XIFRADOR 1.0

Josep Maria Buyó Llunart
(ETIS)

Antoni Martínez Ballesté

17/01/2008

Dedicatòria:

A ella, sincerament, de tot cor, gracies.

Agraïments:

A la UOC.

A dissenyat un espai virtual d'aprenentatge que ens ha permès accedir als coneixements, als consells i a la guia dels docents.

Resum

Aquest TFC pretén investigar en el concepte de generadors de seqüencies pseudoaleatories amb la finalitat d'implementar un generador amb qualitats òptimes per al xifratge de dades.

També proporciona una aplicació per tal de fer proves amb aquest generador a l'ora que permet xifrar i desxifrar dades utilitzant les eines de la criptografia moderna.

Paraules clau

xifratge d'arxius, generadors de seqüencies pseudoaleatories, criptografia, seguretat informàtica. complexitat lineal, algoritme de Berlekamp-Massey, xifra de Vernam, Maurer, Menezes., infraestructura de clau pública, signatures digitals.

Àrea del TFC

SEGURETAT INFORMÀTICA

ÍNDEX GENERAL

ÍNDEX DE FIGURES I TAULES

1 Objectius del Projecte

La idea de realitzar aquest projecte sorgeix de l'experiència acumulada en les diferents assignatures relacionades amb la seguretat informàtica, especialment amb la de criptografia. allí vam estudiar, entre altres, les propietats dels generadors pseudoaleatoris, concretament la del generador de Massey-Rueppel.

Vaig considerar la idea d'utilitzar aquest generador per dur a terme la implementació d'una aplicació pel xifratge i posterior desxifrat d'informació que em servis per al treball de fi de carrera en l'àrea de seguretat informàtica.

Després d'aprofundir una mica més al mon dels generadors i comprovar que és un camp complex i què hi cal fer investigació, vaig entendre que les prestacions d'aquest generador no eren les més adequades per dur a terme el treball, no obstant li vaig donar voltes i sé em va acudir la idea d'implementar un algoritme que fos capaç de mesclar de forma pseudoaleatoria els bits que generessin 3 generadors de Massey_Rueppel.

El primer pas va ser implementar aquest algoritme i comprovar que el seu comportament era correcte si és donaven certes condicions. El següent era implementar el conjunt format pels 3 generadors i l'algoritme, a aquest conjunt l'anomenarem El Nostre Generador.

A tot això, per una banda, cal afegir-li la implementació d'una sèrie de test que serveixin per verificar el comportament de ElNostreGenerador i per l'altra, el disseny i la implementació de l'aplicació que utilitzi aquest generador.

l'objectiu final és aconseguir desenvolupar una aplicació que ens permeti experimentar amb el generador proposat, a l'ora que ens xifri i desxifri els arxius i carpetes que desitgem utilitzant les diferents tècniques de la criptografia moderna.

Serà una primera versió d'una aplicació que ha d'anar creixent amb les aportacions futures per tal de millorar tant les interfícies com qualsevol detall tècnic que afecti al propi generador.

1.1 Calendari

Figura 1. taula del calendari del projecte.

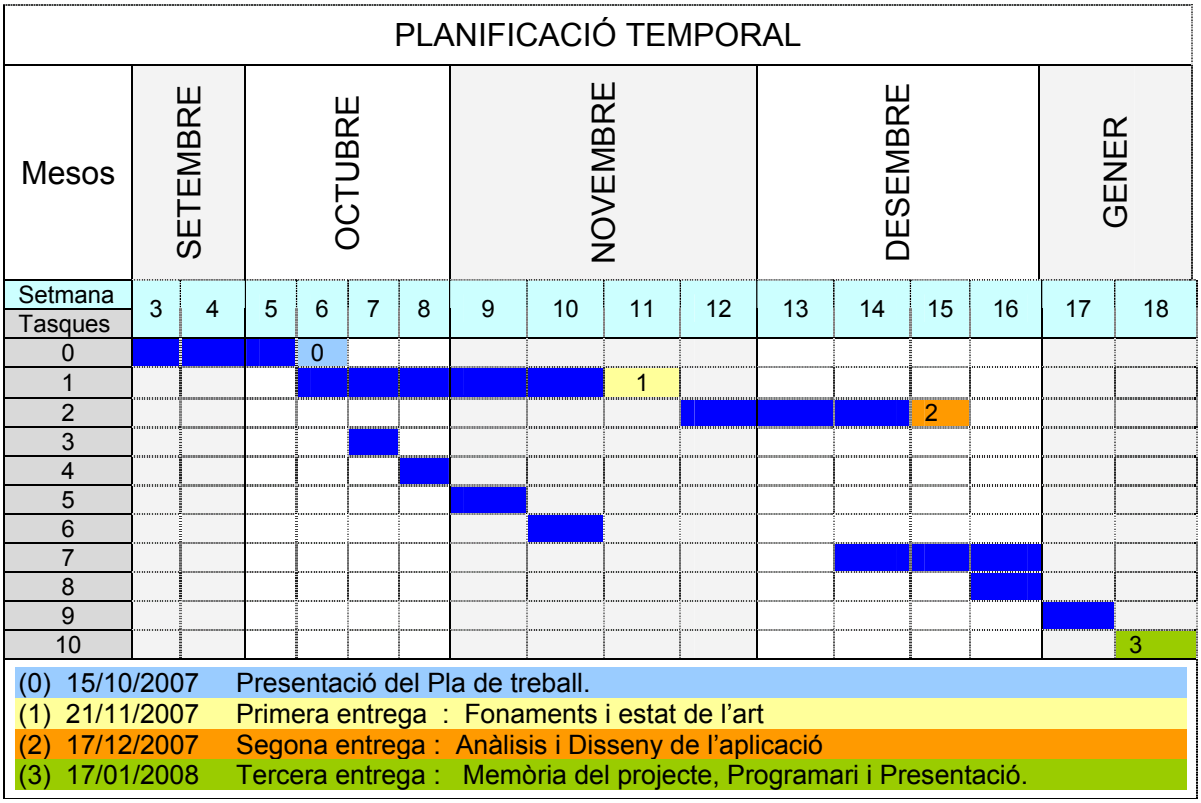


Figura 2. taula per les tasques principals a desenvolupar.

Tasques Principals	
0	Confeccionar el Pla de treball.
1	Desenvolupar els temes per l'apartat de Fonaments i estat de l'art
2	Anàlisis i Disseny de l'aplicació
3 4 5 6	Cercar informació per dur a terme les diferents tasques.
7	Efectuar les corresponents proves del generador implementat amb els diferents tests estadístics. Proves de programari.
8	Elaboració de la memòria
9	Confeccionar presentació.
10	Lliurament final:

1.2 Eines que farem servir

Les principals eines que és proposen fer servir son:

- Ordinador amb Processador Intel Pentium 4 CPU 2,53GHz.
- Plataforma Windows Xp.
- Llenguatge de programació Java.
- Llibreries i programari :
    - jdk1.5.0_13.
    - Llibreria criptogràfica de bouncycastle versió 1.37, API i exemples.
    - Generador de certificats i claus Win32OpenSSL.
    - Eina Keytool de Java per la generació de certificats.
    - Entorn de desenvolupament IDE Eclipse versió 3.2.
    - Entorn StarUML per a la realització dels diferents diagrames.
    - Entorn Ofimatic de Microsoft (Word, Excel, PowerPoint).
    - Entorn Adobe per a generar documents pdf.
    - MINITAB 13 per verificar resultats de les anàlisis estadístiques
    - Scientific Notebook per a determinats càlculs.

2 Fonaments i estat de l'art

2.1 Fonaments.

2.1.1 Fonaments de Criptografia. [uoc1]

Criptografia és un terme d'origen grec que prové dels mots *krypto* ('amagar') i *grapho* ('escriure'). Podem dir que la **criptografia** és la ciència i l'estudi de l'escriptura secreta.

Inicialment, la criptografia va aparèixer per a resoldre la necessitat de comunicar-se en presència d'un adversari (normalment en un context militar o diplomàtic). Actualment, engloba altres problemes; per citar-ne només uns quants, podem parlar de xifratge, autentificació, distribució de claus, etc.

La criptografia moderna proporciona els fonaments teòrics necessaris per a poder:

- Entendre exactament els problemes que acabem d'enumerar.
- Avaluar els protocols que en teoria poden resoldre aquests problemes.

- Construir protocols en la seguretat dels quals puguem confiar.

La criptografia i una disciplina complementària anomenada **criptoanàlisi** es coneixen conjuntament amb el nom de criptologia. La criptografia s'ocupa del disseny de xifres. La criptoanàlisi s'ocupa de trencar xifres. La motivació del criptoanalista pot ser l'interès intrínsec de descobrir el text en clar xifrat i/o la clau emprada, o bé ser de caire cientificotècnic (verificació de la seguretat de la xifra). El vessant cientificotècnic de la criptoanàlisi és essencial per a la depuració de les xifres i és molt útil per al progrés de la criptografia.

La criptografia com a ciència moderna.

La publicació l'any 1949 per part de C.E. Shannon de l'article "Communication Theory of Secrecy Systems" va inaugurar l'era de la criptologia científica de clau compartida. Shannon, que era enginyer i matemàtic, va elaborar una teoria dels sistemes secrets gairebé tan completa com la teoria de les comunicacions que havia publicat l'any anterior. Entre altres coses, l'article del 1949 demostrava la seguretat incondicional de la ***xifra de Vernam***. Però a diferència de l'article sobre comunicacions del 1948, que va fer néixer la teoria de la informació com a disciplina, l'article del 1949 no va suposar un impuls comparable per a la recerca criptogràfica.
L'eclosió real de la criptografia s'esdevingué amb la publicació l'any 1976 per part de W. Diffie i M.E. Hellman de l'article "New Directions in Cryptography".
Diffie i Hellman van mostrar per primer cop que era possible la comunicació secreta sense cap transferència de clau secreta entre l'emissor i el receptor, i van encetar així **l'època de la criptografia de clau pública** en la qual ens trobem actualment.

2.1.2 Fonaments de la teoria de la informació[uoc2]

Una **variable aleatòria** es pot definir informalment com una variable que adopta un valor entre un conjunt de valors possibles, de manera que cada valor possible té una certa probabilitat no nul·la de ser adoptat. Si el conjunt de valors que pot prendre la variable és discret, llavors es diu que la variable aleatòria és discreta. Així, doncs, tenim que:

- Un missatge *M* és vist pel criptoanalista com una variable aleatòria discreta que pot prendre com a valors uns textos concrets.
- Una clau *K* també és vista pel criptoanalista com una variable aleatòria discreta.

El criptoanalista voldria saber quin valor pren la variable *K*, però en principi només pot conèixer (a tot estirar) la distribució de probabilitats de la clau, és a dir, les probabilitats que té la clau d'adoptar cadascun dels valors possibles. Si la clau és aleatòria en el sentit de la **xifra de Vernam**, la probabilitat de cadascun dels valors possibles és la mateixa.

2.1.3 Secret Perfecte i Autenticitat perfecta [uoc2]

El teorema de la fita fonamental de Shannon per al secret perfecte, diu el següent:

" en un criptosistema perfectament secret, la incertesa de la clau secreta K ha de ser almenys tan gran com la incertesa del text en clar que pretén amagar".

La fita de Shannon ens diu que, en un criptosistema perfectament secret, la clau *K* requereix més bits que el text en clar *M*, és a dir, la clau secreta ha de ser més llarga que no el text que es vol xifrar.

La **xifra de Vernam** és l'única coneguda que ofereix un secret perfecte.

La criptografia pretén assegurar el secret i l'autenticitat dels missatges. Però, de fet, el secret i autenticitat són atributs independents. Si l'Anna comparteix una clau amb en Bernat i l'Anna rep un criptograma que es desxifra bé amb la clau compartida, pot estar segura que el criptograma va ser enviat per en Bernat? La resposta és no. G.J. Simmons va publicar el 1984 l'article "Authentication theory/coding theory", que presenta una teoria de l'autenticitat anàloga en molts aspectes a la teoria del secret publicada per Shannon el 1949.

Com Shannon, Simmons suposa que la clau de xifratge es fa servir un sol cop. A l'escenari suposat per Simmons el criptoanalista enemic es troba entre l'emissor i el receptor i genera un criptograma fraudulent Ĉ. Hi ha dos tipus d'atac:

- **Suplantació**: el criptoanalista genera un criptograma fraudulent Ĉ. L'atac té èxit si el receptor accepta Ĉ com a bo. Anomenem P_{sup} la probabilitat d'èxit d'un atac de suplantació.
- **Substitució**: el criptoanalista canvia un criptograma autèntic *C* enviat per l'emissor per un criptograma fraudulent . Aquesta mena d'atac té èxit si el

receptor accepta Ĉ com a bo i Ĉ ≠ *C*. Anomenem P_{sub} la probabilitat d'èxit d'un atac de substitució.

Suposant que el criptoanalista enemic triarà el tipus d'atac que li sigui més favorable, es defineix la **probabilitat d'engany**, P_e, com:

$$P_e = max(P_{sup}, P_{sub})$$

Anomenem #C el nombre de criptogrames c que tenen una probabilitat no nul·la d'aparèixer, és a dir, tals que P(C = c) ≠ 0; de manera anàloga, anomenem #M i #K el nombre de missatges en clar i de claus, respectivament, amb una probabilitat no nul·la d'aparèixer. Llavors per a cada clau k hi ha d'haver #M criptogrames amb probabilitat no nul·la: són els criptogrames resultants de xifrar amb k els #M missatges en clar que tenen una probabilitat no nul·la.

Per tant, si el criptoanalista enemic fa un atac de suplantació triant a l'atzar un dels #C criptogrames que tenen una probabilitat no nul·la, la seva probabilitat d'èxit serà:

$$P_{sub} \geq \frac{\#M}{\#C}$$

Com que $P_e \geq P_{sup}$, aquesta desigualtat mostra que no és possible aconseguir una protecció total contra l'engany. Una bona manera de lluitar contra l'engany és que el conjunt de criptogrames possibles sigui molt més gran que el conjunt de textos en clar possibles, és a dir, agafar #C ▷ #M.

Pel fet que la protecció total és impossible, Simmons va definir l'autenticitat perfecta com la màxima protecció possible contra l'engany. Cal admetre que aquesta definició presenta alguns "casos patològics", ja que quan #M = #C haurem de considerar perfectament autèntic un sistema , ja que $P_e = 1$.

La definició d'autenticitat perfecta es fa en termes d'informació mútua. Per a això necessitem conèixer el **teorema de la fita de Simmons**, que diu: si P_{sup} és la probabilitat de suplantació amb èxit, es compleix:

$$P_{sub} \geq 2^{-I(C,K)}$$

Per tant, un criptosistema té la propietat d'autenticitat perfecta si P_e pren el mínim valor possible, és a dir, si $P_e = 2^{-I(C,K)}$

2.1.5 criptoanàlisi elemental [uoc2]

2.1.5.1 Suposició de Kerckhoff

Una suposició quasi universal en criptografia és que el criptoanalista enemic té accés al criptograma. Gairebé tan universal és la suposició de Kerckhoff, formulada per l'holandès A. Kerckhoff (1835-1903), segons la qual la seguretat de la xifra ha de residir totalment en la clau secreta.

La **suposició de Kerckhoff** diu que tot el mecanisme de xifratge, excepte el valor de la clau secreta, és conegut pel criptoanalista enemic.

2.1.5.2 Redundància i distancia d'unicitat

Si tenim en compte que $H(M|C) \leq H(M,K|C) = H(K|C) + H(M|C,K) = H(K|C) \leq H(K)$ i la suposició de Kerckhoff es dedueix que pot emprar-se $H(K|C)$ per a mesurar el secret d'una xifra. La magnitud $H(K|C)$ s'anomena **ambigüitat de clau** i representa la incertesa que queda sobre la clau un cop es veu un criptograma. Si $H(K|C) = 0$, no hi ha incertesa i la xifra es pot trencar en teoria si disposem de prou recursos de càlcul. Normalment, $H(K|C)$ decreix a mesura que augmenta la longitud N del criptograma conegut C.

La distància d'unicitat d'una xifra és la menor longitud N de criptograma que fa que $H(K|C)$ sigui propera a zero. Altrament dit, és la quantitat de text xifrat que cal per a què la clau quedi determinada de manera única.

Shannon va anomenar idealment secrets aquells criptosistemes que, tot i no proporcionar un secret perfecte, són irrompibles perquè no donen prou informació per a determinar-ne la clau.

La majoria de criptosistemes són massa complexos per a trobar-ne la distància d'unicitat. Shannon va formular un model anomenat de xifra aleatòria que permet aproximar la distància d'unicitat en alguns casos. Vegem com funciona el model, però per a fer-ho necessitem algunes definicions prèvies.

Per a un llenguatge determinat, considerem el conjunt de tots els missatges de llargada N caràcters. La **taxa del llenguatge per a missatges X de longitud N,** r**,** es defineix com:

$$r = H(X)/N$$

és a dir, el nombre mig de bits d'informació per caràcter.

La **taxa absoluta d'un llenguatge**, R, es defineix com el nombre màxim de bits d'informació que poden ser codificats en cada caràcter suposant que totes les seqüències de caràcters són equiprobables. Si hi ha L caràcters al llenguatge, la taxa absoluta és:

$$R=\log_2 L$$

La **redundància d'un llenguatge** amb taxa r i taxa absoluta R es defineix com $D = R - r$.

Suposem que cada text en clar i cada missatge xifrat vénen d'un alfabet finit de L símbols. Llavors hi ha 2^{RN} possibles missatges de longitud N, on $R = \log_2 L$, que es poden dividir en un conjunt de 2^{rN} **missatges amb sentit** i un conjunt de $2^{RN} 2 - 2^{rN}$ **missatges sense sentit,** on r és la taxa del llenguatge.

Se suposa que tots els missatges amb sentit tenen la mateixa probabilitat d'aparició $1/2^{rN} = 2^{-rN}$, mentre que els missatges sense sentit tenen probabilitat zero Suposarem també que hi ha $2^{H(K)}$ claus, totes equiprobables, on H(K) és l'entropia de la clau (nombre de bits de la clau). La probabilitat de totes les claus k és: $P(K = k) = 1/2^{H(K)} = 2^{-H(K)}$.

Una **xifra aleatòria** és aquella en la qual, per a cada clau k i per a cada text xifrat c, el desxifratge $D_{k(c)}$ és vist pel criptoanalista com una variable aleatòria independent (d'altres $D_{k'}(c')$) i distribuïda uniformement sobre tots els 2^{RN} missatges, amb sentit o sense.

Considerem el text xifrat $c = E_{k(m)}$ per k i m donats. Hi ha un **desxifratge espuri** quan el xifratge sota una altra clau k' pot donar c; és a dir, $c = E_{k'}(m)$ per al mateix missatge m, o bé $c = E_{k'}(m')$ per a un altre missatge amb sentit m'. Un criptoanalista que intercepti c no podrà decidir si la clau correcta és k o bé k'.

Ara bé, per cada desxifratge correcte d'un text xifrat determinat hi ha $2^{H(K)} - 1$ claus restants, cadascuna de les quals té la mateixa probabilitat, q, de produir un desxifratge espuri, que és el nombre de missatges amb sentit dividit pel nombre de missatges possibles:

$$q = 2^{rN}/2^{RN} = 2^{(r-R)N} = 2^{-DN}$$

Si denotem per F el nombre esperat de desxifratges espuris utilitzant una clau d'entre les restants, tenim:

$$F = (2^{H(K)} - 1) \cdot q = (2^{H(K)} - 1) \cdot 2^{-DN} \approx 2^{H(K)-DN}.$$

A causa del decreixement ràpid que experimenta F quan N creix, es pren $\log_2 F = H(K) - DN = 0$ com el punt on el nombre de solucions falses és prou petit perquè la xifra es pugui trencar. Per tant, la distància d'unicitat, N, és a dir, la quantitat de text necessari per a trencar la xifra és la següent:

$$N = \frac{H(K)}{D}$$

Si el criptoanalista disposa de N caràcters i d'una capacitat il·limitada de càlcul, llavors podrà trobar l'única clau que pot produir els N caràcters xifrats.

Cal dir que si el text en clar no tingués redundància (és a dir, si consistís en una seqüència aleatòria de bits), la distància d'unicitat fóra infinita.

En la **xifra de Vernam,** per cada N el nombre de claus possibles és tan gran com el nombre de missatges possibles. Llavors, $H(K) = \log_2 (2^{RN}) = RN$ i tenim:

$$H(K) - DN = (R - D) N = rN \neq 0,$$

Això implica que la Xifra de Vernam és irrompible en teoria.

2.2 Xifres de clau compartida.

2.2.2.1 xifres de flux.

Les xifres de flux s'inclouen dins els anomenats criptosistemes de clau compartida. Els criptosistemes de clau compartida són aquells en els quals l'emissor i el receptor comparteixen una mateixa clau per a xifrar i desxifrar missatges.

Així, doncs, en aquests criptosistemes, la clau que s'utilitza per a xifrar el missatge és la mateixa que es fa servir per a desxifrar-lo; per tant, emprant sempre la mateixa clau, en qualsevol moment l'emissor pot passar a fer de receptor i a l'inrevés.

D'una manera esquemàtica, un criptosistema de flux es pot representar com mostra la figura següent:

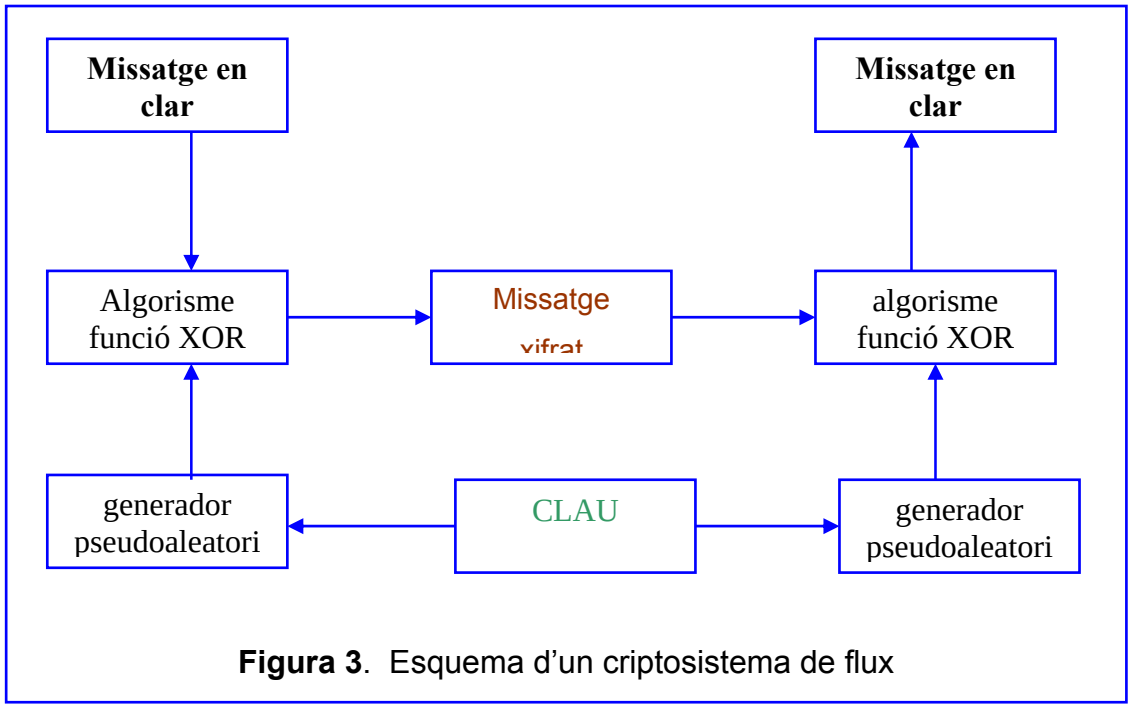


Figura 3. Esquema d'un criptosistema de flux

Tant l'emissor com el receptor disposen de la mateixa CLAU , anomenada **llavor del generador**, i d'un mateix algorisme determinista, anomenat **generador pseudoaleatori**. En proporcionar la clau com a entrada a l'algorisme, aquest genera en la sortida una seqüència anomenada **seqüència de xifratge.**

Per a xifrar el missatge, l'emissor va sumant cada bit del missatge en clar, amb cada bit de la seqüència de xifratge. Quan el receptor rep el missatge xifrat fa servir el mateix algorisme determinista, que en la figura anterior hem denotat com *generador pseudoaleatori* i *clau* , que comparteix amb l'emissor, per a obtenir la mateixa seqüència de xifratge. Així, sumant bit a bit el missatge que li arriba amb la seqüència resultant de *l'algorisme funció Xor*, el receptor obté el *missatge en clar* enviat per l'emissor.

Perquè aquest criptosistema sigui segur, és fonamental que la seqüència de xifratge no sigui coneguda; és a dir, que no es pugui saber en cap moment quin serà el bit de sortida següent. Idealment, el que es necessita per a garantir la seguretat incondicional és que la clau, en aquest cas la seqüència de xifratge, sigui completament aleatòria.

En el nostre esquema no es pot donar aquesta condició, ja que el generador que utilitzem ha de ser determinista per a què l'emissor i el receptor obtinguin la mateixa seqüència quan donen com a entrada la mateixa clau secreta.

Així, doncs, la seqüència de xifratge tindrà propietats molt properes a les que té una seqüència completament aleatòria i s'anomenarà **seqüència pseudoaleatòria.**

2.2.1.1 Generadors

La finalitat d'un generador és la de generar seqüències pseudoaleatoris que siguin el més aleatòries possibles.
Per aconseguir això hi ha una infinitat d'algorismes i processos de tota mena. Nosaltres farem una distinció entre els generadors lineals i no lineals i detallarem el funcionament del LFSR i del Generador de Massey-Rueppel.

2.2.1.1.1 Generadors lineals [uoc3]

Els generadors lineals són aquells que només executen operacions lineals sobre els elements d'entrada per a obtenir la seqüència de sortida.

2.2.1.1.1.a Generadors congruents

Els generadors congruents es basen en equacions modulars recurrents del tipus: $x_n = (ax_{n-1} + b) \bmod m$.

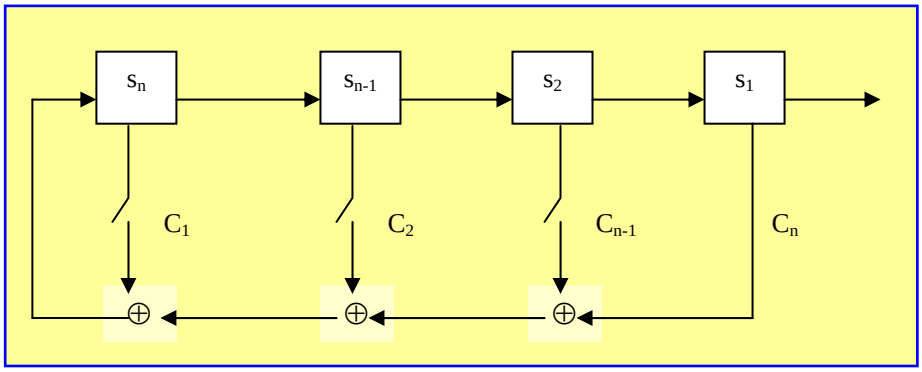
En aquest cas, el valor x_0 seria la llavor de la seqüència de xifratge. Un criptosistema que empri un generador d'aquest tipus ha de tenir com a clau secreta els valors $\{x_0, a, b, m\}$, i perquè el període sigui màxim s'ha de complir que $mcd(a,m) = 1$.
Val a dir, però, que aquests tipus de generadors pseudoaleatoris no són segurs des d'un punt de vista criptogràfic, ja que s'ha pogut demostrar que amb pocs valors x_i coneguts ja es poden esbrinar els paràmetres secrets $\{x_0, a, b, m\}$. Fins i tot coneixent només una part dels bits que formen els x_i (però, això sí, coneixent els paràmetres $\{a, b, m\}$) es pot arribar a determinar el valor de la llavor x_0.
Malgrat això, aquests tipus de generadors són molt utilitzats en sistemes informàtics per a aplicacions no criptogràfiques.

2.2.2.1.1.b LFSR

LFSR vol dir: Registre de desplaçament realimentat linealment.

Un LFSR, és un dispositiu físic o lògic format per n cel·les de memòria i n portes lògiques, tal com mostra la figura següent:

Figura 4. Esquema general d'un LFSR



Inicialment, les cel·les contenen els valors d'entrada, i a cada impuls de rellotge el contingut de la cel·la s_i es desplaça a la cel·la s_{i-1} fent les operacions associades. D'aquesta manera es genera un nou element, s_{n+1}, que és determinat per:

$$s_{n+1} = c_1 s_n + \ldots + c_n s_1,$$

en què els $c_i \in \{0, 1\}$ corresponen als valors de les portes lògiques de l'esquema. És a dir, els coeficients seran 1 si hi ha una connexió, i 0 si no n'hi ha. Aquest nou element, s_{n+1}, se situa a la cel·la s_n, que ha quedat buida a causa del desplaçament. El conjunt de valors continguts en cada cel·la en un instant de temps s'anomena estat. L'estat **inicial** és l'estat en què es troba l'LFSR en el moment de començar el procés.

El **polinomi de connexions d'un LFSR** de longitud n és el polinomi de grau n, que té la forma següent:

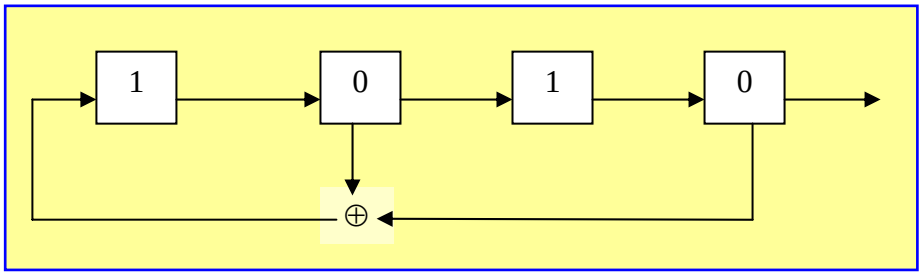
$$P(x) = 1 + c_1 x^1 + c_2 x^2 + \ldots + c_n x^n$$

Un LFSR es determina pel seu polinomi de connexions, i una seqüència, pel polinomi de connexions i per l'estat inicial.

Funcionament d'un LFSR:

Donat el següent LFSR de 4 celes amb polinomi de connexions $P= 1+X^2+X^4$, tenim:

Figura 5. Estat inicial d'un LFSR de quatre cel·les



Podem observar que aquest registre de desplaçament té com a estat inicial 1010, que correspon a l'impuls de rellotge t=0. L'evolució de l'estat en cada cicle del processador es mostra a la taula següent:

Impuls de rellotge (t)	Estat				Sortida (st_n)
		X^2		X^4	
	s_n	s_{n-1}	s_2	s_1	
	c_1	c_2	c_{n-1}	c_n	
(inici) 0	1	0	1	0	0
1	0	1	0	1	1
2	0	0	1	0	0
3	0	0	0	1	1
4	1	0	0	0	0
5	0	1	0	0	0
(període) 6	1	0	1	0	0
7	0	1	0	1	1

Figura 6. Taula d'estats d'un LFSR de quatre cel·les

És a dir, la sortida sempre val el contingut de la cel·la s_1, en l'estat t_0 , només cal fer: sortida $t_0 = s_1 = 0$.

Per la resta d'estats (t_n)fem:

1. Operació XOR –entre les cel·les **actives**, definides pels coeficients de les incògnites del polinomi– de l'estat t_{n-1}.
2. Desplaçar les cel·les a la dreta => $s_1=s_2$, $s_2=s_{n-1}$, $s_{n-1}= s_n$ i s_n = resultat de l'operació XOR feta a (1).
3. Assignar el valor de sortida, sortida $t_n = s_1=1$,

Podem observar que a t_6 tornem a tenir l'estat inicial, així doncs, tenim una repetició de la seqüència, per tant, aquesta seqüència té període = 6.

Cal fer notar que l'estat inicial d'un LFSR no pot ser tot de zeros. Si fos així, la seqüència que produiria seria també de zeros, ja que totes les operacions són lineals. Es diu que l'estat que tan sols té zeros és un **estat absorbent.** També convé destacar que el període màxim d'un LFSR és 2^n-1, i que s'obté de considerar tots els estats possibles 2^n i eliminar-ne l'estat nul.

Les característiques d'un LFSR d'acord amb les del seu polinomi de connexions son:

- **Polinomi de connexions factoritzable**: els LFSR que tenen com a polinomis de connexions polinomis factoritzables generen seqüències que depenen de l'estat inicial. A més, el període d'aquestes seqüències és sempre més curt que el període màxim que pot tenir un LFSR.
- **Polinomi de connexions irreductible**: les seqüències generades pels LFSR que tenen polinomis de connexions irreductibles no depenen de l'estat inicial, sinó que simplement queden desplaçades. En aquest cas, el període serà un divisor de 2^n - 1.
- **Polinomi de connexions primitiu:** un LFSR amb polinomi de connexions, primitiu, té la seqüència de sortida de període màxim, 2^n - 1. Aquesta seqüència de període màxim s'obté per a qualsevol estat inicial, llevat de l'estat absorbent.

Considerant les propietats de les seqüències segons els seus polinomis de connexions, veiem que per a esquemes de xifratge de flux és aconsellable fer servir la que determina el període màxim. A més:

- Les seqüències generades per un LFSR amb un polinomi de connexió primitiu compleixen els tres postulats de Golomb, gràcies a la longitud màxima del seu període i al fet que han de passar per tots els estats possibles

2.2.1.1.1.c Limitacions dels generadors lineals

Un inconvenient d'aquests generadors és que, perquè el període 2^n-1 sigui llarg, cal que la longitud de l'LFSR també sigui llarga.

Això pot representar un problema, ja que el cost de trobar polinomis primitius de grau alt és força elevat.

Malgrat els avantatges i inconvenients dels generadors lineals, la raó principal per la qual no serveixen per a implementar sistemes de xifratge de flux és que són fàcilment predictibles.

Per tant, a l'hora d'utilitzar un generador per a un procés de xifratge de flux cal que ens fixem també en la predictibilitat que té, és a dir, en la seva complexitat lineal. Una seqüència generada per un LFSR de longitud n té, complexitat lineal n, una complexitat molt baixa comparada amb el període.

Per a disminuir la predictibilitat de la seqüència de xifratge cal, doncs, augmentar la complexitat lineal de la seqüència de xifratge, que convindria que fos de llargada propera a la del període. Una manera de fer-ho és basant-se en operacions no lineals.

2.2.1.2 Generadors no lineals [uoc3]

Els generadors no lineals són els que executen, a més, operacions no lineals, com ara permutacions. El fet de no realitzar operacions lineals permet un augment de la complexitat lineal.

Els registres de desplaçament realimentats no linealment, NLFSR, tenen un esquema molt semblant als LFSR, amb la variant que la funció que els realimenta no és lineal

2.2.1.2.a Combinadors no lineals

Els combinadors no lineals, com indica el seu nom, basen el funcionament en la combinació d'un cert nombre d'LFSR. Segons com utilitzem i combinem aquests LFSR podem distingir bàsicament els tres tipus de generadors següents:

- Generadors basats en combinacions no lineals d'LFSR, que fan servir les sortides de diferents LFSR com a entrada d'una funció no lineal que generarà la seqüència final.

- Generadors basats en un control pas per pas, que utilitzen diferents LFSR, alguns dels quals regulen els impulsos de rellotge d'altres.
- Generadors de seqüència multirellotge, formats per diferents LFSR regits per cicles de rellotge de velocitats diferents.

Tot seguit detallarem el funcionament del generador Multivelocitat de Massey-Rueppel, que és el que ens interessa per al nostre projecte

2.2.1.2.b Generador multivelocitat de Massey-Rueppel

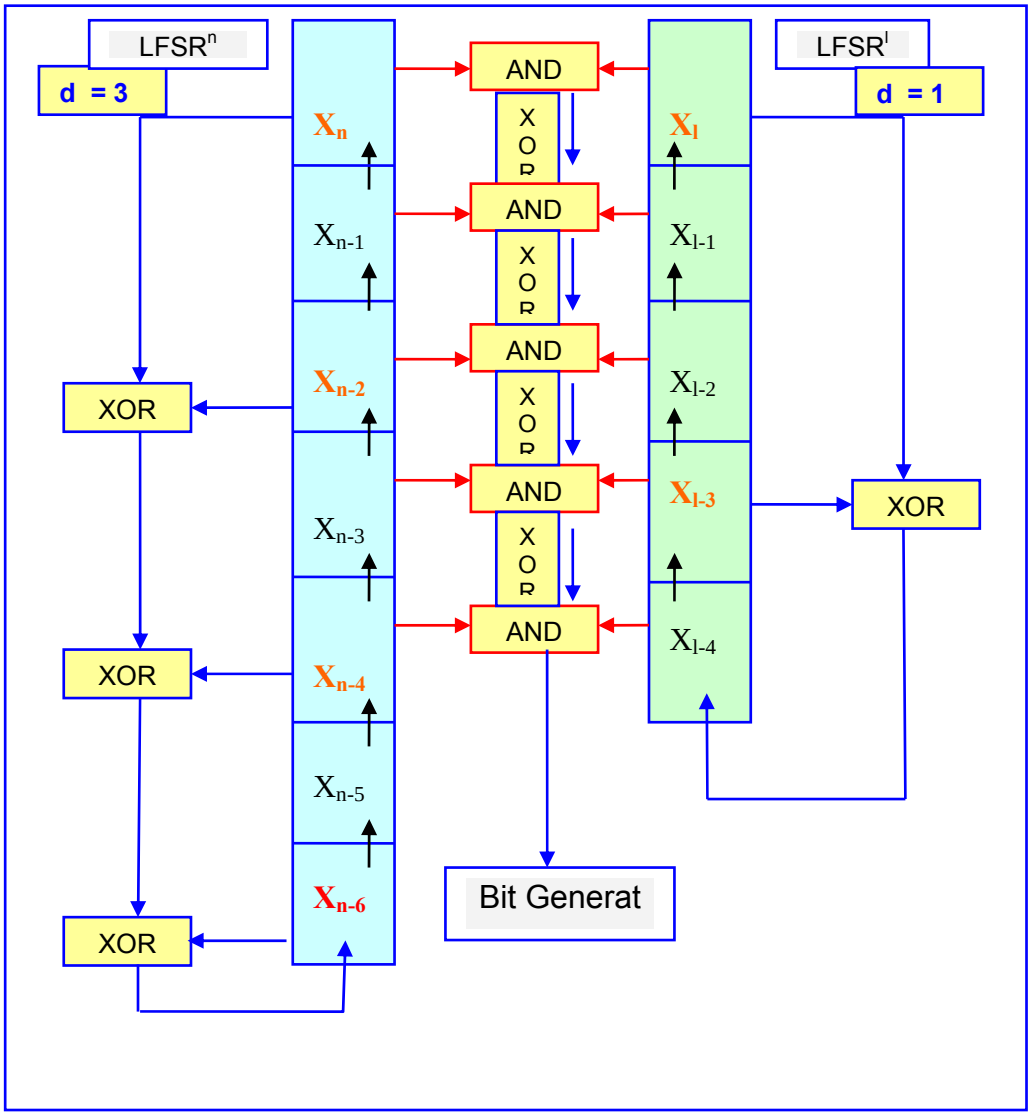
Aquest generador fa servir dos LFSR que funcionen a una velocitat diferent. L'$LFSR^n$, que té n cel·les, va d vegades ($d \geq 2$) més de pressa que l'$LFSR^l$, de l cel·les (on $n \geq l$).

L'expressió algebraica de la seqüència de sortida del generador és la :

$$s_t^f = \sum_{i=1}^{f} s_{dt+i}^n \bullet s_{t+i}^n$$

A la figura següent tenim un esquema detallat del funcionament del generador multivelocitat de Massey-Rueppel. En aquest cas tenim que l'$LFSR^n$ té per polinomi $1+ X^1+X^3+X^5+X^7$ i l'$LFSR^l$ té per polinomi $1+X^2+X^5$ i l'$LFSR^n$ va 3 cops més de pressa que l'$LFSR^l$.

Figura 7. Esquema d'un generador de Massey-Rueppel



Per obtenir un bit el procediment és el següent:

1. Un cop estan carregats els LFSR amb el seu estat inicial, per obtenir el primer bit tenim que fer l'operació AND entre les cel·les dels LFSR –L'índex d'aquestes cel·les anirà des de la que conté el major grau (X^n i X^l) fins la que conte el valor que correspon a X^1 del LFSR més curt– Tot seguit és fa l'operació XOR entre aquests resultats i ja tenim el primer bit.

2. Ara, per obtenir la resta de bits cal moure cadascun dels LFSR, és a dir, en aquest cas, l'$LFSR^n$ el mourem 3 cops i el $LFSR^l$ el mourem un cop, i tornem a fer l'operació AND i la XOR tal com en fet a (1).

Quan diem “moure l’LFSR” ens referim a efectuar l'operació XOR entre les cel·les que l’LFSR té activades, –les identificades pel seu polinomi– Un cop feta l’operació XOR, desplacem el contingut de les cel·les i carreguem a la cel·la que correspon a X^1 (és la cel·la de realimentació) de cadascun dels LFSR, el valor de la seva operació XOR.

Cal fixar-se que en aquest generador, la sortida dels LFSR no l’utilitzem per a res.

Podem observar que el cost d’obtenir un bit implica realitzar un nombre elevat d'operacions, això provocarà que aquest generador sigui una mica lent.

La complexitat lineal (L) de les seqüències generades és $L = n \cdot l$ sempre que els polinomis que defineixen als LFSR siguin primitius, i es compleixi que mcd(l,n)=1 i mcd(d,2^n-1) = 1, llavors el període de la seqüència val $p = (2^n-1)\cdot(2^l-1)$

2.2.2 Xifres de Bloc. [uoc5]

Una alternativa al xifratge de flux és el que s’anomena **xifratge de bloc**. Aquest tipus de xifratge s’inclou també dins el grup de criptosistemes de clau compartida, ja que la clau que s’utilitza per a xifrar i desxifrar és la mateixa i la comparteixen emissor i receptor. La diferència bàsica entre el xifratge de flux i el xifratge de bloc és la utilització de memòria en els algorismes de xifratge.

En el xifratge de bloc això no passa, perquè les xifres de bloc actuen sense memòria i el text xifrat només pot dependre del text en clar i de la clau. Així, doncs, en el xifratge de bloc dos textos en clar iguals es xifren sempre de la mateixa manera quan s’utilitza la mateixa clau.

Pel que fa a la utilització, els xifradors de bloc són força emprats, ja que aconsegueixen una velocitat acceptable de xifratge. En concret, el xifrador de bloc més utilitzat és el Data Encryption Standard (DES), pel fet que s’ha establert com a estàndard. Actualment, però, ha estat seleccionat l’Advanced Encryption Standard (AES), ja que en molts àmbits es considera que el DES (que va ser creat al final dels anys 70) no ofereix les mateixes garanties de seguretat que presentava quan va ser

creat, tenint en compte la rapidesa amb què s'ha desenvolupat la tecnologia informàtica aquests darrers anys.

2.2.2.1 Estructura del xifratge de bloc.

2.2.2.1.a Els principis de confusió i difusió de Shannon

Perquè un criptosistema sigui segur, cal que la informació que proporciona el text xifrat sigui la mínima possible. Això comporta, en particular, aconseguir que les propietats estadístiques del text en clar no es mantinguin en el text xifrat.

Shannon va proposar les dues tècniques que han de seguir els criptosistemes de xifratge de bloc si volen evitar eventuals atacs basats en mètodes estadístics. Aquestes tècniques proposades per Shannon son:

- **La confusió,** que inclou substitucions per tal que la relació entre la clau i el text xifrat sigui tan complicada com es pugui.
- **La difusió,** que utilitza transformacions que dissipen les propietats estadístiques del text en clar entre el text xifrat.

2.2.1.b Característiques comuns i modes de xifratge de bloc

Des d'un punt de vista general, tots els esquemes de xifratge de bloc tenen la mateixa estructura bàsica de funcionament. Tots aquests, a partir d'un bloc de text en clar, B, d'una longitud fixada, i una clau, K, executen determinades operacions més o menys complicades fins a obtenir el text xifrat corresponent, Y, de manera que: $Y = E_k(B)$.

A més, les operacions anteriors han de permetre que es pugui tornar a obtenir el mateix text en clar a partir del text xifrat Y i la clau K, si es porta a terme el procés de desxifratge següent: $B = D_K(Y)$.

Així mateix, les operacions que s'executen sobre el bloc que cal xifrar només combinen els mateixos elements que hi ha en el bloc i en la clau. Això ocorre, perquè aquests mètodes de xifratge no incorporen memòria, de manera que, en el cas que la clau K sigui la mateixa, dos blocs que siguin idèntics produiran textos xifrats idèntics.

Normalment, l'estructura bàsica dels criptosistemes de bloc moderns consta de dues transformacions, una d'entrada i una altra de sortida, entre les quals hi ha un nombre determinat d'iteracions d'una certa funció f, no lineal, que combina els elements que

formen part del bloc de text en clar amb els elements que formen part de la clau. De fet, generalment la clau, K, s'utilitza per a generar un seguit de subclaus, K_i, a partir d'una certa funció fK prou complicada, i són aquestes subclaus les que actuen en cada iteració.

Els modes de xifratge de bloc són:

Mode ECB

ECB és la sigla d'electrònic code Book.

Com que la majoria de vegades el text que s'ha de xifrar té una llargada superior a la longitud del bloc amb el qual treballa el criptosistema, el que es fa és partir el text que cal xifrar, M, en diversos blocs, M1, M2, ..., cada un dels quals té la llargada corresponent al bloc per a xifrar. D'aquesta manera es xifra cada un dels blocs amb una mateixa clau, K, per a obtenir el text xifrat resultant: $Y = Y_1 Y_2 \ldots$

Una vegada fixada una clau, K, a cada bloc de text en clar correspon un bloc concret de text xifrat, com si tinguéssim un diccionari per a cada clau K i anéssim buscant quina paraula (bloc) de text xifrat correspon a cada paraula de text en clar.

Encara que el mode ECB és el que sembla més natural d'utilitzar, des de la perspectiva de la seguretat té força inconvenients: el fet que el xifratge de dos blocs sigui totalment independent fa que sigui vulnerable a determinats atacs. Per exemple:

1. Podria ser que un atacant esborrés blocs de text xifrat, i com que això no afectaria el procés de desxifratge de la resta, el receptor no ho podria detectar.
2. L'atacant podria inserir blocs de text xifrat.
3. El fet que els mateixos blocs quedin xifrats sempre de la mateixa manera pot facilitar els atacs de tipus estadístic per a obtenir la clau K.

La utilització del xifratge de bloc mitjançant el mode ECB sobre missatges llargs i, en especial, en textos que repeteixen determinats patrons, es desaconsella totalment.

Mode CBC

El CBC és la sigla de Cipher block chaining .

Consisteix en l'encadenament dels blocs per al xifratge, de manera que es creï una dependència del xifratge de cada bloc amb l'immediat anterior.

Mode CFB

CFB és l'acrònim de Cipher feedback.

Aquest mode de xifratge utilitza indirectament el xifrador de bloc. Per això, la llargada dels blocs que s'han de xifrar no cal que sigui la mateixa que la dels blocs del criptosistema amb què actua, sinó que pot ser més petita.

Mode OFB

OFB és l'acrònim d'Output FeedBack.

El mode de xifratge utilitza el criptosistema de bloc com a generador pseudoaleatori. És un sistema molt semblant al mode CFB, l'única diferència que presenta és que el vector inicial es realimenta directament amb el resultat del xifratge de bloc abans de fer la suma bit a bit amb el bloc de text en clar.

Com que el xifrador de bloc actua com un generador pseudoaleatori, cal que els criptosistemes de bloc que emprem amb el mode OFB compleixin les característiques requerides per als generadors pseudoaleatoris, tant pel que fa a la impredictibilitat de la seqüència resultant com a la complexitat lineal.

2.2.2.2 Criptosistemes de xifratge de bloc.

2.2.2.2.a L'estàndard DES

DES és l'acrònim de Data Encryption Standard.

L'any 1977, el National Bureau of Standards (NBS), una secció del Departament de Defensa dels EUA, va publicar un criptosistema estàndard creat amb la finalitat de protegir qualsevol tipus de dades, el DES. En el desenvolupament d'aquest sistema van participar l'empresa IBM i la National Security Agency (NSA).

El criptosistema DES és un criptosistema de xifratge de bloc que xifra blocs de dades de 64 bits de llargada per mitjà d'una clau de 56 bits. El DES aconsegueix complir tant el principi de confusió com el de difusió, gràcies a les accions de les caixes S que conté.

a [uoc6] pag. 13 a 20, podem trobar una descripció detallada del funcionament d'aquest algoritme i de les seves particularitats.

2.2.2.2.b El criptosistema IDEA

IDEA és la sigla d'international Data Encryption Algorithm.
El criptosistema IDEA va ser desenvolupat per J. Massey i X. Lai l'any 1990. És un sistema que xifra blocs de text en clar de 64 bits de llargada per mitjà d'una clau de 128 bits. El seu funcionament es basa en vuit iteracions idèntiques seguides d'una transformació de sortida.
En aquest criptosistema queden garantits els principis de confusió i difusió. Pel que fa als processos de xifratge i desxifratge, l'algorisme que s'hi utilitza és el mateix, però l'obtenció de les subclaus amb què operen és diferent.

a [uoc6] pag. 21 a 23, podem trobar una descripció detallada del funcionament d'aquest algoritme i de les seves particularitats.

2.2.2.2.c El criptosistema AES

AES és l'acrònim de Advanced Encription Standard.
El National Institute of Standards and Technology (NIST) del govern nord-americà va fer una crida el setembre de 1997 perquè es presentessin propostes per a un nou estàndard de xifratge per tal de seleccionar l'Advanced Encription Standard (AES).

El 26 de maig del 2002 el FIPS va anunciar l'aprovació de l'Advanced Encryption Standard sota el codi FIPS-197. El **criptosistema de Rijndael** xifra blocs de text en clar de 128, 192 o 256 bits de longitud, encara que l'estàndard aprovat pel NIST només fa servir blocs de 128 bits. La longitud de les claus de xifratge que aquest criptosistema empra també pot variar entre 128, 192 o 256 bits. Les operacions criptogràfiques es basen en un grup finit d'ordre 28.

a [uoc6] pag. 25 a 36, podem trobar una descripció detallada del funcionament d'aquest algoritme.

2.3 Xifres de clau pública [UOC7]

La criptografia de clau compartida presenta tres inconvenients:

1. **La distribució de claus:** dos usuaris han d'elegir una clau secreta abans de començar-se a comunicar. En aquest cas, o bé s'han de trobar personalment o

bé han de confiar en un canal segur per a distribuir-se les claus. També és perfectament possible que no es puguin trobar personalment i que no disposin de cap canal segur.

2. **La gestió de claus:** en una xarxa de n usuaris, cada parella d'usuaris ha de tenir la seva clau compartida particular, la qual cosa implica un total de n(n-1) /2* claus per a tota la xarxa.

3. **No hi ha signatura digital:** la signatura digital és l'equivalent de les signatures manuals en el cas de la informació electrònica; amb un criptosistema de clau compartida, generalment no hi ha la possibilitat de signar els missatges, pel mateix fet que totes les claus són compartides almenys per dos usuaris.

El concepte de criptosistema de clau pública, que permet superar els inconvenients anteriors, va ser proposat per W. Diffie i M.E. Hellman en l'article "New directions in cryptography" aparegut l'any 1976. La idea era permetre un intercanvi segur de missatges entre emissor i receptor sense ni tan sols haver-se de trobar prèviament per acordar una clau secreta comuna.

2.3.1 Concepte de clau publica

Un criptosistema de clau pública s'adreça més sovint a una xarxa d'usuaris que no a una sola parella. En aquest criptosistema cada usuari, u, té associada una parella de claus < P_u,S_u >: la **clau pública**, P_u, que es publica amb el nom de l'usuari en un directori públic que tothom pot llegir, mentre que la **clau privada**, S_u, només la coneix u. Els parells de claus es generen mitjançant un algorisme de generació de claus.

Per a enviar un missatge secret m a u, tothom de la xarxa fa servir el mateix mètode:

1. Cercar Pu.
2. Calcular c =E(Pu,m), en què E és un algorisme públic de xifratge.
3. Enviar c a l'usuari u.

En rebre el text xifrat c, l'usuari u el pot desxifrar de la manera següent:

1. Cercar la seva clau privada S_u.

2. Calcular $D(S_u,c)$, en què D és un algorisme públic de desxifratge.

Perquè aquest procediment funcioni cal que $D(S_u,E(P_u,m) = m$. D'aquesta manera la gestió de claus queda simplificada, perquè ara només hi ha n parelles de claus per a n usuaris, en comptes de les n(n-1)/2 claus que calien amb la criptografia de clau compartida.

2.3.2 Intercanvi de claus de Diffie-Hellman

En un article de 1976, W. Diffie i M.E. Hellman van descriure un protocol d'intercanvi de claus que evita l'inconvenient de la distribució de claus. El fet innovador és que dos usuaris poden pactar una clau secreta compartida sobre un canal insegur.

Protocol 1: Intercanvi Diffie-Hellman

Per a dur a terme correctament el protocol d'intercanvi de Diffie-Hellman, s'han de seguir els passos següents:

1. Els dos usuaris, A i B, trien públicament un grup multiplicatiu finit G d'ordre n i un generador $\alpha \in G$.
2. A genera un nombre aleatori a, calcula α^a dins de G i transmet el resultat a B.
3. B genera un nombre aleatori b, calcula α^b dins de G i transmet el resultat a A.
4. A rep α^b i calcula $(\alpha^b)a$ dins de G.
5. B rep α^a i calcula $(\alpha^a)b$ dins de G.

Al final del protocol 1, A i B han pactat un element α^{ab} del grup G que és comú entre ells dos i secret per a la resta d'usuaris. La seguretat de l'intercanvi de Diffie-Hellman es basa en la dificultat del problema de Diffie-Hellman generalitzat.

2.3.3 Criptosistemes de clau pública

L'**aritmètica modula**r i, més generalment, la **teoria dels nombres** són la base sobre la qual s'implementen els criptosistemes de clau pública.

Algorisme de generació de claus de l'RSA

Per a generar les claus que es fan servir en l'RSA cal seguir els passos següents:

1. Cada usuari, u, tria dos nombres primers, p i q, de com a mínim 200 dígits. Llavors calcula n = pq, amb la qual cosa el grup multiplicatiu que farà servir u és

$G=Z^*_n$. L'ordre d'aquest grup és $\varphi(n)= \varphi(pq) = (p-1)(q-1)$. Per a u és fàcil de calcular aquest ordre, ja que coneix p i q.

2. L'usuari u tria un enter positiu, e, que compleixi les relacions $1\leq e \leq \varphi(n)$ i $mcd(e,\varphi(n))=1$.
3. Amb l'algorisme d'Euclides estès, u calcula l'invers d de e a $Z\varphi(n)$. Tenim, doncs, $ed \bmod \varphi(n) = 1$.
4. La clau pública de u és el parell (n,e), mentre que la seva clau privada és d. Per descomptat, p, q i $\varphi(n)$ també romanen secrets.

Si un usuari A vol enviar un missatge $m \in Z^*_{nb}$ a un altre usuari B, fa servir la clau pública de B, (n_b,e_b), per a calcular el valor $m^{eb} \bmod n_b = c$, el qual és enviat a B.

Per a recuperar el missatge original, B calcula:

$$c^{db} \bmod n_b = (m^{eb})^{db} \bmod n_b = m^{eb,db} \bmod n_b = m^{1+t\,\varphi(nb)} \bmod n_b =$$
$$= (m \bmod nb)\,(m^{\varphi(nb)} \bmod nb)^t \bmod nb = m$$

En l'última igualtat anterior s'ha fet servir el teorema d'Euler, que garanteix que $m^{\varphi(nb)} \bmod n_b = 1$ si $mcd(m,n_b) =1$ (i això darrer és gairebé segur que passarà, perquè n_b és molt gran i només té dos divisors, p_b i q_b).

Protocol de generació de claus de l'ElGamal

Per a generar les claus que es fan servir en l'ElGamal cal seguir els passos següents:

- Es tria un grup finit, G, i un element $\alpha \in G$ que no ha de ser necessàriament un generador.
- Cada usuari A tria un nombre aleatori, a, que serà la seva clau privada, i calcula α^a dins de G, que serà la seva clau pública.

Si un usuari A vol enviar un missatge, $m \in G$, a un usuari B, llavors fa el següent:

- A genera un nombre aleatori v i calcula α^v dins de G.
- A mira la clau pública de B, α^b, i calcula $(\alpha^b)^v$ i $m \cdot \alpha^{bv}$ dins de G.
- A envia a B el parell $(\alpha^v, m \cdot \alpha^{bv})$.

Per a recuperar el missatge original, B ha de fer el següent:

- B calcula $(\alpha^v)^b$ dins de G.
- B obté m només dividint la segona part del criptograma:

$$\frac{m \cdot \alpha^{vb}}{\alpha^{vb}}$$

2.4 Signatures Digitals [UOC8]

Les signatures mantenen una estreta relació amb la criptografia de clau pública. Més concretament, cada usuari crea un parell de claus: una de pública i una de privada. Per a signar un missatge, l'usuari fa servir la seva clau privada; per a verificar una signatura, qualsevol pot fer servir la clau pública del signatari.

El verificador queda convençut que el missatge no ha estat alterat perquè està signat. A més, amb aquest procediment, el signatari no pot repudiar més tard el fet d'haver signat el missatge, perquè ningú, llevat del signatari no té la clau privada necessària per a produir la signatura.

2.4.1 Esquemes de signatura digital

En aquest apartat comentarem tres dels esquemes de signatura digital més emprats:

- El primer esquema es basa en el criptosistema RSA i, per tant, la seva infalsificabilitat està relacionada amb la dificultat que suposa el problema de la factorització.
- El segon esquema es basa en el criptosistema d'ElGamal i, per tant, la seva infalsificabilitat està relacionada amb la dificultat que suposa el problema del logaritme discret.
- El tercer esquema és l'algorisme estàndard de signatura homologat pel govern dels EUA, que, de fet, és molt semblant a la signatura d'ElGamal (amb algunes millores).

Signatura RSA

Suposem que un usuari A té una clau pública RSA (n_A,e_A) i una clau privada d_A. Per a signar digitalment el missatge m amb l'RSA, l'usuari A calcula la signatura $s = m^{dA} \bmod n_A$. Tot seguit A publica el missatge signat difonent la parella (m,s).

Per a verificar la signatura s, qualsevol usuari B pot mirar si es compleix $s^{eA} \bmod n_A = m$. En cas afirmatiu, la signatura és vàlida; en cas negatiu, la signatura no ho és.

Cal tenir en compte que el protocol de verificació suposa implícitament que B disposa de l'autèntica clau pública de l'usuari A, una manera d'assegurar-se que la clau pública de A és correcta és recuperar-la a partir d'un certificat emès per un gestor de directori de claus públiques o, generalment, per una autoritat de certificació.

Signatura d'ElGamal

Suposem que el signatari A ha fet una generació de claus d'ElGamal dins un grup G usant un element $\alpha \in G$. Suposem que a és la clau privada de A, que $\alpha^a \in G$ és la seva clau pública i n és l'ordre del grup G. Si A vol signar un missatge $\alpha \in G$, llavors fa el procés següent:

- A genera un nombre aleatori h tal que $mcd(h,\varphi(n)) = 1$.
- A calcula $r = a^h$ dins de G.
- A troba el valor de s resolent la congruència $m \equiv ar+hs \bmod \varphi(n)$.

La signatura digital per al missatge m és la parella (r,s).

Per a verificar la signatura del missatge m, qualsevol usuari B pot fer el procés següent:

- B calcula $r^s = (\alpha^h)^s$ dins de G. També calcula $(\alpha^a)^r$ dins de G.
- B verifica si es compleix: $(\alpha^a)^r(\alpha^h)^s = \alpha^m$, on totes les operacions són dins de G.

L'estàndard DSS

Aquest mètode és molt similar a la signatura d'ElGamal.

2.4.2 Funcions hash

Les signatures digitals en ús actualment són lentes (amb relació a criptosistemes de clau compartida, com el DES). Per tant, és desitjable signar només un resum del missatge en comptes del missatge sencer. Les funcions hash serveixen per a crear resums.

Una **funció hash** és una funció que fa correspondre a un missatge m de mida variable una representació H(m) de mida fixa. Típicament, H(m) té de 64 a 160 bits i s'anomena el valor hash del missatge.

Si una funció hash ha de ser emprada per a aplicacions criptogràfiques, no n'hi ha prou que resumeixi la seva entrada de manera aparentment aleatòria. Cal que sigui també unidireccional.

Una **funció hash unidireccional** és una funció hash H tal que, per a qualsevol missatge m' del recorregut de H, és difícil de trobar m tal que m=5 H(m).

Així, doncs, una funció hash unidireccional és una funció hash que és també una funció unidireccional. La combinació de funcions hash unidireccionals amb signatures digitals dóna peu al protocol de signatura amb hash i al corresponent protocol de verificació.

Protocol de signatura amb hash

Si A vol signar el missatge m amb una funció hash fa dues accions:

- Calcula H(m), on H és una funció hash unidireccional públicament coneguda.
- Signa H(m) amb la seva clau privada i obté la signatura s. s és considerada la signatura de m.

Protocol de verificació amb hash

Per a verificar la signatura s del missatge m, qualsevol usuari B pot fer:

- Primer calcula H(m).
- Tot seguit, verifica si s és una signatura vàlida per a H(m).

Les funcions hash més utilitzades són els message digest de Rivest, coneguts com MD2, MD4 i MD5. Aquestes funcions produeixen resums de 128 bits i l'únic atac que es coneix contra aquestes, és el de la cerca exhaustiva.

Donat que MD5 ja es pot trencar en un dia de procés d'un ordenador, actualment s'utilitza la funció Secure Hash Algorithm (SHA-1) i SHA-2, que generen resum de 160 i 200 bits.

Els algorismes que implementen aquestes funcions tenen una aparença semblant als dels criptosistemes de tipus DES amb la diferència que no depenen de cap clau. Llur velocitat en programari és superior a la dels criptosistemes de tipus DES.

2.5 Infraestructura de clau publica [UOC8]

2.5.1 Distribució de claus

La criptografia de clau pública permet l'intercanvi de missatges confidencials i íntegres de manera àgil, sempre que disposem de la clau pública de l'interlocutor amb qui ens comuniquem. El problema és com obtenir aquesta clau pública i poder estar segurs que pertany a qui ens pensem.

Diffie i Hellman (1976) van pensar que la distribució de claus es resoldria amb un directori segur en línia en el qual s'establís de manera unívoca el lligam entre un nom distintiu d'usuari i una clau pública. El directori públic hauria de signar totes les seves transaccions de manera que ningú no en pogués suplantar la identitat. El problema raïa en el baix rendiment que el sistema oferia en poblacions d'usuaris mitjanes o grans. L. Kohnfelder (1978) es va basar en la idea d'una autoritat central de confiança introduïda per Diffie i Hellman i va proposar crear uns registres de dades signades –els certificats– que permetrien que la distribució de claus es fessin des de directoris públics que no requerissin confiança.

Un **certificat digital** és una estructura de dades que conté informació del propietari de les claus criptogràfiques, la clau pública, i una signatura digital dels dos camps anteriors que hi dóna validesa.

La signatura, realitzada per un usuari o entitat externa lleial, assegura la integritat contra una possible modificació no desitjada de les dades. La confiança en el signatari s'estén al subjecte del certificat.

2.5.2 infraestructura de clau pública (PKI)

PKI és l'acrònim de públic key infrastructure.

L'objectiu d'una infraestructura de clau pública és la gestió eficient i confiable de les claus criptogràfiques i els certificats per a què es puguin utilitzar per a funcions d'autenticació, integritat, no-repudi i confidencialitat. La infraestructura de clau

pública crea un marc segur d'intercanvi de dades en un entorn típicament insegur com Internet.

Una **infraestructura de clau pública** (**PKI**) és el conjunt de maquinari, programari, persones, polítiques i procediments necessaris per a crear i gestionar certificats digitals basats en criptografia de clau pública.

El certificat és l'element central de la infraestructura de clau pública al voltant del qual es crea aquesta infraestructura de suport que abraça serveis com el registre d'usuaris, l'emissió de certificats, la seva distribució des de directoris públics, la seva renovació i revocació, la recuperació de claus, etc.

2.5.3 Components d'una infraestructura de clau pública

Els principals components d'una infraestructura de clau publica són:

Autoritat de certificació

L'**autoritat de certificació** (**CA**) és la responsable d'emetre i revocar certificats. És l'entitat de confiança que dóna legitimitat a la relació d'una clau pública amb la identitat d'un usuari o servei.

Autoritat de registre

L'autoritat de registre (RA) és l'encarregada de verificar el lligam entre les claus públiques i la identitat dels seus titulars.

Subscriptors i entitats finals

Els **subscriptors** i les **entitats finals** són aquells que posseeixen un parell de claus (pública i privada) i un certificat associat a la clau pública.

Usuaris

Els usuaris són els agents que validen signatures digitals i la seva ruta de certificació a partir de claus públiques emeses per autoritats de certificació de confiança. També poden xifrar documents per a subscriptors i entitats finals.

Repositoris

Els **repositoris** són les estructures encarregades d'emmagatzemar la informació relativa a la infraestructura de clau pública. Els dos repositoris més importants en una infraestructura de clau pública són el repositori de certificats i el repositori de llistes de revocació de certificats.

Una **llista de revocació de certificats** (**CRL**) inclou tots aquells certificats que per diversos motius són invàlids abans de la data de caducitat establerta en el mateix certificat.

Autoritat de validació

L'**autoritat de validació** (**VA**) és l'encarregada de comprovar la validesa dels certificats digitals.
sobre l'estat actual del certificat (revocat, suspès, vàlid o estat desconegut) o relativa a la cadena de certificació necessària per a validar l'autenticitat del certificat.

Autoritat de segellat de temps

L'**autoritat de segellat de temps** (**TSA**) és l'encarregada de signar un missatge amb la finalitat de provar que existia abans d'un determinat instant de temps.

2.5.4 Certificat X.509

El format de certificats més àmpliament acceptat en infraestructura de clau pública està definit per l'ISO/IEC JTC1 SC21 i es coneix com X.509v3.
L'**estàndard internacional X.509v3**, que també es publica amb el nom d'**ITU-T Recommendation X.509**, és important per dues raons bàsiques: defineix el marc per a la provisió de serveis d'autenticació i un format de certificat per a les claus públiques.

El format dels certificats X.509 ha evolucionat a partir de tres versions en diferents edicions de l'estàndard. La primera versió va aparèixer el 1988. El 1993 es va revisar i se n'obtingué la versió 2, en què s'hi van afegir dos camps opcionals d'identificació única. L'ús de l'X.509 junt amb l'Internet privacy enhanced mail (PEM) va fer sorgir la necessitat de dotar el certificat de més flexibilitat. El juny de 1996 va aparèixer la versió 3 del certificat, que inclou la possibilitat de tenir camps d'ampliació.

a [uoc9] pag. 28 a 32 podem trobar un detall del format i contingut d'un certificat.

2.6 Java, seguretat i criptografia

[uoc10] L'any 1991 James Goslingn el creador de Java, va anomenar aquest llenguatge OAK perquè un gran roure (oak, en anglès) era el que veia des de la finestra de la seva oficina de Sun Microsystems.

Més tard, l'equip de desenvolupament de Java va descobrir que Oak era el nom d'un altre llenguatge de programació i que, per tant, calia trobar-ne un altre. Mentre estaven en una cafeteria se'ls va ocórrer d'anomenar-lo Java, que és una manera col·loquial de referir-se al cafè als Estats Units. Aquesta és l'explicació del símbol de Java: una tassa de cafè que fumeja. Java, va néixer el maig de 1995. És, d'una banda, un nou llenguatge simplificat de Sun basat en objectes i sistemes oberts, amb què els desenvolupadors de programari poden crear aplicacions que es poden distribuir a Internet mitjançant el web (o amb altres mitjans) i, de l'altra, un ordinador virtual (anomenat tècnicament màquina virtual Java) que permetrà que les aplicacions basades en Java siguin omnipresents, que es puguin executar en qualsevol ordinador independentment del seu maquinari i el seu sistema operatiu.

La maquina virtual de Java és un processador CISC. Te la característica que cada opcode mesura un byte, per això del codi Java se'n diu bytecode. Però una característica encara mes important és que és un llenguatge ensamblador orientat a objectes.
a [JAVA1] tenim els links corresponents a tutorials i manuals relacionats amb la seguretat i criptografia implementades a Java. En els següents apartats comentarem algunes d'aquestes característiques.

2.6.1 Seguretat a Java

Teòricament l'emulació a Java anul·la la possibilitat de buffers overflows ja que el sistema té totes les àrees de memòria controlades. L'única manera de carregar codi a una maquina virtual és forçant la carrega d'una classe, tot i que pot ser que el sistema segueixi protegit amb una "sandbox" mitjançant un classloader especial. Quan creem un nou objecte mitjançant new, s'invoca al classloader que té el sistema per defecte i que carrega la classe. Si canviéssim el classloader que hi ha per

defecte (que tècnicament s'anomena bootstrap class loader) es podria limitar l'accés a les classes del classpath mitjançant classes proxy.

Podem canviar el classloader d'un fil mitjançant thread. setContextClassLoader(ClassLoader cl). Per exemple, podríem limitar l'accés als fitxers modificant el classloader de la classe File, aleshores quan es creés una instància de Fitxer es crearia una instància que implementaria la interfície, però que actuaria de forma diferent. Generalment quan trobem per Internet que han trobat un error a Java es refereixen a alguna maquina virtual, i no en al disseny de la plataforma Java.
Degut a la dificultat d'executar codi arbitrari a Java, podem considerar-lo un llenguatge adequat per aplicacions que requereixin un alt grau de seguretat.

2.6.2 Criptografia a Java

L'API criptogràfic de Java es troba dividida en dues parts, la JCA (Java Cryptography Architecture i el JCE (Java Cryptography Extension). El motiu d'aquesta divisió no és només una decisió de disseny sinó que també es deu a motius polítics.
La criptografia sempre ha estat considerada una eina perillosa per governs com el d'EE.UU., ja que permet a qualsevol persona utilitzar-la per amagar informació sense que ningú, ni el propi govern, en pugui conèixer el contingut. Per aquest motiu, quan Sun va decidir incloure la criptografia dins el llenguatge sense incomplir les lleis americanes, va haver de condicionar el disseny de l'API criptogràfic a aquestes lleis.
El JCA, són tot el conjunt d'interfícies que defineixen les operacions criptogràfiques que es poden dur a terme usant el llenguatge Java. Aquestes interfícies, defineixen com usar una signatura digital, un algorisme de xifrat o una funció hash, sense proporcionar la implementació concreta de cap algorisme. Aquestes interfícies, conegudes com Engines o motors criptogràfics defineixen de forma molt general, quins són els paràmetres necessaris per un algorisme criptogràfic. Podem entendre un motor criptogràfic, com una "caixa buida" amb certes dades que entren i que en surten.
La JCA defineix tot aquest conjunt de "caixes", no només pel xifrat, sinó també per les signatures, funcions hash, etc. Tot i així, aquestes caixes es troben buides, no fan res, ja que simplement són interfícies d'alt nivell. Els proveïdors criptogràfics o

JCE (Java Cryptography Extension) són paquets software que proporcionen implementacions concretes per als motors criptogràfics que proporciona el JCA. Aquests proveïdors els pot escriure qualsevol persona que vulgui proporcionar un conjunt d'implementacions concretes d'algorismes criptogràfics. El JCE (Java Cryptography Extension) proporcionat per Sun defineix el contingut d'aquestes caixes per alguns algorismes criptogràfics.

2.6.2.1 JCE

JCE: Java Cryptography Extension

El JCE és un framework per a criptografia que forma part de la distribució estàndard de la JVM (màquina virtual de Java). Ofereix un API (application programming interface) que permet:

- Generació de claus (claus secretes i parelles de claus pública y privada),
- xifrat simètric (DES, 3DES, IDEA, etc.)
- Xifrat asimètric (RSA, DSA, Diffie-Hellman, ElGamal...)
- Funcions de resum (MD5 y SHA1) i Algoritmes MAC Message Authentication Code)
- Acord de claus.

Les implementacions dels algoritmes de xifrat, generació de claus, etc. son ofrenades por paquets externs anomenats **providers**.

La distribució bàsica inclou per defecte el provider "SUN" amb implementacions dels algoritmes més representatius. Altres fabricants ofereixen providers addicionals que inclouen algoritmes o implementacions alternatives dels que ja estan inclosos en el provider "SUN". Així és possible dotar al JCE amb noves funcionalitats sense la necessitat de canviar l'API bàsica y permetre la distribució d'algoritmes criptogràfics amb limitacions d'exportació.

2.6.2.2 Generació i ús de Claus

Key Factories

Les Key Factories són entitats que permeten construir objectes Java que representen claus criptogràfiques, objectes del tipus Key, PublicKey, PrivateKey, etc. La diferència entre una KeyFactory i un KeyGenerator és que aquests últims generen claus del no res, generalment de forma aleatòria. Les Key Factories en

canvi, creen claus a partir de cert "material criptogràfic". Aquest material pot ser, per exemple, un array de bits que hem generat nosaltres aleatòriament, uns nombres n, p i q, en cas de què usem RSA, o bé, a partir d'un array de bytes que representa una clau generada prèviament emmagatzemada usant una codificació determinada.

La classe KeyFactory és una classe motor que s'encarrega de generar les claus a partir de material previ.

2.6.2.2.a Criptografia Simètrica

Les claus de criptografia simètrica venen representades per la interfície SecretKey. Aquesta interfície també deriva de la interfície general Key. Per crear claus simètriques disposem, igual que en criptografia asimètrica, de dues maneres diferents segons vulguem generar claus de forma aleatòria o a partir d'un material previ:

- *KeyGenerator* Aquesta classe permet generar claus de forma aleatòria per a diferents algorismes de criptografia simètrica com DES, Triple DES, Blowfish, o AES.
- *SecretKeyFactory* Aquesta classe permet generar a claus a partir de cert material previ. Aquest material (pot ser una colla de bits), ha d'anar encapsulat en una classe del tipus KeySpec. Si l'algorisme utilitzat per generar les claus és DES o TripleDES, podem utilitzar les classes DESKeySpec o DESedeKeySpec respectivament. En el cas de què l'algorisme utilitzat sigui un altre, podem utilitzar la classe SecretKeySpec que és de propòsit general per a claus de qualsevol algorisme simètric. Aquesta última classe, que també implementa la interfície SecretKey, pot ser utilitzada directament com una clau sense necessitat d'utilitzar una KeyFactory prèviament.

PBE: Password-Based Encryption

La Password-Based Encryption ens permet generar claus d'un algorisme simètric a partir d'un password de forma segura. La forma de dur a terme aquest procés està definida a l'estàndard PKCS#5 [4] de RSA Laboratories.

El principal avantatge d'utilitzar aquest tipus de claus, és que no cal emmagatzemar la clau usada a cap lloc ja que simplement recordant el password a partir de la qual l'hem creat podem obtenir-la.

La diferència principal a l'hora d'utilitzar aquest tipus de xifrat respecte el vist anteriorment són fonamentalment dues.

En primer lloc, el procés de creació de claus es du a terme mitjançant una SecretKeyFactory ja que estem creant una clau a partir de material previ (un password). L'algorisme amb el qual instanciarem aquesta SecretKeyFactory serà un algorisme propi de PBE, com per exemple PBEWithMD5AndTripleDES. Una vegada instanciada la SecretKeyFactory construirem la clau passant-li un objecte de la classe PBEKeySpec inicialitzat amb el password que haguem triat.

En segon lloc, a l'hora de xifrar utilitzant un motor criptogràfic Cipher hem de tenir en compte que els xifrats usant PBE necessiten ser inicialitzats amb més paràmetres a part de la clau i el mode d'operació. Aquests paràmetres són el salt i el nombre de rounds utilitzats per generar la clau a partir del password. Aquests paràmetres els especificarem a l'inicialitzar el motor Cipher per mitjà dels mètodes *PBEKeySpec(char[] password)* i *PBEParameterSpec(byte[] salt, int rounds)* de la classe *PBEParameterSpec*.

2.6.2.2.b Criptografia asimètrica o de clau publica

Per començar a treballar amb qualsevol tipus d'algorisme relacionat amb la criptografia asimètrica, el primer que cal és generar les claus necessàries per fer servir aquests algorismes.

L'arquitectura criptogràfica de Java ens proporciona una classe motor per a dur a terme aquest propòsit anomenada KeyPairGenerator. Com totes les classes motor, disposa dels mètodes getInstance, que permeten obtenir un objecte del tipus KeyPairGenerator que generi claus per a un algorisme determinat. El proveïdor que proporciona Sun per defecte, proporciona els següents algorismes de generació de claus:

- DSA (Digital Signature Algorithm)
- RSA (Rivest Shamir Adleman)

2.6.2.3 Signat de dades

Un motor criptogràfic que podem fet servir per al signat de dades és la classe *Signature* que es troba dins el paquet java.security. Com tots els motors criptogràfics, aquesta classe no disposa de constructor, sinó del mètode getInstance(String Algorisme), que permet crear "signadors" segons un algorisme determinat.

- ECDSA Signatura amb DSA basat en corbes el.líptiques
- MD2withRSA Signatura amb RSA i el hash MD2
- MD5withRSA Signatura amb RSA i el hash MD5
- NONEwithDSA Signatura amb DSA sense hash Les dades han de ser exactament 20 bytes.
- SHA1withDSA Signatura amb DSA i el hash SHA1
- SHA..withRSA Signatura amb RSA i el hash SHA1,SHA-256, SHA-383, i SHA-512, (160, 256, 383, y 512 bits, respectivament)

Signar un objecte.

Java ens permet signar objectes en el moment de serialitzar-los. El mètode a utilitzar és :

```
SignedObject so =new SignedObject(objecte , clau , signature);
```

On signature és un objecte de la classe Signature inicialitzat d'acord amb l'algoritme que desitgem; objecte, és l'objecte que volen serialitzar i signar; i clau, és la clau privada per efectuar la signatura.

2. 6.2.4 Xifrat de dades

La classe *Cipher* ens permet xifrar i desxifrar dades usant algorismes simètrics o asimètrics. Aquesta classe és un motor criptogràfic pel que la inicialització sempre es durà a terme mitjançant el típic mètode:

- públic static Cipher getInstance(String transformation);
- públic static Cipher getInstance(String transformation, String provider);

On transformation te la forma :

- "algoritme / mode / farciment " o "algoritme", per exemple :

```
Cipher c1 = Cipher.getInstance("DES/ECB/PKCS5Padding");
```

```
Cipher c1  = Cipher.getInstance("RSA/ECB/PKCS1Padding", "BC");
Cipher c1 = Cipher.getInstance("DES");
```

Els possibles modes de treballar de l'algoritme son :

- Cap (None)
- ECB (Electronic Code Book)
- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback Mode)
- OFB (Output Feedback Mode)
- PCBC (Propagating Cipher Block Chaining)

El farciment és degut a què generalment el missatge no és múltiple de la mida d'un bloc, per tant, ens trobem que a l'últim bloc cal afegir-hi bytes per tal que tingui la mida d'un bloc. Aquest farciment s'anomena padding. Els diferents valors per aquest paràmetre son:

- NoPadding
- PKCS5
- OAEP
- SSL3

Quan utilitzem el motor criptogràfic *Cipher*, ens trobem que hem d'utilitzar bucles que vagin cridant successives vegades el mètode *update()* i finalment *doFinal()*. Java proporciona dues classes Stream que permeten automatitzar molt més aquest procés, la CipherInputStream i la classe CipherOutputStream .

2.6.2.5 l'entorn PKI

2.6.2.5.a Key stores (Magatzems de claus)

Aquestes entitats són repositoris que permeten emmagatzemar, en un sol arxiu, múltiples claus i/o certificats de forma segura. Els Key Stores es manipulen a través de la classe KeyStore. Alguns dels mètodes més destacables d'aquesta classe són:

- *getInstance(String Algorithm)* Crea un objecte KeyStore. Els tipus especificats al camp algorisme poden ser "JKS", "JCEKS", o bé "PKCS12".
- *store(OutputStream os, char[] password)* Emmagatzema la Key Store en l'OutputStream especificat.

- *load(InputStream is, char[] password)* Carrega el contingut d'una Key Store que es llegeix a través de l'stream especificat.
- *setCertificateEntry(String alias, Certificate cert)* Afegeix un certificat al nostre KeyStore. Aquest certificat serà referenciat a través de l'alias que es passa per paràmetres.
- *setKeyEntry(String alias, Key key, char[] password, Certificate[] chain)* Afegeix una parella clau privada-certificat al Key Store. La clau privada és emmagatzemada xifrada utilitzant el password especificat. L'entrada corresponent també és referenciada a través d'un àlias.

2.7 Proposta de solució per al nostre projecte

Un cop feta la presentació dels coneixements que cal comprendre per dur a terme aquest projecte, passem a descriure la nostra proposta per fer realitat el projecte.

2.7.1 El Generador

Per una banda, els LFSR amb polinomi primitiu compleixen els tres postulats de Golomb, és a dir, la seqüència té bones propietats aleatòries. Per l'altra banda, hem presentat les característiques i el funcionament del generador de Massey-Rueppel.

Hem vist que la complexitat lineal d'aquest generador esta definida pel grau dels LFSR que el formen, és a dir, si tenim un generador amb dos LFSR (de polinomi primitiu) de grau 7 i grau 11, tenim que la seva complexitat lineal és 7*10 = 70.

Com les seqüències produïdes per un generador de Massey-Rueppel tenen bones qualitats aleatòries, ens centrarem com millorar el valor de la complexitat lineal, és a dir, cal trobar un procediment que ens permeti augmentar aquest valor sense perdre qualitats aleatòries i sense allargar massa el temps de procés de xifratge.

Com ja vam proposar, el nostre generador estarà basat en una combinació de generadors de Massey-Rueppel, concretament, volem combinar la sortida de 3 generadors d'igual període, per tant, si volem aprofitar tots els bits produïts i que la seqüència no perdi les seves característiques aleatòries i la complexitat sigui propera al període, considerem que tenim que tenim que aconseguir dos fets:

1. A mida que la seqüència generada augmenta, cadascun dels generadors tindrà que haver generat un nombre de bits similars, és a dir, si estem calculant el bit nº 1000, el generador nº 1 en pot haver generat 335, el generador nº 2 en pot haver generat 327 i el nº 3 ha generat 338.

2. L'ordre com intervenen els generadors ha de ser aleatori.

Per aconseguir aquest comportament, proposem el següent algoritme:

- Definicions:
  - De cadascun dels generadors de Massey-Rueppel que formen part del nostre generador, siguin **g1**, **g2** i **g3**, el conjunt d'operacions necessàries per generar un bit amb valor **b**.
  - Siguin **c1**, **c2**, **c3**, nombres enters que comptabilitzen el nombre de cops que s'ha efectuat l'operació **g1**, **g2**, **g3**, respectivament.
  - Sigui $\mathbf{b_n}$ **,** el valor **b**, de l'operació XOR numero **n** del procés intern de **g1**, **g2**, o **g3,** per generar el bit anterior **,** on **n** = grau del LFSR més curt - 2

- L'algoritme és:

Iniciar valors	bn = true (o false); c1 = c2 = c3 = 0,	
Generar els bits demanats	Si c1 ≥ c2 i	Si b_n = false => fer operació g3 => c3 = c3 + 1
		Si b_n = true => fer operació g2 => c2 = c2 + 1
	Sinó Si c2 ≥ c3 i	Si b_n = false => fer operació g1 => c1 = c1 + 1
		Si b_n = true => fer operació g3 => c3 = c3 + 1
	Sinó Si c3 ≥ c1 i	Si b_n = false => fer operació g2 => c2 = c2 + 1
		Si b_n = true => fer operació g1 => c1 = c1 + 1

Figura 8. Taula de l'algoritme per combinar les sortides del nostre generador

Anàlisis de la condició b_n=0 o b_n=1

Si analitzem el funcionament del Generador de Massey-Rueppel, observarem que el valor **b** del bit de sortida ha sofert una evolució, diguem que es va fent més aleatori a mida que es van realitzant les operacions internes (and i xor), per tant, nosaltres utilitzarem el valor $\mathbf{b_n}$, –ja que ha de ser un valor prou bo per obtenir un grau

d'incertesa propera al 50%– per determinar quin, de dos generadors, executa les seves operacions.

Per comprovar-ho simularem el funcionament d'un generador Massey_Rueppel, a partir del moment que els seus LFSR ja han efectuat les seves operacions XOR, amb el mètode Math.random() de la llibreria Java.

El mètode per simular el resultat de l'operació AND entre dues cel·les serà :

```
if(Math.random()<0.75) and = false     // llavors  comptabilitzem un false
else and = true                        // comptabilitzem un true
```

Ara només cal anar fent l'operació XOR entre els valors que va retornant aquest mètode i observar com evoluciona el valor **b.**

Per exemple, simulem 5000 cops el funcionament del generador de Massey-Rueppel Suposem que aquest generador, per obtenir el valor **b,** ha realitzat 5 operacions AND i 4 XOR, és a dir, la longitud del lfsr mes curt és de 5 cel·les (grau 5) llavors: **n** = 5-2 = 3 i b_1 = and ^ and => b_2 = b_1^ and => $\mathbf{b_n}$ = b_2^ and i **b** = b_n^ and

Si comptabilitzem en % el nombre de cops que b_1, b_2, i $\mathbf{b_n}$ han assolit el valor 1 o 0 (ignorem l'ultima operació que és el valor del bit de sortida), podrem observar l'evolució del grau d'aleatorietat del valor **b** (b_1,b_2, ..., b_n) al llarg de les successives operacions XOR (els valors propers al 50% son els desitjats)**.** El resultat és:

```
Simulació d'un Generador de Massey-Rueppel en generar 5000 bits
  Anàlisi de l'evolució del valor b:
  uns_b1 = 37.16%      zeros_b1 = 62.84%
  uns_b2 = 43.4%       zeros_b2 = 56.6%
  uns_bn = 48.22%      zeros_bn = 51.78%
```

Els resultats ens confirmen l'evolució del valor **b,** i com, el valor $\mathbf{b_n}$ és prou bo per utilitzar-lo a l'ora de triar un dels generadors .

Anàlisis de la condició Si $c_1 \geq c_2$ Sinó $c_2 \geq c_3$ Sinó $c_3 \geq c_1$

Aquesta condició ens ha de permetre que els tres generadors intervinguin el mateix nombre de cops, però cal que la incertesa de $\mathbf{b_n}$ no quedi afectada, per tant, per comprovar-ho, efectuarem la simulació anterior i comptabilitzarem quants cops ha

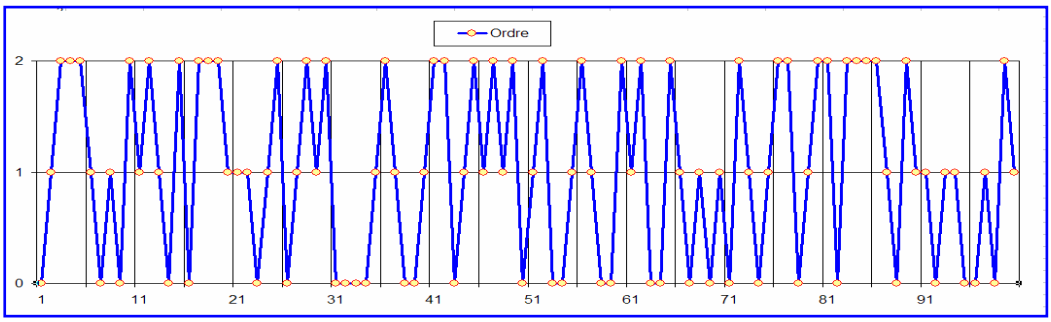
intervingut un generador, i, d'aquestes intervencions, mirarem el % de 1'ns i 0's que li ha correspost del valor b. El resultat és:

```
Simulació amb 3 generadors de Massey-Rueppel en generar 5000 bits
  Anàlisi de la distribució del valor bn per a cada Generador :
  g1=1665 => 33.3%  g2=1667 => 33.34%  g3=1668 => 33.36%
  La distribució és :
  Generador_1 trues = 47.27%     falses = 52.73%
  Generador_2 trues = 47.63%     falses = 52.37%
  Generador_3 trues = 49.76%     falses = 50.24%
```

Tal com podem comprovar els valors estan bastant equilibrats, per tant és un bon comportament pseudoaleatori i a més, els generadors intervenen aproximadament el mateix nombre de cops.

Gràficament podem representar, per a cada bit, quin és el generador que el genera i comprovar si aquesta incertesa amaga algun patró. La gràfica següent ens mostra l'ordre com intervenen els generadors. A l'eix Y tenim les rectes de cada generador i a l'eix X tenim els primers 100 bits generats .

Figura 9. Gràfica de l'ordre com intervenen els generadors



Podem observar com els generadors 1 i 2 han generat 34 bits i el generador 0 ha generat 32 bits, també podem veure com inicialment no s'aprecia cap patró en l'ordre que han intervingut.

Cal entendre que fins ara només hem tractat la manera de fer aleatòria la combinació de les sortides del 3 generadors, per tant, la qualitat final de la seqüència generada dependrà de la suma de dos fets, per una banda el comportament pròpiament dit d'un generador de Massey-Rueppel i per l'altre el comportament de l'algoritme de mescla que hem presentat.

Per tant, partint de la premissa del bon comportament d'un generador de Massey-Rueppel més la qualitat pseudoaleatòria de l'algoritme de mescla, podem esperar que **aquest algoritme ens ha de permetre obtenir una seqüència amb bones qualitats aleatòries i amb una complexitat lineal tan elevada com el propi període de la seqüència.** És a dir, si tenim 3 generadors de Massey-Rueppel amb període $p_1 = p_2 = p_3$, construirem un generador amb període p = p1+p2+p3 i una complexitat lineal propera a p.

Un possible atac contra aquest algoritme consistiria a endevinar l'ordre d'intervenció dels generadors, però donada la incertesa aconseguida considerem que caldria provar totes les possibles combinacions.

Per comprovar que això és cert, caldrà implementar el generador i analitzar les seqüències produïdes amb els diferents tests proposats. També avaluarem la complexitat lineal amb l'algoritme de Berlekamp-Massey i decidirem si cal o no modificar l'algoritme de mescla que hem presentat.

7.2. Els requeriments

Presentem tot seguit les principals funcionalitats que l'usuari pot demanar que tingui la nostra aplicació:

1. L'usuari vol que el xifrat del material és realitzi utilitzant la xifra de Vernam.
2. L'usuari ens demana que només vol introduir un pasword quan calgui xifrar / desxifrar un arxiu o una carpeta a més, aquest pasword ha de permetre desxifrar qualsevol material xifrat en qualsevol moment, també ens demana que pugui canviar el pasword quan vulgui sense afectar a la funció de desxifrat ni al material xifrat anteriorment.
3. l'usuari vol tenir la seguretat que el material xifrat no ha estat manipulat.
4. L'usuari demana que si ha xifrat tot el contingut d'una carpeta, vol tenir la possibilitat de desxifrar qualsevol dels arxius que hi ha a la carpeta sense tenir que desxifrar tot el seu contingut.
5. L'usuari ens demana que l'aplicació li permeti fer un test d'una part de la seqüència que s'ha fet servir per xifrar un determinat arxiu.

La solució que nosaltres proposem:

L'apartat 1 el resoldrem utilitzant criptografia de clau compartida amb la implementació del generador presentat.

Per resoldre l'apartat 2 utilitzarem la criptografia de clau publica i la infraestructura PKI, és a dir, l'usuari disposarà d'un certificat digital on guardarà un parell de claus que li permetran utilitzar l'aplicació tal com demana. El pasword servirà per accedir a la clau privada del certificat digital, per tant quan vulgui canviar el pasword no hi haurà cap problema . Aquesta solució ens condiciona per una banda a importar un proveïdor que implementi els algorismes per treballar amb claus RSA, nosaltres utilitzarem el proveïdor Bouncy Castle [BC1] i a utilitzar l'eina Keytool de Java per tal de construir la infraestructura PKI.

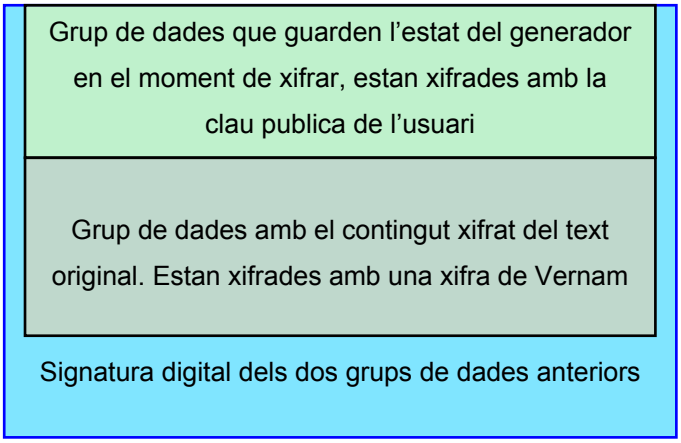
L'apartat 3 el resoldrem utilitzant la metodologia de la signatura digital aprofitant que l'usuari disposa d'un certificat digital. qualsevol producte xifrat quedarà signat, per tant, l'usuari tindrà la seguretat que el material no ha estat manipulat.
Per resoldre l'apartat 4 caldrà que cadascun dels arxius guardi l'estat del xifrador en el moment que va ser xifrat per tal que l'aplicació el pugui recuperar-lo i efectuar el desxifratge. Aquestes dades aniran xifrades amb la clau publica de l'usuari.

L'apartat 5 el resoldrem aprofitant la implementació realitzada per tal de verificar el generador proposat i implementant l'opció en la interfície de l'usuari.

A les solucions propostes s'han d'afegir els següents complements:

1. qualsevol arxiu xifrat estarà constituït per tres grups de dades:
    a. El primer grup de dades desarà l'estat del xifrador en el moment de xifrar el document.
    b. Un segon grup de dades desarà el text xifrat amb la xifra de Vernam.
    c. El tercer grup de dades estarà constituït per la signatura dels dos grups anteriors.

Figura 10. Esquema d'un arxiu xifrat



2. Tot el material xifrat conservarà el seu nom original concatenat amb el sufix **_xif** , aquest sufix permetrà a l'aplicació distingir el material xifrat per tal d'evitar xifrar-lo més d'un cop o bé desxifrar-lo.
3. Tot el material xifrat conservarà el seu Path, per tant, quan recuperem material xifrat caldrà tenir en compte que si els arxius originals no van ser destruïts seran reescrits.
4. Nosaltres com a proveïdors i mantenidors de l'aplicació i d'acord amb l'usuari generarem un estat inicial per al xifrador. Aquestes dades és desaran xifrades i signades a arxiu, per tal de tenir garanties que l'arxiu és correcte. Aquest arxiu s'anirà actualitzant cada cop que l'aplicació es tanqui o finalitzi una sessió de xifrat.

Donem per finalitzada la proposta de la nostra solució. Pot semblar que hem deixat massa clar com funcionarà tot plegat, però si fem cas de la **suposició de kerckhoff**, l'únic que cal vigilar és que l'usuari no comprometi el seu pasword d'accés al certificat ja que encara que pugui tenir un accés no autoritzat a la seva màquina, el desxifrat dels arxius o el desxifrat de l'estat del generador no és possible actualment, sense tenir el pasword del certificat.

Tal com vam comentar caldrà avaluar la velocitat de l'aplicació en el moment de xifrar i desxifrar i notificar-ho a l'usuari per tal que sigui conscient que l'aplicació necessita una certa quantitat de temps per dur a terme la seva gestió.

2.8 Altres productes i solucions del mercat

Al mercat podem trobar una gamma diversa de productes destinats al xifratge de la informació, tant de pagament, com de lliure distribució.

Per als usuaris de Windows XP, el propi sistema operatiu incorpora la funcionalitat de xifrar/desxifrar arxius i carpetes, sempre que facin servir una partició de disc NTFS. Cal tenir en compte que els arxius xifrats no es poden comprimir amb la mateixa funcionalitat de Windows, fet que implica fer servir un producte extern per a comprimir els fitxers xifrats. http://support.microsoft.com/kb/308989/es

Windows Vista en el seu servide Pack 1 incorpora ell generador de nombres aleatoris **Dual_EC_DRGB**, però sembla que ha suscitat controvèrsies degut a què és un generador que utilitza corbes el·líptiques creades amb constants. http://www.techtear.com/2007/12/18/polemico-generador-de-numeros-aleatorios-en-el-sp1-de-vista/

També podem trobar motors criptogràfics (proveïdors) que implementen un generador de nombres, per exemple el generador **VIA Padlock RNG**, amb aleatorietat quàntica integrada. És un generador basat en el propi hardware del processador, concretament utilitza el soroll elèctric del xip del processador. http://es.viatech.com/es/initiatives/padlock/index.jsp

Dintre del software de pagament disponible al mercat, un dels de més pes potser és el Security Suite 7 de Steganos, el qual permet, entre altres funcionalitats, xifrar arxius i carpetes fent servir l'algoritme AES de 256 bits, i és compatible amb processadors de 64 bits. http://www.steganos.com/es/

Dintre de les aplicacions gratuïtes la més coneguda és PGP. Permet xifrar arxius i carpetes. Aquest programa fa servir un xifratge asimètric. http://www.pgp.com

Dintre dels programes que fan servir xifratge simètric, la més interessant potser és CrypText 3.4 És molt fàcil d'instal·lar i executar, algunes de les seves característiques són: accés des del menú contextual, xifratge simètric fent servir una

combinació dels algorismes RC4 i SHA-1 amb una clau de 160 bits,. http://cryptext.archivospc.com/

Com a producte didàctic i gratuït tenim Cryptool , incorpora eines criptogràfiques i mòduls d'aprenentatge, permet xifrar i desxifrar, realitza test d'entropia i correlació. http://www.secude.es/services/it-security/cryptool_es.php

3 Anàlisi i disseny de l'aplicació

En l'apartat corresponent a la proposta de solució ja vam presentar les funcionalitats que ha de tenir la nostra aplicació.

Per efectuar el disseny i la corresponent implementació d'aquesta aplicació analitzarem i dissenyarem els mòduls principals i els anirem relacionat per mitja d'un element comú per tal d'obtenir l'estructura final.

Els mòduls a considerar son:

- Mòdul Generador.
- Mòdul Test.
- Mòdul Xifrador/desxifrador.
- Mòdul Interfícies.

3.1 Disseny del Mòdul Generador.

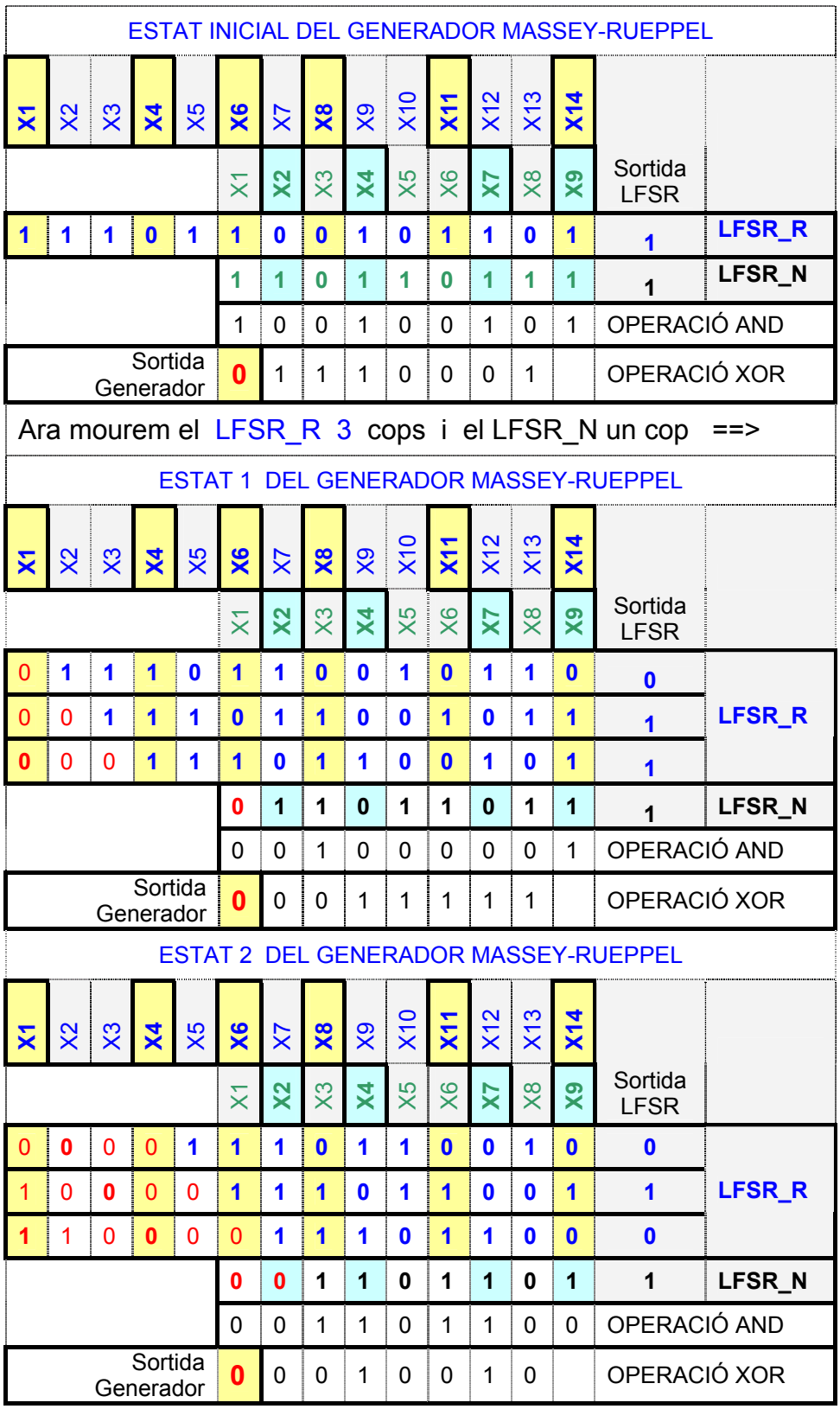
El generador de l'aplicació és un agrupament de tres generadors del tipus Massey-Rueppel on les seves sortides es combinen per mitja de l'algoritme presentat.

En l'apartat corresponent de l'estat de l'art ja vam donar un esquema detallat de com funciona un generador de Massey-Rueppel. Per tal de justificar els atributs que farem servir, dibuixarem la taula de l'estat inicial i dels dos estats següents, en aquest cas el generador té els següents paràmetres:

- Polinomi del LFSR_R (el més ràpid) 100100101010011 (grau14)
- Estat Inicial del LFSR_R 11101100101101

- Velocitat del LFSR_R 3
- Polinomi del LFSR_N (el més lent) 1010010101 (grau 9)
- Estat Inicial del LFSR_N 110110111
- Velocitat del LFSR_N 1

Figura 11. Taula d'estats del generador Massey-Rueppel.



ESTAT INICIAL DEL GENERADOR MASSEY-RUEPPEL															
X1	X2	X3	**X4**	X5	**X6**	X7	**X8**	X9	X10	**X11**	X12	X13	**X14**	Sortida LFSR	
					X1	**X2**	X3	**X4**	X5	X6	**X7**	X8	**X9**		
1	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**LFSR_R**
					1	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**LFSR_N**
					1	0	0	1	0	0	1	0	1	OPERACIÓ AND	
				Sortida Generador	**0**	1	1	1	0	0	0	1		OPERACIÓ XOR	

Ara mourem el LFSR_R 3 cops i el LFSR_N un cop ==>

ESTAT 1 DEL GENERADOR MASSEY-RUEPPEL															
X1	X2	X3	**X4**	X5	**X6**	X7	**X8**	X9	X10	**X11**	X12	X13	**X14**	Sortida LFSR	
					X1	**X2**	X3	**X4**	X5	X6	**X7**	X8	**X9**		
0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	
0	0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**LFSR_R**
0	0	0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	
					0	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**LFSR_N**
					0	0	1	0	0	0	0	0	1	OPERACIÓ AND	
				Sortida Generador	**0**	0	0	1	1	1	1	1		OPERACIÓ XOR	

ESTAT 2 DEL GENERADOR MASSEY-RUEPPEL															
X1	X2	X3	**X4**	X5	**X6**	X7	**X8**	X9	X10	**X11**	X12	X13	**X14**	Sortida LFSR	
					X1	**X2**	X3	**X4**	X5	X6	**X7**	X8	**X9**		
0	**0**	0	0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	
1	0	**0**	0	0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**LFSR_R**
1	1	0	**0**	0	0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	
					0	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**LFSR_N**
					0	0	1	1	0	1	1	0	0	OPERACIÓ AND	
				Sortida Generador	**0**	0	0	1	0	0	1	0		OPERACIÓ XOR	

Tal com podem observar el valor dels primers 3 bits de sortida és 000.

En aquest mòdul farem servir un diagrama estàtic i definirem totes les classes que són necessàries per tal de representar l'estructura del nostre generador.

Per tal d'implementar el funcionament ens caldrà definir la classe **Lfsr** , la classe **MasseyRueppel,** la classe **ElNostreGenerador** i la classe **Principal** que serà l'element d'unió entre els diferents mòduls. Inicialment els atributs i mètodes amb més rellevància d'aquestes classes son:

Classe **Lfsr:**

Aquesta classe permet construir objectes que simulen el comportament d'un LFSR. Estableix una relació d'agregació amb la classe Massey-Rueppel ja que cadascun dels generadors, esta format per dos LFSR.

Atributs

- taulaIdCelesActives : int[] Taula per desar les cel·les actives d'un LFSR.
- totalCelesActives : int Atribut per guardar en nombre de cel·les actives.
- matriu : boolean [] Taula per desar l'estat de totes les cel·les.
- estatInicial : boolean [] Taula per desar l'estat inicial.
- velocitat : int Atribut per desar la velocitat de funcionament.
- grau : int Atribut per desar el grau.
- període : Integer Atribut per desar el període = (2^grau)-1.
- polinomi : String Atribut per desar el polinomi primitiu en format binari (1010....1).
- idCelaX1 : int Atribut per desar l'índex de la matriu amb el valor de la cel·la X1.
- idCelaXn : int Atribut per desar l'índex de la matriu amb el valor de la cel·la Xn .
- posInicial : int Atribut per reservar el primer índex de la matriu amb el valor de la cel·la X1.
- posFinal : int Atribut per desar el primer índex de la matriu amb el valor de la cel·la Xn .

Mètodes

Els mètodes per la funcionalitat son:

+lfsr() constructor sense paràmetres, no executa cap acció.

+lfsr(polinomi:String, velocitat:int) constructor, realitza els càlculs necessaris per iniciar tots els atributs menys l'estat inicial.

+lfsr(polinomi:byte[], velocitat:byte) constructor, realitza els càlculs necessaris per iniciar tots els atributs menys l'estat inicial.
init (estatInicial:String) carrega la matriu amb l'estat inicial i el desa al seu atribut.
estatSegüent() : boolean[] aquest mètode implementa el moviment, tants cops com val l'atribut velocitat, dels valors del LFSR i retorna l'estat assolit.
getEstatActualBoolean() : boolean[] Retorna l'estat actual del LFSR.
reset() Aquest mètode carrega el valor de l'atribut estatInicial en l'atribut matriu
buscarPrimitiu(int grau) Calcula els polinomis primitius de grau igual al paràmetre.

Classe **GeneradorMasseyRueppel:**

Aquesta classe permet construir objectes que simulen el comportament d'un generador de Massey-Rueppel. Estableix una relació d'agregació amb la classe ElNostreGenerador ja que ElNostreGenerador esta format per tres generadors Massey-Rueppel.

Atributs

- lfsr_R : Lfsr Atribut per desar un objecte Lfsr amb velocitat >= 2. que la de lfsr_N.
- lfsr_N : Lfsr Atribut per desar un objecte Lfsr.
- període : Integer Atribut per desar el període = (2^grau_R) - 1 * (2^grau_N) - 1
- comptador : int Atribut per acumular el nombre de bits generats.
- ID_BN : int Atribut que determina en quina iteració del funcionament del generador tenim que capturar el seu l'estat intern idbn =. grau_N - 2

Mètodes

Els mètodes per la funcionalitat son:

+masseyRueppel(pR:String, pN:String, vR:int vN:int) constructor, realitza els càlculs necessaris per iniciar tots els atributs, rep per paràmetre els polinomis i les velocitats dels seus LFSR.
+masseyRueppel(pR:byte[], pN: byte[], vR: byte[] vN: byte[], comptador:byte) constructor, realitza els càlculs necessaris per iniciar tots els atributs, rep per paràmetre els polinomis, les velocitats dels seus LFSR i el valor d'inici del comptador de bits generats.
init (eR:String, eN:String) carrega el mètode init(estatInicial:String) de cadascun dels LFSR amb els paràmetres rebuts.

init (eR: byte[], eN: byte[]) carrega el mètode init(estatInicial: byte[]) de cadascun dels LFSR amb els paràmetres rebuts.

getBit() : boolean aquest mètode implementa el moviment del generador, retorna el bit generat, incrementa el comptador i actualitza el valor **bn**.

reset() Aquest mètode executa el reset() dels LFSR i inicia l'atribut comptador =0.

Classe **ElNostreGenerador:**

Aquesta classe permet construir l'objecte que simularà el comportament del generador proposat .

Atributs

- ms_1 : MasseyRueppel Atribut per desar un objecte MasseyRueppel
- ms_2 : MasseyRueppel Atribut per desar un objecte MasseyRueppel
- ms_3 : MasseyRueppel Atribut per desar un objecte MasseyRueppel
- període : BigInteger Atribut per desar el període = (periode_ms_1)+ (periode_ms_2)+ (periode_ms_3)
- bn: boolean Atribut per desar el valor **bn** del generador Massey-Rueppel que ha generat l'ultim bit,

Mètodes

Els mètodes per la funcionalitat son:

+ElNostreGenerador(g1:String[], g2:String[], g3:String[],) constructor, realitza els càlculs necessaris per iniciar tots els atributs, rep per paràmetre les taules amb el contingut dels atributs de cadascun dels generadors Massey-Rueppel.

+ElNostreGenerador(p1:byte[], p2:byte[], p3:byte[], p4:byte[], p5:byte[], p6:byte[], p7:byte[]) constructor, realitza els càlculs necessaris per iniciar tots els atributs, rep per paràmetre els polinomis i velocitats de cadascun dels LFRS que formen part dels generadors dels atributs de cadascun dels generadors Massey-Rueppel.

init (g1:String[], g2:String[], g3:String[],) crida el mètode init (estatInicialR:String, estatInicialN:String) de cadascun dels generadors Massey-Rueppel amb els paràmetres rebuts.

getBit() : boolean aquest mètode implementa l'algoritme de mescla proposat. Retorna el bit generat per un dels 3 generadors.

reset() Aquest mètode executa el reset() dels objectes MasseyRueppel i inicia l'atribut bn =true.

resetComptadors() Permet modificar el comptador de bits generats de cadascun dels 3 generadors:

* Al generador amb menys bits generats se li adjudicarà el valor zero.

* Als altres dos sé lis adjudicarà el valor de la diferencia entre el seu comptador i el comptador del generador amb menys bits generats.

Classe **Principal:**

Aquesta Classe anirà evolucionant al llarg de tot el disseny. Permet construir l'objecte que simularà el comportament general de l'aplicació.

Atributs

#generador : ElNostreGenerador Atribut per desar l'objecte ElNostreGenerador, el fem protegit static per tal d'accedir des dels altres mòduls

Mètodes

Els mètodes per la funcionalitat son:

+principal() constructor de la classe, construeix un objecte ElNostreGenerador

El diagrama de Classes per aquest mòdul és :

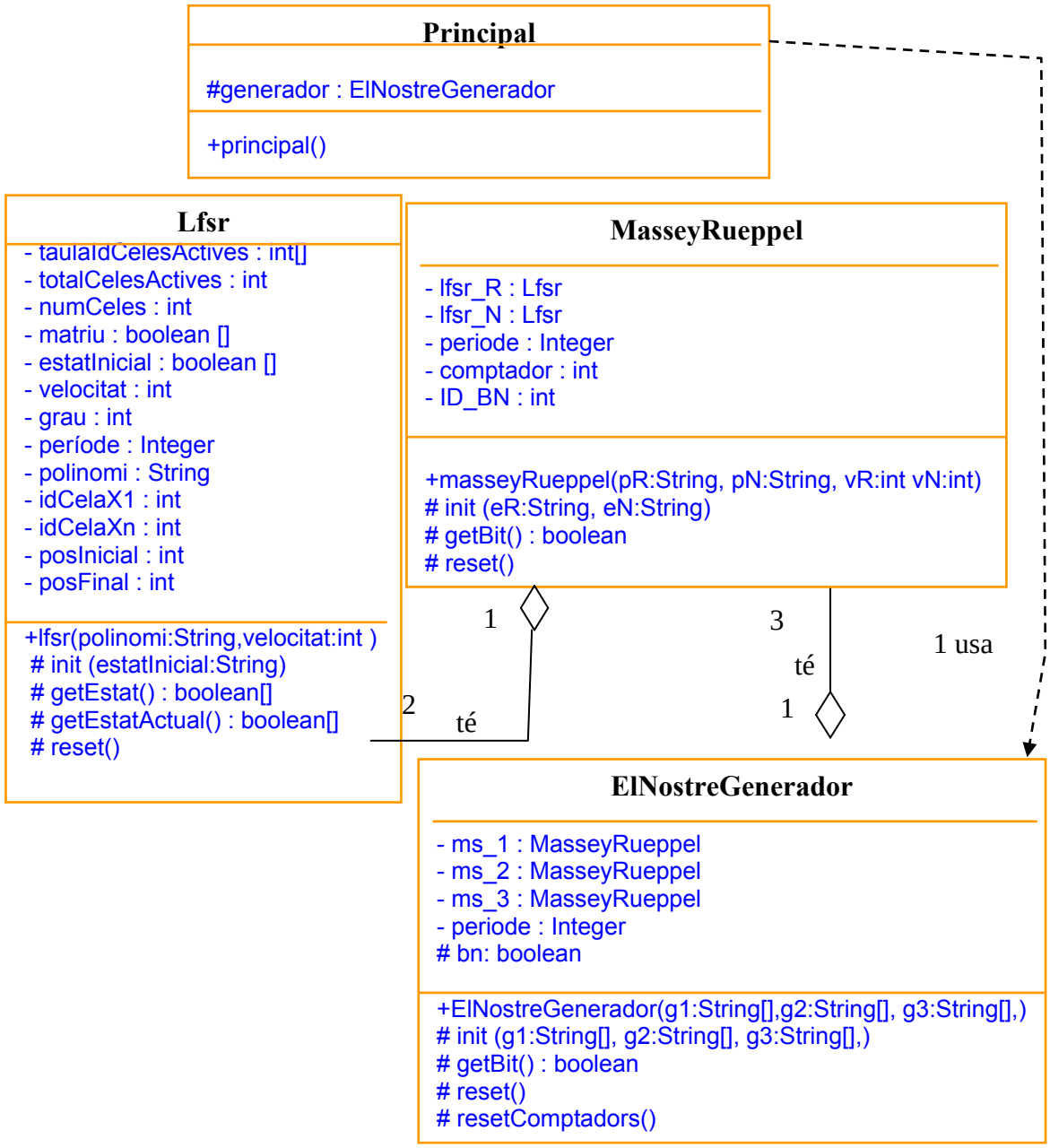


Figura 2. Diagrama de classes del mòdul generador.

3.2 Disseny del Mòdul Test.

Centrarem el disseny dels tests pensant en els problemes d'espai que podem tenir quan les seqüències analitzades tenen molta allargada.

Una solució, és analitzar els bits en el moment de ser generats sense tenir que desar-los en lloc. És clar que, depenen del test, caldrà desar un determinar nombre de bits. Com a constants generals definirem :

- LONG_SEQUENCIA = quantitat de bits per generar o analitzar.
- MAX_BITS_DESAR = quantitat de bits generats o analitzats que volem desar en un arxiu o taula , on MAX_BITS_DESAR ≤ LONG_SEQUENCIA

Així doncs, només acumularem els bits per als següents tests:

- **Test de correlació.**
  Per realitzar aquest test, cal acumular tants bits com el paràmetre **d** i després efectuar la corresponent comparació amb la resta de bits generats, donat que aquest paràmetre pot assolir un valor = LONG_SEQUENCIA - 1, fixarem el seu màxim amb la constant MAX_BITS_DESAR.
- **Test de correlació de Golomb.**
  Aquí cal acumular tants bits com el paràmetre **k** i després efectuar la corresponent comparació amb tota la seqüència, Però donat que ens interessa el valor de la funció AC(k) per a tot k <= LONG_SEQUENCIA, el que farem és fixar el valor màxim que pot assolir k amb la constant MAX_BITS_DESAR.
- **Test del perfil de la complexitat lineal.**
  En aquest test també hem d'acotar el nombre de bits que volem que passin el test, cal recordar que en cal analitzar 2 períodes de la seqüència per tal d'analitzar el perfil de la complexitat lineal. Per tant, el nombre de bits per acumular serà <= (2* MAX_BITS_DESAR)+10 només quan la seqüència superi els 3000 bits.
- **Test de Maurer.**
  En aquest test, tampoc acumularem els bits. Considerem que és millor generar de nou la seqüència per a cada longitud de bloc que vulguem avaluar.

  Per tant, com ens interessa passar el test per a la màxima longitud de bloc que el període del generador permeti, el que farem serà fer un reset() del generador per a cada possible valor L
  Per als blocs de longitud L=6 tenim: Q= 640 blocs i K= 64000 blocs., això implica generar (640*6)+(64000*6) = 387.840 bits, i per als blocs de longitud L=

16 tenim: Q= 655.360 blocs K= 65.536.000 blocs, llavors caldrà generar (655.360*16)+(65.536.000*16) = 1.059.061.760 bits, és a dir, aproximadament 2.731 cops els bits necessaris per a blocs de L= 6, és a dir, aquest test ens ocuparà bastant de temps .

El disseny esta orientat a l'objecte, en aquest cas cadascun dels tests son objectes que reben missatges i executen alguna funció, per tant les classes que caldrà implementar son:

- **Classe TestFreqüències.**
- **Classe TestSeries**
- **Classe TestPoker**
- **Classe TestRàfegues.**
- **Classe TestCorrelació.**
- **Classe TestCorrelacióGolomb**
- **Classe TestComplexitatLineal.**
- **Classe TestMaurer.**
- **Classe PrincipalTest.**

La idea és que el generador generi els bits i els objectes de les classes els vagin analitzant sense acumular-los en lloc, només acumularem els bits mínims que siguin necessaris per al funcionament precís del test, això ens permetrà analitzar seqüències de milers de milions de bits.

Donat que les classes dels tests tenen que implementar les formules detallades l'annex nº2, en els apartats corresponents només comentarem que han de tenir els atributs necessaris per poder implementar les formules i com a mètodes comuns considerarem :

setBit_1(idBit:int) ; Per decidir que fer quan és rep un 1.
setBit_0(idBit:int) Per decidir que fer quan es rep un 0.
getResultat():String Per retornar el resultat del test.
La resta quedarà detallat en la pròpia implementació.

Definirem ara els atributs i mètodes de la classe PrincipalTest i tot seguit realitzarem un diagrama de seqüències per tal d'entendre el funcionament de tot plegat.

Classe **PrincipalTest:**

Aquesta classe implementarà l'execució de tots els tests. per fer-ho establirà relacions d'agregació amb la resta de les classes del mòdul menys amb la classe TestMaurer, ja que aquesta accedirà al generador directament. en canvi les altres rebran els bits del generador des de la classe PrincipalTest.

Atributs

- tRafegues : TestRafegues Objecte per efectuar el test de Ràfegues.
- tPoker : TestPoker Objecte per efectuar el test de Poker.
- tFrequencia : TestFrequencia Objecte per efectuar el test de Freqüència
- tSeries : TestSeries Objecte per efectuar el test de Series
- tCorrelacio : TestCorrelacio Objecte per efectuar el test de Correlació
- tCorrelacioGolomb : TestCorrelacioGolomb Objecte per efectuar el test de la correlació de Golomb.
- tComplexitat : TestComplexitat Objecte per efectuar el test del perfil de la complexitat lineal
- desarDades : boolean permet determinar si cal desar les dades dels tests en arxius.
- testPoker : boolean permet determinar si s'ha de fer el test del poker.
- testCorrelacio : boolean permet determinar si s'ha de fer el test de correlació.
- longitudSeqüencia :int Atribut per guardar la longitud de la seqüència a analitzar.
- maxBitsDesar :int Atribut comptabilitzar els bits que volem desar.
- idBit :int Atribut comptabilitzar els bits analitzats.

Mètodes

Els mètodes per la funcionalitat son:

+principalTest(area: InterficieText, desarDades:booelan, int longBloc) constructor, inicialitza tots els objectes identificats pels seus atributs, més una instanciació de la classe TestMaurer a la que li passa l'atribut longBloc. Rep per paràmetre l'objecte on s'escriuran els resultats dels diferents tests. Demanarà al generador tants bits com LONG_SEQUENCIA, i acumularà en una taula tants bits com MAX_BITS_DESAR.

En funció de la longitud de la seqüència decideix si cal fer el test de poker(la seqüència >=10) o el test de correlació (seqüència >8).

-ferTest(area: InterficieText) Aquest mètode realitza els tests, crida a la resta de mètodes.

- analitzaPrimerBit () Amb aquest mètode demanen el primer bit inicialitzem els tests.

- analitzaBit () Aquest mètode s'ha d'encarregar de demanar un bit al generador i enviar un missatge amb el resultat a la resta d'objectes

-imprimirResultats(resultats : InterficieText) Aquest mètode s'encarregarà d'imprimir els resultats obtinguts dels diferents tests en l'objecte rebut per paràmetre.

Classe **Principal:**

La classe principal quedarà modificada de la següent manera.

Atributs

#generador : ElNostreGenerador Atribut per desar l'objecte ElNostreGenerador.

#LONG_SEQUENCIA : integer Constant que guardarà el valor màxim de la seqüència a generar.

#MAX_BITS_DESAR: int Constant que guardarà el nombre màxim de bits per desar.

Mètodes

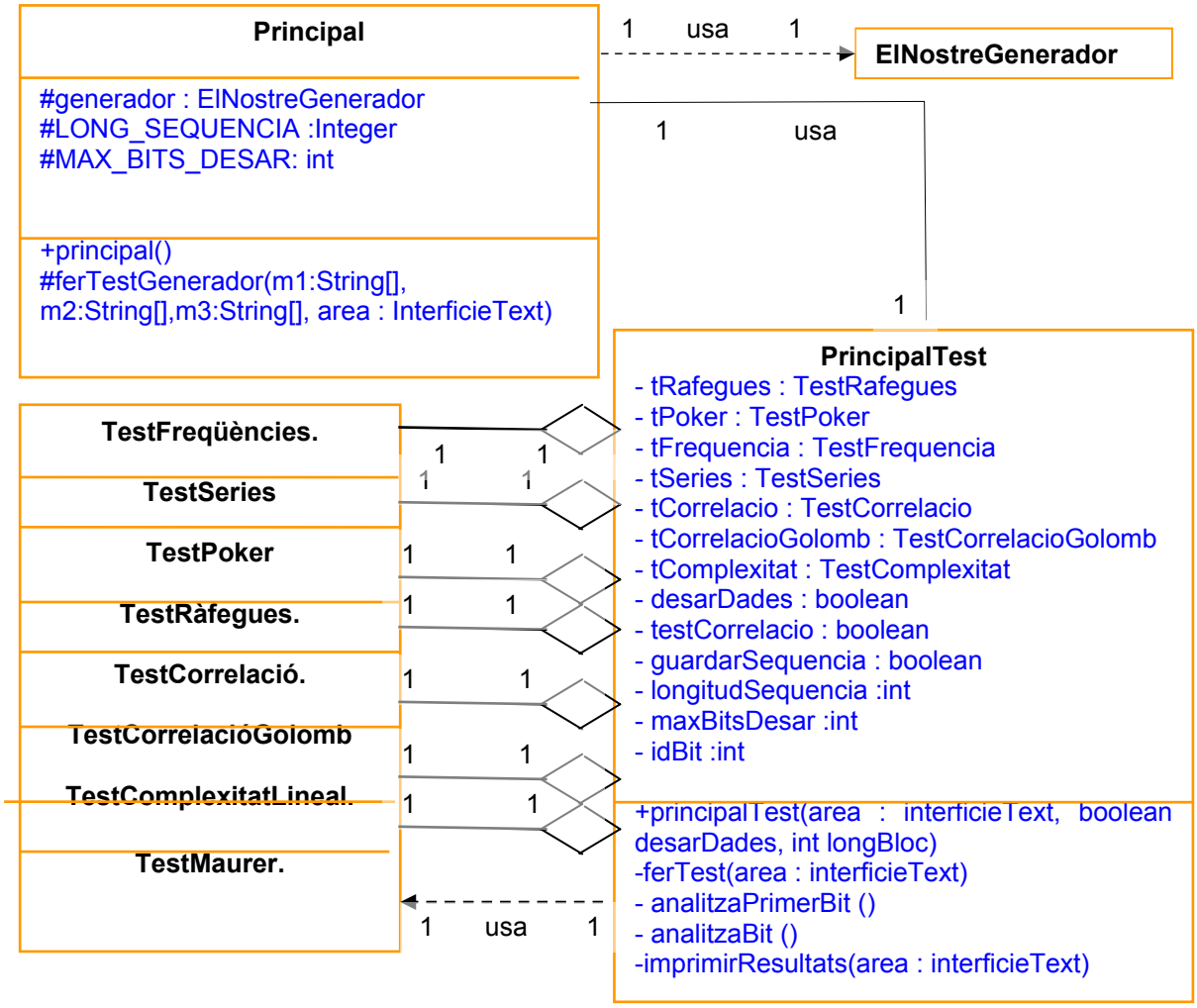
Els mètodes per la funcionalitat son:

+principal() constructor de la classe

#ferTestGenerador(m1:String[], m2:String[], m3:String[], area:InterficieText) Aquest mètode permet crear un objecte ElNostreGenerador per generar la seqüència, inicialitza les constants i utilitza una instància de la classe PrincipalTest, per realitzar els tests, imprimeix els resultats a l'area de text

El diagrama de Classes per aquest mòdul és :

Figura 13. Diagrama de classes del mòdul tests.



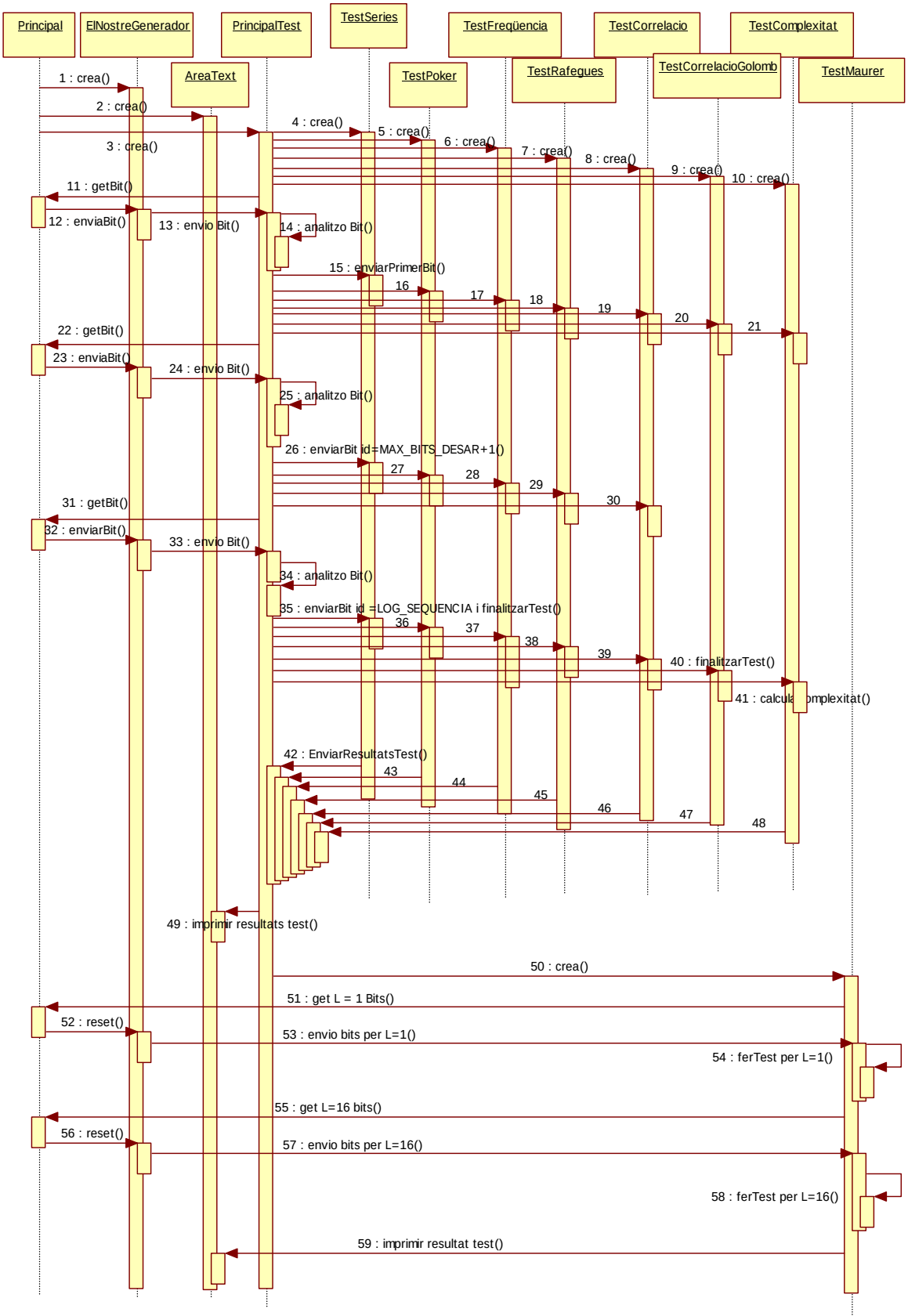
Plantejarem tot seguit un guio per aquest mòdul i generarem un diagrama de seqüències:

- Principal i PrincipalTest creen una instància de classe.
- PrincipalTest demana a Principal el primer bit del generador, determina si és un 1 o un 0 i envia el mateix missatge a tots els seus objectes, cadascun dels objectes en funció de la seva implementació realitza les tasques adients.
- PrincipalTest demana a Principal el bit numero MAX_BITS_DESAR.+1 del generador, determina si és un 1 o un zero i envia el mateix missatge a tots els

seus objectes menys a l'objecte TestCorrelacioGolomb i a l'objecte TestComplexitat. Així fins arribar a LONG_SEQUENCIA

- PrincipalTest imprimeix els resultats d'aquests tests.
- PrincipalTest crea l'objecte TestMaurer.
- L'objecte TestMaurer executa les seves operacions per a tots els valors L = longBloc i imprimeix el resultat del test per a cada longitud de bloc.

Figura 14. Diagrama de seqüencies del mòdul tests.



3.3 Mòdul Xifrador/desxifrador.

Per al disseny d'aquest mòdul considerarem els requisits plantejats en l'apartat de requeriments.

Per una banda volem xifrar i desxifrar informació utilitzant una xifra de Vernam, per tant crearem la classe **Vernam** per simular la Xifra de Vernam.

Un dels requisits, ens demana que l'usuari només tingui que introduir una contrasenya en el moment de xifrar i que el pugui modificar quan vulgui sense que això afecti al material que ja havia estat xifrat, tal com vam dir, això ho solucionarem fent servir la criptografia de clau publica, per tant aprofitarem aquest fet per cobrir un dels altres requisits que ens demana tenir la seguretat que el material xifrat no ha estat modificat posteriorment, per tant caldrà signar-lo i més tard verificar la signatura. Tot això ho tractarem en la classe **UtilitatsRSA.**

Un altre requisit ens demana que l'usuari pugui fer el test a una part dels bits generats per xifrar els seus documents, per satisfer aquest requisit, el que farem és capturar a un arxiu tants bits generats com MAX_BITS_DESAR , llavors l'usuari podrà triar l'opció per fer el test d'aquesta seqüència. Aquest requisit el tractarem a la classe **LaNostraSequencia**.

Per coordinar les associacions entre les diferents classes, farem servir la classe **Principal,** per tant caldrà afegir-hi els corresponents mètodes per realitzar aquesta tasca.

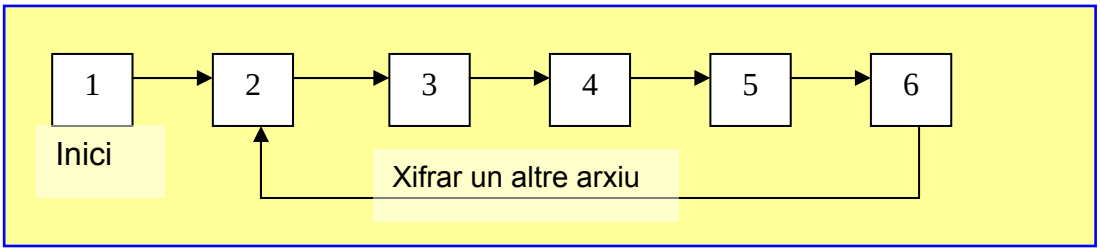
Tot seguit farem un guio de con ha d'anar el procés de xifrat i desxifrat.

Procés de Xifrat

1. Carregar l'estat actual del generador. Cal obtenir-lo des del seu document auxiliar, caldrà, doncs, verificar la signatura d'aquest document amb la clau publica RSA i desxifrar el contingut amb la clau privada RSA.
2. Obtenir les dades que volem xifrar i xifrar-les amb la Xifra de Vernam.
3. Carregar l'estat actual del generador a la capçalera del document de sortida. Cal xifrar-lo prèviament amb la clau publica RSA.

4. Signar el resum del conjunt de dades més capçalera amb la clau privada RSA.
5. Carregar les dades xifrades al document de sortida.
6. Carregar al document auxiliar del generador, l'estat actual del generador en finalitzar el xifrat d'un document . Aquestes dades han d'anar xifrades amb la clau publica i signades amb la clau privada. Mentre l'aplicació no canviï l'operació (passar de xifrar a desxifrar i després tornar a xifrar) , l'estat actual del generador no cal carregar-lo.

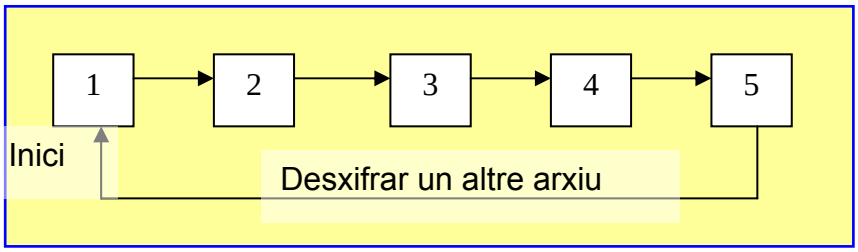
Figura 15. Esquema del procés de xifrat.



Procés de Desxifrat.

1. Obtenir les dades que volem desxifrar.
2. Verificar la Signatura del document amb la clau publica RSA.
3. Obtenir, des de la capçalera del document, l'estat del generador i carregar-lo. Cal desxifrar-lo amb la clau privada.
4. Desxifrar les dades amb la xifra de Vernam.
5. Muntar l'arxiu de sortida.

Figura 16. Esquema del procés de desxifrat.



Tot seguit especificarem els Atributs i mètodes de les diferents classes que són necessàries per a dur a terme el procés de xifrat i desxifrat.

Classe **Vernam:**

Aquesta classe implementa els mètodes necessaris per aplicar l'operació XOR a les dades .

Atributs

Cap

Mètodes

Els mètodes per la funcionalitat son:

#ferXOR(byte[] : entrada) : byte[] mètode per realitzar l'operació XOR, bit a bit , entre els bits generats pel generador i els bits dels Bytes de la taula d'entrada.

Classe **UtilitatsRSA:**

Aquesta classe implementa els mètodes necessaris per treballar amb les eines de la criptografia de clau publica.

Atributs

-cipher : Chiper és el xifrador/desxifrador per utilitzar la clau privada/publica

-sig : Signature és l'objecte que ens permet signar i verificar els documents.

- privada : RSAPrivateKey és l'objecte que desarà la clau privada.

- publica : RSAPublicKey és l'objecte que desarà la clau publica

- ks : KeyStore és l'objecte que ens permetrà accedir al magatzem de claus.

- aliasCertificat : String és el alias per accedir al certificat..

Mètodes

Els mètodes per la funcionalitat son:

+UtilitatsRSA(contrasenyaKeyStore:String, aliasCert:String, URLKeystore:String) constructor, inicialitza tots els atributs menys la clau privada.

#setPrivada(pasword: String) obté la clau privada del magatzem de claus per mitja de l'atribut passat per paràmetre i al carrega en el paràmetre privada.

#setPublica(pasword: String) obté la clau publica del magatzem i la carrega en el paràmetre pública.

#xifrarRSA(dades : byte[]) : byte[] Mètode per xifrar les dades rebudes per paràmetre amb la clau publica.

#desXifrarRSA(dades : byte[]) : byte[] Mètode per desxifrar les dades rebudes per paràmetre amb la clau privada.
#signarDocument(serialitzada : Serialitzada) : SignedObject, signa l'objecte passat per paràmetre amb un resum fet amb SHA1 i la clau privada i el retorna.
#verificarSignatura(objecteSignat : SignedObject) : bolean, verifica la signatura del document passat per paràmetre amb un resum fet amb SHA1 i la clau publica i retorna el resultat de l'operació.

Classe **LaNostraSequencia:**

Aquesta classe implementa la funcionalitat per poder llegir una seqüència de bits des d'un arxiu. Hereta de la classe ElNostreGenerador i sobreescriu els mètodes getBit() i reset()

Atributs

-taulaSequencia: boolean[] és la taula que guardarà la seqüència.
- idBit : int és l'índex del bit llegit.

Mètodes

Els mètodes per la funcionalitat son:

+LaNostraSequencia(seqüència : boolean[]) constructor, Rep per paràmetre la taula de boolean amb el contingut de la seqüència. Inicialitza la classe pare i els diferents Atributs.
#getBit() :boolean obté un bit de la taula i el retorna a qui l'ha demanat.
#reset() reinicia el valor de l'atribut idBit.

Classe **Principal:**

A La classe principal afegirem els següents mètodes i atributs.

Atributs

- serialitzada : Serialitzada objecte per serialitzar les dades.
-path : String Atribut per desar el path del material a desxifrar o xifrar.
-volemXifrar: Boolean Atribut per notificar que volem fer una operació de xifrat.
-volemDesxifrar: Boolean Atribut per notificar que volem fer una operació de desxifrat.

Mètodes

Els mètodes afegits per la funcionalitat son:

#ferTestSequencia(seqüència: boolean[], area InterficieText) Aquest mètode permet crear un objecte ElNostreGenerador instanciant un objecte LaNostraSequencia inicialitza les constants i utilitza una instància de la classe PrincipalTest, per realitzar els tests, rep per paràmetre l'area de text on imprimirà els resultats.

#ferTestXifrador(area InterficieText) Aquest mètode permet fer un test a una seqüència generada per ElNostreGenerador amb la configuració que utilitza per xifrar arxius, així podem observar com son les seqüencies que utilitzem per xifrar.

#iniciarProces(llistaObjectes: List) Aquest mètode processa el material arrossegat per l'usuari al damunt de la interfície per desxifrar o xifrar, s'encarrega de fer el recorregut per les carpetes i de cridar als mètodes corresponents per tal d'obtenir un producte xifrat o desxifrat.

-iniciarGenerador() tal com vam detallar en l'apartat de la proposta de solució, tot arxiu xifrat porta incorporada una capçalera que correspon a l'estat del generador en el moment de xifrar-lo, per tant, aquest mètode s'encarrega de recuperar aquesta capçalera per iniciar el generador.

-guardarEstatGenerador() Aquest mètode serveix per desar l'estat del generador després de realitzar una operació de xifrat. Tal com vam dir, cal tenir un estat inicial per al moment que l'aplicació comença una nova sessió de xifrat, per tant l'aplicació disposarà d'un arxiu auxiliar (base de dades) on desarà l'estat del generador en el moment de finalitzar un xifrat per tal de recuperar-lo en el següent xifrat, és a dir, això ens assegurà que les seqüències de xifrat sempre seran originals fins que el generador esgoti el seu període.

-desxifrar(arxiuEntrada:File, arxiuSortida:File) Aquest mètode s'encarregarà de fer les crides i comprovacions necessàries per tal de verificar i desxifrar l'arxiu d'entrada passat per paràmetre i desar-lo a l'arxiu de sortida passat per paràmetre.

-xifrar(arxiuEntrada:File, arxiuSortida:File) Aquest mètode s'encarregarà de fer les crides i comprovacions necessàries per tal de signar i xifrar l'arxiu d'entrada passat per paràmetre i desar-lo a l'arxiu de sortida passat per paràmetre.

3.4 Mòdul Interfícies

Arribats a aquest punt, podem dir que pràcticament la funcionalitat de l'aplicació ja esta del tot detallada, caldrà esperar a la fase d'implementació per tal de polir el disseny final.

Per tal de determinar quines interfícies necessitem, dictarem les funcionalitats que l'usuari pot trobar a la nostra aplicació:

1. Xifrar; desxifrar arxius, carpetes, dades en general, (cal introduir una contrasenya) i presentar el procés per pantalla.
2. Fer un test al generador quan esta en mode xifrador (cal introduir una contrasenya), i presentar els resultats per pantalla.
3. Modificar la contrasenya d'accés.
4. Configurar el generador a voluntat per efectuar un test del resultat i presentar els resultats per pantalla.
5. Efectuar un test a una seqüència introduïda per teclat o des d'un arxiu i presentar els resultats per pantalla.
6. Presentar l'ajuda que expliqui a l'usuari com utilitzar les diferents funcionalitats de l'aplicació.

Per tant, dissenyarem les següents interfícies gràfiques:

Interfície gràfica per realitzar les tasques de Xifrat i Desxifrat. Aquesta interfície quedarà implementada per la classe **InterficieXifrador**.

Interfície gràfica per realitzar un test al generador. Aquesta interfície quedarà implementada per la classe **InterficieElNostreGenerador**,

Interfície gràfica per realitzar un test a la seqüència. Aquesta interfície quedarà implementada per la classe **InterficieTestSequencia**.

Interfície gràfica per a què l'usuari pugi accedir a l'ajuda. Aquesta interfície quedarà implementada per la classe **InterficieAjuda.**

Barra de menús per accedir a totes les funcionalitats. Aquesta interfície quedarà implementada per la classe **InterficiePrincipal**. La barra de menús disposarà d'uns els menús permanents anomenats Arxiu, Tasques i Ajuda .

- El menú Arxiu estarà equipat amb les etiquetes anomenades :
    - Sortir Permet a l'usuari tancar l'aplicació.
    - Canviar Contrasenya Permet a l'usuari canviar la contrasenya.
- El menú Tasques estarà equipat amb les etiquetes anomenades :
    - Xifrador Menú temporal que permet a l'usuari accedir a les operacions de xifrat i desxifrat.
    - Test Seqüència Menú temporal que permet a l'usuari accedir a les operacions que és poden realitzar amb una seqüència de bits.
    - Test generador Permet a l'usuari realitzar un test al generador.

La resta de funcionalitats de la barra de menú les anirem afegint a mida que desenvolupem el disseny de les diferents interfícies.

Ens interessa que qualsevol element interactiu que sigui activat en una situació errònia generi el corresponent missatge i orienti a l'usuari en el camí correcte, per tant, analitzarem els casos d'us més rellevants que poden sorgir de la interacció de l'usuari amb l'aplicació per a cadascuna de les interfícies.

Tot seguit especificarem les funcionalitats d'aquestes interfícies i els guions d'interacció que generen i els casos d'us que considerem més interessants.

3..4.1 Interfícies gràfiques per xifrar / desxifrar i barra de menús

Funcionalitats de la barra de menús per aquesta interfície.

El menú temporal Xifrador estarà equipat amb dues etiquetes anomenades:

- Xifrar Permet a l'usuari realitzar l'operació de xifrat .
- Desxifrar Permet a l'usuari realitzar l'operació de desxifrat.
- Test xifrador Permet a l'usuari realitzar un test del generador quan esta xifrant.

Funcionalitats de la Interfície per les tasques de xifrar i desxifrar.

Per a desxifrar i xifrar els documents, ja vam avançar que volem que l'usuari arrossegui els documents i carpetes damunt la finestra d'aplicació, per tant equiparem aquesta interfície amb una àrea de text que per una banda tindrà les propietats adients per realitzar aquesta funcionalitat i per l'altra, permetrà informar a l'usuari del resultat de l'operació de xifrat o desxifrat d'un document, és a dir, permetrà a l'usuari saber el temps que s'ha trigat en fer l'operació, el nombre de bytes processats i el nom de l'arxiu.

Aquesta interfície ha de permetre a l'usuari d'introduir la contrasenya que l'autoritzi a efectuar les operacions de Xifrat i Desxifrat, per tant, l'equiparem amb un camp de text anomenat contrasenya , on l'usuari haurà d'escriure la contrasenya que li doni accés al magatzem de claus.

Guio 1: Interacció de l'usuari amb l'aplicació per realitzar la tasca de Xifrat.

L'usuari ha iniciat l'aplicació. Introdueix la contrasenya en la casella corresponent, la confirma, i per mitja de les opcions de la barra de menú activa el boto amb el nom Xifrar, tot seguit arrossegarà el document o carpeta que vol xifrar damunt la pantalla d'aplicació.

casos d'us del guio 1.

cas_1 Activar el boto de ràdio amb el nom "Xifrar"

Descripció: Permet notificar a l'aplicació que és farà una operació de Xifrat
Actor: usuari de l'aplicació
Precondició: l'usuari ha executat l'aplicació, a la pantalla hi tenim la finestra d'aplicació equipada amb una barra de menús fixos anomenats Arxiu, Tasques i Ajuda i un camp de text editable amb el nom contrasenya,
Flux normal: 1. L'usuari escriu la contrasenya. 2. L'usuari polsa la tecla enter i el sistema confirma que la contrasenya és correcte canviant l'estat de la casella a no editable. 3. L'usuari desplega el menú temporal Tasques i activa el menú temporal

amb el nom Xifrador , llavors es despleguen dos botons de radio i l'usuari activa el boto amb el nom "Xifrar"

4. L'usuari arrossega l'arxiu o carpeta que vol xifrar damunt la pantalla d'aplicació.

Flux alternatiu_1:

1. L'usuari polsa la tecla enter i el sistema per mitja d'una caixa de diàleg confirma que la contrasenya no és correcte .
2. Executar el flux normal.

Flux Alternatiu_2:

1. l'usuari activa el boto amb el nom "Xifrar"
2. El sistema obra una caixa de diàleg per notificar a l'usuari que primer ha d'introduir la contrasenya i confirmar-la
3. Executar el flux normal.

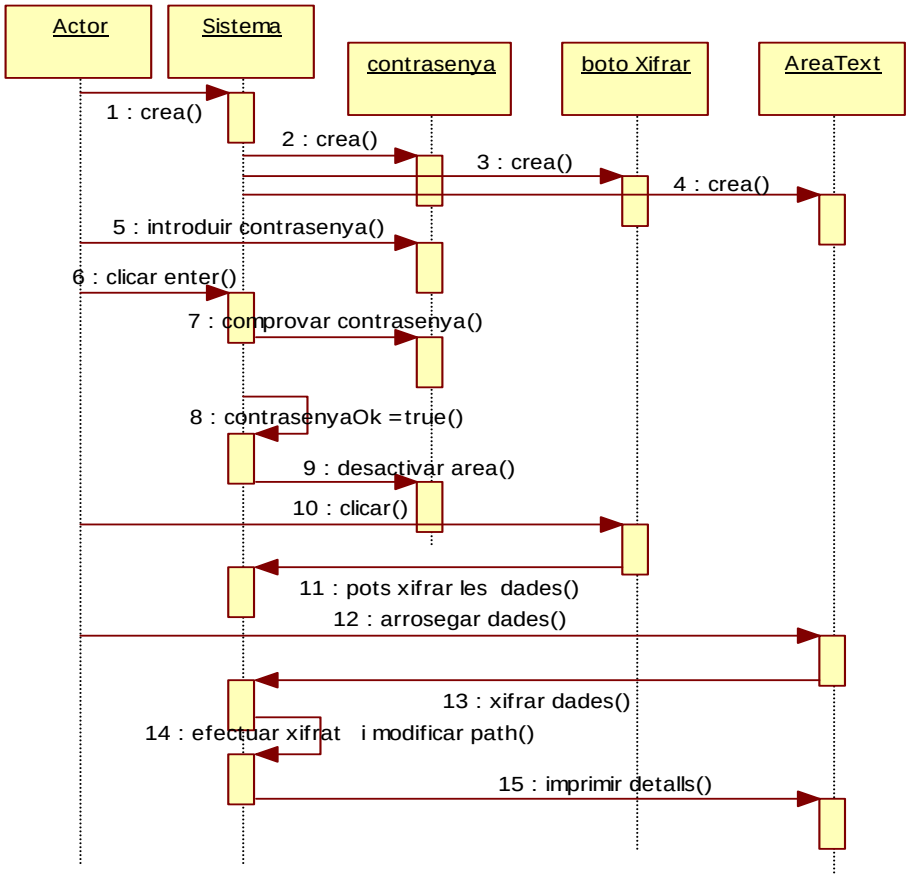
Flux Alternatiu_3:

1. L'usuari arrossega un document o carpeta damunt la pantalla d'aplicació.
2. El sistema obra una caixa de diàleg per notificar a l'usuari que primer cal introduir la contrasenya, confirmar-la i triar una operació
3. Executar el flux normal.

Poscondició:

1. El boto de ràdio Xifrar esta activat.
2. L'aplicació a xifrat el document o carpeta. S'ha generat un nou document o carpeta amb el mateix path concatenat amb el sufix **_xif**
3. A l'àrea de text hi tenim el nom original dels arxius xifrats, la seva allargada i el temps que s'ha trigat en xifrar-los.

Figura 17. Diagrama seqüència per al cas : Activar el boto de ràdio amb el nom "Xifrar", flux normal.



Guio 2: Interacció de l'usuari amb l'aplicació per realitzar la tasca de Desxifrat.

L'usuari ha iniciat l'aplicació. Introdueix la contrasenya en la casella corresponent, la confirma i per mitja de les opcions de la barra de menú activa el boto amb el nom Desxifrar, tot seguit arrossegarà el document o carpeta que vol xifrar damunt la i pantalla d'aplicació.

casos d'us del guio 2

cas_1 Activar el boto de ràdio amb el nom "Desxifrar"

En aquest cas passa exactament el mateix que pel cas d'us de clicar el boto Xifrar, la diferencia està en què el resultat final és un arxiu desxifrat amb el mateix path i sense el sufix **_xif.** i el boto de ràdio "Desxifrar" esta activat i el boto de ràdio "Xifrar" esta desactivat.

Disseny de les classes per aquesta interfície

Per tal d'implementar les funcionalitats descrites en aquest apartat, dissenyarem tot seguit la classe **InterficiePrincipal,** la classe **interficieXifrador** i la classe **interficieTest,** també afegirem a la classe **Principal** els atributs per controlar les operacions de xifrat, desxifrat.

Classe **InterficiePrincipal:**

Aquesta classe s'encarregarà d'iniciar totes les interfícies. Implementa la barra de menús.

Atributs

interficieXifrador : InterficieXifrador Objecte per implementar la interfície d'usuari per les operacions de xifrat i desxifrat

interficieElNostreGenerador : InterficieElNostreGenerador Objecte per implementar la interfície d'usuari per tal de fer un test al generador

interficieSequencia : InterficieTestSequencia Objecte per implementar la interfície d'usuari per tal de fer un test a una seqüència.

- barraMenus : JMenuBar Objecte per implementar totes les funcionalitats de la barra de menús.

- principal : JPanel panell per carregar les interfícies.

Mètodes

Els mètodes afegits per la funcionalitat són:

+InterficiePrincipal () constructor de la classe, inicialitza l'objecte Pare.

crearInterficie() Executa les diferents crides a tots els mètodes necessaris per crear la interfície amb totes les seves funcionalitats. Construeix els objectes de la resta d'interfícies.

- crearBarraMenus() Aquest mètode s'encarrega d'implementar la barra de menús.

- afegirPanels() Aquest mètode s'encarrega de muntar l'estructura de panells necessaris per la interfície.

+actionPerformed(event :ActionEvent) Aquest mètode captura la interacció de l'usuari amb la barra de menús.

Classe **InterficieXifrador:**

Aquesta classe implementa la pantalla d'aplicació per les operacions de xifrat i desxifrat, i les seves funcions són:

- Permetre a l'usuari introduir la contrasenya. i mostrar els missatges d'error si la contrasenya no és valida.
- capturar el material que l'usuari arrossega damunt la interfície per tal de desxifrar-lo o xifrar-lo, mostrant els missatges d'error convenients.
- Mostrar els resultats del procés de xifrat o desxifrat.

Atributs

- areaText : InterficieText Objecte per escriure els resultats i equipar-lo amb les propietats adients per controlar el material arrossegat a sobre d'ell.

- dropTarget : DropTarget Objecte que permet la interacció amb el material arrossegat.

-llista: List llista dels objectes arrossegats damunt la interfície.

- PannelContrasenya : JPanel Objecte per gestionar els espais destinats a la interacció amb l'usuari per tal que aquest pugui introduir la contrasenya.

- contrasenya : JTextField Objecte per a què l'usuari pugui escriure la contrasenya.

Mètodes

Els mètodes afegits per la funcionalitat son:

+InterficieXifrador () constructor de la classe, inicialitza l'objecte Pare.

crearInterficie() Executa les diferents crides a tots els mètodes necessaris per crear la interfície amb totes les seves funcionalitats.

- cearAreaPerContrasenya() Aquest mètode s'encarrega d'implementar l'espai per a què l'usuari escrigui la contrasenya.

- cearAreaPerArrosegar() Aquest mètode s'encarrega d'implementar l'espai per a què l'usuari arrossegui el material per xifrar o desxifrar.

- cearFinestraTestXifrador() Aquest mètode s'encarrega d'implementar l'espai per a mostrar el resultat de fer un test al xifrador.

- afegirPannels() Aquest mètode s'encarrega de muntar l'estructura de panells necessaris per la interfície.

getArea() : InterficieText Permet accedir a la zona d'escriptura de la interfície.

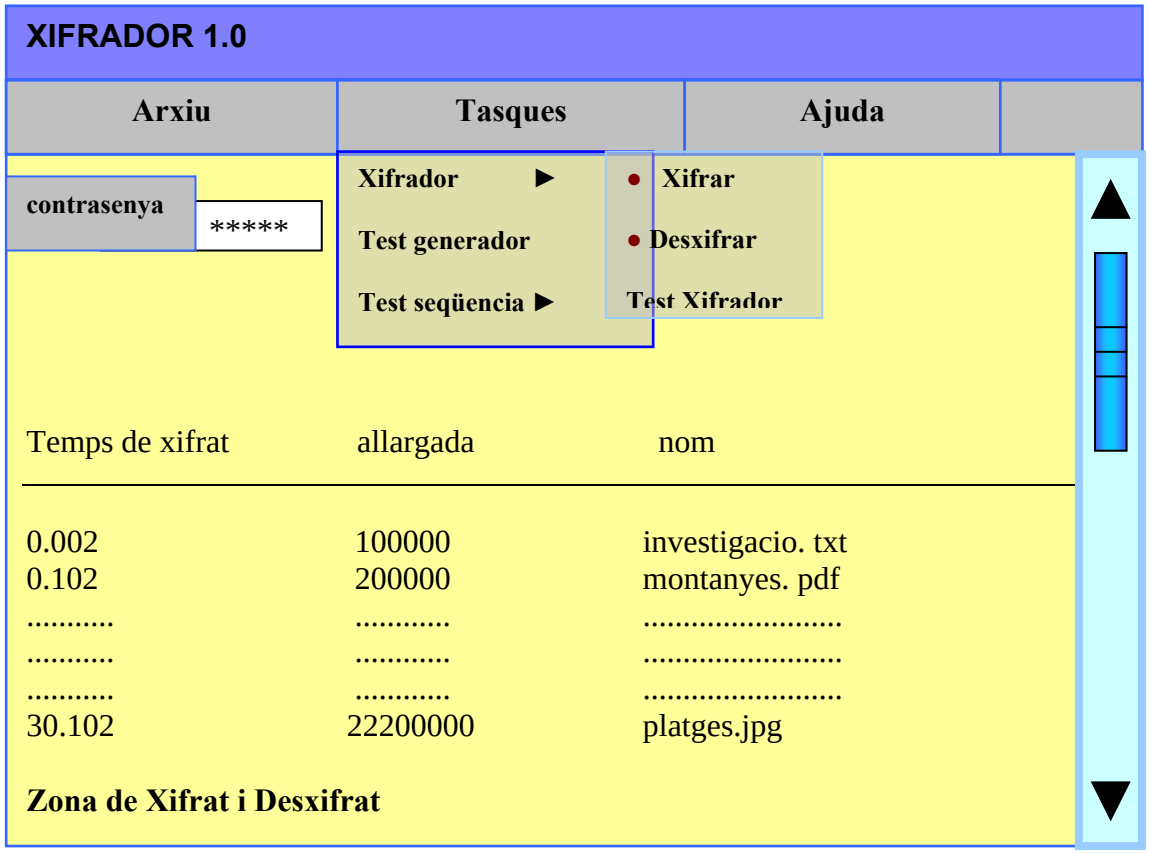
+actionPerformed(event :ActionEvent) Aquest mètode captura la interacció de l'usuari amb l'àrea per escriure la contrasenya.
+drop(event :DropTargetDropEvent) Aquest mètode captura el material arrossegat per l'usuari damunt la pantalla d'aplicació.

Classe **InterficieText:**
Aquesta classe implementa la interfície destinada a presentar les dades per pantalla.

Atributs
-panelPrincipal : JScrollPane panell per crear l'estructura de la interfície, esta equipat amb cursor
-titolArea : String nom de l'àrea de text.

Mètodes
Els mètodes afegits per la funcionalitat son:
+InterficieTest (títol: String) constructor de la classe, inicialitza l'objecte Pare , inicialitza l'atribut titolArea amb el valor passat per paràmetre.
crearInterficie() Executa les diferents crides als mètodes necessaris per crear la interfície amb totes les seves funcionalitats.
anarAlFinal() Aquest mètode permet situar-se al final de la mascara del text imprès a l'area de text.
getPanelPrincipal() : JScrollPane Permet l'accés a l'atribut panelPrincipal.

Figura 18. Disseny de la pantalla d'aplicació per Xifrar i Desxifrar



3.4.2 Interfície gràfica per fer el test al generador.

Funcionalitats de la barra de menú per aquesta interfície.

Com ja hem dit la barra de menús dintre del menú Tasques disposarà de l'etiqueta Test generador que permet a l'usuari obrir la pantalla d'aplicació per realitzar un test al generador.

Funcionalitats de la interfície.

Aquesta interfície ens ha de permetre per una banda introduir els paràmetres de configuració de l'estat inicial del nostre generador i per un altre visualitzar el resultat dels diferents tests. També ens ha de permetre l'opció d'iniciar el test un cop les dades han estat introduïdes i acceptades.

La interfície gràfica que permet l'entrada de dades ha de tenir tants camps de text com les variables que defineixen l'estat inicial dels 3 generadors MasseyRueppel, que son, per cada generador :

- Polinomi del LFSR_R (el més ràpid)
- Estat Inicial del LFSR_R
- Velocitat del LFSR_R
- Polinomi del LFSR_N (el més lent)
- Estat Inicial del LFSR_N
- Velocitat del LFSR_N

Les propietats que han de complir aquestes variables són:

1 Els camps dels polinomis i dels estats inicials són dígits amb valor 0 o 1.
2 El nombre de dígits del polinomi del LFSR_R ha de ser més llarg o igual que el del polinomi del LFSR_N
3 El nombre de dígits de l'estat Inicial sempre serà inferior en una unitat al nombre de dígits del polinomi.
4 El primer i últim dígit d'un polinomi sempre ha de valer 1.
5 L'últim dígit d'un estat inicial sempre ha de valer 1 si la resta val 0
6 El valor de la velocitat del LFSR_R és d vegades (d >= 2) més gran que la del LFSR_N
7 Si el polinomi d'un lfsr és: $X^6+ X^3+ X^1+1$, la seva representació binària és: 1001011, i el seu estat inicial pot ser 000001, és a dir, el dígit de l'esquerra del polinomi correspon a la representació de la cel·la que té la màxima potencia (Xn), en canvi, en l'estat inicial, el dígit de la dreta, és el que ens indica el valor que correspon a la cel·la de màxima potencia (Xn).

Un exemple correcte del format d'aquestes dades es:

----- LFSR _R -----
Polinomi = 11111111111111111110111111110101 (és primitiu)
Estat inicial = 00000001001010001001011010010011
velocitat = 3

----- LFSR _N -----

Polinomi = 1011111011111111111 (és primitiu)

Estat inicial = 011000111001001101

velocitat = 1

Guio d'interacció de l'usuari amb l'aplicació per fer un test al generador

L'usuari inicia l'aplicació i activa l'etiqueta amb nom Test generador, tot seguit l'aplicació obra una nova finestra d'aplicació on a l'esquerra té tots els camps de text per omplir i a la dreta té una àrea de text amb un parell de botons, un té el nom Acceptar i esta activat, l'altre té el nom Nou i esta desactivat.

L'usuari introdueix les dades dels Atributs dels diferents generadors Massey-Rueppel que configuren el Nostre Generador. L'usuari clica el boto Acceptar, l'aplicació valida els valors, desactiva aquest boto i executa el test.

L'aplicació imprimeix els resultats del test en l'àrea de text i activa el boto Nou . Si l'usuari vol tornar a repetir el test, només ha de clicar el boto amb el nom Nou, llavors, s'esborraran les dades de l'àrea de text i s'activarà el boto Acceptar per tal d'iniciar de nou el procés.

casos d'us del guio per al cas d'Introduir les dades inicials dels generadors.

cas_1 Introduir el valor d'un camp de text.

Descripció: Permet introduir el valor d'una de les variables que necessita ElNostreGenerador per operar.
Actor: usuari de l'aplicació
Precondició: l'usuari executa l'aplicació, desplega el menú temporal Tasques i activa l'etiqueta amb el nom Test generador. L'aplicació obra una nova finestra d'aplicació on hi ha tota una sèrie de camps de text identificats amb un nom al costat esquerra del camp de text.
Flux Normal:

1. L'usuari omple els camps de text, segons els criteris establerts
2. L'usuari clica el boto Acceptar .
3. L'aplicació valida les dades segons els criteris establerts i inicia el test.

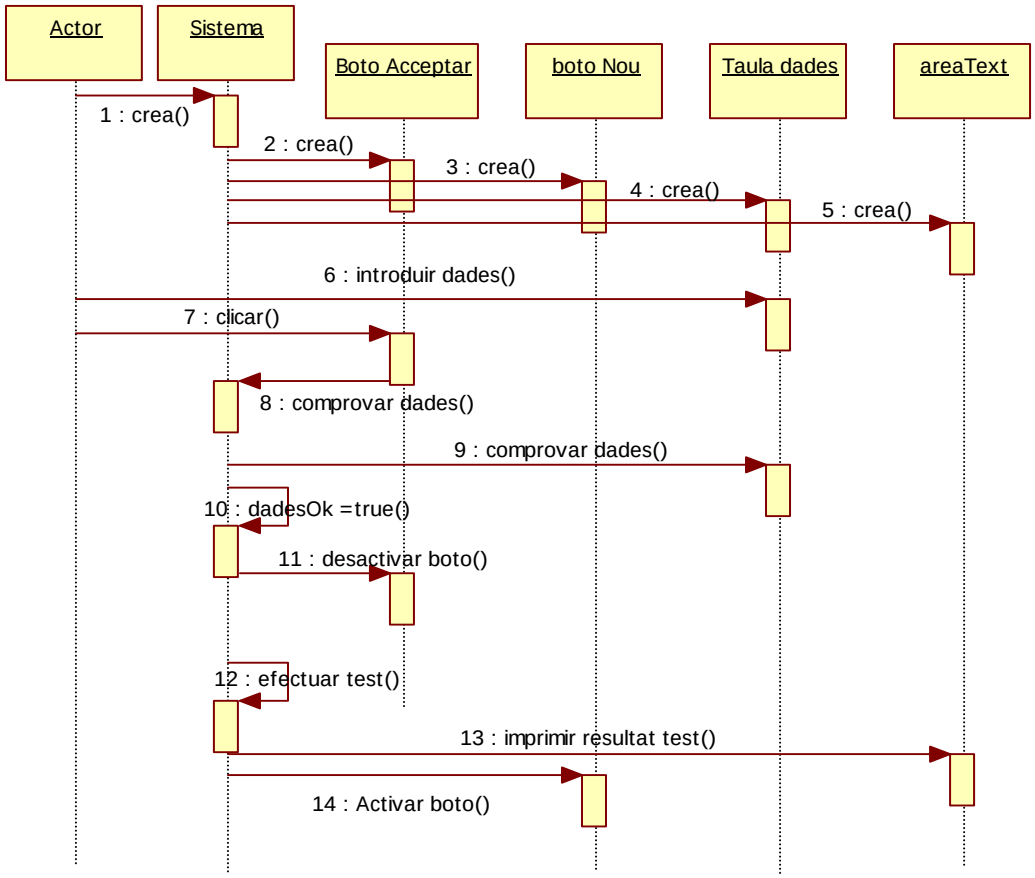
Flux Alternatiu_1:

1. L'aplicació valida les dades segons els criteris establerts i notifica a l'usuari que les dades introduïdes no son correctes.
2. Executar flux normal.

Poscondició:

1. El boto Acceptar esta desactivat.
2. El boto Nou esta activat.
3. L'àrea de text conté el resultat dels diferents tests.

Figura 19. Diagrama seqüència per al cas : Introduir el valor d'un camp de text, flux normal



Disseny de les classes per aquesta interfície

Repartirem les propietats d'aquesta Interfície en tres classes:

1. La pròpia classe **InterficieElNostreGenerador**, equipada amb els botons interactius ("Nou" per iniciar un procés nou i "Acceptar" per verificar les dades introduïdes i realitzar el test),.
2. La classe **InterficieMasseyRueppel** per tal d'implementar la interfície gràfica d'entrada de dades d'un generador de Massey-Reuppel
3. La classe **InterficieText** que ens ha de permetre mostrar les dades del resultat del test.

Classe **InterficieElNostreGenerador:**

Aquesta classe implementa la pantalla d'aplicació per fer el test al generador. i les seves funcions són:

- Permetre a l'usuari introduir les dades dels generadors Massey-Rueppel.
- Mostrar els resultats del test.

Atributs

- ms_1 : InterficieMassey-Rueppel interfície gràfica per capturar les dades del primer generador de Massey-Rueppel.
- ms_2 : InterficieMassey-Rueppel interfície gràfica per capturar les dades del segon generador de Massey-Rueppel.
- ms_3 : InterficieMassey-Rueppel interfície gràfica per capturar les dades del tercer generador de Massey-Rueppel.
- areaText : InterficieTest Objecte per presentar el resultat del test.
- panellDreta : JPanel panell per muntar la part dreta de la interfície.
- panellEsquerra : JPanel panell per muntar la part esquerra de la interfície.
- panellElNostreGenerador : JPanel panell per muntar les interfícies de ElNostreGenerador.
- taulaBotons : [] JButoon per guardar els botons per interactuar amb l'usuari.
- panellBotons : JPanel panell per muntar els botons.

Mètodes

Els mètodes afegits per la funcionalitat son:

+InterficieElNostreGenerador () constructor de la classe, inicialitza l'objecte Pare.

crearInterficie() Executa les diferents crides a tots els mètodes necessaris per crear la interfície amb totes les seves funcionalitats.

- crearPanelGenerador() mètode que permet crear l'estructura del generador, carrega les interfícies Massey-Rueppel.

- setValorsInicials() Mètode que carrega amb un valor determinat els camps de text de les interfícies dels generadors de Massey-Rueppel.

- setDades() Mètode que carrega els camps de text del període i el grau.

- prepararBotons() Mètode que permet construir els botons de la interfície.

- afegirPanells() Aquest mètode s'encarrega de muntar l'estructura de panells necessaris per la interfície.

- getArea() : InterficieText Permet accedir a la zona d'escriptura de la interfície.

+actionPerformed(event :ActionEvent) Aquest mètode captura la interacció de l'usuari amb els elements interactius.

Classe **InterficieMasseyRueppel:**

Aquesta classe implementa la interfície destinada a capturar els paràmetres necessaris per iniciar un generador MasseyRueppel.

Atributs

- etiquetes : [] JTextField Taula per guardar les etiquetes identificadores de camp.

- campsText : [] JTextField Taula per guardar els camps de text.

- valors : [] String Taula per guardar els valors dels camps de text.

- idMassey : int permet adjudicar als objectes un nombre per identificar-los.

- taulaPanels : [] JPanel Taula per guardar els panells necessaris per implementar les interfícies.

- vora : Border vora per decorar els panells.

Mètodes

Els mètodes afegits per la funcionalitat son:

+InterficieMasseyRueppel (idMassey:int) constructor de la classe, inicialitza l'objecte Pare i l'atribut idMassey.

crearInterficie() Executa les diferents crides a tots els mètodes necessaris per crear la interfície amb totes les seves funcionalitats.

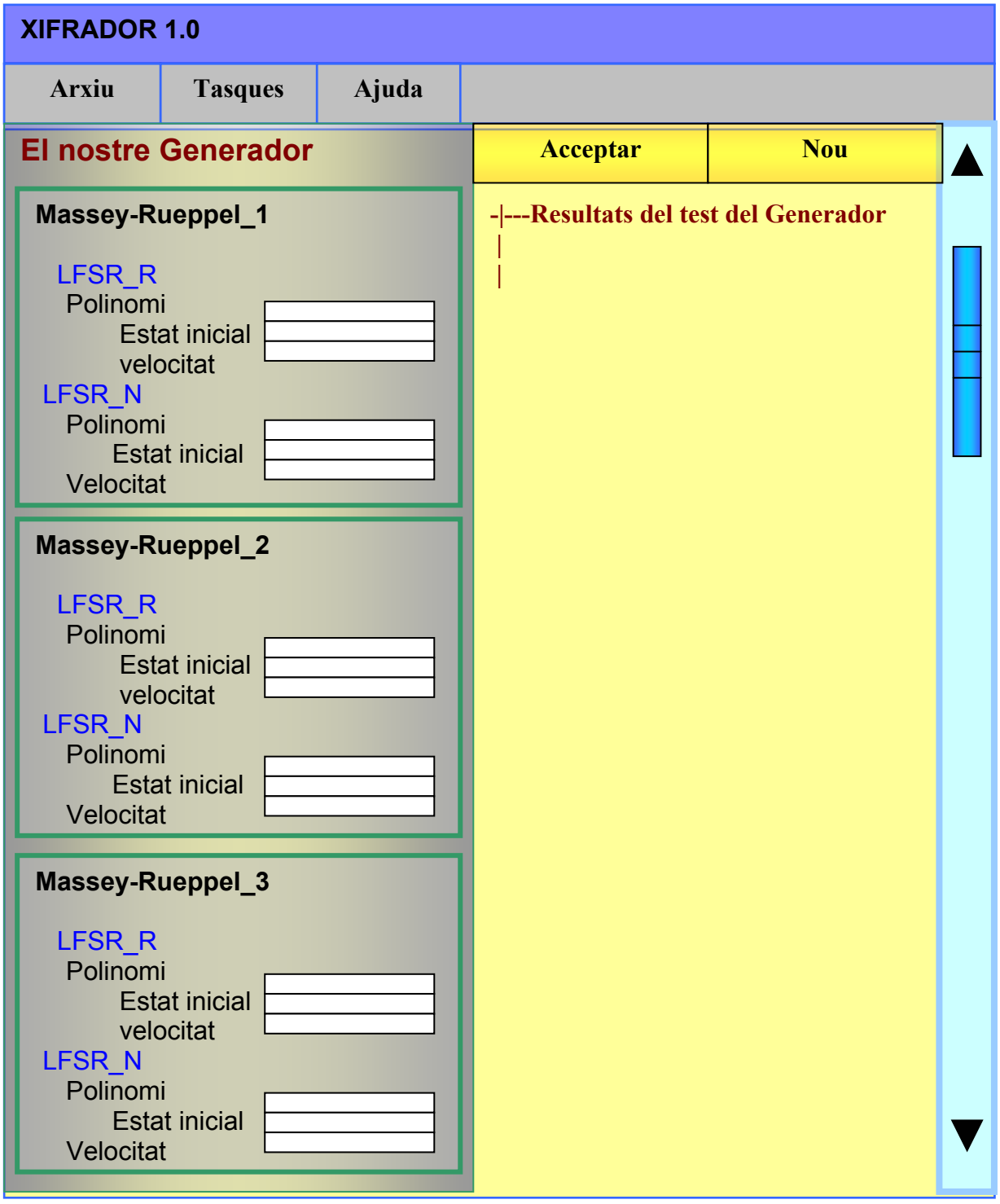
- crearCamps() Aquest mètode s'encarrega de muntar l'estructura d'etiquetes i camps de text.
- crearEstructuraPanels() Aquest mètode s'encarrega de muntar l'estructura de panells necessaris per la interfície.

getDades() : int retorna el contingut dels camps de text.

setDades(text:String, idDades:int) carrega el camp de text amb índex idDades amb el valor text.

getId() : int retorna el valor de l'atribut IdMassey.

Figura 20. Disseny de la pantalla d'aplicació per fer el test del generador.



3.4.3 Interfície gràfica per fer el test de la seqüència.

Funcionalitats de la barra de menú per aquesta interfície.

Per a què l'usuari pugui realitzar el test d'una seqüència, utilitzarem el menú temporal Test seqüència que estarà equipat amb tres etiquetes anomenades:

- Guardada Permet a l'usuari realitzar el test d'una porció de la seqüència que esta desada a l'arxiu guardada.txt.
- menezes Permet a l'usuari comprovar que la implementació del test són correctes i que, per tant, el resultat del test del generador només depèn de les dades introduïdes per teclat. Això ho aconseguim fent el test d'una seqüència de la qual en coneixem el resultat a priori. En aquest cas utilitzem la seqüència analitzada en [MEN1].
- Nova Permet a l'usuari introduir una seqüència per teclat i fer-li un test.

Funcionalitats de la interfície.

Aquesta interfície ens ha de permetre fer al test a una seqüència en les següents condicions:

- La seqüència és introduïda per teclat.
- La seqüència s'obté des d'un arxiu.
- La seqüència s'obté des de la pròpia aplicació.

Les propietats que ha de complir aquesta seqüència son:

- La seqüència ha de tenir una llargada >=4
- La seqüència només pot contenir el caràcter 1 o 0.

Guio 1: Interacció de l'usuari per realitzar el test d'una seqüència introduïda per teclat

L'usuari ha iniciat l'aplicació i a traves dels menús de la barra de menú activa l'etiqueta amb nom Nova, tot seguit l'aplicació obra una nova finestra d'aplicació equipada amb:

- Un parell de botons, un té el nom Acceptar i esta activat, l'altre té el nom Nou i esta desactivat.

- Una àrea de text titulada “Espai per escriure la seqüència”. Aquesta àrea de text és editable i és la que l’usuari farà servir per introduir la seqüència que vol estudiar.
- Una àrea de text on l’aplicació imprimirà el resultat del test.

L’usuari introdueix la seqüència i clica el boto Acceptar. L’aplicació desactiva el boto Acceptar, realitza el test, imprimeix els resultats i activa el boto Nou per tal que l’usuari torni a començar el procés..

Guio 2: Interacció de l’usuari per realitzar el test d’una seqüència desada a un arxiu
L’usuari ha iniciat l’aplicació i a traves dels menús de la barra de menú activa l’etiqueta amb nom Guardada, tot seguit l’aplicació obra una nova finestra d’aplicació equipada igual que al Guio 1.

l’àrea de text per introduir la seqüència no és editable i hi tenim el missatge “Analitzarem la seqüència desada a l’arxiu guardada.txt”.

L’usuari clica el boto Acceptar. L’aplicació desactiva el boto Acceptar, realitza el test, imprimeix els resultats i activa el boto Nou per tal que l’usuari torni a començar el procés..

Guio 3: Interacció de l’usuari per realitzar el test a la seqüència de menezes.
L’usuari ha iniciat l’aplicació i a traves dels menús de la barra de menú activa l’etiqueta amb nom Menezes, tot seguit l’aplicació obra una nova finestra d’aplicació equipada igual que al Guio 1.

l’àrea de text per introduir la seqüència no és editable i hi tenim el missatge “Analitzarem la seqüència de menezes”.

L’usuari clica el boto Acceptar. L’aplicació desactiva el boto Acceptar, realitza el test, imprimeix els resultats i activa el boto Nou per tal que l’usuari torni a començar el procés.

casos d'us del guio1.

cas_1 Introduir la seqüència per teclat..

Descripció: Permet introduir una seqüència de zeros i uns per a què l'aplicació li faci un test.
Actor: usuari de l'aplicació
Precondició: L'usuari ha iniciat l'aplicació i a traves dels menús de la barra de menú activa l'etiqueta amb nom Nova. L'aplicació obra una nova finestra d'aplicació on hi ha un cap de text editable que duu per títol " Espai per escriure la seqüència".
Flux Normal: 1. L'usuari introdueix la seqüència en funció els criteris establerts. 2. L'usuari clica el boto Acceptar . 3. L'aplicació valida les dades segons els criteris establerts i inicia el test. **Flux Alternatiu_1**: 1. L'aplicació valida les dades segons els criteris establerts i notifica a l'usuari que les dades introduïdes no son correctes. 2. Executar flux normal.
Poscondició: 1. El boto Acceptar esta desactivat. 2. El boto Nou esta activat. 3. L'àrea de text conté el resultat dels diferents tests.

Disseny de les classes per aquesta interfície

Implementarem la classe **InterficieTestSequencia**, aquesta interfície utilitza la classe **interficieText.** Estarà equipada amb un camp de text que permet l'entrada de la seqüència per teclat i amb els botons interactius ("Nou" per iniciar un procés nou i "Acceptar" per verificar les dades introduïdes i realitzar el test),.

Classe **InterficieTestSequencia:**

Aquesta classe implementa la interfície destinada a presentar les dades del resultat del test fet a una seqüència. i les seves funcions són:

- Permetre introduir a l'usuari una seqüència per teclat.
- Permetre a l'usuari triar la seqüència desada a l'arxiu guardada.txt.
- Permetre fer un test a una determinada seqüència, concretament la que trobarem a [MEN1] pag.182.

Atributs

-sequenciaTeclat : JTextField Àrea de text per escriure la seqüència.

-areaText : interficieText Area de text per escriure els resultats.

- botons : [] JButton taula per desar els botons (Nou i Acceptar).

-panellBotons : JPanel panell per suportar els botons.

-MENEZES : String String per desar la seqüència que surt a [MEN1].

-NOVA : String String per desar la seqüència per defecte.

-tipusSequencia : String Aquest atribut ens permet emmagatzemar l'opció triada per l'usuari (Guardada, Menezes, Nova) a la barra de menús.

Mètodes

Els mètodes afegits per la funcionalitat son:

+InterficieTestSequencia () constructor de la classe, carrega les constants.

crearInterficie() Executa les diferents crides als mètodes necessaris per crear la interfície amb totes les seves funcionalitats.

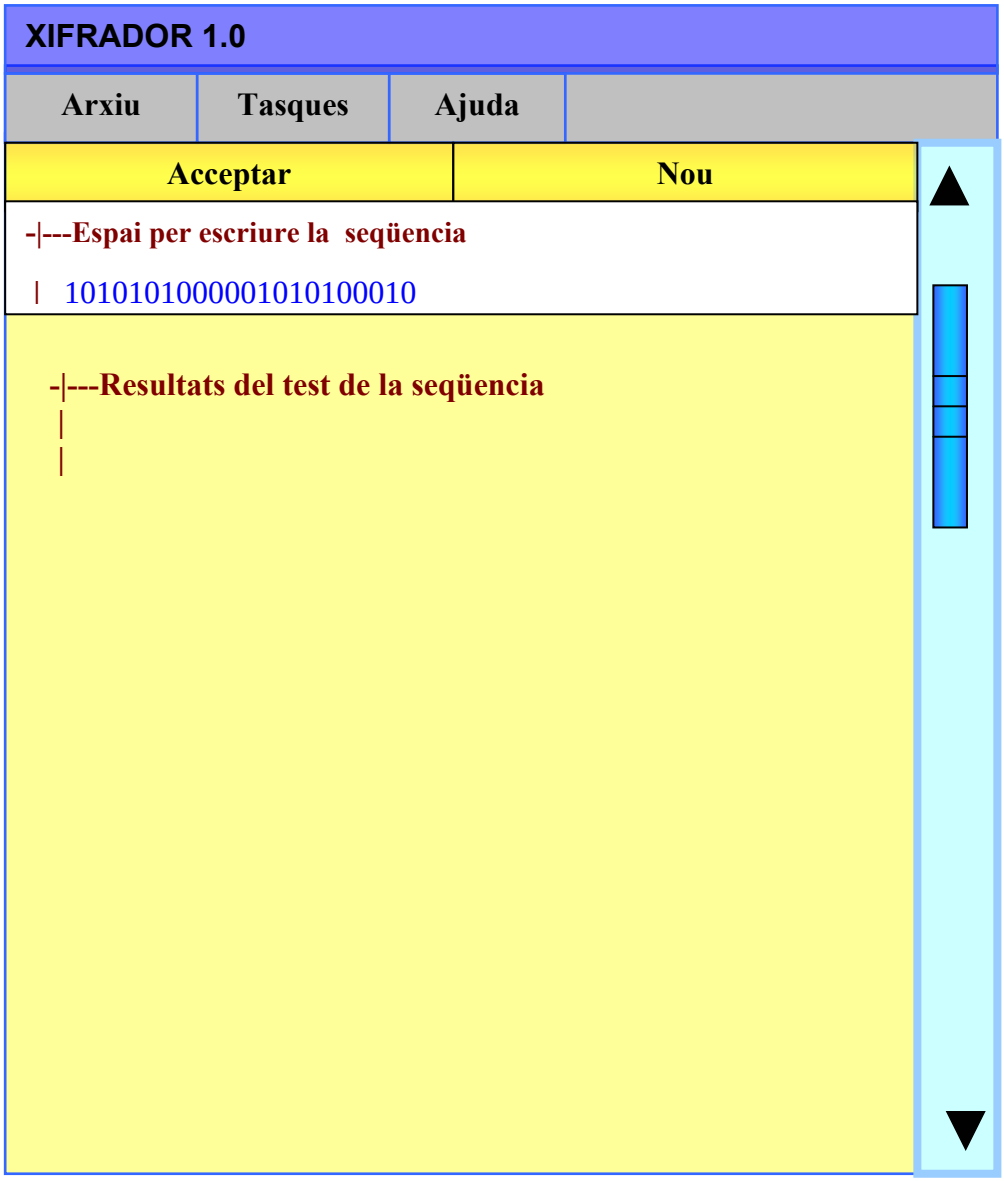
- crearAreaPerCapturarSequencia() Aquest mètode s'encarrega de crear l'àrea de text per a què l'usuari pugui introduir la seqüència.

- crearBotons() Aquest mètode crea els botons i implementa les seves funcionalitats.

-afegirPanels() Mètode per a muntar l'estructura de panells necessaris per la interfície.

+actionPerformed(event :ActionEvent) Aquest mètode captura la interacció de l'usuari amb els elements interactius.

Figura 21. Disseny de la pantalla d'aplicació per fer el test a una seqüència.



3.4.4 Interfície gràfica per l'ajuda.

Funcionalitats de la barra de menú per aquesta interfície.

El menú Ajuda estarà equipat amb tres etiquetes anomenades:

- Ajuda Xifrador Permet a l'usuari consultar les característiques relacionades amb el xifrador, com funciona, limitacions, etc.
- Ajuda Generador Permet a l'usuari consultar les característiques relacionades amb el test del generador, com funciona, limitacions, etc.
- Ajuda Seqüència Permet a l'usuari consultar les característiques relacionades amb el test a una seqüència, com funciona, limitacions, etc.

Funcionalitats de la interfície.

Aquesta interfície ens ha de permetre informar a l'usuari de les diferents funcionalitats de l'aplicació.

L'accés a l'ajuda és realitzarà des de l'etiqueta "Ajuda" de la barra de menús. Per cada tipus d'ajuda es generarà una nova pantalla d'aplicació on l'àrea de text podrà tenir els següents títols :

- Ajuda per a les funcionalitats del xifrador.
- Ajuda per a les funcionalitats del generador.
- Ajuda per a les funcionalitats de la seqüència.

Disseny de les classes per aquesta interfície

Implementarem la classe **InterficieAjuda**, aquesta interfície utilitza la classe **interficieText.**

Classe **InterficieAjuda:**

- Aquesta classe implementa la interfície destinada a presentar les ajudes de les funcionalitats de l'aplicació.

Atributs

-areaText : interficieText Àrea de text per escriure la seqüència.

AJUDA1: String Constant amb valor = "Ajuda per a les funcionalitats del xifrador"

AJUDA2: String Constant amb valor = "Ajuda per a les funcionalitats del generador"

AJUDA3: String Constant amb valor = "Ajuda per a les funcionalitats de la seqüència"

Mètodes

Els mètodes afegits per la funcionalitat son:

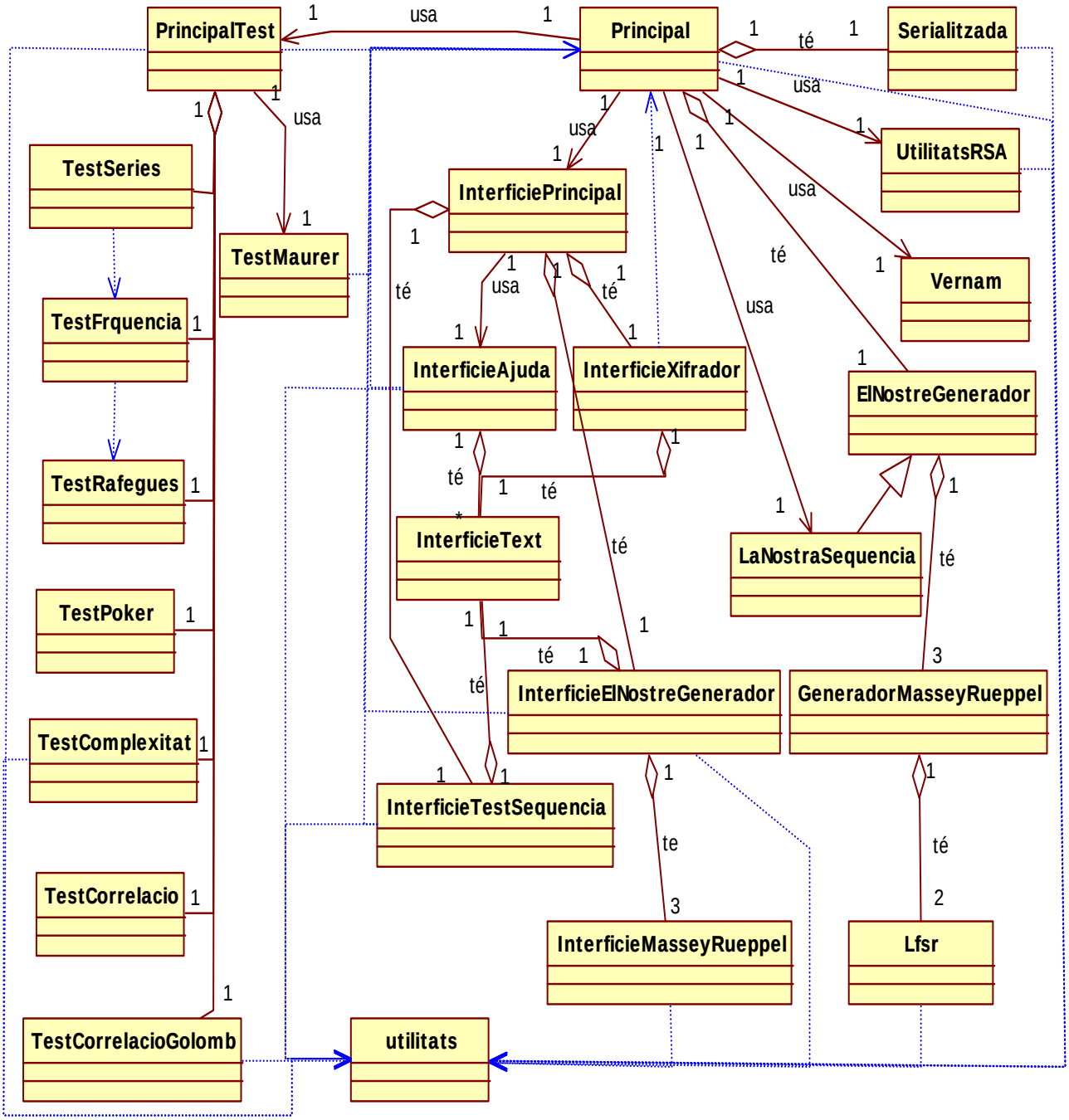
+InterficieAjuda () constructor de la classe. Inicia una nova pantalla.

- crearInterficie(tipus:int) construeix un objecte interfície text amb el títol determinat per la constant tipus.

+actionPerformed(event :ActionEvent) Aquest mètode captura la interacció de l'usuari amb els elements interactius, mostra l'ajuda triada per l'usuari.

3.5 Diagrama de Classes.

Figura 22. Diagrama de classes de l'aplicació.



4 Aspectes concrets de la Implementació.

La implementació s'ha realitzat utilitzant el llenguatge Java, les seves llibreries i aplicant les tècniques que permet aquest llenguatge.

Per al control d'errors, s'han utilitzat els blocs try- catch combinats amb la classe JoptionPanel. Aquesta classe té mètodes statics que permeten llançar una finestra de diàleg amb l'usuari i notificar-li el tipus d'error o com ha d'executar les comandes del menú.

Les classes han establert les relacions d'agregació declarant atributs de tipus objecte.

Per la funcionalitat de tots els mòduls s'ha implementat la **classe Utilitats.** Aquesta classe s'encarrega principalment de les tasques relacionades amb els buffers per la lectura i escriptura d'arxius i objectes. També proporciona els mètodes encarregats de notificar a l'usuari alguns dels possibles errors i implementa alguns mètodes per convertir estructures de Strings a booleans o a bytes. Aquesta classe establirà relacions de dependència amb la resta de classes.

La majoria de classes han sobreescrit el mètode toString() i han incorporat el mètode main per fer les proves corresponents.

Per treballar amb claus RSA s'ha utilitzat les llibreries proporcionades pel proveïdor Bouncy Castle.

Durant la implementació han sorgit noves idees i possibles millores, però ens hem limitat a seguir el guio establert al disseny i efectuar les mínimes variacions. Tot seguit explicarem les variacions més importants realitzades per tal d'implementar totes les funcionalitats.

4.1 Implementació del mòdul Generador.

Les relacions entre les classes no han estat afectades.

S'ha creat la carpeta documents per desar l'arxiu guardada.txt Aquest arxiu ha modificat el seu contingut, ara conté 5000 dels bits generats pel generador en el moment de fer-li un test.

Classe **Lfsr:**

S'ha eliminat l'atribut numCeles ja que fa el mateix que l'atribut grau.

Entre altres, s'ha implementat el mètode # buscarPrimitiu(grau :int) que permet buscar tots els polinomis primitius del grau passat per paràmetre.

Classe **GeneradorMasseyRueppel:**

l'atribut - idbn : int és una constant i passa a anomenar-se - ID_BN : int

Classe **ElNostreGenerador**

Implementa un parell més de constructors i mètodes per accedir als comptadors dels seus generadors. També implementa un mètode per minimitzar el valor dels comptadors, és a dir, si a l'inici d'un xifratge els comptadors tenen per valor 1000, 1003 i 1007, el valor que és carregarà als arxius serà 0,3 i 7, ja que pel desxifrat és suficient (al comptador de menys valor sempre valdrà zero).

4.2 Implementació del Mòdul Test.

Apareixen relacions de dependència entre la classe TestRafegues i TestFrequencia i entre les classes TestFrequencies i TestSeries.

El constructor de la classe principal d'aquest mòdul ha incrementat i modificat el nombre de paràmetres:

+PrincipalTest(àrea: InterfícieText, desarDades : boolean , longBloc:int)

L'atribut longBloc permet limitar la longitud del bloc per al test de Maurer. En aquesta implementació limitem la longitud a 10, per tal de no allargar massa els càlculs.

L'atribut desarSequencia serveix per decidir si cal o no cal desar la seqüència a analitzar i el resultat dels tests en arxius.

L'atribut InterfícieText és una àrea de text on s'escriuran els resultats dels tests.
Com a característica de la implementació, destaquem el fet que Java no controla el desbordament del primitiu int, per tant ha calgut tractar aquesta situació. Per fer-ho hem utilitzat la classe BigInteger per calcular les dades i dividir-les en parts iguals al màxim que pot representar un int.

En els següents casos no farem alguns dels tests:

- No farem el test del poker en seqüencies <=10 bits
- No Imprimirem el resultat del test de ràfegues en seqüencies < 38 bits, però si que el farem degut a la relació de dependència entre les classes TestRafegues i TestFrequencia.
- El valor del paràmetre d en el test de correlació, esta fixat a 8, per tant, no farem aquest test en seqüencies <=8 bits.

Els arxius generats en el test de la complexitat lineal i el test de la correlació de Golomb s'han arxivat a la carpeta documents.

Els valors de les variables a l'ora de fer el test del generador serà:

- LONG_SEQUENCIA= 25 milions de bits.
- MAX_BITS_DESAR = 5000 bits.

Els valors de les variables a l'ora de fer el test d'una seqüència serà:

- LONG_SEQUENCIA= longitud seqüència.
- MAX_BITS_DESAR = longitud seqüència.

4.3 Implementació del mòdul Xifrador/desxifrador.

Per desar l'estat del generador, hem creat un arxiu anomenat generador_xif. Aquest arxiu és reescriu cada cop que és xifra un document, esta signat i xifrat amb les claus RSA de l'usuari. Aquest arxiu quedarà desat a la carpeta privat. Per crear aquest arxiu per primer cop, s'han substituït a la classe Principal les crides del mètode guardarEstatGenerador() per valors que hem considerat correctes en funció de les especificacions (polinomis primitius, etc..).

La classe UtilsRSA a canviat el constructor per UtilitatsRSA(contrasenyaKeyStore: String ,aliasCertificat: String, URLKeystore String) Aquest constructor rep per paràmetre la contrasenya d'accés al magatzem de claus, l'alies del certificat i l'adreça on es troba el certificat.

Per efectuar la signatura digital i encriptar l'estat del generador, hem generat a la carpeta privat un arxiu anomenat USUARI.Keystore amb l'eina Keytool i amb les següents comandes.

```
keytool -genkey -keyalg rsa -keysize 1024 -alias XIF
        -keystore USUARI.keystore -keypass 123456 -dname "cn= USUARI "
        -storepass USUARI -storetype JCEKS -validity 360
```

Per permetre a l'usuari canviar la contrasenya de l'arxiu USUARI.Keystore, hem generat a la carpeta privat, l'arxiu canviarContrasenya.bat amb el següent contingut:

```
keytool   -keypasswd -v -alias XIF  -keystore USUARI.keystore
          -storepass USUARI. -storetype JCEKS
```

Per accedir a aquesta funcionalitat, hem creat una nova opció en el menú Arxiu anomenada "Canviar contrasenya"

Per la funcionalitat d'aquest mòdul, hem implementat la classe **Serialitzada.** Aquesta classe implementa la classe serializable, això ens ha permès convertir l'estat del generador i les dades xifrades en un objecte per serialitzar-lo i signar-lo/verificar-lo i així generar el document de sortida que l'usuari vol xifrar/desxifrar i actualitzar les dades de l'arxiu generador_xif.

S'ha decidit modificar el fet d'arxivar una part de la seqüència generada pel generador quan l'usuari esta efectuant l'operació de xifrat. En el seu lloc, el que s'ha fet, és inserir una opció més al menú Tasques anomenada "Test xifrador". Aquesta opció permet a l'usuari fer tota la col·lecció de test als propers 20000 bits que el generador generarà (el test de Maurer només fins arribar a blocs de longitud L=10) i imprimir el resultat per pantalla. Per fer-ho, recuperem l'estat actual del generador de l'arxiu generador_xif i posem en marxa el generador. Aquests bits no

és perden, ja que no modifiquem les dades de l'arxiu generador_xif i tampoc els desem en lloc.
Per tant la classe **LaNostraSequencia** passa a formar part del disseny del mòdul generador.

Els valors de les variables a l'ora de fer el test del generador en mode xifrador serà:

- LONG_SEQUENCIA= 20.000 bits.
- MAX_BITS_DESAR = 3000 bits.

En aquesta versió, a l'ora de xifrar, desem les dades en matrius, per tant, limitarem la longitud màxima dels arxius que volem xifrar a 3 MBytes. Si l'usuari vol xifrar documents amb major longitud, tindrà que desmuntar els arxius en blocs de < 3 Mbytes.

4.4 Implementació del mòdul Interfícies.

La creació de les interfícies s'ha aconseguit fent que les classes heretin de la classe Jpanel i Jframe.

Per obtenir la interacció entre l'usuari i l'aplicació, les classes han implementat les interfícies MouseListener , ActionListener i DropTargetListener .

Per aconseguir una fluïdesa en la presentació de dades per pantalla les classes han implementat la interfície Runnable.

Com ja hem comentat, el menú Arxiu incorpora un nou ítem anomenat "canviar contrasenya", en clicar-lo, apareix una caixa de diàleg que informa a l'usuari del procediment que cal fer per canviar la contrasenya. El menú Tasques incorpora un nou ítem anomenat "Test xifrador", en clicar-lo s'obra una nova finestra equipada amb una àrea de text on l'usuari pot observar com van apareixent els resultats de fer un test de 20000 bits generats pel generador amb les dades de l'arxiu generador_xif.

S'ha eliminat la classe **InterficieTest** i en el seu lloc s'ha utilitzat la classe **InterficieText.** La finalitat ha estat unificar en un objecte les característiques comunes de totes les àrees de text de les diferents pantalles d'aplicació, per tant,

aquesta classe establirà relacions d'us amb la resta d'interfícies Això ha provocat que la classe **InterficieTextSequencia,** passi a ser anomenada **InterficieTestSequencia** i passi a heretar de la classe JPanel.

La classe **InterficiePrincipal** incorpora un nou mètode canviarEstatBotons(cas :int) La finalitat d'aquest mètode és desactivar els botons de la barra de menú mentre l'aplicació esta realitzant tasques amb el generador, així evitem que els diferents mòduls puguin accedir a l'ora al generador o a l'arxiu generador_xif.

La classe **InterficieMasseyRueppel** implementa la interfície MousseListener per tal de capturar les eventualitats de l'usuari amb el ratolí en els camps de text que aquest pot emplenar. L'usuari podrà realitzar les següents operacions:

- Si selecciona un o més de dígits dels camps de text polinomi o estat inicial podrà:
    - Si clica el boto dret, substituir-los per un 0.
    - Si clica el boto central, substituir-los per un 1.

- Si se situa en un dels camps de text polinomi o estat inicial i no ha seleccionat cap dígit podrà:
    - Si clica el boto dret, inserir un 0 al final del valor del camp.
    - Si clica el boto central, inserir un 1 al final del valor del camp.

- Si se situa o selecciona el camp de text velocitat podrà:
    - Si clica el boto dret, reduir una unitat el valor del camp (valor >= 1).
    - Si clica el boto central, incrementar una unitat el valor del camp.

La classe **InterficieElNostreGenerador** implementa el mètode setValorsInicials() Aquest mètode serveix per iniciar els camps de text dels Generadors Massey-Rueppel en el moment d'iniciar la interfície.

La classe **InterficieXifrador** incorpora un nou mètode crearFinestraTestXifrador():InterficieText Aquest mètode s'encarrega de realitzar el test al Xifrador.

5 Manual d'instal·lació i us.

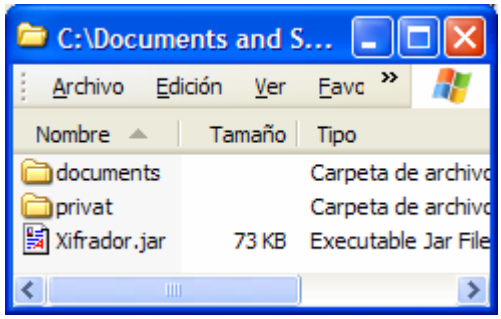
Per executar l'aplicació calen els següents requisits i procediments:

1. Tenir instal·lat la J2SE Runtime Environment 5.0_xx

2. Editar l'arxiu de Java, anomenat java.security (aquest arxiu el trobarem a la carpeta anomenada .....jre/lib/security/j) i a l'apartat List of providers and their preference orders a la línia 2, afegir la línia - security.provider.2=org.bouncycastle.jce.provider.BouncyCastleProvider i modificar la numeració de la resta de línies adequadament.

3. Descomprimir l'arxiu **Xifrador1.0.zip** i obtindrem la carpeta Xifrador1.0. A la següent figura podem observar el seu contingut.

Figura 23. Contingut de la carpeta Xifrador1.0



4. Aquests documents son:
   - **Carpeta documents:** Aquesta carpeta serveix per a desar els arxius generats per l'aplicació a l'ora de realitzar les tasques. Inicialment genera els següents arxius:
     - guardada.txt, conté 5000 dels bits generats al fer el test del generador. Aquest arxiu el podem carregar amb altres seqüencies per tal de fer-hi un test.
     - Complexitat_Desviacio.txt, conte la desviació calculada en el test de la complexitat lineal, per estudiar-la en una gràfica.
     - Complexitat_Perfil.txt, conte el valor que la complexitat va assolint a mida que processa els bits, per estudiar-la en una gràfica.

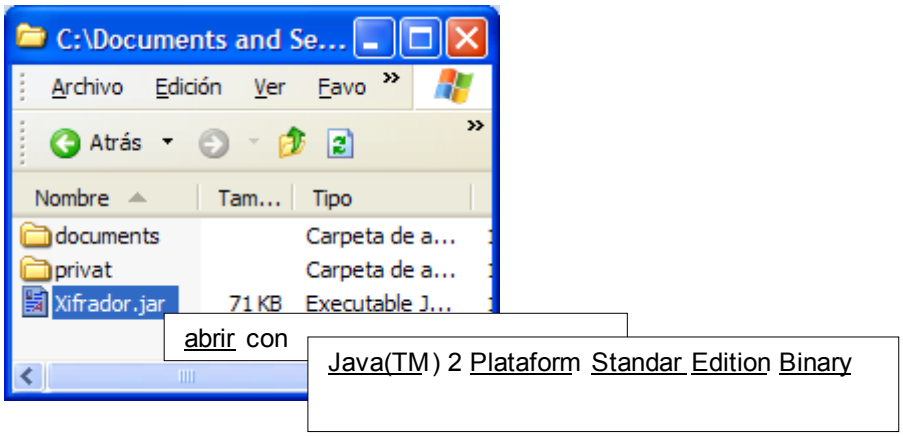
- correlacioGolomb.txt, conté el valor de la funció AC(k) per a tots els k < que la longitud de la seqüència, per analitzar els pics en un gràfic.

- **Carpeta privat:** Aquesta carpeta serveix per a desar els arxius que fa servir l'aplicació per fer les diferents tasques, aquests arxius són:
  - ajuda/ajuda sequencia.txt informa com s'ha de fer un test d'una seqüència de bits.
  - ajuda/ajuda generador.txt Informa com s'ha de fer un test al generador.
  - ajuda/ajuda xifrador.txt informa que s'ha de fer per xifrar i desxifrar un document i com fer un test al generador amb la configuració que utilitza per xifrar.
  - USUARI.keystore magatzem de claus de l'usuari de l'aplicació.
  - generador_xif (estat inicial del generador).
  - canviarContrasenya.bat permet a l'usuari canviar la contrasenya d'accés a la clau privada.
  - proveidor.jar conte els arxius del proveïdor de serveis criptogràfics.

- **Arxiu xifrador.jar:** Aquest arxiu conté els arxius de l'aplicació i és el que permet iniciar l'aplicació.

5. Iniciar l'aplicació clicant al damunt de l'arxiu **xifrador.jar** (aconsellem crear un accés directe i situar-lo a l'escriptori. Si l'aplicació **no** s'executa caldrà indicar-li al sistema que volem que obrir aquest arxiu amb Java(TM) 2Plataform Standard Edition binary.

Figura 24. Procediment per determinar qui obra l'aplicació



6. Un cop s'ha clicat l'arxiu xifrador.jar, s'obrirà la pantalla d'aplicació equipada amb un menú que permet accedir a les diferents tasques. Aquesta pantalla inicial és la interfície per a les tasques de xifrat i desxifrat. A les següents figures tenim un detall de les diferents pantalles de l'aplicació. Des de la barra de menús podem accedir al menú **Ajuda** per editar les diferents ajudes i veure com funciona tot plegat.

Figura 24 . Contingut del document ajudaGenerador.txt

TEST DEL GENERADOR.

1 - Clicar a la barra de menús Tasques > Test generador.

2 - Omplir els camps de dades dels polinomis, dels estats inicials i de les velocitats.

- Per defecte ja estan emplenades.

- Les dades es poden introduir per teclat o bé amb els botons central i dret del ratolí de la següent manera:

Si selecciona un o més de dígits dels camps de text polinomi o estat inicial podrà:
Si clica el boto dret, substituir-los per un 0.
Si clica el boto central, substituir-los per un 1.

Si se situa en un dels camps de text polinomi o estat inicial i no ha seleccionat cap dígit podrà:
Si clica el boto dret, inserir un 0 al final del valor del camp.
Si clica el boto central, inserir un 1 al final del valor del camp.

Si se situa o selecciona el camp de text velocitat podrà:
Si clica el boto dret, reduir una unitat el valor del camp (valor >= 1).
Si clica el boto central, incrementar una unitat el valor del camp.

- les dades han de complir els següents requisits:

a) Els camps dels polinomis i dels estats inicials són dígits amb valor 0 o 1.
b) El nombre de dígits del polinomi del LFSR_R ha de ser més llarg o igual que el del polinomi del LFSR_N
c) El nombre de dígits de l'estat Inicial sempre serà inferior en una unitat al nombre de dígits del polinomi.
d) El primer i últim dígit d'un polinomi sempre ha de valer 1.
e) L'últim dígit d'un estat inicial sempre ha de valer 1 si la resta val 0.
f) El valor de la velocitat del LFSR_R és d vegades (d >= 2) més gran que la del LFSR_N.
g) Si el polinomi d'un lfsr és X6+ X3+ X1+1, llavors tenim:
La seva representació binària és: 1001011.
El seu estat inicial pot ser: 000001. és a dir, el dígit de l'esquerra del polinomi correspon a la representació de la cel·la que té la màxima potencia (Xn),
en canvi, en l'estat inicial, el dígit de la dreta,
és el que ens indica el valor que correspon a la cel·la de màxima potencia (Xn).

3 - Un cop s'han introduït les dades, cal clicar el boto "Acceptar" .

- Si les dades compleixen els requisits es realitzarà el test i els resultats aniran apareixent a la interfície a mida que estén llestes, cal esperar un determinat temps.

- El test de Maurer només és fa per a un màxim de longitud de bloc L=10, és pot editar el codi

i modificar aquest paràmetre a les crides dels mètodes de la classe principal.

- El test de series, freqüència, poker i ràfegues és realitza sobre una seqüència de 25 Mbits

- El test de la correlació (A(8)), el test de correlació de golomb i el test de la complexitat lineal es fan per a un tram de la seqüència de 5 Kbits.

- A la carpeta documents trobarem els arxius generats pels tests de correlació de Golomb i complexitat lineal,
també trobarem un arxiu (guardada.txt) que contindrà 5 Kbits de la seqüència generada.

- Mentre és fa el test, el boto "Acceptar" i els menús "Arxiu" i "Tasques" romandran desactivats.

- Quan finalitzi el test, es tornaran a activar els menús i el boto "Nou", llavors si és vol realitzar un nou test, només caldrà clicar el boto "Nou", s'esborraran les dades de la pantalla i s'activarà el boto "Acceptar" per tal d'iniciar un nou test.

- Si les dades no compleixen els requisits, s'avisarà de l'error concret, llavors, caldrà corregir-lo i tornar a clicar "Acceptar".

****
Fi ajuda.

Figura 25 . Contingut del document ajudaSequencia.txt

TEST D'UNA SEQÜÈNCIA DESADA A UN ARXIU.
****

- Amb aquesta opció analitzem les dades contingudes a l'arxiu guardada.txt, cal tenir present que cada cop que fem un test al generador, aquest arxiu és reescriu. Inicialment conté la seqüència generada de fer un test al generador amb les dades introduïdes per defecte a l'ora de crear la interfície per fer un test del generador.

1 - Clicar Tasques > Test seqüència > Guardada

- la interfície modificarà la seva estructura i apareixerà una nova pantalla.

2 - Clicar el boto "Acceptar" i esperar la sortida dels resultats.

- Mentre dura l'anàlisi els botons i el menú romandran desactivats, un cop a finalitzat l'anàlisi, s'activarà el boto "Nou", si el cliquem, s'esborrarà la pantalla i s'activarà el boto "Acceptar", llavors podem repetir l'anàlisi.

TEST DE LA SEQÜÈNCIA DE MENEZES :
****

- Analitzem la seqüència que trobarem al llibre:
A. Menezes, P. Van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC. Press, 1997, pag. 182
També el podem descarregar gratuïtament des de http://www.cacr.math.uwaterloo.ca/hac/

- la seqüència de 160 bits és:
1110001100010001010011101111001001001001111000110001000101001110111100100100100111100011000100010100111011110010010010011110001100010001010011101111001001001001111000110001000101001110111100100100100111

- l'anàlisi d'aquesta seqüència ens permet confirmar el correcte comportament de la implementació dels tests.

1 - Clicar Tasques > Test seqüència > Menezes

- la interfície modificarà la seva estructura i apareixerà una nova pantalla.

2 - Clicar el boto "Acceptar" i esperar la sortida dels resultats.

- Mentre dura l'anàlisi els botons i el menú romandran desactivats, un cop a finalitzat l'anàlisi, s'activarà el boto "Nou", si el cliquem, s'esborrarà la pantalla i s'activarà el boto "Acceptar", llavors podem repetir l'anàlisi.

TEST D'UNA SEQÜÈNCIA INTRODUÏDA PER TECLAT.
****

1 - Clicar Tasques > Test seqüència > Nova

- la interfície modificarà la seva estructura i apareixerà una nova pantalla.

2 - Introduir la seqüència per mitja del teclat en l'àrea titulada "espai per escriure la seqüència".

- Inicialment l'àrea de text conte la seqüència 10010011110001001110

3 - Clicar el boto "Acceptar" i esperar la sortida dels resultats.

- La seqüència només pot estar formada per zeros i uns, sinó, sortirà un missatge d'error.

- Mentre dura l'anàlisi els botons, el menú i l'àrea de text romandran desactivats, un cop a finalitzat l'anàlisi, s'activarà el boto "Nou", si el cliquem, s'esborrarà la pantalla i s'activarà el boto "Acceptar", llavors podem repetir l'anàlisi o modificar la seqüència.

****
Fi ajuda.

Figura 26 . Contingut del document ajudaXifrador.txt

XIFRAR / DESXIFRAR

- Atenció, aquesta versió no xifra documents amb una allargada superior als 3 MBytes, recomanem fer particions dels arxius o comprimir-los.

- Les carpetes poden tenir la mida que és vulgui, si és volen comprimir, cal fer-lo després de xifrar-les.

1 - Introduïm la contrasenya i la validem amb la tecla "enter".

- Si la contrasenya no és valida, sortirà un missatge d'avis i caldrà tornar a introduir-la.

- Mentre no es canviï la contrasenya, el seu valor és "123456".

- Per canviar la contrasenya cal clicar Arxiu > Canviar contrasenya i seguir les instruccions.

2 - Clicar Tasques > Xifrador > Xifrar, o Tasques > Xifrador > Desxifrar.

- Ara, només cal arrossegar l'arxiu/carpeta que vulguem xifrar o desxifrar damunt de la interfície.

- A l'àrea de text anirà sortint la informació del procés que s'està realitzant.

- Cal tenir present que la velocitat de Xifrat/Desxifrat és aproximadament de 6 segons per cada MByte.

- Mentre duri el procés, els menús Arxiu i Tasques romandran desactivats, i l'àrea d'arrossegament també romandrà desactivada.

- En el cas de fer un XIFRAT, obtindrem un nou arxiu/carpeta amb el mateix path i que finalitzarà amb el sufix "_xif", és a dir, si tenim l'arxiu path/A.dat, obtindrem un nou arxiu path/A.dat_xif i si path/A.dat_xif EXISTEIX, el SOBREESCRIURÀ!!.

- En el cas de fer un DESXIFRAT, obtindrem l'arxiu/carpeta amb el path original, és a dir, si tenim path/A.dat_xif, obtindrem un nou arxiu path/A.dat i si path/A.dat EXISTEIX, el SOBREESCRIURÀ!!.

- Tant en el procés de Xifrat com en el de desxifrat, els arxius/carpetes ARROSSEGATS no seran destruïts.

- Si Desxifrem o Xifrem una carpeta amb continguts barrejats (material xifrat amb material

desxifrat), tindrem dos possibles resultats:

a) Si no hi ha la possibilitat de trobar un arxiu amb nom repetit al final dels processos, tot el material xifrat es dipositarà a la carpeta de xifrats i tot el material desxifrat es dipositarà a la carpeta de desxifrats.

b) Si és dona el cas de trobar una situació amb material d'igual nom al final dels processos, el material duplicat romandrà a la seva carpeta original, és a dir, suposem que volem xifrar una carpeta que conté els arxius A.dat_xif i A.dat, el resultat, serà la carpeta amb el sufix _xif que contindrà l'arxiu A.dat_xif (que correspon al xifratge de A.dat), en canvi si el contingut és A.dat_xif i AB.dat el resultat serà la carpeta amb el sufix _xif que contindrà l'arxiu A.dat_xif (que estava a la carpeta original) i AB.dat_xif, i a la carpeta original només tindrem l'arxiu AB.dat.

TEST DEL XIFRADOR.

1 - Introduïm la contrasenya i la validem amb la tecla "enter".

2 - Clicar Tasques > Xifrador > Test xifrador.

- S'obrirà una nova finestra on aniran apareixent les dades referents a fer un test al generador amb la mateixa configuració amb la que xifra.

- El test de series, freqüència, poker i ràfegues és realitza sobre una seqüència de 20 Kbits.

- El test de la correlació (A(8)), el test de correlació de Golomb i el test de la complexitat lineal es fan per a un tram de la seqüència de 3 kbits.

- El test de Maurer és fa per una allargada de bloc de L=10.

- Aquest test no genera cap arxiu de dades, només els resultats trets per pantalla.

****
Fi ajuda.

Observacions Importants.

Ës molt important **no** manipular (sense coneixement de causa) els arxius que formen part de l'aplicació, especialment l'arxiu **USUARI.Keystore i generador_xif.**

La destrucció de l'arxiu **USUARI.Keystore** suposa no poder recuperar cap document xifrat. **Aconsellem fer una copia de seguretat d'aquest document.** Aquest arxiu

conté la clau publica i privada de l'usuari, i s'ha construït amb l'eina Keytool amb els següents paràmetres:

keytool -genkey -keyalg rsa -keysize 1024 -alias XIF -keystore USUARI.keystore -keypass **123456** -dname "cn= USUARI " -storepass USUARI -storetype JCEKS

Per tant, l'usuari pot canviar la contrasenya (keypass) d'accés a la clau privada sempre que vulgui, només ha de clicar l'arxiu canviarContrasenya.bat i seguir les instruccions.

Si és vol substituir el certificat, cal fer-ho abans de realitzar qualsevol xifrat. Aquesta versió de l'aplicació ja té configurats per defecte la contrasenya d'accés del magatzem de claus ("USUARI") i el alias del certificat ("XIF") , per tant, el nou certificat tindrà que tenir el mateix valor per aquests paràmetres, sinó, caldrà modificar la crida que fa el constructor de la classe Principal i tornar a compilar el projecte

Si substituïm el certificat caldrà generar un nou arxiu generador_xif. Per fer-ho, cal crear un **projecte** i substituir a la classe Principal les crides del mètode guardarEstatGenerador() per valors que l'usuari consideri correctes en funció de les especificacions (polinomis primitius, estats inicials, velocitats, valor bn, i comptadors de bits).

De fet, l'aplicació és responsabilitat de l'usuari. Si l'usuari vol, pot generar un nou projecte per capturar la configuració actual del generador, modificar l'entorn PKI, comprovar el funcionament individual de les classes, etc...

6 Joc de proves

Per no allargar aquest apartat, ja que caldria omplir moltes pagines amb les proves realitzades, mostrarem les mínimes i necessàries, que són:

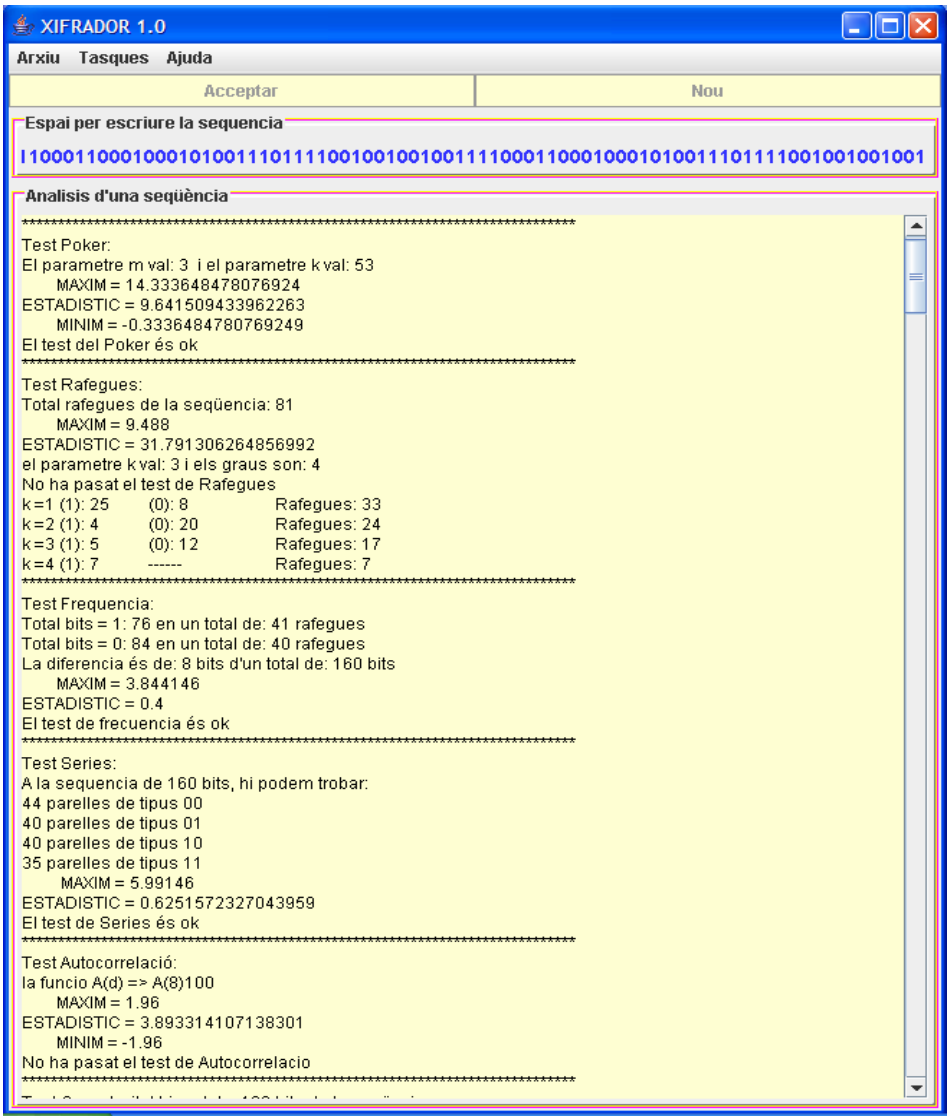
1. Comprovar que els tests implementats realitzen els càlculs correctament.

2. Comprovar que el generador proposat genera seqüencies pseudoaleatories capaces de superar els diferents tests.
3. Comprovar que el test fet al generador configurat amb les dades que conté l'arxiu generador_xif, passa els tests.
4. Comprovar que el procés de xifrat d'un document i posterior desxifrat, genera el document original.

6.1 Verificar la implementació correcta dels tests.

Per fer la primera prova farem passar el test a la seqüència que podem trobar a [MEN1] pag. 182. Aquesta seqüència ja esta analitzada, per tant, si la nostra implementació és correcte, els nostres resultats han de ser idèntics. Com podem comprovar a la següent figura, el resultat obtingut és el mateix. és :

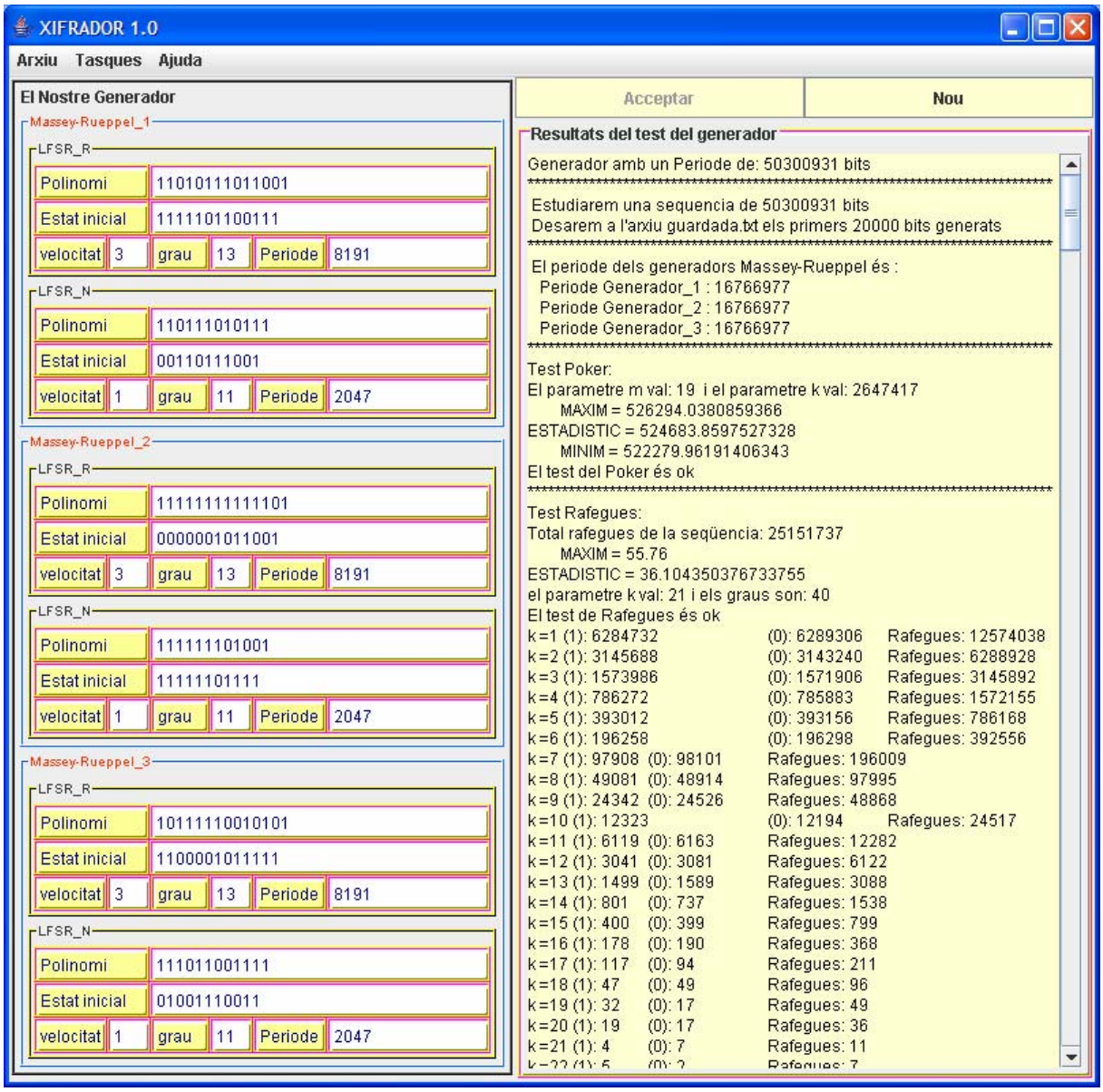
Figura 27 . Captura pantalla per verificar implementació tests.



6.2 Verificar el comportament pseudoaleatori del generador

Per efectuar la segona prova farem un test del generador, les dades les introduirem des de la interfície, en aquest cas avaluarem tot el període d'un generador (50300931 milions de bits) i farem el test de Maurer fins a una longitud de bloc L=12 que és la màxima que permet el seu període. A la següent figura podem observar l'aspecte de la interfície amb una part dels valors. La totalitat dels valors obtinguts la tenim annexada al final del document.

Figura 28. Captura pantalla per fer un test del generador.

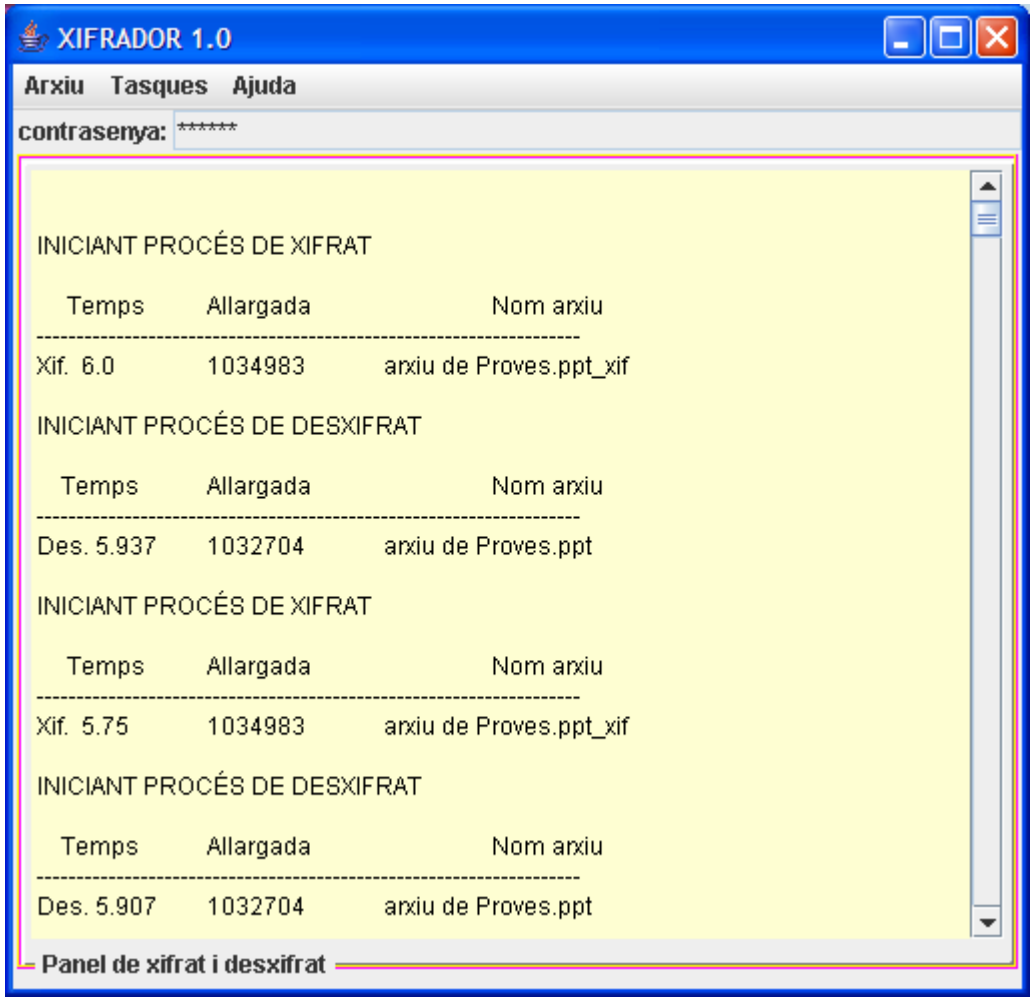


6.3 Verificar el procés de xifrat i desxifrat

La tercera i quarta prova la farem conjuntament, és a dir, xifrarem un document d'aproximadament 1Mbyte i el desxifrarem, llavors farem un test al generador, aquest test és realitzarà sobre els següents 20000 bits (cal recordar que aquests bits no és perden).

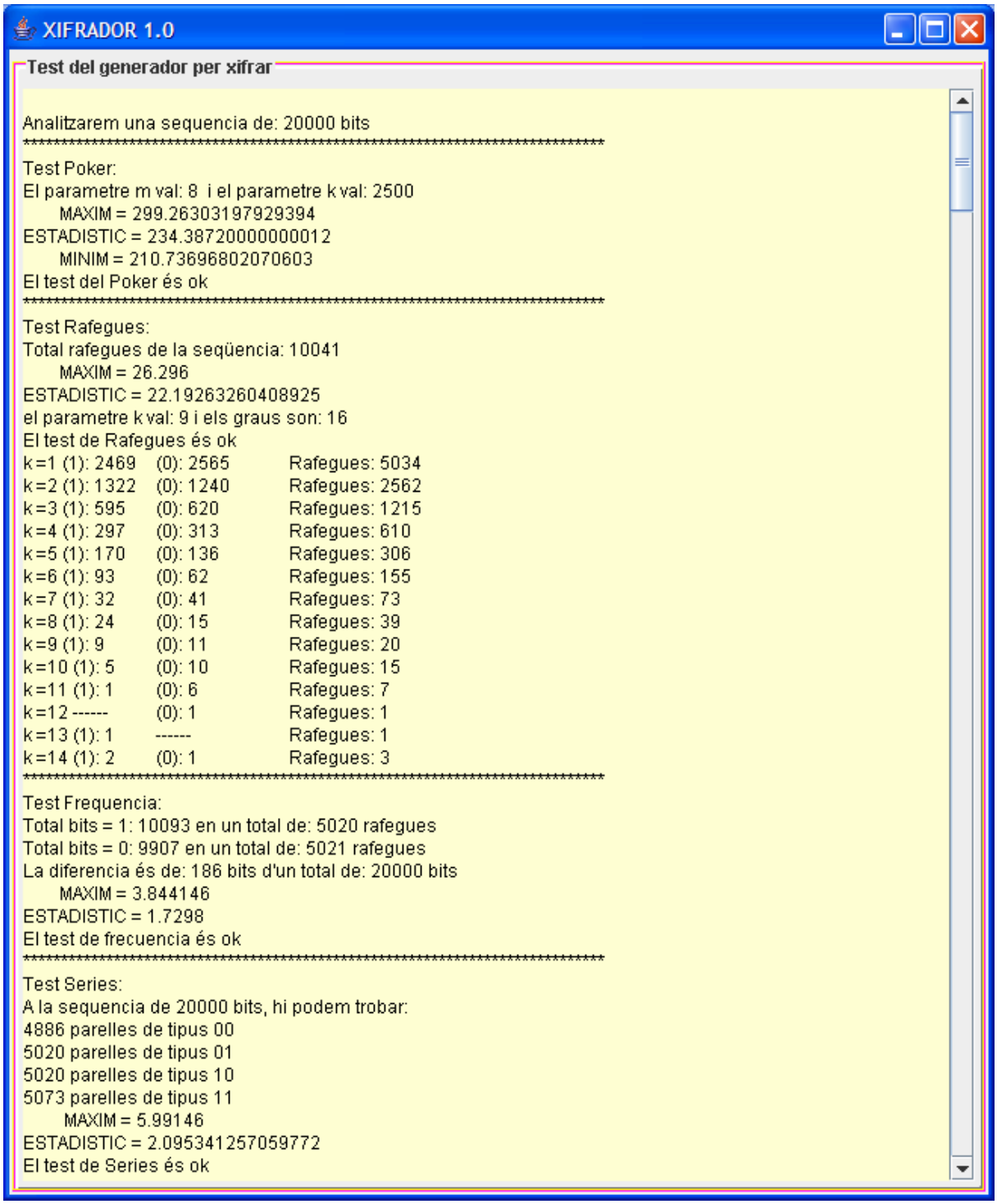
A la seGüent figura podem veure la interfície en el procés de xifrat i desxifrat , la diferencia de bytes és l'espai ocupat per la capçalera i la signatura del document.

Figura 29. Captura pantalla en el procés de xifrat i desxifrat.



A la seGüent figura podem observar la pantalla després de fer el test dels bits que genera el generador amb la configuració per xifrar.

Figura 30. Captura pantalla en el procés fer un test dels bits per xifrar.



Podem concloure que el resultat de les proves és correcte i compleix les expectatives inicials.

7 Conclusions.

Donat que en altres TFC's que he llegit, es fan comentaris no massa encoratjadors de les prestacions del llenguatge Java- voldria comentar que en el meu cas tot ha anat perfecte i que la utilització de les llibreries que proporciona el llenguatge Java és idònia per realitzar la implementació d'aquesta aplicació (interfícies gràfiques incloses) i obtenir un perfecte disseny orientat a l'objecte, de fet l'aplicació esta perfectament definida gracies a les classes de Java. L'entorn PKI és d'allò més senzill i el fet de serialitzar els objectes encara facilita més la feina. Respecte de treballar a nivell de bit, quan ha estat necessari s'han convertit els 1's i 0's a booleans i per fer les operacions a nivell de bit només ha calgut fer l'operació XOR amb els flags apropiats. La implementació de la interacció també és molt simple gracies a les interfícies que les classes poden implementar "implements".

Arribat aquest moment sembla que ens hem fet un fart de treballar per realitzar quelcom que ja esta fet, de fet de generadors en n'hi ha un grapat, i d'aplicacions que xifren un altre, però el treball d'investigar, buscar i intentar dissenyar un generador pesudoaleatori amb les prestacions aconseguides a estat molt enriquidor.

Això si, cal efectuar moltíssims més anàlisis i proves per tal de verificar els resultats obtinguts.

Aquest TFC implica el correcte coneixement de diferents eines informàtiques i suficient destresa a l'ora de fer-les anar. Si no es té aquesta destresa cal invertir molt més temps per dur a terme les tasques i aquest ha estat el problema principal.

Per l'altra banda s'han aconseguit les fites proposades en el pla de treball i el resultat final és exactament el que s'havia proposat:

- Dur a terme una nova implementació d'un mòdul generador de bits per utilitzar-lo en la construcció d'una aplicació per xifrar i desxifrar arxius, a l'ora que aquesta mateixa aplicació ens permet experimentar amb el generador i observar el seu rendiment al llarg del seu període de xifratge.

El producte obtingut és una aplicació per xifrar i desxifrar arxius utilitzant la xifra de Vernam, on el generador de bits presenta molt bones qualitats pseudoaleatories, amb una velocitat de xifratge de 1MB cada 6 segons per a un període de 6.755.392.995.459.075 bits, aproximadament 786.431GB (si reduïm el període obtindrem més velocitat ja que els polinomis son més curts i conseqüentment reduïm el nombre d'operacions per bit.). Per tant considerem que cobreix perfectament les demandes de qualsevol usuari mentre aquestes no siguin industrials.

Aquesta és la primera versió de l'aplicació XIFRADOR 1.0. Aquesta aplicació esta dissenyada en base a 4 mòduls:

- Mòdul xifrador/Desxifrador.
- Mòdul Generador.
- Mòdul test
- Mòdul interfícies.

Cadascun d'aquests mòduls pot evolucionar i perfeccionar-se tant en el seu disseny com la seva implementació per tal d'aconseguir millorar les prestacions.

El mòdul xifrador/desxifrador pot millorar o implementar altres procediments relacionats amb l'entorn PKI.

El mòdul generador pot millorar la implementació dels seus procediments per tal de fer-lo més eficient a l'ora de generar els bits.

El mòdul test pot implementant més test i millorar la seva eficiència.

El mòdul interfícies es pot dissenyar per tal de crear unes interfícies més entenedores i amb moltes més prestacions. (gràfics, etc.).

Considerem que el producte obtingut és una primera versió que necessita evolucionar per convertir-se en un producte de millor qualitat. El seu disseny i la seva implementació faciliten aquesta tasca.

8 Terminologia, Glossari.

AES: Advanced Encryption Standard. Criptosistema Rijndael, que xifra blocs de 128 bits per mitjà d'una clau que pot variar la longitud entre 128, 192 o 256 bits.

Ambigüitat de clau: incertesa que queda sobre el valor de la clau un cop es coneix un criptograma.

Autenticitat perfecta: propietat que té una xifra quan la probabilitat d'engany amb èxit del receptor per part d'un criptoanalista és mínima.

Atac criptoanalític només amb text xifrat, En aquest atac el criptoanalista ha de trobar la clau basant-se només en el text xifrat que ha pogut interceptar. El mètode de xifratge, la llengua en què és escrit el text en clar i algunes paraules probables es poden suposar coneguts.

Atac criptoanalític amb text en clar conegut, En aquest atac el criptoanalista sap uns quants parells de text en clar/text xifrat i n'intenta deduir la clau o algun text en clar que no coneix.

API : Conjunt de rutines (procediments i funcions que en la sintaxi de la programació orientada a l'objecte, s'anomenen mètodes) que el llenguatge ofereix al programador.

Atac criptoanalític amb text en clar escollit, En aquest atac el criptoanalista que intenta deduir la clau és capaç d'adquirir el text xifrat corresponent a un text en clar escollit per ell mateix. Aquest atac representa la situació més favorable per al criptoanalista, i és, per tant, el més perillós. Les bases de dades que guarden la informació en forma xifrada es presten a aquesta mena d'atacs si l'enemic pot inserir registres en clar i observar els canvis en el text xifrat emmagatzemat.

Atac criptoanalític amb text xifrat escollit només té sentit en criptosistemes de clau pública, en els quals una de les dues transformacions de xifratge/desxifratge és pública. En aquesta modalitat d'atac, el criptoanalista és capaç d'adquirir el text en

clar corresponent a un text xifrat escollit per ell mateix. Tot i que és poc probable que el text en clar que ha obtingut sigui intel·ligible, pot ajudar a deduir-ne la clau.

Autenticitat: propietat de trobar-se, en relació amb la informació, en el mateix estat en què va ser produïda, sense modificacions no autoritzades; és sinònim d'integritat.

Autentificació: comprovació de l'autenticitat.

bytecode: m Resultat de la compilació d'arxius codi font Java i que es carrega, verifica i executa en l'intèrpret.

Clau: paràmetre, normalment secret, que controla els processos de xifratge i/o de desxifratge.

Criptoanàlisi: ciència que s'ocupa de trencar xifres, és a dir, descobrir la clau o el text en clar usats com a entrades de la xifra.

Criptografia: ciència i estudi de l'escriptura secreta.

Criptograma: text xifrat.

Criptologia: denominació conjunta de la criptografia i de la criptoanàlisi.

Criptosistema: xifra.

CBC: Cipher Bloc Chaining. Mode de xifratge de bloc en què es crea un encadenament dels blocs, de manera que el xifratge d'un bloc depèn de l'anterior per mitjà d'un bloc inicial aleatori per al xifratge.

CFB: Cipher Feedback. Mode de xifratge de bloc en què la llargada dels blocs de text no ha de coincidir amb la dels blocs del criptosistema.
Confusió: tècnica que té per objectiu que la relació entre la clau i el text xifrat sigui tan complicada com es pugui.

Complexitat lineal d'una seqüència: nombre de cel·les de l'LFSR més curt que és capaç de generar una seqüència.

Criptosistema de clau compartida: criptosistema en què tant l'emissor com el receptor comparteixen una sola clau que fan servir tant per a xifrar com per a desxifrar.

Criptosistema de flux: sistema de xifratge que utilitza un generador pseudoaleatori per a xifrar un missatge, sumant bit a bit el text en clar amb la seqüència pseudoaleatòria que resulta del generador.

Criptosistema històric: criptosistema emprat abans de l'aparició dels ordinadors.

DES: Data Encryption Standard. Criptosistema de xifratge de bloc que xifra blocs de dades de 64 bits de llargada per mitjà d'una clau de 56 bits i l'acció de caixes S.

Desxifratge: procés de transformació del text xifrat en text en clar.

Difusió: tècnica que té per objectiu la dissipació de les propietats estadístiques del text en clar mitjançant el text xifrat.

Desxifratge espuri: desxifratge que, donat el text xifrat *c* 5 *Ek*(*m*), existeix quan el xifratge sota una altra clau *k*' pot donar *c*; és a dir, *c* 5 *Ek*'(*m*) per al mateix missatge *m*, o bé *c* 5 *E k*'(*m*') per a un altre missatge amb sentit *m*'.

Distància d'unicitat: nombre mínim de caràcters de text xifrat tal que existeix una única clau que produeix aquests caràcters de text xifrat a partir d'un text en clar amb sentit.

Entropia de Shannon: entropia d'una variable aleatòria, que mesura, en bits, la incertesa sobre el valor que prendrà la variable.

ECB: Electronic Code Book. Mode de xifratge de bloc en què el xifratge dels blocs és independent l'un de l'altre i es porta a terme amb una mateixa clau.

Estat d'un LFSR: conjunt de valors continguts en cada cel·la d'un LFSR en un instant de temps.

Funció d'autocorrelació d'una seqüència periòdica: nombre de coincidències menys nombre de no-coincidències entre la successió original i la mateixa successió desplaçada k posicions, dividit pel període de la seqüència original.

Funció hash: funció que dóna com a sortida un resum de longitud fixa a partir d'una entrada consistent en un missatge arbitràriament llarg.

Funció hash unidireccional: funció hash que a més és unidireccional; és a dir, la sortida és fàcil de calcular a partir de l'entrada, però l'entrada és difícil de calcular a partir de la sortida.

Generador lineal: generador de seqüències de bits que només executa operacions lineals sobre els elements d'entrada per a obtenir la seqüència de sortida.

Generador no lineal: generador de seqüències de bits que executa operacions no lineals, com ara permutacions, sobre els elements d'entrada per a obtenir la seqüència de sortida; a més, pot fer servir també operacions lineals.

Generador pseudoaleatori: procés determinista capaç de generar una seqüència pseudoaleatòria.

IDEA: International Data Encryption Algorithm. Criptosistema de xifratge de bloc que xifra blocs de text en clar de 64 bits de llargada per mitjà d'una clau de 128, per mitjà de vuit iteracions idèntiques i una transformació de sortida.

Integritat: propietat de no haver sofert, en relació amb la informació, modificacions ni supressions parcials no autoritzades.

LFSR: registre de desplaçament realimentat linealment.

NLFSR: registre de desplaçament realimentat no linealment.
OFB: Ouput Feedback. Mode de xifratge de bloc en què el vector inicial es realimenta directament amb el resultat del xifratge de bloc.

Període: enter més petit p tal que $s_{i+p} = s_i$ per a tot $i \geq 0$, en què $\{s_i\}_{i\geq 0}$ és una seqüència periòdica.

Privacitat: dret de les persones a salvaguardar la seva intimitat, especialment pel que fa a les dades de què disposen les entitats públiques o privades.

Probabilitat d'engany: probabilitat que un criptoanalista enemic aconsegueixi enganyar el receptor fent-li acceptar com a bo un missatge modificat o un missatge inserit.

RSA: criptosistema de clau pública, basat en el problema de la factorització, publicat per Rivest, Shamir i Adleman l'any 1978.

Suposició de Kerckhoff: suposició segons la qual els algorismes de xifratge i de desxifratge d'un criptosistema són públics i el secret queda restringit a la clau emprada.

Signatura digital: procediment per a signar documents en format electrònic que consisteix en un algorisme de signatura privat del signatari i un algorisme públic per a la verificació de la signatura.

Signatura DSS: signatura digital estàndard del govern dels EUA, molt semblant a la signatura d'ElGamal.

Signatura d'ElGamal: signatura digital basada en el criptosistema de clau pública d'ElGamal.

Signatura RSA: signatura digital basada en el criptosistema de clau pública RSA.

Teoria de la informació: teoria introduïda per Shannon que mesura la informació des d'un punt de vista quantitatiu.

Triple DES: protocol de xifratge triple que utilitza el DES com a base per a obtenir un xifrador amb un espai de claus superior.

Variable aleatòria: variable que pren un valor entre una colla de valors possibles, de tal manera que cada valor possible té una certa probabilitat no nul·la de ser adoptat.

Verificació: comprovació que una signatura és vàlida, és a dir, que ha estat efectuada pel pretès signatari. Ha de ser possible per a tothom, és a dir, no ha de requerir coneixement de paràmetres secrets.

xifra o **criptosistema:** és un mètode secret d'escriptura, mitjançant el qual un text en clar es transforma en un text xifrat o criptograma. El procés de transformar text en clar en text xifrat s'anomena *xifratge*; el procés invers, transformar text xifrat en text en clar, s'anomena *desxifratge*. Tant el xifratge com el desxifratge són controlats per una o més claus criptogràfiques.

xifra de transposició: reordena els bits o els caràcters del text en clar; la clau de la xifra és el criteri de reordenació emprat.

xifra de substitució: canvia bits, caràcters o blocs de caràcters per substituts; la clau és el criteri de substitució emprat.

Xifres trencables o febles: xifres per a les quals el criptoanalista té prou recursos de càlcul per a determinar el text en clar o la clau a partir del text xifrat, o per a determinar la clau a partir de parells de text en clar/text xifrat.

Xifres computacionalment segures o fortes: xifres que no poden ser trencades a partir d'una anàlisi sistemàtica amb els recursos de què disposa el criptoanalista.

Xifres incondicionalment segures: una xifra ho és si, independentment de la quantitat de text xifrat interceptada pel criptoanalista, no hi ha prou informació al text xifrat per a determinar el text en clar de manera única.

Xifra de Vernam: consisteix a sumar una clau *K* aleatòria al text en clar *M* per a obtenir el text xifrat *C*. La diferència és que *M*, *C* i *K* prenen valors a {0, 1} i que la suma és mòdul 2 (és a dir, una *or exclusiva*): $C = M \text{ XOR } K$. altrament si M i K són naturals tenim C = (M+K) mod 2. La innovació fonamental introduïda per Vernam fou fer servir la clau només una vegada, és a dir, xifrar cada bit de text en clar amb un nou bit de clau escollit a l'atzar. Això requereix la transferència segura (amb missatgers armats, per exemple) d'emissor a receptor de tants bits de clau com text en clar vulguem xifrar més tard. Malgrat aquest inconvenient, aquesta és l'única xifra incondicionalment segura.

Xifra: mètode secret d'escriptura, mitjançant el qual un text en clar es transforma en un text xifrat.
Xifratge: procés de transformació d'un text en clar en un text xifrat.

9 Bibliografia

Llibres, Mòduls:

[uoc1] **Josep Domingo Ferrer** *Introducció a la Criptografia* **UOC P03/05024/02259.**

[uoc2] **Josep Domingo Ferrer Jordi Herrera Joancomart**i *Fonaments de Criptografia* **UOC P03/05024/02260.**

[uoc3] **Jordi Herrera Joancomart**i *Xifres de clau compartida: xifres de flux* **UOC P03/05024/02261.**

[uoc4] **Carles Rovira Escofet** *Contrast d'hipòtesis* **UOC P03/05057/00338**

[uoc5] **Josep Gibergans Bàguena** *Contrast de variàncies* **UOC P03/05057/00340**

[uoc6] **Jordi Herrera Joancomart**i *Xifres de clau compartida: xifres de bloc* **UOC P03/05024/02262.**

[uoc7] **Josep Domingo Ferrer** *Xifres de clau pública* **UOC P03/05024/02263.**

[uoc8] **Josep Domingo Ferrer** *Signatures digitals* **UOC P03/05024/02264.**

[uoc9] **Helena Rifà Pous** Infraestructura de clau pública PKI **UOC P03/05024/02265**

[uoc10] **Marta I. Tarrés Puertas** El llenguatge Java **UOC P03/05063/00950**

www:

[UPC1] **Ernesto Cruselles Forner**, **José Luis Melús Moreno,** *Secuencias pseudoaleatorias para telecomunicaciones,* 1996, cap 6
http://www.edicionsupc.es/virtuals/fmatcat.htm

[UPC2] **Ernesto Cruselles Forner**, **José Luis Melús Moreno,** *Secuencias pseudoaleatorias para telecomunicaciones,* 1996, cap 5
http://www.edicionsupc.es/virtuals/fmatcat.htm

[MAU1] **U. M. MAURER.** "A Universal Statistical Test for Random Bits Generators". Journal of Cryptography, vol 5, nº 2. 1992, pg. 89-105
ftp://ftp.inf.ethz.ch/pub/crypto/publications/Maurer92a.pdf.

[MEN1] ***A. Menezes, P. Van Oorschot and S. Vanstone,*** *Handbook of Applied Cryptography, CRC. Press, 1997.*
http://www.cacr.math.uwaterloo.ca/hac/

Les funcions hash

- http://www.anf.es/security/incidencias/ATAQUESHA1.pdf
- http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf

Xifradors, generadors, nombres aleatoris, algoritme Berlekamp-massey

- http://www.rhodes.edu/Psych/wetzel/random/intro.html
- [http://www.ciphersbyritter.com/
- [***P. M. Alcover Garau, J. M. García Carrasco y L. Hernández Encinas,*** *"Diseño de un Nuevo Generador de Secuencias de Bits Aleatorios por Entrada de Teclado", Novática, No. 174, marzo-abril, 2005.* http://ditec.um.es/~jmgarcia/papers/novatica05.pdf
- http://www.mindspring.com/~pate/crypto.html
- PERALTA, Francisco I, DUCHEN, Gonzalo I y VAZQUEZ, Rubén. **Complejidad Lineal y Algoritmo Berlekamp-Massey para la Construcción de Generadores de Secuencias Pseudoaleatorias**. *Inf. tecnol.*, 2006, vol.17, no.3, p.167-178. ISSN 0718-0764. http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-07642006000300023&lng=es&nrm=iso&tlng=es

[JAVA1] **Manuals sobre seguretat i criptografia amb Java.**
http://dymas.ii.uam.es/~flh/macprog/SCCEJ.pdf
http://www.programacion.com/tutorial/security1dot2/
http://www.uv.es/~sto/cursos/seguridad.java/html/sjava.html#toc6
http://java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html#PBEEx
http://ccia.ei.uvigo.es/docencia/SSI/practicas/jce.html
http://java.sun.com/javase/6/docs/technotes/guides/security/certpath/CertPathProgGuide.html

[BC1] **Obtenir la llibreria criptogràfica de Bouncy Castle versió 1.37, API i exemples** http://www.bouncycastle.org/docs/

[OP1] **obtenir openssl** http://www.slproweb.com/products/Win32OpenSSL.html

obtenir jdk1.5.0_13

https://sdlc1d.sun.com/ECom/EComActionServlet;jsessionid=40477A0F0FD4B6898B8B5DA79F41D402

10 Annexos

10.1 Requisits de les seqüències de xifratge de flux [uoc3]

10.1.1 Període [uoc3]

Hem vist que per a implementar un criptosistema de xifratge de flux necessitem un algorisme que ens doni com a sortida la seqüència de xifratge. El fet que aquest algorisme sigui determinista implica que la seqüència que en resulta no sigui completament aleatòria i, per consegüent, que a partir d'un cert moment es repeteixi. Ja hem dit que aquesta subseqüència que es va repetint és el període.

Formalment, podem definir el període de la manera següent:

"sigui $\{s_i\}_{i \geq 0}$ una seqüència periòdica, el **període** p és l'enter més petit tal que $s_{i+p} = s_i$ per a tot $i \geq 0$."

Tenint en compte que el període es repeteix, una vegada conegut és possible determinar exactament tota la seqüència de xifratge i trencar el criptosistema.

Per això, les seqüències que s'utilitzen per al xifratge de flux cal que tinguin un període molt llarg, ja que d'aquesta manera triguen molt a repetir-se i és més difícil predir-ne la sortida.

10.1.2 Postulats de Golomb [uoc3]

Per a aconseguir una seqüència que ens serveixi per al xifratge de flux no n'hi ha prou que el període sigui llarg. Cal també que la distribució dels zeros i els uns que la formen tingui una certa uniformitat. En concret, perquè una seqüència es pugui considerar pseudoaleatòria ha de complir els tres postulats de Golomb. Abans d'enunciar els postulats, introduirem la terminologia que emprarem.

Definirem una **ràfega** com un conjunt de bits consecutius iguals; és a dir, una ràfega de longitud **k** és el conjunt dels elements $s_t, \ldots, s_{t+k-1}$ tals que $s_{t-1} \neq s_t = s_{t+1} = \ldots = s_{t+k-1} \neq s_{t+k}$.

La **funció d'autocorrelació, AC(k)**, d'una seqüència periòdica $\{s_i\}_{i \geq 0}$ amb un període p es defineix com:

$$AC(k) = \frac{A - D}{p}$$

en què A i D són, respectivament, el nombre de coincidències i de no-coincidències de tot el període entre la successió $\{s_i\}_{i \geq 0}$ i la mateixa successió desplaçada k posicions, $\{s_{i+k}\}_{i \geq 0}$. És a dir:

- $A = |\{0 \leq i < p \text{ tal que } s_i = s_{i+k}\}|$
- $D = |\{0 \leq i < p \text{ tal que } s_i \neq s_{i+k}\}|$

Cal fixar-se que si k és un múltiple de p, llavors AC(k) = 1.

Els postulats de Golomb, que ha de verificar tota seqüència pseudoaleatòria, diuen el següent:

- Dins el període d'una seqüència pseudoaleatòria, el nombre de zeros i d'uns ha de ser el mateix o ha de diferir com a màxim d'una unitat; és a dir, ha de ser p/2 si p és parell i (p $\pm$ 1) / 2 si és senar.
- El nombre total de ràfegues de longitud k en un període ha de valer com a mínim $n/2^k$, essent n el nombre total de ràfegues del període.
- La funció d'autocorrelació AC(k) és bivaluada, és a dir, només pren dos valors: 1 si k és múltiple de p, i un altre valor constant si p no divideix k.

10.1.3 Complexitat lineal [uoc3]

Per a utilitzar una seqüència pseudoaleatòria per al xifratge de flux, cal que cada bit depengui el mínim possible de l'anterior. És a dir, la seqüència de xifratge ha de tenir un grau elevat d'impredictibilitat. Cal que la probabilitat que surti un 0 o bé un 1 sigui propera a 0,5 i que depengui en mínima mesura del bit de sortida anterior.
El concepte de **complexitat lineal** ens mesura aquesta impredictibilitat, és a dir, ens informa de quina part de la seqüència ens cal conèixer per a poder-la predir totalment, no obstant cal tenir en compte que una complexitat lineal elevada no implica necessàriament que la seqüència sigui impredictible, per exemple la seqüència 000000.....01 sempre tindrà una complexitat lineal màxima però és fàcilment predictible.

Per tant, donada una seqüència de bits, tindrem que calcular la seva complexitat lineal, però també caldrà efectuar l'estudi de com creix aquesta complexitat lineal. Per a calcular la complexitat lineal farem servir l'algorisme de Berlekamp-Massey.

10.1.4 Propietats Estadístiques

Per naturalesa pròpia, les seqüències de bits tenen un comportament que es pot analitzar amb mètodes estadístics anomenats **Test d'aleatorietat** on avaluem un determinat **estadístic de contrast.** Els resultats donats per aquests tests s'han d'avaluar per mitja d'un **contrast d'hipòtesi** que ens ha de permetre decidir amb un determinat **grau de significació α** si una seqüència és o no és aleatòria.
De tests aleatoris n'hi ha un bon grapat, nosaltres tractarem els que considerem més significatius.

10.1.4.a Contrast d'hipòtesis [uoc4]

El contrast d'hipòtesis és una tècnica estadística que ens permet discernir respecte d'un resultat que inicialment no compleix amb la teoria. Per exemple podem imaginar que hem comprat una màquina que fabrica rodes i que, segons el fabricant, en fa com a màxim un 2% de defectuoses. Després de fabricar 200 rodes, trobem que n'hi ha 6 de defectuoses (segons el fabricant haurien d'haver estat 4). Aquest augment del nombre de rodes defectuoses indica que la màquina no funciona bé, o podem pensar que només és casualitat?
Quan volem estudiar aquest tipus de problema és determinar bé les hipòtesis. Cal que en determinem dues, ja que la resolució del problema consistirà, en certa mesura, a escollir-ne una entre aquestes dues.
La **hipòtesi nul·la,** que indicarem per $\mathbf{H_0}$, és la hipòtesi de partida. Ha de recollir el fet que volem posar a prova.
La **hipòtesi alternativa** és la que, com el seu nom indica, oferim com a alternativa a la nul·la. Aquesta hipòtesi, que denotem per $\mathbf{H_1}$, representa que s'ha produït un canvi respecte de la situació descrita per la hipòtesi nul·la.
Per a poder plantejar correctament les hipòtesis, caldrà que entenguem bé l'estructura estadística que hi ha darrere de les dades i, per tant, haurem de conèixer quina distribució segueixen. Les hipòtesis s'expressaran normalment en termes d'algun paràmetre de la distribució de les dades que estudiem.

Un cop plantejades les hipòtesis nul·la i alternativa hem de prendre una decisió a partir de les observacions. D'una banda, hi ha dues decisions possibles:

- Acceptar la hipòtesi nul·la.
- Rebutjar la hipòtesi nul·la.

D'altra banda, hi ha dues situacions possibles:

- La hipòtesi nul·la és certa.
- La hipòtesi nul·la és falsa.

Això fa que puguem cometre dues classes d'errors diferents: podem acceptar la hipòtesi nul·la quan aquesta és falsa o podem rebutjar la hipòtesi nul·la quan aquesta és certa, tal com es veu en la taula següent:

		Decisions	
		Acceptar H_0	Rebutjar H_0
Situacions	H_0 és certa	Decisió correcta	Error tipus I
	H_0 és falsa	Error tipus II	Decisió correcta

Figura 2. Taula hipòtesis

Podem mesurar aquests errors amb la probabilitat que es donin les situacions respectives.

Així, podem considerar els errors següents:

- **L'error de tipus I** és la probabilitat que rebutgem la hipòtesi nul·la quan aquesta és certa.
- **L'error de tipus II** és la probabilitat que acceptem la hipòtesis nul·la quan aquesta és falsa.

Ara necessitem una regla de decisió que ens permeti de determinar si hem d'acceptar o rebutjar la hipòtesi nul·la a partir de les observacions. Com que podem cometre dos tipus d'errors diferents, ens interessaria prendre la decisió que els minimitzés tots dos. La solució d'aquest problema no és fàcil. Normalment, l'experiència ens ensenya que quan establim una regla de decisió que fa molt petit un dels dos errors, l'altre es fa gran.

Com que no podem fer petits els dos errors al mateix temps utilitzarem l'estratègia següent: buscarem regles de decisió que ens permetin de tenir limitat **l'error de tipus I**. Aquesta estratègia és conservadora, tendim a no rebutjar la hipòtesi nul·la excepte si els resultats són molt poc probables sota aquesta.

Una bona manera de fer petits els dos errors i, per tant, de millorar els nostres resultats, és augmentar la mida de les mostres que utilitzem.

10.1.4.b El nivell de significació α [uoc4]

El nivell de significació α d'un contrast és **l'error màxim de tipus I** que estem disposats a assumir.

Determinar el nivell de significació que cal fixar quan es comença l'estudi no és un problema estrictament matemàtic, ja que depèn de cada cas particular. Usualment es fa servir el valor estàndard de a = 0,05. Altres nivells utilitzats són de l'ordre de 0,1, de 0,01 o fins i tot de 0,001 (si volem augmentar la precisió).

A l'hora d'escollir el nivell de significació, indica que detectem un canvi respecte de la hipòtesi nul·la quan, en realitat, no n'hi ha. Per tant, podem tenir en compte les consideracions següents:

- Si detectar aquest canvi –que, en realitat, no existeix i només és conseqüència del caràcter aleatori de les dades– no és gaire greu, aleshores podem escollir un nivell de significació alt (per exemple, de l'ordre de 0,1).
- Si detectar aquest canvi –que, en realitat no existeix– és un error greu, aleshores hem de fixar un nivell de significació petit.

Però cal anar en compte. Com més petit sigui el valor de **α** que fixem, més tendència tindrem a acceptar la hipòtesi nul·la. El cas extrem seria fixar un nivell de significació 0. En aquest cas acceptarem sempre la hipòtesi nul·la, de manera que mai no es donarà l'error de tipus I, però és clar que si hem d'acceptar sempre, l'estudi que hem fet no ens aporta res de nou.

El fet de tenir fitat **l'error de tipus I** fa que els nostres contrastos siguin conservadors i tendeixin a acceptar la hipòtesi nul·la, a menys que hi hagi evidències molt clares que l'hem de rebutjar. Quan acceptem la hipòtesi nul·la no estem segurs que sigui realment certa, ja que no controlem **l'error de tipus II,** en canvi, quan rebutgem la hipòtesi nul·la estem segurs que l'hem de rebutjar ja que tenim fitat **l'error de tipus I**

Un cop tenim fixades les hipòtesis i també l'error de tipus I que estem disposats a assumir. Per a decidir si rebutgem la hipòtesi nul·la o no la rebutgem utilitzarem el que anomenarem **estadístic de contrast.**

Un estadístic de contrast és una funció de la mostra de la qual en coneixem la distribució sota la hipòtesi nul·la.

10.2 Tests Estadístics [UPC1][MEN1]

En aquest apartat descriurem els tests que considerem més rellevants per tal de determinar la qualitat pseudoaleatòria de les seqüències de bits.

10.2.1 Test de Freqüències

Aquest test pretén avaluar si la seqüència analitzada conté el mateix nombre d'uns i de zeros.

Si definim com :

- n_0 =nombre de zeros de la seqüència.
- n_1 =nombre d'uns de la seqüència.
- n= longitud total de la seqüència , on n= n_0+ n_1.

La funció de l'estadístic queda determinada per :

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

En aquest test l'estadístic de contrast segueix una distribució χ^2 amb 1 grau de llibertat. Estimarem un grau de significació α = 0.05 i acceptarem el test com a bo quan $X_1 < 3{,}844146$.

10.2.2 Test de Series

Aquest test avalua si a la seqüència hi ha igual distribució de les parelles 00, 01, 10, i 11 , és a dir, si tenim la seqüència 010100 i comptabilitzem totes les parelles consecutives, tenim : 01, 10, 01, 10, 00.

Si definim com :

- n_0 =nombre de zeros de la seqüència.
- n_1 =nombre d'uns de la seqüència.
- n= longitud total de la seqüència , on n= n_0+ n_1.

- n_{00}, n_{01}, n_{10}, n_{11} = nombre de parelles de cada tipus, on $n_{00}+n_{01}+n_{10}+ n_{11} = n-1$

La funció de l'estadístic queda determinada per :

$$X_2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

En aquest test l'estadístic de contrast segueix una distribució χ^2 amb 2 graus de llibertat. Estimarem un grau de significació α = 0.05 i acceptarem el test com a bo quan $X_1 < 5{,}99146$

10.2.3 Test del Poker

Aquest test avalua si a la seqüència hi ha igual nombre de subseqüències d'una determinada allargada i tipus.

Si definim com :

- n = longitud total de la seqüència.
- m = nombre enter positiu tal que . $\left\lfloor \frac{n}{m} \right\rfloor \geq 5 * 2^m$
- k = nombre enter positiu tal que. $k = \frac{n}{m}$
- n_j = nombre enter tal que $1 \leq j \leq 2^m$

Ara, dividirem la seqüència en k parts disjuntes, cadascuna d'allargada m i representada per j, llavors cada n_j comptabilitza el nombre de cops que apareix aquesta subseqüència.

Exemple, suposem que tenim :

0101101110101010110111010101011011101010010110111010101011011101010
1011011101010010110111010101011011101010101101110101001011011101010
10110111010101011011101010

Llavors n = 160, m = 3,i k = 53 i les parts disjuntes a comptabilitzar seran:

010 110 111 010 101 011 011 101 010 101 101 110 101 001 011 011 101 010 101
101 110 101 010 110 111 010 100 101 101 110 101 010 110 111 010 101 011 011
101 010 010 110 111 010 101 011 011 101 010 101 101 110 101 0

Ara cal comptabilitzar (n_j) totes les aparicions amb valor j tal que $1 \leq j \leq 2^m$

Per j = 1 => 000 $n_1 = 0$ Per j = 2 => 001 $n_2 = 1$ Per j = 3 => 010 $n_3 = 12$

Per j = 4 => 011 $n_4 = 8$ Per j = 5 => 100 $n_5 = 1$ Per j = 6 => 101 $n_6 = 19$

Per j = 7 => 110 $n_7 = 8$ Per j = 8 => 111 $n_8 = 4$ Total 53 parts

La funció de l'estadístic queda determinada per :

$$X_3 = \frac{2^m}{k}\left(\sum_{j=1}^{2^m} n_j^2\right) - k$$

En aquest test l'estadístic de contrast segueix una distribució χ^2 amb 2^m-1 graus de llibertat. Quan n és gran, aproximarem per una Normal $((2^{m-1}),(2^{m-1})^{0,5})$ i el grau de significació α = 0.05 (0.025 a cada cua), llavors acceptarem el test com a bo quan: $2^{m-1} - 1{,}96\cdot(2^{m-1})^{0,5} < X_3 < 2^{m-1} + 1{,}96\cdot(2^{m-1})^{0,5}$

10.2.4 Test de Ràfegues

Aquest test comptabilitza totes les ràfegues de la seqüència i avalua si la seqüència compleix el segon postulat de Golomb.

Si definim com :

- n = longitud total de la seqüència.
- e_j = nombre total de ràfegues d'allargada j que ha de tenir una seqüència aleatòria on, per cada j i tipus de ràfega (ràfegues de zeros o ràfegues d'uns) tenim: $e_j = \frac{n - j + 3}{2^{j+2}}$
- k = nombre enter positiu que és el valor més gran de j tal que $e_j \geq 5$
- G_j i B_j = nombre total de ràfegues de tipus 0 i tipus 1 respectivament, de longitud j tal que $1 \leq j \leq k$

És a dir, comptabilitzem (G_j i B_j) les ràfegues d'allargada j on $1 \leq j \leq k$, però tenint en compte que k = j_{max} tal que $e_{j\,max} \geq 5$, per tant primer caldrà trobar el valor k, seguidament calcularem els paràmetres e_j i llavors ja podem comptar les diferents ràfegues .

La funció de l'estadístic queda determinada per :

$$X_4 = \sum_{j=1}^{k} \frac{(G_j - e_j)^2}{e_j} + \sum_{j=1}^{k} \frac{(B_j - e_j)^2}{e_j}$$

En aquest test l'estadístic de contrast segueix una distribució χ^2 amb (2·k-2) graus de llibertat. Estimarem un grau de significació α = 0.05 i acceptarem el test com a bo quan : X_4 < {valor de la taula que correspon al grau (2·k-2) per a un α = 0.05 }

10.2.5 Test de correlació

En aquest test fem l'operació XOR entre els bits de la mateixa seqüència, però desplaçats una determinada distancia, anem sumant els resultats amb un 1 i després avaluem aquest valor per mitja de l'estadístic.

Si definim com :

- n = nombre enter. Determina la longitud total de la seqüència.
- d = nombre enter, és la distancia entre bits tal que $1 \le d \le \lfloor n/2 \rfloor$.
- A(d) = funció que retorna el nombre de bits diferents entre la posició j i la posició (j+d). on $A(d) = \sum_{j=0}^{n-d-1} s_j \oplus s_{j+d}$, és a dir, realitzem la funció XOR entre els bits de la seqüència (fins arribar a n-d-1) separats una distancia d i acumulem el resultat . {$(1 \oplus 1) = (0 \oplus 0) = 0$ i $(1 \oplus 0) = (0 \oplus 1) = 1$}

La funció de l'estadístic queda determinada per :

$$X_5 \frac{2\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$$

En aquest test l'estadístic de contrast segueix una distribució Normal N(0,1). Estimarem un grau de significació α = 0.05 (0.025 a cada cua), llavors acceptarem el test com a bo quan: $-1{,}96 \le X_5 \le 1{,}96$

El 3er postulat de Golomb fa referència a la correlació d'una seqüència de període p, cal entendre que aquí, la forma de mirar la correlació és diferent, en aquest cas la comparació entre bits es fa amb una copia de la seqüència, però cal mirar la copia com una roda que fem rotar k posicions a l'esquerra o a la dreta i tot seguit comptabilitzem els bits diferents (A) i els bits iguals (D). En aquest cas cal aplicar la funció d'autocorrelació:

$$AC(k) = \frac{A - D}{p}$$

Per tant caldrà aplicar aquest càlcul quan considerem que la seqüència a estudiar és un període que es repeteix infinitament i considerar la forma d'avaluar els resultats. Per exemple:

- Sabem que el 3er postulat de Golomb es diu que la funció ha de ser bivaluada, però sabem que la nostra seqüència no és aleatòria, sinó que és pseudoaleatòria, per tant podem fer una gràfica de AC(k) per a tots els valors k tal que $1 \leq k \leq p$ i determinar si l'estudi d'aquesta correlació permet endevinar el comportament de la seqüència.

10.2.6 Test universal de Maurer, l'entropia per bit

Al 1990 i 1992, U. M. Maurer [MAU1] va presentar i millorar un test estadístic per a generadors de bits pseudoaleatoris basant-se en els algoritmes universals de compressió de dades de Elias i de Willems. Aquest test pretén detectar una desviació significativa de l'estadístic d'una seqüència per mitja de mesurar l'entropia de cadascun dels bits generats pel generador [UPC2].

Aquest test és considera universal en el sentit que detecta diversos defectes estadístics en un sol càlcul, la idea és calcular la mitjana i la variància i poder calcular llavors la desviació estàndard, i avaluar-la ja que com més petita sigui la desviació estàndard millor qualitat aleatòria tindrà la seqüència. Per tant, una seqüència perfectament aleatòria segueix una llei N(0,1).

Si definim com :

s^N = seqüència a estudiar .

- bloc = conjunt de bits consecutius identificat pel seu valor en decimal.
- L = enter que determina l'allargada d'un bloc, tal que $1 \leq L \leq 16$.
- Q = determina el nombre de blocs consecutius que ha de tenir el segment d'inici tal que $Q = 10 \cdot 2^L$
- Segment d'inici = porció de bits que va des de 0 fins a (Q·L.-1)
- K = determina el nombre de blocs que ha de tenir el segment de prova tal que $K = 1000 \cdot 2^L$
- Segment de prova = porció de bits que va des de (Q·L) fins (K.·L)

- $A_n(s^N)$ = Per cada bloc n del segment d'inici, anotem l'índex de la ultima posició on hem trobat aquest bloc dintre d'aquest segment. Seguidament busquem aquests blocs al segment prova i retornem la diferencia entre els índexs, (entre on estava i on esta). Tot seguit actualitzem el valor de l'índex (on estava = on esta) i tornem a fer la mateixa operació fins arribar a K.

La funció de l'estadístic queda determinada per :

$$X_6 = \frac{1}{k}\sum_{n=Q+1}^{Q+K} \log_2 A_n(s^N)$$

El que estem avaluant és l'entropia màxima que aporta cadascun dels diferents blocs. També cal observar que donada una seqüència infinita, podem situar-nos en qualsevol bit i començar en aquella posició per realitzar aquest test.

L'estadístic de contrast segueix una distribució Normal $N(\mu,\sigma^2)$. Aquest estadístic és el valor esperat de la mitjana i és tindrà que calcular per a cada valor de L.

Estimarem un grau de significació $\alpha = 0.01$ (0.005 a cada cua), llavors acceptarem el test com a bo quan: $\mu - 2{,}58 \cdot \sigma \leq X_6 \leq \mu + 2{,}58 \cdot \sigma$

Per tant calcularem la desviació estàndard σ com: [MEN1]

$$\sigma = c\sqrt{\frac{\sigma^2(L)}{K}}$$

on α^2 (L) i μ son valors obtinguts de la taula següent corresponent als valors reals per a una seqüència aleatòria i c és un factor de correcció que és calcula com:

1. segons [MEN1] $c \cong 0.7 - \frac{0.8}{L} + \left(1.6 + \frac{12.8}{L}\right)\cdot\left(\frac{1}{K}\right)^{\frac{4}{L}}$ per $K \geq 2^L$
2. segons [MAU1] $c \cong 0.7 - \frac{0.8}{L} + \left(4 + \frac{32}{L}\right)\cdot\left(\frac{1}{K}\right)^{\frac{-3}{L}} \cdot \frac{1}{15}$ per $K \geq 2^L$

L	μ	α^2		L	μ	α^2
1	0.7326495	0.690		1	8.1764248	3.311
2	1.5374383	1.338		2	9.1723243	3.356
3	2.4016068	1.901		3	10.1700323	3.384
4	3.3112247	2.358		4	11.1687649	3.401
5	4.2534266	2.705		5	12.1680703	3.410
6	5.2177052	2.954		6	13.1676926	3.416
7	6.1962507	3.125		7	14.1674884	3.419
8	7.1836656	3.238		8	15.1673788	3.421

Figura 31. Taula (μ α^2) per seqüències aleatòries

10.2.7 Test del perfil de la complexitat lineal

És imprescindible efectuar un estudi de la complexitat lineal d'una seqüència. En el nostre cas estudiarem la complexitat lineal proposada per Rueppel.

Per calcular la complexitat lineal d'una seqüència cal trobar l'**LFSR** mes curt que la generi [UPC1]. Aquest lfsr és en realitat una porció (estat) de la pròpia seqüència, però donat el seu comportament, és capaç de generar-la totalment, i a més, aquesta porció de seqüència és pot representar per mitja d'un polinomi tal com:

$$P(s^N) = c \cdot X^n + c \cdot X^{n-1} + c \cdot X^{n-2}.+....+ c \cdot X^1+1$$

on c només pot valer 0 o 1 i n és l'enter que identifica el grau màxim del polinomi.

Per trobar aquest lfsr de longitud mínima i comptabilitzar la complexitat lineal farem servir una implementació de l'algoritme que Massey i Berlekamp van proposar al 1969 [MEN1] . Aquest algoritme, a mida que processa els bits de la seqüència estudiada, va retornant un valor per la complexitat lineal a l'ora que calcula els coeficients del LFSR més curt que pot generar la seqüència estudiada.

La complexitat lineal $L(s^n)$ esperada d'una seqüència aleatòria (s^n) és:

$$E\left(L\left(s^n\right)\right)=\frac{n}{2}+\frac{4+n \bmod 2}{18}-2^{-n}\left(\frac{n}{3}+\frac{2}{9}\right)$$

El procediment serà el següent.:

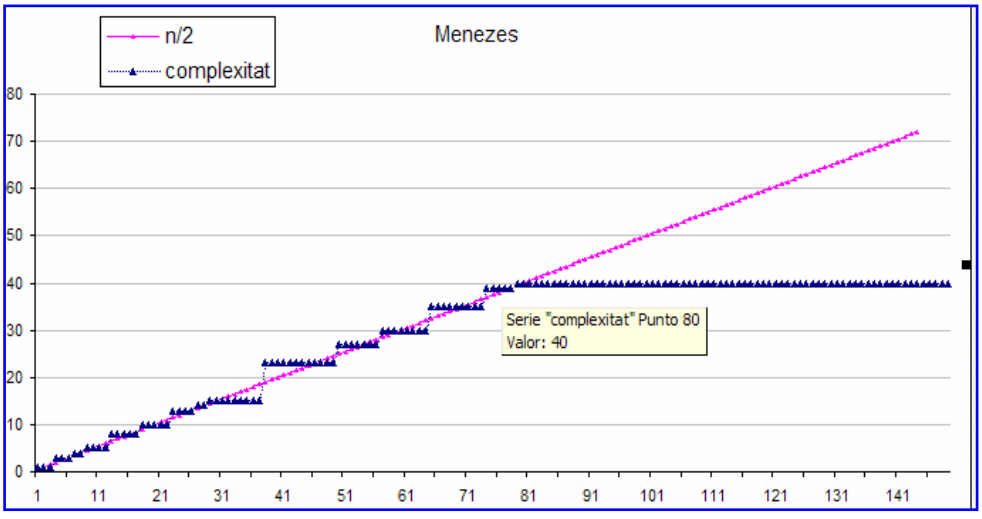
El creixement de la complexitat lineal de les seqüències aleatòries és irregular però té tendència a seguir la línia n/2, per tant, tindrem que comprovar que la nostra seqüència també es comporta de forma similar, és a dir, si la nostra seqüència és de període p, caldrà analitzar dos períodes complerts i comprovar que el valor de la complexitat s'apropa al valor del període p. Per cadascun dels bits analitzats l'algoritme de Berlekamp-Massey ens retornarà un valor per la complexitat, per tant nosaltres acumularem aquests valors i dibuixarem la corresponent gràfica per tal de comprovar quin és el comportament respecte a la recta n/2.

A la figura següent podem observar la recta n/2 i el comportament pròxim a aquesta línia de la seqüència que podem trobar a [MEN1] pag 182:

s^{40} = 11100 01100 01000 10100 11101 11100 10010 01001

Podem observar com el creixement del valor L = 40 de la complexitat s'estabilitzat a partir de n = 80 , és a dir, després d'haver processat 2 períodes de la seqüència. En aquest cas la complexitat és màxima i igual al període.

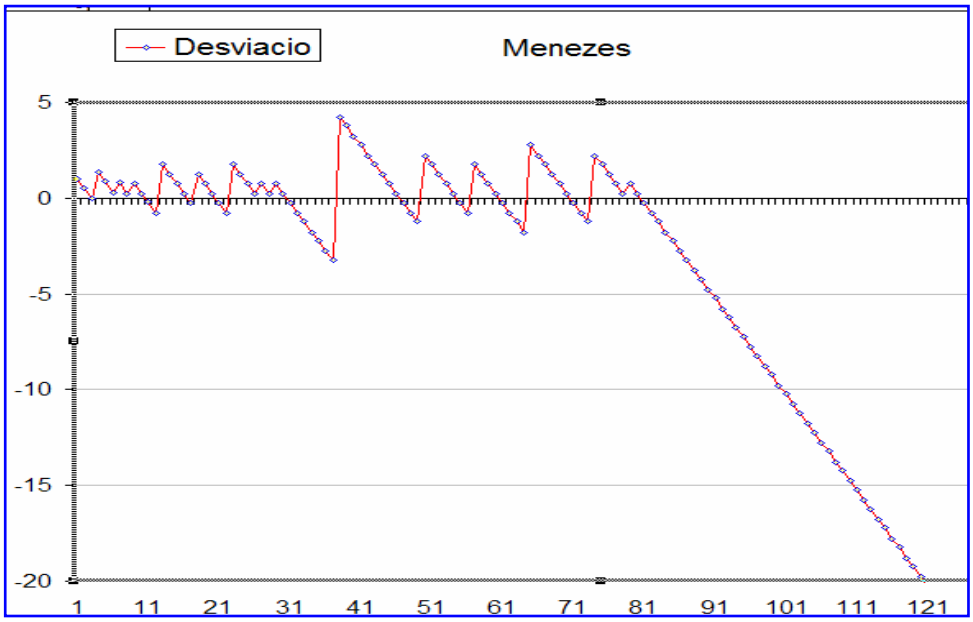
Figura 32. Gràfica del creixement del valor de la complexitat Lineal



Evidentment, caldrà rebutjar qualsevol seqüència on el valor de la complexitat no creixi de forma irregular al voltant de n/2.

Per una altra banda, calcularem $E\left(L\left(s^n\right)\right)$, és a dir, la complexitat esperada i la compararem amb la complexitat retornada per l'algoritme, això ens donarà un perfil de la desviació. Com més a prop del zero millor resultat. En aquest cas podem observar com a partir de L=40, el comportament comença a empitjorar fins allunyar-se dels valors òptims.

Figura 33. Gràfica de la variació del valor de la complexitat Lineal



10.3 Resultat de fer un test al generador.

```
Generador amb un Període de: 50300931 bits
********************************************************************************
 Estudiarem una seqüència de 50300931 bits
 Desarem a l'arxiu guardada.txt els primers 5000 bits generats
********************************************************************************
 El període dels generadors Massey-Rueppel és :
   Període Generador_1 : 16766977
   Període Generador_2 : 16766977
   Període Generador_3 : 16766977
********************************************************************************
Test Poker:
El paràmetre m val: 19  i el paràmetre k val: 2647417
       MAXIM = 526294.0380859366
ESTADISTIC = 524683.8597527328
       MINIM = 522279.96191406343
El test del Poker és ok
********************************************************************************
Test Ràfegues:
Total ràfegues de la seqüència: 25151737
       MAXIM = 55.76
ESTADISTIC = 36.104350376733755
el paràmetre k val: 21 i els graus son: 40
El test de Ràfegues és ok
k =1 (1): 6284732  (0): 6289306Ràfegues: 12574038
k =2 (1): 3145688  (0): 3143240Ràfegues: 6288928
k =3 (1): 1573986  (0): 1571906Ràfegues: 3145892
k =4 (1): 786272    (0): 785883  Ràfegues: 1572155
k =5 (1): 393012    (0): 393156  Ràfegues: 786168
k =6 (1): 196258    (0): 196298  Ràfegues: 392556
k =7 (1): 97908 (0): 98101    Ràfegues: 196009
k =8 (1): 49081 (0): 48914    Ràfegues: 97995
k =9 (1): 24342 (0): 24526    Ràfegues: 48868
k =10 (1): 12323    (0): 12194    Ràfegues: 24517
k =11 (1): 6119 (0): 6163 Ràfegues: 12282
k =12 (1): 3041 (0): 3081 Ràfegues: 6122
k =13 (1): 1499 (0): 1589 Ràfegues: 3088
k =14 (1): 801   (0): 737   Ràfegues: 1538
k =15 (1): 400   (0): 399   Ràfegues: 799
k =16 (1): 178   (0): 190   Ràfegues: 368
k =17 (1): 117   (0): 94    Ràfegues: 211
k =18 (1): 47(0): 49     Ràfegues: 96
k =19 (1): 32(0): 17     Ràfegues: 49
k =20 (1): 19(0): 17     Ràfegues: 36
k =21 (1): 4  (0): 7  Ràfegues: 11
k =22 (1): 5  (0): 2  Ràfegues: 7
k =23 (1): 2  ------   Ràfegues: 2
k =24 (1): 2  ------   Ràfegues: 2
********************************************************************************
Test Frequencia:
Total bits = 1: 25153535 en un total de: 12575868 ràfegues
Total bits = 0: 25147396 en un total de: 12575869 ràfegues
La diferencia és de: 6139 bits d'un total de: 50300931 bits
       MAXIM = 3.844146
ESTADISTIC = 0.7492370469246384
El test de freqüència és ok
********************************************************************************
Test Series:
```

```
A la seqüència de 50300931 bits, hi podem trobar:
12571527 parelles de tipus 00
12575868 parelles de tipus 01
12575868 parelles de tipus 10
12577667 parelles de tipus 11
        MAXIM = 5.99146
ESTADISTIC = 0.8781874179840088
El test de Series és ok
********************************************************************************
Test Autocorrelació:
la funció A(d) => A(8)25146497
       MAXIM = 1.96
ESTADISTIC = -1.1179707491853768
       MINIM = -1.96
El test d'Autocorrelació és ok
********************************************************************************
Test Complexitat Lineal de: 5000 bits de la seqüència.
Imprimim una part del perfil:
0,2,2,2,3,3,4,4,5,5,5,7,7,7,7,9,9,9,10,10,10,12,12,12,13,13,13,15,15,15,
16,16,16,18,18,18,18,18,21,21,21,21,21,21,24,24,24,24,25,25,26,26,27,27,28,28,28,30,30,30,
30,30,30,34,34,34,34,34,35,35,36,36,36,38,38,38,39,39,40,40,41,41,42,42,43,43,44,44,44,46,
46,46,47,47,47,49,49,49,49,49,49,49,49,49,56,56,56,56,56,56,56,56,57,57,58,58,59,59,60,60,
61,61,61,61,64,64,64,64,65,65,65,67,67,67,68,68,68,68,71,71,71,71,72,72,73,73,73,73,76,
La complexitat calculada és: 5000
********************************************************************************
Test Correlació Golomb per a una porció de: 5000
Imprimirem els pics superiors a 0.05 dels primers 1000 bits :

********************************************************************************
El bits generats per cadascun dels Generadors de Massey_Rueppel son:
El generador 1 ha produït: 16766976 bits
El generador 2 ha produït: 16766977 bits
El generador 3 ha produït: 16766978 bits
********************************************************************************
********************************************************************************
Test Maurer, avaluarem fins els blocs d'allargada: 12
L= 1 bits,   Q= 20 blocs/20 bits,  K= 2000 blocs/2000 bits
Total bits generats :2020
         MAXIM = 0.7374411580964887
ESTADISTIC = 0.730950294983651
          MINIM = 0.7278578419035112
El test és Ok.     temps de càlcul: 0.0
--------------------------------------------------------------------------------
L= 2 bits,   Q= 40 blocs/80 bits,  K= 4000 blocs/8000 bits
Total bits generats :8080
         MAXIM = 1.551594721921427
ESTADISTIC = 1.5249706858470837
          MINIM = 1.523281878078573
El test és Ok.     temps de càlcul: 0.0
--------------------------------------------------------------------------------
L= 3 bits,   Q= 80 blocs/240 bits,  K= 8000 blocs/24000 bits
Total bits generats :24240
         MAXIM = 2.4188459082820004
ESTADISTIC = 2.411218893592408
          MINIM = 2.384367691718
El test és Ok.     temps de càlcul: 0.016
--------------------------------------------------------------------------------
L= 4 bits,   Q= 160 blocs/640 bits,  K= 16000 blocs/64000 bits
Total bits generats :64640
         MAXIM = 3.3269025951976823
ESTADISTIC = 3.312574611841584
```

```
        MINIM = 3.2955468048023175
El test és Ok.     temps de càlcul: 0.031
--------------------------------------------------------------------------------
L= 5 bits,   Q= 320 blocs/1600 bits,  K= 32000 blocs/160000 bits
Total bits generats :161600
        MAXIM = 4.266268300509686
ESTADISTIC = 4.250949042105155
        MINIM = 4.240584899490314
El test és Ok.     temps de càlcul: 0.078
--------------------------------------------------------------------------------
L= 6 bits,   Q= 640 blocs/3840 bits,  K= 64000 blocs/384000 bits
Total bits generats :387840
        MAXIM = 5.227680970359833
ESTADISTIC = 5.221600770712275
        MINIM = 5.2077294296401675
El test és Ok.     temps de càlcul: 0.219
--------------------------------------------------------------------------------
L= 7 bits,   Q= 1280 blocs/8960 bits,  K= 128000 blocs/896000 bits
Total bits generats :904960
        MAXIM = 6.203764458492585
ESTADISTIC = 6.199348049968969
        MINIM = 6.188736941507416
El test és Ok.     temps de càlcul: 0.437
--------------------------------------------------------------------------------
L= 8 bits,   Q= 2560 blocs/20480 bits,  K= 256000 blocs/2048000 bits
Total bits generats :2068480
        MAXIM = 7.189216884946545
ESTADISTIC = 7.183453952387054
        MINIM = 7.178114315053455
El test és Ok.     temps de càlcul: 0.985
--------------------------------------------------------------------------------
L= 9 bits,   Q= 5120 blocs/46080 bits,  K= 512000 blocs/4608000 bits
Total bits generats :4654080
        MAXIM = 8.180475573639995
ESTADISTIC = 8.17710076141788
        MINIM = 8.172374026360004
El test és Ok.     temps de càlcul: 2.141
--------------------------------------------------------------------------------
L= 10 bits,   Q= 10240 blocs/102400 bits,  K= 1024000 blocs/10240000 bits
Total bits generats :10342400
        MAXIM = 9.175255389216082
ESTADISTIC = 9.171989409548893
        MINIM = 9.169393210783918
El test és Ok.     temps de càlcul: 4.703
--------------------------------------------------------------------------------
L= 11 bits,   Q= 20480 blocs/225280 bits,  K= 2048000 blocs/22528000 bits
Total bits generats :22753280
        MAXIM = 10.172141611301091
ESTADISTIC = 10.170780590407729
        MINIM = 10.16792298869891
El test és Ok.     temps de càlcul: 10.282
--------------------------------------------------------------------------------
L= 12 bits,   Q= 40960 blocs/491520 bits,  K= 4096000 blocs/49152000 bits
Total bits generats :49643520
        MAXIM = 11.17027705334929
ESTADISTIC = 11.168820935366819
        MINIM = 11.167252746650709
El test és Ok.     temps de càlcul: 22.188
--------------------------------------------------------------------------------
Fi Test Maurer
```

10.4 implementació de l'algoritme de simulació

Aquest algorisme permet simular el comportament del generador proposat i avaluar el grau d'aleatorietat de l'ordre com intervenen els generadors a l'ora de generar els bits.

```
public class Algoritme {
//constants i atributs.
   //nombre de bits per generar.
   private final static int LONG_SEQUENCIA =5000;
   //longitud del LFSR més curt
   private final static int LONG_LFSR = 5;
   //nombre total de operacions AND
   private final static int OPER_AND = LONG_LFSR;
   //nombre total de operacions XOR (ignorem les dels LFSR`s)
   private final static int OPER_XOR = LONG_LFSR-1;
   // valor del paràmetre n
   private final static int N =LONG_LFSR-2;
   //nombre total de generadors
   private final static int NUM_GENERADORS =3;
   //per anar desant el resultat de les operacions XOR
   private static boolean bn ;
   //per anar desant el resultat de les operacions AND
   private static boolean and;
   //taula per acumular el nombre de cops que intervé un generador.
   private static  int[] taula_Comptadors = new int[NUM_GENERADORS];
//taula per desar el nombre d'1s i 0s de cadascuna de les operacions XOR,
   // des de la primera operació XOR fins la N operació XOR.
   private static  int[][] taula_Xor=new int[N][2];
   //taula per desar el nombre d'1s i 0s de la operació XOR corresponent
   //al valor bn que ha fet intervenir a algun dels generadors.
   private static  int[][] taula_Bn_Generador=new int[NUM_GENERADORS][2];
   //buffer per arxivar l'ordre en que intervenen els generadors
   public static PrintStream buferOrdreGeneradors =null;

   /**
    * mètode que simula l'operació AND per mitja del mètode Math.Random().
    * Realitza N operacions XOR i acumula els resultats.
    */
   private static void calculaBn(){
     //carreguem la primera operació AND amb el valor de bn
     if(Math.random()<=0.75d)  bn =false;
     else bn=true;
     for (int i = 0; i<N;i++){
       //fem la següent operació AND
       if(Math.random()<0.75d) and=false;
       else and=true;
       //fem l'operació XOR
       bn = bn^and;
       //guardem el resultat de la operació XOR
       if(bn)taula_Xor[i][1] ++;//acumulem els uns
       else taula_Xor[i][0] ++;//acumulem els zeros
     }
   }
```

```
   /**
    * mètode que comptabilitza el nombre de cops que un generador intervé
en el càlcul.
    * Comptabilitza el valor de bn que ha activat el generador passat per
paràmetre.
    * Guarda en un arxiu l'índex del generador passat per paràmetre
    * @param idGenerador int amb l'índex del generador triat per generar un
bit.
    */
   private static void operacionsG(int idGenerador){
     //simulem el funcionament fins obtenir el valor bn.
     calculaBn();
     //incrementem el comptador que controla el nombre de cops
     //que intervé aquest generador.
     taula_Comptadors[idGenerador]++;
     //guardem el valor de bn que li ha correspost a aquest generador.
     if(bn)taula_Bn_Generador[idGenerador][1]++;
     else taula_Bn_Generador[idGenerador][0]++;
     //desem el generador que ha intervingut.
     buferOrdreGeneradors.print(idGenerador+"\n");
   }
/**
    * mètode que executa l'algorisme.
    */
   private static void executaAlgoritme(){
     //calculem els bits
     for (int i = 0 ;i <LONG_SEQUENCIA;i++){
       if(taula_Comptadors[0] >= taula_Comptadors[1]){
         if(bn)operacionsG(2);
         else operacionsG(1);
       }else if(taula_Comptadors[1] >= taula_Comptadors[2]){
         if(bn)operacionsG(0);
         else operacionsG(2);
       }else if(taula_Comptadors[2] >= taula_Comptadors[0]){
         if(bn) operacionsG(1);
         else operacionsG(0);
       }
     }
   }
/**
    * mètode per escriure a la consola els resultats obtinguts.
    * @param argv
    */
   private static void ToString(){
     System.out.println("Dades generals: ");
     System.out.println("\tEl nombre de cel•les del LFSR mes curt és:" +
LONG_LFSR);
     System.out.println("\tEl nombre d'operacions AND és :"+ OPER_AND);
     System.out.println("\tEl nombre d'operacions XOR és :"+ OPER_XOR);
     System.out.println("\tEl paràmetre n val: "+ N);
//   ***********************************************************************

     System.out.println("\nSimulació d'un Generador de Massey-Rueppel al
generar "+LONG_SEQUENCIA+ " bits");
     System.out.println("\tAnàlisi de l'evolució del valor b:");
     for (int i=0 ; i<N;i++){
       if(i+1==N)System.out.println("\tuns_bn =
"+(taula_Xor[i][1]*100d/LONG_SEQUENCIA)+"%"+"\tzeros_bn =
"+(taula_Xor[i][0]*100d/LONG_SEQUENCIA)+"%");
       else System.out.println("\tuns_b"+(i+1)+" =
"+(taula_Xor[i][1]*100d/LONG_SEQUENCIA)+"%"+"\tzeros_b"+(i+1)+" =
"+(taula_Xor[i][0]*100d/LONG_SEQUENCIA)+"%");
```

```
    }
  //*********************************************************************
    System.out.println("\nSimulació d'un Generador amb 3 generadors de
Massey-Rueppel al generar "+LONG_SEQUENCIA+ " bits");
    System.out.println("\tAnàlisi de la distribució del valor bn per a
cada Generador :");
    System.out.println("\tEl nombre de cops que ha intervingut cada
generador és :");
    System.out.println("\tg1="+ taula_Comptadors[0]+" =>
"+(taula_Comptadors[0]*100d/LONG_SEQUENCIA)+"%"
        +"  g2="+ taula_Comptadors[1] +" =>
"+(taula_Comptadors[1]*100d/LONG_SEQUENCIA)+"%"
        +"  g3="+ taula_Comptadors[2]+" =>
"+(taula_Comptadors[2]*100d/LONG_SEQUENCIA)+"%");
    System.out.println("\tLa distribució és :");
    for (int i=0 ; i<NUM_GENERADORS;i++)
      System.out.println("\tGenerador_"+(i+1)+"\ttrues =
"+(taula_Bn_Generador[i][1]*100d/taula_Comptadors[i])+"%"
          +"\tfalses =
"+(taula_Bn_Generador[i][0]*100d/taula_Comptadors[i])+"%");
  }
/**
   * metode main
   * @param argv
   */
  public static void main(String[] argv){
    //obrim un buffer per desar els índex dels generadors
    //per realitzar la gràfica o l'estudi estadístic de la Autocorrelació.
      try {
        buferOrdreGeneradors = new PrintStream
("sequenciaGeneradors.txt");
        } catch (FileNotFoundException e) {
        e.printStackTrace();
      }
    //escrivim la capçalera
        buferOrdreGeneradors.print("Ordre\n");
    executaAlgoritme();
    //buidem i tanquem el buffer.
      buferOrdreGeneradors.flush();
      buferOrdreGeneradors.close();
    //imprimim els resultats.
    ToString();
  }
}
```