitzat per a determinar que el programa funciona correctament.

Tot i que cada classe disposa de la seva pròpia funció 'main' que li permet utilitzar de forma independent i per tant provar-la de manera aïllada, s'han generat dos jocs de proves independents, un per a validar el procés de xifrat-desxifrat, i l'altre per avaluar el generador de números pseudo-aleatoris, que inclou la validació de la classe TFCBit.

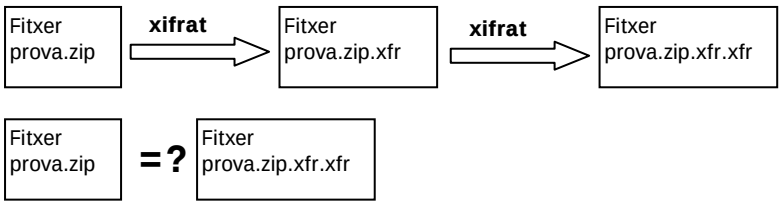
5.1. Joc de Proves. Validació del procés de xifrat - desxifrat.

El mètode triat per a realitzar aquesta part, consisteix en encriptar un fitxer de format .pdf i un fitxer amb format .zip, l'elecció d'aquests tipus de fitxer no es atzarosa, respon a que ambdós tipus de fitxers inclouen validacions similars a les incloses en el nostre programari per a validar la congruència dels fitxers, es a dir, que no estan corruptes. La prova s'ha repetit en dos tipus de fitxer, un fitxer amb un pes mig (aproximadament 100 Kbytes), i un altre d'un pes prou important (>2Mb.).

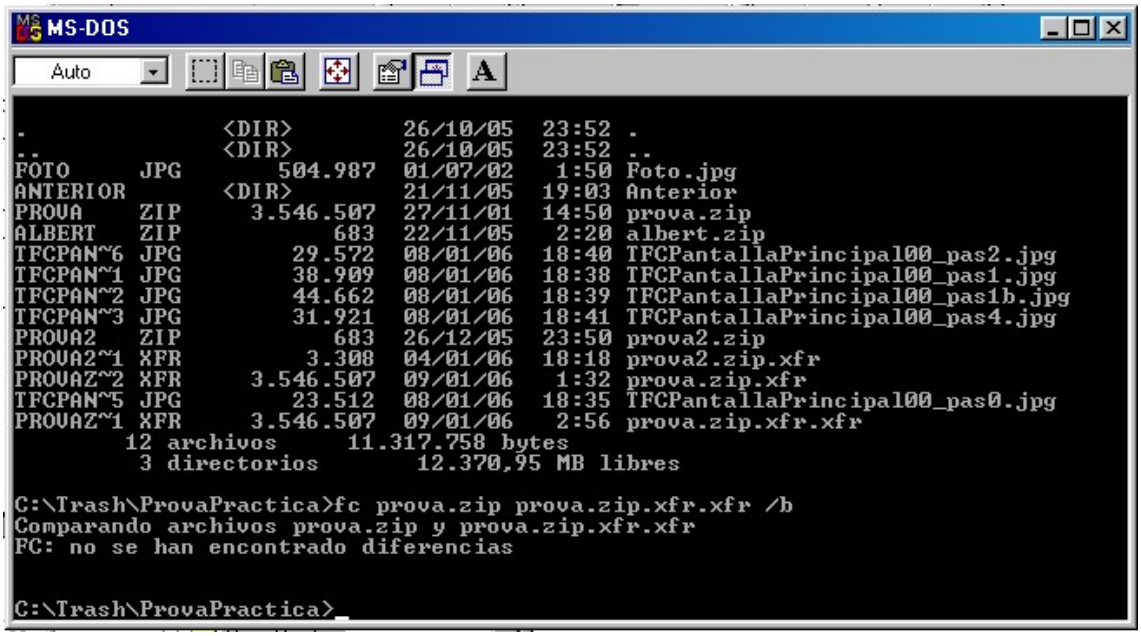
En qualsevol dels casos el joc de proves consisteix en primer lloc en aplicar dues vegades el procés d'encriptació sobre el mateix fitxer, utilitzant el mateix password. Val a dir que el procés que s'utilitzarà, serà l'execució reduïda del nucli de l'aplicatiu que s'encarrega exclusivament del xifratge, es a dir, en aquest joc de proves, no s'inclouen camps de control addicionals, ja que alterarien el resultat, i ens portarien a conclusions errònies.

El motiu pel que repetim el procés de xifratge dues vegades, es perquè el xifrat del xifrat dóna el mateix text en clar, donat que es realitza l'enmascarement de cada byte utilitzant l'operació lògica XOR, amb el respectiu byte aleatori, obtingut d'un algorisme determinista, tal i com ja s'havia explicat en anteriorment.

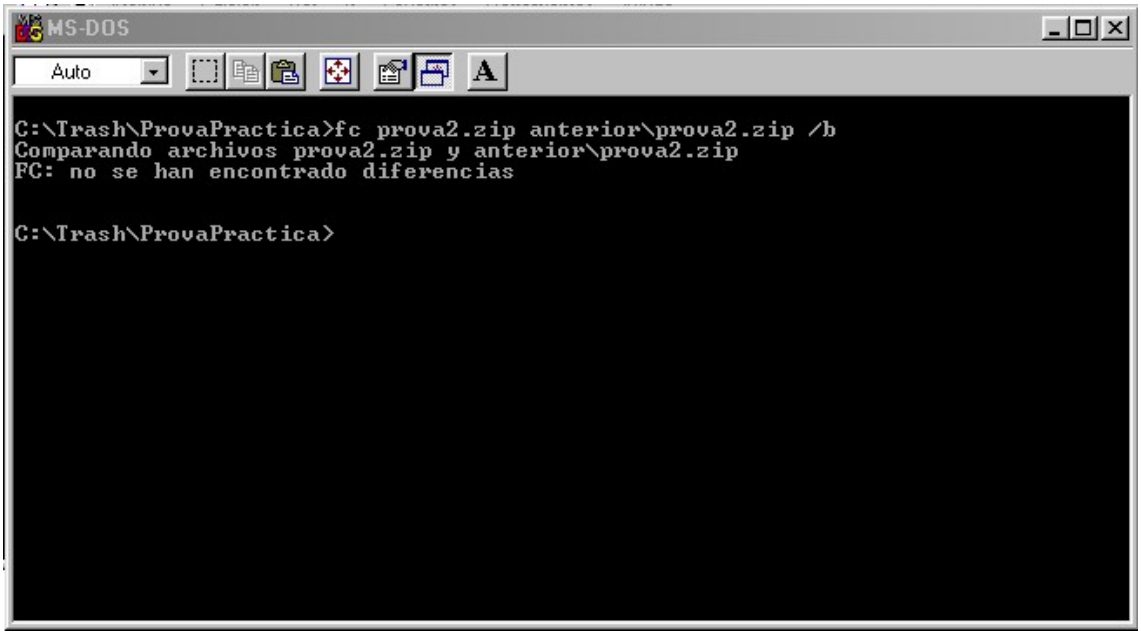
Finalment, un cop tenim els dos fitxers, l'original i la imatge (obtingut del segon cicle de xifrat), els compararem a nivell de bit, i per fer-ho utilitzarem la comanda FC a nivell de bit del MSDOS. Val a dir que aquesta verificació NO es pot fer mitjançant una comparació ASCII doncs no tindríem una certesa absoluta sobre el resultat. El procés que seguim s'esquematitza a continuació:



A continuació mostrem els resultats de la comparació en forma de captura de pantalla:



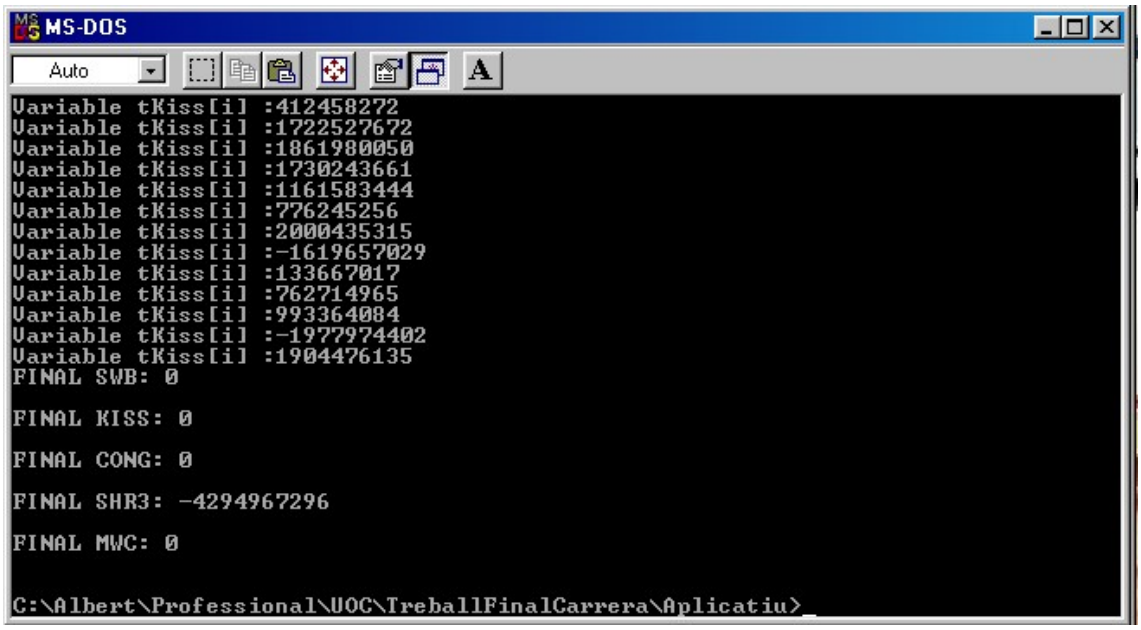
La segona part del joc de proves d'aquest mateix apartat, consisteix en verificar que el programari es capaç de xifrar un fitxer en clar i posteriorment desxifrar obtenint el fitxer original, aquest cop hem tornat a utilitzar els mateixos fitxers utilitzats en la prova anterior, i finalment un cop finalitzat el procés tornem a realitzar una comparació amb la comanda FC de MS-DOS, com sempre a nivell de bit, el resultat el mostrem en la següent captura:



5.2. Joc de Proves. Validació del KISS.

El generador de números pseudo-aleatoris utilitzat el KISS, inclou un joc de proves per a validar que la seqüència de valors es l'esperada. Aquest va ser el motiu pel que es va detectar les

anomalies amb el tractament a nivell de bits que ja hem comentat. Les funcions semblaven que estiguessin perfectament ben implementades, però no aconseguíem la mateixa seqüència que el programa original. Fins i tot es va realitzar la implementació en C per a comparar pas a pas quin era el problema. Un cop solventat el mateix, es pot aplicar el test que incorpora la classe 'TFCRngKiss.class' i que s'arrenca des del 'main'. El test, consisteix en generar 1000000 de valors de la seqüència amb una incialització de les llavors concretes. Al finalitzat el procés tots els generadors involucrats en el KISS ofereixen un resultat 0, adjuntem una captura de pantalla per mostrar que el generador implementat compleix perfectament aquesta premissa.



En la imatge veiem que el valor corresponent a SHR3 no es 0, però respon a un problema de representació, en primer lloc s'utilitza un variable sobredimensionada on s'ha incrementat el darrer bit (bit de més a l'esquerra) i el Java no permet fer representacions de variables com 'unsigned' (-4294967296 = 0001 0000 0000 0000 0000 0000 0000 0000 0000).

6. Comentaris i conclusions.

El projecte a estat interessant, però hem tingut masses problemes alhora de realitzar la implementació. Ens vàrem decantar per utilitzar la plataforma Java per que era una de les recomanades, però després d'haver sofert tots els problemes de la implementació estem convençuts de que d'haver utilitzat Delphi (que era l'altre opció que vàrem valorar), ens haguéssim estalviat molts dels problemes trobats.

Concretament en Java resulta extremadament complicat moure un entorn gràfic. S'ha d'implementar tot el codi per tal d'obrir i tancar finestres i sobretot manipular tots els events, certament no hi teníem massa experiència amb l'ús de les llibreries Swing, però m'ha decebut molt. No es raonable que una plataforma tan estesa i tant reconeguda, presenti tantes dificultats per a desenvolupar un entorn gràfic bàsic.

Fent un punt i a part, com dèiem el projecte a estat interessant, molt ben enfocat des del punt de vista dels objectius a assolir. Com sempre, el problema es el temps de dedicació, el semestre es fa curt i no es possible implementar tot allò que inicialment ens havíem proposat, per exemple, altres tipus de xifrat alternatius.

Per a finalitzar, sota el nostre parer, el projecte que hem desenvolupat cobreix els punts demanats, i com a conseqüència d'haver-nos recolzat en estàndards criptogràfics, creiem que els fitxers xifrats amb aquest programari ofereixen una alt grau de seguretat.

7. Bibliografia.

Llibres:
- CRC Press. *Criptography Theory and Practice*: Boca Raton.
- Josep Domingo Ferrer (2004). *Criptografia*: Universitat Oberta de Catalunya (UOC).
- Donald E. Knuth (1998). *Seminumerical algorithms - The Art of Computer programming VOL2*: Secon Edition - Addison Wesley.
- Murray R. Spiegel (1991). *Estadística*: McGraw-Hill.

eBooks:
- Sun: *The Java Tutorial*
- Bruce Eckel's *Thinking in Java, 2nd Edition*

Revistes:
- Revista @rroba, núm 90
- Revista PC Actual, núm 171

Web:
- http://stat.fsu.edu, Documentació sobre RNG (George Marsaglia)
- http://www.iro.umontreal.ca/~lecuyer/address.html, Documentació sobre RNG (Piere Lecuyer)
- http://webs.ono.com/usr005/jsuarez/disclaim.html, Codi Font RNG KISS.
- http://www.geocities.com/SiliconValley/Way/4302/java.html, Programació en Java.
- http://www.arrakis.es/~abelp/ApuntesJava/Introduccion.htm, Programació en Java.
- http://www.gamarod.com.ar/articulos/129.asp, Instal.lació de Java.
- http://www.java.com/es/download/help/5000010500.xml, Instruccions de descàrrega de Java.