Received 19 Juny, 2025; revised ; Date of publication 22 Juny, 2025; date of current version 21 Juny, 2025.

# Programació criptogràfica: Encriptació amb RSA

**Daniel García**

[1]Universitat de les Illes Balears, Departament d'Enginyeria Informàtica

Autor de contacte: Daniel García (email: daniel.garcia19@estudiant.uib.es)

**ABSTRACT**

En aquest document es presenta el desenvolupament d'una aplicació implementada en Java que permet l'encriptació i desencriptació d'arxius. L'objectiu principal és determinar la viabilitat d'aquesta tècnica
L'algorisme criptogràfic emprat es el RSA, de tipus de clau pública, que està basat en la factorització de nombres primers molt grans.
Com a funcionalitats addicionals, s'ha implementat la possibilitat de comprimir amb Huffman els arxius encriptats, a més d'una gestió persistent de les claus, ús de concurrència i reportatge del temps emprat.

**INDEX TERMS** RSA, nombres primers, Criptografia, Swing, MVC, Huffman, Clau pública, Clau privada, Ciberseguretat, Java

## I. Introducció

La criptografia és una branca molt important de la informàtica, sobretot en un mon hiperconnectat on la privadesa i la seguretat estan en primera plana.
En el marc de l'assignatura d'Algorismes Avançats, aquesta pràctica se centra en aplicar la teoria donada sobre el xifrat RSA a una pràctica on es pugui demostrar els coneixements adquirits al llarg del curs.

Aquest projecte explora l'ús d'algorismes probabilístics basats en mostreig Monte Carlo [2] per calcular ràpidament nombres primers.
A més, s'implementa l'algorisme RSA per obtenir una encriptació de clau pública .

La solució es construeix segons el patró Model–Vista–Controlador (MVC) [5], garantint la separació de responsabilitats entre la capa de dades i lògica (Model), la interfície d'usuari (Vista) i el control del flux (Controlador). Finalment, la GUI es desenvolupa íntegrament amb Java Swing [7], complint els requisits d'interacció i visualització sense utilitzar la consola en cap moment.

## II. Marc Teòric

### A. RSA

RSA es sistema criptogràfic [1] àmpliament utilitzat en l'àmbit de la criptografia, tant per xifrar arxius con per signar-los. RSA es un sistema de clau publica i privada [3], que permet la transmissió segura d'informació per la xarxa, entre d'altres. El xifrat es basa en la complexitat computacional que suposa factoritzar un nombre semiprimer, un nombre que només es pot formar per la multiplicació de dos nombres primers.

### B. Mostreig Monte Carlo

El mostreig Monte Carlo és una tècnica per estimar propietats d'una població a partir de mostres aleatòries [2]. Aplicat a nombres primers, el test de primalitat de Miller-Rabin [6] permet trobar nombres primers de gran mida amb una alta probabilitat d'encert. Això es possible al triar testimonis de forma aleatòria en un temps de $O(log^3 n * log(1/\epsilon))$ sent $\epsilon$ l'error causat. Aquest test es àmpliament usat en aplicacions on es depèn de nombres primers.

### C. Codificació de Huffman

La codificació de Huffman [4] es altament emprada per comprimir sense pèrdua arxius de forma eficaç. L'algorisme consisteix en crear una taula de freqüències per després generar un arbre binari per assignar un codi a cada símbol.

## III. Entorn de Programació

Per al desenvolupament d'aquesta pràctica s'ha configurat un entorn de treball basat en les eines i tecnologies següents:

- **Llenguatge de programació:** Java SE 23. S'ha triat per la seva robustesa, portabilitat i per disposar de llibreries estàndard per a la gestió de fils a alt nivell `ExecutorService`.
- **Interfície gràfica:** Java AWT i Swing. Swing s'empra per construir la GUI complint el patró MVC [5] i

permet gestionar esdeveniments i components gràfics de manera eficient [7].

- **IDE:** IntelliJ IDEA. Aquest entorn integrat facilita la navegació pel codi, la refactorització automàtica, la detecció d'errors en temps real i la integració amb sistemes de compilació com Maven o Gradle.
- **Control de versions:** Git amb repositori a GitHub. Permet portar un historial complet de canvis, gestionar branques de desenvolupament i col·laborar de forma distribuïda.
- **Documentació científica:** LaTeX a Overleaf mitjançant la plantilla IEEEtran. Facilita la redacció col·laborativa, la gestió de citacions i el format segons l'estàndard IEEE.

L'estructura del projecte segueix el patró Model–Vista–Controlador:

**Model:** Classes `Dades` i les diferents classes com `EncriptadorDesencriptador`, responsables de l'emmagatzematge de les claus, generar les claus RSA i l'encriptació i desencriptació.
**Vista:** Classe `Finestra` (extensió de `JFrame`), amb tots els components Swing (botons, etiquetes, `JFileChooser`, etc.).
**Controlador:** Classe `Main` (implementa la interfície `Comunicar`), que enllaça la vista i el model, registra listeners d'esdeveniments i invoca les operacions de xifrat, gestió i compressió.

## IV. Desenvolupament i Metodologia

### A. Arquitectura general del sistema

El projecte s'ha estructurat seguint el patró **Model–Vista–Controlador (MVC)**, que permet una clara separació de responsabilitats:

- **Model**: La interfície `Comunicar.Java` juntament amb la classe principal del programa `Main.java` són els responsables d'establir la comunicació i alinear les parts del programa per poder comunicar les peticions de l'usuari, executar l'algorisme classificador corresponent i mostrar els resultats obtinguts.
- **Vista**: Implementada amb Swing, proporciona una interfície gràfica intuïtiva i flexible per a la interacció amb l'usuari. Mostra els resultats del xifrat com les claus disponibles.
- **Controlador**: Actua com a intermediari, processant els esdeveniments de la GUI i invocant accions sobre el **Model**, a través de la interfície *Comunicar* i la classe *Main.java*.

### B. Estructura de paquets i classes

L'estructura del programa segueix les bones pràctiques d'encapsulament de Java. Els paquets principals es poden identificar d'acord amb el disseny de l'arquitectura Model-Vista-Controlador. La figura següent il·lustra l'estructura del nostre programa.

Els quadres de colors representen diferents elements: els blaus, les **classes**; els grisos, les **interfícies**;i els altres, els **paquets**. CAMBIAR

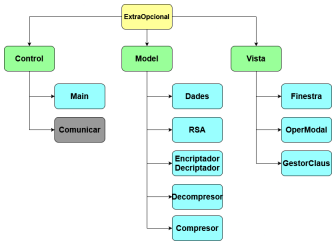


**FIGURE 1.** **Classes principals del projecte**

### C. Implementació del xifrat RSA

El xifrat RSA està implementat a la classe `RSA.java`, que realitza els passos següents:

1) Obtenció amb algorismes probabilístics dels nombres primers $P$ i $Q$.
2) Càlcul de $N = PQ$.
3) Càlcul de $\phi(N) = (P-1)(Q-1)$.
4) Càlcul de $E$ tal que $1 < E < \phi(N)$ i $E$ i $\phi(N)$ son coprimers.
5) Càlcul de $D$ tal que $(DE \mod \phi(N) = 1$
6) Crear les claus públiques i privades:
   - Clau pública (encriptar) $(E, N)$
   - Clau privada (desencriptar) $(D, N)$
7) Per operar amb una clau es calcula $res = val^{E|D} \mod N$

```
// Excerpt de RSA.generar()
for (int i = 0; i < 3; i++) {
    calls.add(executor.submit(() -> prim.
        otroProbablePrimo(new BigInteger("1" + "0".
        repeat(n)))));
}
P = calls.get(0).get();
Q = calls.get(1).get();
E = calls.get(2).get();

calls.add(executor.submit(() -> P.multiply(Q)));
calls.add(executor.submit(() -> P.add(new
    BigInteger("-1")).multiply(Q.add(new BigInteger
    ("-1")))));
N = calls.get(3).get();
FN = calls.get(4).get();

if (E.compareTo(FN) > -1){
    throw new RuntimeException("E is greater than
        FN");
}

D = E.modInverse(FN);
```

### D. Xifrat i desxifrat d'informació

Per xifrar un arxiu, o desxifrar, primer es divideix en tantes parts com nuclis té el processador, a continuació cada part s'opera respectivament de forma concurrent i finalment es forma el fitxer final.

```
byte[] fileIn = bis.readAllBytes();

int _step = fileIn.length / N_THREADS;
final int step = _step + 1 - (_step % N_THREADS);
for (int i = 0; i < N_THREADS -1; i++) {
    final int j = i;
    byte[] finalFileIn = fileIn;
    addConcurrent(() -> {
        encriptar(finalFileIn, j, j * step, (j+1) *
             step);
    });
}
byte[] finalFileIn1 = fileIn;
addConcurrent(() -> {
    encriptar(finalFileIn1, N_THREADS-1, (N_THREADS
        -1) * step, finalFileIn1.length);
});

waitAll();

try(BufferedOutputStream bos = new
    BufferedOutputStream(new FileOutputStream(
    outPath))){
    byte[] header = CryptHeader.createHeader(rsa,
        comprimir);
    bos.write(header);
    for (ArrayList<Byte> chunk : fileChunksOut) {
        for(byte b : chunk) {
            bos.write(b);
        }
    }
}
```

Per desencriptar es anàlog, però es te en compte que cada byte original es correspon a varis en el fitxer encriptat.
Si està activa la compressió del fitxer, s'han re realitzar passes extra, en el cas de estar encriptant, primer es xifra el fitxer cap a un resultat temporal, després es comprimeix el resultat intermedi i s'obté el resultat final. En el cas de desencriptar primer s'ha de descomprimir l'arxiu i després es desxifra.
Per mantenir un control del arxiu es crea un header dins el fitxer amb metadades per poder interpretar l'arxiu. La clase `CryptHeader` defineix, crea i comproba la validesa d'un header, aquest esta conformat per 4 elements:

1) *Magic number*: Identifica el fitxer com un arxiu encriptat, aquest és 0x444756 (DGV en hexadecimal)
2) *Comprimit?*: Byte que marca si les dades encriptades necessiten descompresió
3) *Checksum*: Checksum (SHA-256) de la clau utilitzada per xifrar l'arxiu original

### E. Gestió de claus

Per a gestionar les claus RSA intervenen varies classes

- `Dades`: emmagatzema els noms de les claus disponibles
- `Main`: crida a `RSA` per generar i guardar en fitxers les claus
- `GestorClaus`: La interfície de la gestió de les claus
- `CryptHeader`: Guarda quina clau s'ha emprat per el xifrat.

### F. Algorisme de Huffman

L'algorisme de Huffman és un *algorisme àvid òptim* que construeix, de baix a dalt, un arbre binari ple seleccionant en cada iteració els dos subarbres de freqüència mínima [4]. Aquesta construcció permet obtenir una codificació de longitud mínima per a cada símbol del conjunt, maximitzant l'eficiència en la compressió de dades.

El procés comença amb la construcció de la taula de freqüència de símbols, on es recull la freqüència d'aparició de cada símbol en l'entrada. A partir d'aquestes freqüències, es construeix iterativament un arbre binari, on els nodes més superiors representen els símbols amb freqüència més alta.

Una vegada es té l'arbre, es genera la taula de codificació mitjançant el recorregut de l'arbre. Durant aquest recorregut, s'assignen valors: 0 als nodes de l'esquerra i 1 als nodes de la dreta, fins a arribar als nodes fulla que representen els símbols de l'alfabet. Aquesta assignació proporciona un codi prefix-free, on cap codi és prefix d'un altre, garantint així la desambiguació durant la descodificació.

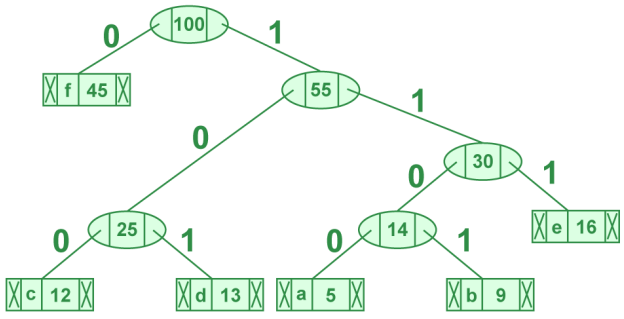


FIGURE 2. **Exemple d'un arbre de Huffman.**

Tot aquest procés construeix la nostra classe de Huffman. Aquesta classe encapsula les funcionalitats necessàries per generar i utilitzar l'arbre de Huffman en la compressió i descompressió de dades.

### G. Gestió de concurrència

Com es costum en les nostres pràctiques, la interfície s'executa en el seu propi thread, a més, cada exe-

cució d'un `Runnable` es fa sobre un thread dins un `ExecutorService`. A més, es controla l'aturada controlada dins els processos amb la funció aturar() de `Comunicar`.

Dins cada procés, també s'utilitza la concurrència per guanyar rendiment:

- A l'hora de crear claus, si varis nombres primers no depenen entre sí es pot calcular paral·lelament
- A l'hora de tractar amb fitxers, es divideix el fitxer en parts per a cada nucli per a poder operar més rapid

### H. Visualització i interacció amb l'usuari

La finestra principal ofereix les següents funcionalitats: permet encriptar o desencriptar un arxiu, gestionar les claus emmagatzemades i controlar els diferents processos en execució.

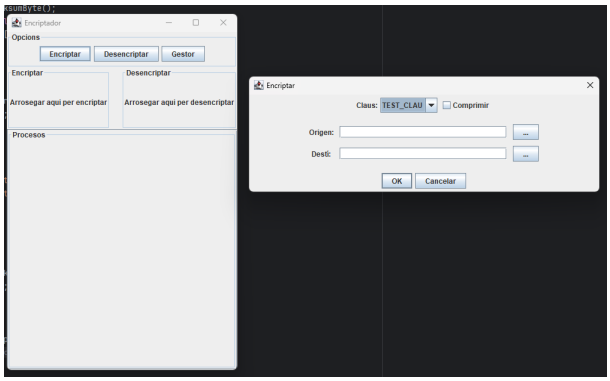


**FIGURE 3.** **Finestra principal amb modal d'encriptació**

A més, durant l'execució del programa s'utilitzen finestres modals per mostrar informació addicional o per demanar a l'usuari l'entrada d'informació

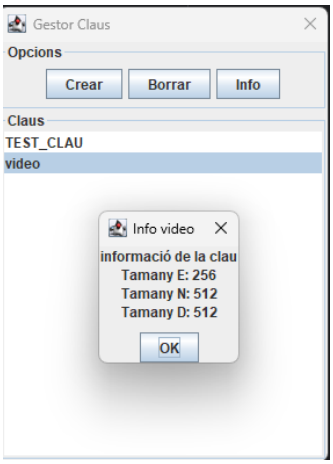


**FIGURE 4.** **Modal del Gestor de Claus amb informació sobre una clau**

### I. Flux de control i comunicació

El flux de control i la comunicació entre components segueixen fidelment el patró Model–Vista–Controlador (MVC). La classe `Main` actua com a controlador principal, gestionant els esdeveniments de la interfície gràfica i delegant les operacions al model de manera asincrònica. Per tal de garantir una arquitectura desacoblada i modular, s'ha dissenyat una interfície pròpia anomenada `Comunicar`, que defineix un conjunt de mètodes estàndard per a la comunicació entre la vista, el controlador i els processos del model.

#### *a: Interfície `Comunicar`*

Aquesta interfície inclou operacions com ara:

- `crearClau()`: Sol·licita la creació d'una nova clau.
- `encriptar()` i `desencriptar()`: activen els processos corresponents.
- `comunicar(String msg)`: canal de comunicació genèric per enviar missatges i estats entre components.
- `actualitzar()`: utilitzat per actualitzar la vista quan hi ha canvis en les dades.
- `aturar()`: Sol·licita la aturada d'un procés en execució.

Aquesta interfície és implementada pel controlador `Main`, i és passada als components que en necessiten accés, com ara la classe `Finestra` i `EncriptadorDesencriptador`.

#### *b: Controlador i gestió de tasques*

Per garantir una execució fluida i no bloquejant, el controlador empra un `ExecutorService` amb 3 fils d'execució:

```
private ThreadPoolExecutor executor = (
    ThreadPoolExecutor) Executors.
    newFixedThreadPool(3);
```

Això permet executar varis procesos de forma concurrent, sense sobrecarregar el sistema. La crida a, per exemple, encriptar es fa així:

```
@Override
public void encriptar(int id, String kName, String
    filePath, String outPath, boolean comprimir) {
    EncriptadorDesencriptador ed = new
        EncriptadorDesencriptador(id, kName);
    executar(id, ed, () -> {ed.encriptar(id,
        filePath, outPath, comprimir);});
}
```

Tant `RSA` i `EncriptadorDesencriptador`) implementa la interfície `Runnable` i `Comunicar`, un cop finalitzat el seu procés, pot comunicar-se amb el controlador a través del mètode `finalitzar()`.

#### *c: Patró* Singleton *i compartició de dades*

La classe `Main` s'implementa com un *singleton*, de manera que l'únic punt d'accés global al controlador és el mètode `getInstance()`. Això permet:

1) Compartir el mateix objecte `Dades` entre tots els components: el model és creat una sola vegada dins `Main` i recuperat des de qualsevol lloc amb:

```
Dades dades = Main.getInstance().getDades();
```

2) Facilitar les crides de la vista cap al controlador sense afegir dependències dures.
3) Garantir coherència de l'estat: com que tots els processos i la vista treballen sobre la mateixa instància de `Dades`, no hi ha duplicació d'informació. A més, qualsevol modificació es fa sempre dins el fil gestionat per l'`ExecutorService`; la vista només llegeix els resultats mitjançant `SwingUtilities.invokeLater`, evitant condicions de carrera.

D'aquesta manera, el *singleton* no trenca la separació de capes sinó que serveix com a mecanisme de registre centralitzat perquè tots els actors comparteixin l'estat i es comuniquin mitjançant la interfície `Comunicar`.

#### *d: Desacoblament i comunicació estructurada*

Aquest enfocament modular permet que el model i la vista no es coneguin directament entre ells. El model, encapsulat en classes com `EncriptadorDesencriptador` i `Dades`, pot operar de manera independent i només comunica resultats a través del controlador.
La vista, representada per la classe `Finestra`, també opera exclusivament mitjançant la interfície `Comunicar`, invocant mètodes sense dependre de la implementació específica del model.

Aquesta arquitectura ofereix una gran flexibilitat i facilita la integració de nous mètodes d'encriptació (com el AES), ja que aquests només necessiten accedir a les dades i notificar el controlador de l'estat de l'execució. Alhora, permet una gestió eficaç d'errors i estats mitjançant missatges textuals controlats centralment al `Main`.

### *J. Decisions de disseny i extres implementats*

Per tenir centralitzat les diferents funcionalitats, s'ha obtat per tenir poques classes, però especialitzades en el seu àmbit, per exemple: `RSA` s'encarrega de tot el relacionat amb les claus RSA, des de generarles, guardarles en arxius, llegir-les, etc. El mateix passa amb `EncriptadorDesencriptador`, que conté tota la funcionalitat per encriptar i desencriptar un arxiu, encara que utilitza les classes `RSA` i `CryptHeader` com a suport.

Per a l'extra de comprimir, s'ha agafat el model de la pràctica 4, aquest model ha estat modificat per ser més compatible, però gràcies al MVC han estat canvis menors. Per a l'extra de mostrar el temps s'ha reutilitzat, sense pràcticament haver fet cap canvi, les barres de càrrega de la pràctica 5.

La classe `PrimoProbable` és la publicada a l'aula digital, ja que no és necessari reinventar la roda, a més, `RSA` està basat també en el projecte de classe, però modificat substancialment per aprofitar la concurrència i la gestió de claus.

## V. Resultats i Comparació

En aquesta secció es posa a prova el funcionament del xifrat. Amb l'obtenció del temps d'execució de diferents claus es pot fer una comparativa del rendiment segons el tamany d'aquesta.

**TABLE 1. Classificació del temps de generació d'una clau segons el seu tamany**

| N | segons |
|---|---|
| 64 | 0,011 |
| 128 | 0,610 |
| 256 | 0,467 |
| 512 | 0,687 |
| 1024 | 19,256 |

**TABLE 2. Classificació del temps (s) d'encriptació i desencriptació segons el tamany de la clau**

| N | encriptar no c | desencriptar no c | encriptar c | desenciptar c |
|---|---|---|---|---|
| 64 | 1.699 | 0,603 | 0,919 | 0,838 |
| 128 | 2.152 | 3,304 | 2,577 | 3,782 |
| 256 | 11,175 | 21,109 | 12,863 | 21,605 |
| 512 | 76,532 | 147,109 | 80,045 | 149,684 |

**TABLE 3. Diferencia de pes comprimit i sense**

| N | no comprimit (MB) | comprimit (MB) |
|---|---|---|
| 128 | 11,0 | 9,98 |
| 256 | 21,9 | 19.9 |

Els resultats obtinguts es poden representar de manera visual mitjançant un gràfic de barres, que permet comparar fàcilment el rendiment dels diferents classificadors.

Els resultats permeten observar una clar augment del temps, de forma no lineal, d'execució en la mesura que augmenta N, també es pot intuir que el temps de desencriptació suposa el doble de temps respecte l'encriptació.

També una altre conclusió que es pot treure dels resultats es que el fet de comprimir o no l'arxiu encriptat afecti significativament al rendiment general.

Finalment, encara que els arxius siguin molt més grans xifrats que els originals, comprimir l'arxiu redueix un 10% del tamay original.

## VI. Conclusions

En conclusió, aquesta pràctica ha estat molt interessant per a mi, la criptologia es una branca que es dona de reüll a les diferents assignatures de les que he estat estudiant, i poder treballar amb ella, encara que sigui a un nivell bàsic, ha estat enriquidor. Tenint el compte la limitant de haver de compaginar aquesta pràctica amb exàmens i altres treballs puc dir que estic orgullós, encara que amb més temps hagués pogut fer altres coses, com implementar claus híbrides amb AES, i optimitzar, encara més, la encriptació, especialment a nivell espacial. També ha estat una ajuda

poder reutilitzar diferents components d'altres pràctiques, sense això segurament hagués necessitat un altre dia de feina.

## REFERENCES


[1] RL. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* https://people.csail.mit.edu/rivest/Rsapaper.pdf

[2] J. M. Hammersley i D. C. Handscomb, *Monte-Carlo-Methods*, Chapman & Hall, 1964.

[3] Martin E. Hellman *An Overview of Public Key Cryptography* https://www-ee.stanford.edu/~hellman/publications/31.pdf

[4] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[5] E. Gamma, R. Helm, R. Johnson i J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[6] Keith Conrad *THE MILLER–RABIN TEST* https://kconrad.math.uconn.edu/blurbs/ugradnumthy/millerrabin.pdf

[7] J. Gafter, "Swing Architecture and Concepts," JavaOne Conference, 1999.