Criptografia: l'art dels missatges secrets

Anàlisi dels fonaments teòrics de la criptografia, així com dels seus mètodes més coneguts (tant dels moderns com dels clàssics) i creació de dos programes informàtics d'encriptació de text

Alumne: David Marín Gutiérrez

Tutora: Eva Garcia Toledano

Centre: INS Salvador Dalí

Classe: 2n A BTX

Departament de matemàtiques

Data: 12/12/2017

Índex

1. Introducció

1.1. Interès personal

La criptografia és un tema que, personalment, mai m'havia cridat l'atenció com a tal. Fins i tot, fins que no vaig començar a fer aquest treball, ni tan sols coneixia de la seva existència. Llavors, perquè escollir aquest tema?

Quan el nostre tutor ens va comunicar que hauríem d'escollir un tema sobre el qual fer un treball, vaig començar a buscar diferents idees i camps per a desenvolupar el meu TR. Un dia, de casualitat, em va sortir a la portada del YouTube un vídeo que es deia "Cómo funciona la criptografía", i el vaig decidir veure per a combatre l'avorriment. El que vaig veure no només em va agradar molt sinó que, quan va acabar el vídeo, ja tenia decidit de quin tema tractaria el meu treball de recerca: de la criptografia.

Personalment, m'agraden molt les matemàtiques i la informàtica, i m'agradaria estudiar una de les dues a la universitat. Aquest tema, llavors, m'ofereix l'oportunitat d'ampliar els meus coneixements de les matemàtiques i dels seus fonaments i donar-los una aplicació pràctica a la vegada que em permet iniciar-me al món de la informàtica i de la creació de programes complexos mitjançant llenguatges de programació moderns.

1.2. Objectius

Els objectius d'aquest treball són, en resum, analitzar, de manera més o menys extensa, la història de la criptografia, la seva evolució (i la dels seus mètodes) i la seva importància al llarg del temps, les seves bases i aspectes teòrics més importants, així com els axiomes i teoremes matemàtics sobre els quals es fonamenta i, com a part pràctica del treball, crear un programa informàtic que realitzi l'acció d'encriptació i de desencriptació de text utilitzant un mètode semblant a l'RSA, i un altre que emuli el funcionament de la màquina Enigma.

Personalment, tractar aquest tema representa un repte per a mi: els meus coneixements previs de la criptografia eren nuls, només sabia el que havia vist a un vídeo d'Internet. Tinc com a objectius personals, llavors, introduir-me al món de la criptografia i la informàtica, així com expandir els meus coneixements matemàtics i de les matemàtiques en les quals la criptografia es basa.

1.3. Estructura

Aquesta memòria està dividida en quatre grans parts i un annex: la introducció (aquesta part); la part d'aspectes teòrics on, apartat a apartat s'explicaran, de la manera més detallada (i, en la major mesura possible, comprensible per al lector no expert), els trets i aspectes teòrics més importants i característics de la criptografia; una tercera part on es tractarà la seva història, evolució, els mètodes clàssics i moderns més importants a destacar, i es farà un incís en la importància del rol que la criptografia juga al món modern; i finalment un quart punt, el d'aspectes pràctics, on es deixarà constància de totes les parts i passes que s'han de portar a terme per tal de crear un programa informàtic (els seus objectius, les seves parts, l'algorisme principal, el llenguatge de programació escollit i el perquè d'aquesta elecció, etc.), i també es mostrarà el procés a través del qual he creat dos programes per a encriptar text (inspirats en la màquina Enigma i en el mètode RSA). També hi haurà un punt de conclusions finals (cinquè punt) on es sintetitzi tot el coneixement obtingut durant la recerca i l'elaboració de la part pràctica, així com les conclusions extretes de la totalitat del treball i de la seva realització.

A l'annex (sisè punt), trobarem les justificacions a veritats matemàtiques que hem considerat certes durant tot el treball (per a facilitar la seva lectura). Aquest treball inclourà esclariments (a peu de pàgina) que es considerin necessaris o d'ajut per al lector no expert en la matèria.

També s'hi inclourà un apartat de bibliografia i webgrafia de recursos utilitzats al llarg del desenvolupament d'aquesta memòria (setè punt).

1.4. Mencions i recursos

Per a la redacció d'aquesta memòria i per a la realització d'aquest treball, m'han estat imprescindibles els llibres *Cryptography: an itroduction* (per Nigel Smart) i *Discrete mathematics and its applications* (per Kenneth Rosen), així com les ponències del Dr. Christof Paar a la Ruhr-Universität Bochum, que ell mateix ha penjat a la web.

També voldria agrair-li molt a la meva professora de matemàtiques i tutora del treball per haver-me fet un seguiment constant del progrés, haver verificat que el contingut fos correcte i haver-me donat idees per a redactar i per fer l'exposició oral.

Finalment, també voldria agrair-li a la meva mare el temps que ha passat llegint-se l'esborrany final del treball, avisant-me de les faltes, corregint oracions incorrectes o poc elaborades i per haver-me ajudat a fer el treball més senzill i intel·ligible per a persones que no tenen molts (o cap) coneixements sobre la criptografia.

2. Fonaments teòrics i matemàtics

2.1. Introducció a la criptografia: definició i camp de treball

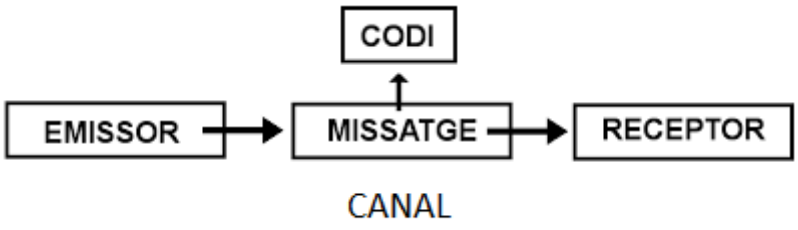
Podem definir la criptografia com la disciplina que s'ocupa de les tècniques de xifrat amb l'objectiu d'alterar el llenguatge en el que ha estat escrit un missatge per tal d'ocultar a usuaris no desitjats la informació que aquest conté. El seu principal i últim propòsit és aconseguir la confidencialitat, és a dir, que només els individus autoritzats puguin llegir un missatge i que encara que aquest missatge xifrat es fes públic, només els destinataris poguessin obtenir la seva informació.

Per a assolir aquest objectiu, la criptografia moderna fa servir com a eines principals les matemàtiques i la computació; les matemàtiques serveixen per a crear models i sistemes complexos que permetin xifrar text de manera sofisticada i la computació serveix per a automatitzar i agilitzar aquest procés de xifrat.

2.2. Teoria de la comunicació

Amb l'evolució i el progrés de la nostra espècie, han anat sorgint necessitats, i una de les principals a la vida de tot ésser humà és la comunicació: el procés a través del qual un subjecte transmet informació a un altre. Per a satisfer aquesta necessitat, vam inventar el llenguatge. El model general de l'acte de la comunicació té els següents elements:

- Emissor: qui, fent ús del llenguatge, transmet informació.
- Receptor: qui rep la informació de l'emissor i la interpreta.
- Missatge: informació que es transmet.
- Codi: llenguatge en el que l'emissor manifesta i expressa el missatge.
- Canal: medi a través del qual es propaga i manifesta el missatge.



Il·lustració 2.1., Esquema del model general de la teoria de la comunicació

Amb el pas del temps i amb l'avanç de la civilització, va sorgir una altra necessitat: poder escollir qui pot accedir a la informació d'un missatge i qui no. Amb altres paraules, va sorgir la necessitat de tenir secrets. La criptografia neix amb l'objectiu de satisfer aquesta necessitat.

Aplicant l'objectiu de la criptografia a la teoria de la comunicació, s'obté el següent:

- L'existència de l'emissor i el receptor roman. No obstant, però, sorgeix una tercera persona, que intentarà, a través de mitjans no lícits, interceptar els missatges entre l'emissor i el receptor. Per convenció i amb l'objectiu de simplificar el contingut, en els exemples, ens referirem a l'emissor com a Alice, al receptor com a Bob, i a un possible interceptor com a Eve.
- El missatge es parteix en dos:
  - El missatge original o sense xifrar es diu text pla.
  - El missatge xifrat o encriptat es diu criptograma.
- Apareix un nou element, la clau: element bàsic de tot criptosistema, el qual és emprat per usuaris teòricament autoritzats (l'Alice o en Bob) per a encriptar i desencriptar un missatge i obtenir la seva informació.
- Apareix una nova disciplina, la criptoanàlisi: branca de la criptologia responsable de trencar[1] criptosistemes.

2.3. Concepte de criptosistema

Un criptosistema o mètode de xifrat és una funció o algorisme que, donat un missatge i una clau d'entrada (que són l'*input*), retorna el missatge xifrat (que és l'*output*) sense tenir en compte el seu llenguatge, significat o sentit i de manera que sigui impossible d'entendre per persones no autoritzades.

Per exemple, si el mètode de xifrat emprat és el de desplaçament, els caràcters del missatge a xifrar experimentaran un desplaçament de n unitats (la paraula HOLA amb desplaçament de n=2 queda com JQNC).

Els seus components i característiques més importants són:

- Espai de missatges $M=\{M_1, M_2, M_3, ..., M_n\}$, on cada M_n és un possible missatge.
- Espai de claus $K=\{k_1, k_2, k_3, ..., k_n\}$, on cada k_n és una clau diferent que xifra un missatge M_n.
- Espai de xifrats $C=\{C_1, C_2, C_3, ..., C_n\}$, on cada C_n és el resultat d'aplicar l'algorisme de xifrat a un missatge M_n.

1: "trencar" un codi o un criptosistema vol dir tenir accés a la seva informació o mètode de funcionament a través de mitjans matemàtics, com ara l'anàlisi de freqüències

- Espai de claus de desxifrat T={t_1, t_2, t_3, ..., t_n}, on cada t_n és la clau necessària per a aplicar a C_n per tal de convertir-lo en M_n. L'espai T pot coincidir o ser dependent de l'espai K o no, com es veurà més endavant quan parlem de clau privada i de clau pública.
- El text pla ha de tenir el mateix nombre de lletres que el criptograma, és a dir, un caràcter de text pla ha d'equivaldre a un caràcter de criptograma.
- Un criptosistema ha de retornar només valors alfanumèrics.

Per tant, si interpretem els criptosistemes com una funció matemàtica, tenen la següent forma:

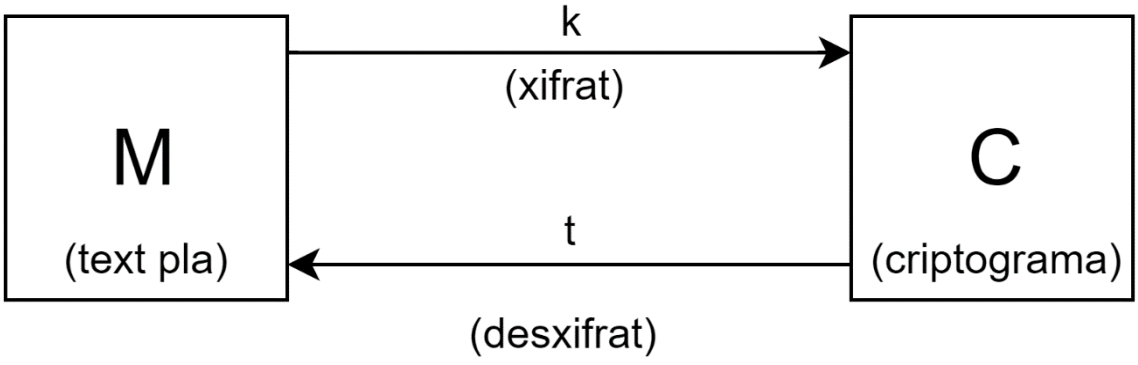
$$f(M_n, k_n) = C_n$$

on $f(M_n, k_n)$ és el xifrat o criptosistema, M_n i k_n, les entrades, són el text pla i la clau de xifrat, respectivament, i C_n l'*output*, el criptograma o text xifrat.

Per a tornar del criptograma al text pla, existeix la funció de desxifrat, la clau de la qual a vegades coincidirà amb o estarà en funció de la clau de xifrat, i d'altres serà una altra variable independent més. Expressat matemàticament:

$$f^{-1}(C_n, t_n) = M_n$$

on $f^{-1}(C_n, t_n)$ és la funció inversa de la de xifrat, C_n és el criptograma, t_n és la clau de desxifrat i M_n és el missatge original.



Il·lustració 2.2., Esquema del funcionament d'un criptosistema

2.4. Criptografia de clau pública i de clau privada

Tot criptosistema requereix l'existència d'una clau que permeti a l'Alice xifrar un missatge i, fent servir una clau (que pot ser la mateixa o una de diferent), permeti a en Bob desxifrar el missatge xifrat prèviament per ella. Els criptosistemes poden ser, en funció de si la seva clau de

xifrat és coneguda per tothom o guardada en secret, de clau pública o de clau privada, respectivament.

2.4.1. Criptografia de clau privada

La criptografia de clau privada, també anomenada criptografia simètrica, és el mètode criptogràfic que empren els criptosistemes on l'Alice xifra un missatge fent servir una clau k i en Bob desxifra el missatge fent servir aquesta mateixa clau. En aquests sistemes, conèixer k permet desencriptar un missatge xifrat amb gran facilitat. Per exemple, si per a enviar un missatge a en Bob, l'Alice realitza un desplaçament de 5 lletres a l'alfabet català, si l'Eve coneix que k=5, podrà fàcilment deduir que per a desxifrar un missatge xifrat d'aquesta manera, haurà de desplaçar tots els caràcters del missatge -5 posicions en l'alfabet.

Per a entendre millor com funciona, es pot fer servir la següent analogia: l'Alice escriu un missatge i el guarda dins una capsa. Llavors, tanca la capsa amb un cadenat del qual només ella té la clau. Una vegada ha tancat la capsa amb el cadenat, queda amb en Bob i li proporciona una còpia de la clau i la capsa original. Per a obrir-la, en Bob només ha de fer servir la clau que li ha donat l'Alice. No obstant, si l'Eve aconsegueix robar-li la clau a en Bob o a l'Alice, podrà llegir el missatge secret. Per això es diu criptografia de clau *privada*, perquè és crucial que només els usuaris autoritzats coneguin la clau de xifrat per al bon funcionament del criptosistema.

Avui dia, aquest tipus de criptografia no es fa servir gaire, ja que es fonamenta en dos principis totalment impràctics i ineficients. En primer lloc, l'Alice i en Bob han de quedar prèviament per a compartir i acordar la clau a través d'un canal no segur (per exemple, quedar en persona, o a través de llibres de claus, com van fer els alemanys a la segona guerra mundial), cosa que disminueix la confidencialitat. A més a més, la seva seguretat depèn del fet que només emissor i receptor coneguin la clau privada, fet que complica la gestió quan un mateix emissor te diferents receptors, com per exemple, en el cas dels bancs.

Exemples de criptosistemes simètrics són la màquina Enigma alemanya, el xifrat Cèsar o el xifratge de Vigenère, dels quals es parlarà en apartats posteriors.

2.4.2. Criptografia de clau pública

La criptografia de clau pública, que també s'anomena criptografia asimètrica, és el mètode de creació de claus amb el que funcionen els criptosistemes on en Bob, el receptor, crea dues claus: una de pública (de xifrat) i una de privada (de desxifrat). La clau pública es fa pública i és la clau que els emissors hauran de fer servir per a xifrar els seus missatges i enviar-los a en

Bob. La clau privada (que el receptor no diu a ningú) servirà per a que el receptor pugui desxifrar els missatges que ha rebut xifrats amb la clau pública.

En contraposició als criptosistemes de clau privada, en aquest tipus de criptografia, conèixer la clau de xifrat (clau pública) no permet desxifrar el missatge, sinó que s'ha de conèixer la clau privada (que ningú exceptuant en Bob no coneix). Qualsevol persona que vulgui enviar un missatge secret a en Bob, l'haurà de xifrar prèviament fent servir la clau pública i, encara que aquest missatge xifrat sigui interceptat per l'Eve, si aquesta no coneix la clau privada, no podrà desxifrar-lo. El receptor rebrà el missatge xifrat i el desencriptarà fent servir la clau privada que només ell coneix.

Aquest tipus de criptosistemes són molt més segurs contra atacs que els simètrics, ja que no cal proporcionar la clau privada a ningú. La seva seguretat depèn directament de la dificultat d'obtenir la clau privada i de la dificultat que presenten a l'hora de ser atacats fent servir una *brute-force solution*[2].

Per a entendre millor com funciona, es pot fer servir el següent exemple: en Bob vol rebre missatges de l'Alice. Llavors, ell crea una capsa amb contrasenya que tothom pot tancar, però només s'obrirà de nou quan s'hi introdueixi la contrasenya correcta d'apertura (que només coneix ell). Llavors, li dona la capsa sense bloquejar a l'Alice, qui escriu un missatge i l'introdueix dins la capsa, que bloqueja. Ara, només la pot desbloquejar qui conegui la contrasenya (teòricament, només en Bob la sap). L'Alice retorna la capsa a en Bob, qui l'obre fent servir la contrasenya que només ell sap. D'aquesta manera, encara que l'Eve robi la capsa abans que aquesta arribi al receptor, ella no podrà obrir-la, ja que no coneix la contrasenya. La seguretat d'aquest criptosistema depèn directament de la dificultat d'esbrinar la contrasenya. Si aquesta és una paraula molt llarga amb caràcters especials i sense sentit, com ara "sdùAhë?sfAJSdkf!uyg88!!5¿i", a l'Eve li serà pràcticament impossible endevinar-la.

Exemples de criptosistemes que siguin asimètrics són l'RSA, l'algorisme elGamal, el DSA o el criptosistema de Merkle-Hellman.

2.5. Principis de Kerckhoffs

Els principis o lleis de Kerckhoffs són uns postulats o normes que qualsevol criptosistema hauria de complir per a ser considerat segur i fiable. Van ser enunciats per Auguste Kerckhoffs a finals del segle XIX, i encara avui dia són una referència a tenir en compte a l'hora de crear un criptosistema. Els seus sis principis, modernitzats, són els següents:

1. Si el sistema no és matemàticament "irrompible", ho haurà de ser a la pràctica.

2: una *brute-force solution* consisteix a provar totes les possibles claus d'un sistema criptogràfic fins a, per sort, trobar la correcta.

2. **L'efectivitat del sistema no ha de dependre del fet que el seu disseny sigui secret. No hauria de significar cap problema que el disseny caigués en les mans de l'enemic.**
3. La clau ha de ser fàcil de memoritzar i no hauria de requerir l'ús de notes escrites.
4. Els criptosistemes han de donar resultats alfanumèrics.
5. Hauria de ser fàcil de transportar i operable per una única persona.
6. El sistema ha de ser fàcil de fer servir: no ha de causar estrès als operaris ni fer-los memoritzar una gran llista de normes.

Actualment, es pot prescindir d'alguna d'aquestes lleis, com per exemple la tercera, ja que, gràcies als ordinadors, es poden emmagatzemar claus molt llargues sense cap tipus de problema.

El primer principi és de sentit comú: hauria de ser pràcticament impossible per a l'enemic poder trencar un criptosistema fent servir una *brute-force solution*.

La llei més important de totes és la segona, que rep el nom de llei de Kerckhoffs. Explicada, aquesta norma vol dir que la seguretat del criptosistema no hauria de ser compromesa encara que el disseny del sistema, els seus algorismes i tots els components menys la clau, siguin públics. En altres paraules, a l'hora de dissenyar un criptosistema, la primera suposició que s'ha de fer és que tothom en coneixerà el seu disseny i el seu funcionament. Aquest principi és un dels més importants de la criptografia moderna, i tots els sistemes moderns el segueixen. Se li ha d'afegir que, a la pràctica, el mètode de xifrat hauria de proporcionar la menor informació possible sobre el text original**.** Té a veure amb que s'hauria d'evitar fer un criptosistema que es pugui trencar fent ús d'anàlisi de freqüències o dels missatges.

En definitiva, qualsevol criptosistema hauria de ser capaç de funcionar correctament i ser útil encara que el seu funcionament sigui conegut per tothom.

La resta de regles fan referència a temes d'eficiència i utilitat, i són les més flexibles de totes.

2.6. Atacs a criptosistemes

Un atac és quan un usuari no desitjat té accés a la informació o a la transmissió d'informació d'un criptosistema. Hi ha dos tipus generals d'atacs que un sistema pot experimentar: els atacs actius i els atacs passius.

Els atacs actius tenen a veure amb la interrupció, modificació i fins i tot la falsificació dels missatges. Per exemple, interrompre la comunicació entre A i B o modificar els seus missatges a través d'interferències. Aquets tipus d'atacs posen en perill la fiabilitat i l'autenticitat dels missatges obtinguts.

Els atacs passius, per la seva banda, consisteixen a robar o a accedir de manera no autoritzada a la informació. Exemples d'aquest tipus d'atacs són l'ús de la força, suborns o trencar criptosistemes per a obtenir directament informació dels missatges. Aquests tipus d'atacs posen en perill la confidencialitat del sistema i dels seus missatges.

El criptoanalista és aquella persona que, fent ús de tota la informació que té a l'abast, té com a tasca desxifrar missatges secrets i, en última instància, descobrir com funciona un criptosistema per a obtenir un mètode sistemàtic de desxifrat (funció de desxifrat) per a aquest sistema.

Sempre és essencial que els criptoanalistes enemics tinguin el menor coneixement possible sobre el funcionament del criptosistema però, encara que ho coneguessin tot menys la clau, els hauria de ser pràcticament impossible de trencar (segona llei de Kerckhoffs). No obstant, això no vol dir que s'hagi de fer públic tot el relatiu a un sistema ja que, mentre menys informació es tingui sobre aquest, més difícil serà trencar-lo.

Relatives al nivell de coneixement que un criptoanalista enemic pot tenir sobre un sistema, es distingeixen les següents situacions:

1. Només té accés a un criptograma. En aquesta situació, el criptoanalista enemic només té accés a un missatge xifrat, és a dir, ha interceptat un missatge ja xifrat i desconeix l'estat original d'aquest i el mètode a través del qual s'ha xifrat. Aquesta situació és la més favorable i segura per a la confidencialitat del sistema.
2. Té accés a un criptograma i al seu text original. En aquesta situació, el criptoanalista té accés a un missatge xifrat, com a l'exemple anterior, però també sap quin era el missatge original. Encara desconeix com funciona el mètode de xifrat ni tampoc té accés a ell.
3. Té accés a qualsevol text xifrat i a l'original però desconeix l'algorisme. En aquesta situació, el criptoanalista té accés al mètode de xifrat però desconeix com funciona. Per exemple, imaginem que un intrús ha estat capaç de robar una màquina de xifrat i se l'entrega al criptoanalista, però aquest és incapaç d'entendre com funciona. Por fer-la servir i xifrar tot el text que vulgui, però desconeix el funcionament dels seus mecanismes interns.
4. Té accés a totes les característiques del sistema: disseny, components, algorismes: tot. Aquesta situació és igual a l'anterior amb l'única diferència que el criptoanalista sí que comprèn el funcionament del sistema, és a dir, té accés al mètode de xifrat i entén com funcionen els seus mecanismes interns. Aquesta situació és la pitjor que es pot

donar en termes de confidencialitat i la situació més favorable per a un criptoanalista enemic.

No obstant, si un criptosistema compleix les dos primeres regles de Kerckhoffs, cap d'aquestes situacions haurien de significar un perill per a la confidencialitat de la seva informació.

Hi ha diferents tipus d'atacs passius que es poden fer a un criptosistema. En aquest apartat, es tractaran alguns dels que assumeixen que el criptoanalista té accés al mètode de xifrat (situacions 3 i 4):

2.6.1. Brute-force solution

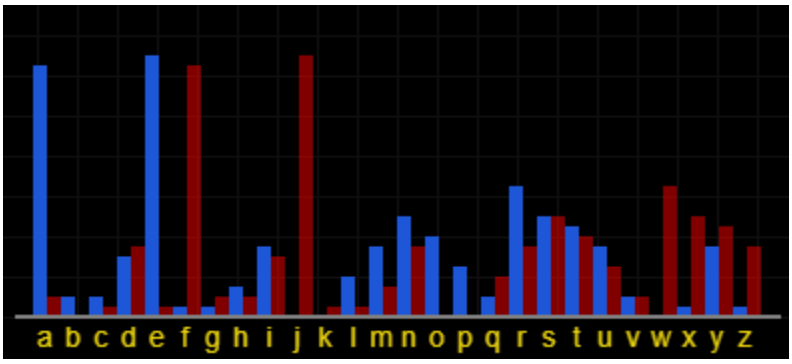
Una vegada s'ha obtingut un missatge xifrat (i es coneix mínimament com funciona el sistema de xifrat), una *brute-force solution* consisteix a provar totes les combinacions diferents que un criptosistema pot tenir fins a trobar la correcta. Per exemple, si el mètode de xifrat és el de desplaçament, aquest tipus d'atac consistiria a anar provant totes les diferents claus a un missatge xifrat fins a obtenir un text que tingui sentit. A mesura que augmenta el grau de complexitat que un criptosistema té, aquest tipus d'atac es fa més impossible o ineficient. Per exemple, a un ordinador modern li portaria més de 300 anys desxifrar un missatge que ha estat xifrat fent servir el mètode RSA. Si es compleix la primera regla de Kerckhoffs, aquest tipus d'atac hauria de ser inútil.

2.6.2. Anàlisi de freqüències

Aquest tipus d'atac, més elaborat que l'anterior, consisteix a agafar un text xifrat i comptar el nombre de lletres i les vegades que apareixen. Ara, cal comparar el nombre de lletres més emprades amb el nombre de lletres que es repeteixen més a cada idioma. Per exemple, a l'espanyol i al català serien segurament les vocals *a* i *e* les lletres més repetides, i les que menys, segurament, les consonants *k*, *x*, o *z*. Amb aquesta informació, es pot establir una relació entre les lletres del missatge original i del xifrat, i es pot esbrinar quina lletra equival a quina altra. Un criptosistema sofisticat evitarà aquest tipus de repeticions i tindrà una distribució alfabètica molt més uniforme. Com més uniforme i igual sigui la repetició de caràcters, més segur serà un criptosistema. Com ja s'ha dit abans: "a la pràctica, el mètode de xifrat hauria de proporcionar la menor informació possible sobre el text original".

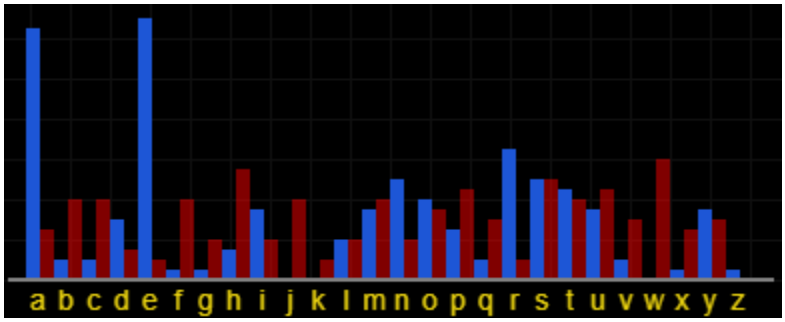
Per exemple, les següents imatges mostren la freqüència amb què apareix cada lletra a un text quotidià en espanyol fent servir el color blau (les lletres més repetides són, en efecte, les vocals *a* i *e*). El color vermell indica la freqüència amb què es repeteixen les lletres al missatge xifrat. A la primera imatge, es veu clarament que les lletres han estat desplaçades cinc

posicions (a=f, e=j, etc.). Fent servir un simple anàlisi de freqüències, un criptoanalista podria obtenir la clau de xifrat del sistema de la següent imatge.



Il·lustració 2.3., Anàlisi de freqüència d'un text quotidià en espanyol encriptat fent servir el xifrat Cèsar

No obstant, a la segona imatge, el mateix text s'ha xifrat fent servir un mètode més sofisticat i complex, un tipus de xifrat polialfabètic. Aquest tipus de xifrat es pot dir que és millor que el primer ja que "camufla" molt millor les freqüències de l'espanyol. Tot i així, la distribució de freqüències del text xifrat no és igual per a totes les lletres, i encara revela un patró que, amb un estudi més dur i llarg, un criptoanalista podria trencar.



Il·lustració 2.4., Anàlisi de freqüència del mateix text encriptat fent servir el xifrat Vigenère

En definitiva, per a fer difícil o evitar que un sistema pugui ser trencat fent servir l'anàlisi de freqüències, cal que aquest sistema produeixi una distribució el més igual possible a l'hora de xifrar text.

Cal destacar que com l'objectiu d'aquest tipus d'atac és obtenir la clau de xifrat, només és útil a l'hora d'atacar sistemes criptogràfics de clau privada. És totalment inútil contra sistemes que fan servir claus públiques, ja que s'obtindria aquesta, cosa que no serviria per a res.

2.6.3. "Enginyeria inversa"

Aquest tipus d'atac requereix més quantitat d'informació: no només requereix un missatge xifrat, sinó també conèixer com funciona el mètode d'encriptació (màxim nivell d'informació

que un criptoanalista pot tenir). Encara que el mètode de xifrat no hauria de proporcionar cap tipus d'informació sobre la forma del missatge original, a la pràctica, sempre hi haurà imperfeccions que permetran a un criptoanalista obtenir informació sobre el missatge original. Aquestes imperfeccions són dissenys incorrectes, algorismes imperfectes o simplement mètodes de xifrat que no s'han acabat de perfeccionar. El criptoanalista, coneixent el disseny del sistema que vol trencar, pot aprofitar-se de totes aquestes imperfeccions i errors de disseny per a intentar desxifrar missatges. Per exemple, va ser un error al seu disseny el que va facilitar molt que els Aliats poguessin trencar Enigma.

2.7. Concepte de secret perfecte i la llibreta d'un sol ús

Podem afirmar que un criptosistema té secret perfecte si el codi que produeix no proporciona cap tipus d'informació sobre el missatge original.

Per que això passi, han d'existir a K tantes claus de xifrat com missatges de n-lletres, de manera que a cada possible missatge de n-lletres li correspongui una única clau. Per tant, la clau de xifrat hauria de tenir la mateixa llargada que el missatge, i només es podria fer servir una vegada, ja que, els missatges que enviem no acostumen a tenir tots la mateixa llargada. És més, la majoria de missatges que enviem tenen diferents llargades, cosa que requeriria una clau diferent per a cada missatge, si volem complir l'anterior condició.

Això fa que l'únic mètode d'encriptació amb secret perfecte conegut fins ara sigui totalment impràctic, ja que produeix claus tan llargues com el missatge original i només pot ser utilitzada una vegada. Aquest fet fa que rebi el nom de *one-time pad* en anglès, que es tradueix al català com "llibreta d'un sol ús". Funciona de la següent manera:

Imaginem que l'Alice vol enviar un missatge de 15 lletres a en Bob i que el vol dotar de secret perfecte. Fent servir la *one-time pad*, ella agafa un dau de 26 cares i, per a cadascuna de les 15 lletres del seu missatge, tira el dau una vegada, i escriu els 15 resultats aleatoris (per exemple, aquesta seqüència podria ser: {1, 26, 4, 8, 6, 4, 12, 15, 24, 13, 17, 19, 23, 11, 21}). Ara, al valor numèric de cada lletra del seu missatge li afegeix el valor de la tirada que li correspongui. A la primera lletra, li afegeix el valor de la primera tirada, i així fins a la quinzena lletra. La clau de xifrat és, doncs, la seqüència aleatòria de 15 nombres. L'Alice ha de compartir aquesta clau amb en Bob. Una vegada fet això, suposant que l'Eve no coneix la clau privada, és matemàticament impossible que pugui esbrinar el missatge original. Encara que disposés de poder computacional il·limitat, encara que tingués un ordinador capaç de realitzar infinites operacions aritmètiques i lògiques per segon, li seria impossible trencar el codi i el millor que

podria fer seria intentar endevinar-ho per pura sort. Això es deu a que el xifrat de la *one-time pad* no proporciona cap tipus d'informació sobre el missatge original i, per tant, totes les seves claus de l'espai K son equiprobables[3] per a cada missatge de n-lletres.

La llibreta d'un sol ús és, però, un mètode molt impràctic i no recomanable, ja que per a missatges grans, les claus són molt grans i, el més important de tot i la raó per la qual la criptografia de clau privada no es fa servir avui dia: l'Alice i en Bob han de compartir entre ells les claus a través d'un canal no segur (quedar en persona, per exemple).

3: dos possibles cassos són equiprobables quan ambdós tenen les mateixes probabilitats d'esdevenir. Per exemple, les tirades d'un dau o tirar una moneda enlaire i que aquesta caigui sobre la cara o sobre la creu.

3. Història de la criptografia

3.1. Introducció

En aquest apartat es realitzarà un estudi històric dels diferents tipus de mètodes criptogràfics en ordre cronològic. Com aquests han estat molts, se seleccionaran i només es cobriran els més importants. Concretament, en aquest treball se li ha volgut donar molta importància a dos mètodes, que són els que es cobriran amb més extensió i rigor. Aquests dos mètodes són la màquina Enigma (de clau privada) i el mètode RSA (de clau pública), els quals considero crucials a la història de la criptografia, i d'aquí que siguin els que més extensament s'explicaran.

La majoria dels mètodes més complexos seran estudiats de manera sistemàtica fent servir l'esquema següent:

1. Història i importància. Breu resum del context històric quan va aparèixer i explicació del seu origen.
2. Fonaments matemàtics i teòrics del mètode.
3. Algorismes. Com funcionen?
   3.1. Xifrat.
   3.2. Desxifrat.
   3.3. Creació de les claus
   3.4. Exemple de xifrat i de desxifrat.
4. Com pot ser trencat? Com ha estat trencat històricament?
5. Avantatges i inconvenients.

Alguns mètodes, sobre tot els primers i més primitius, potser no tindran gaires fonaments matemàtics o importància històrica a destacar (degut a la seva senzillesa de disseny), i alguns d'aquests punts s'ometran (com s'ha fet, per exemple, a l'apartat 3.2.). A mesura que la complexitat dels mètodes augmenti amb el temps, aquests cinc aspectes es cobriran amb més rigor.

Naturalment, és un període històric molt extens (més de dos mil·lennis). Per tant, com ja s'ha dit abans, només es tractaran de cada període històric els mètodes que es considerin més importants per a la història de la criptografia.

3.2. Skytálē espartana

Il·lustració 3.1., Imatge d'una skytálē

Aquest mètode és un dels mètodes de transposició[4] més antics que es coneixen. Es creu que va sorgir a voltants del segle V a.C i que va ser emprat primerament pels espartans i també, més tard, pels romans.

Es fonamenta en escriure un missatge a una graella de manera que, quan s'enrotlli al voltant d'una vara de diàmetre adequat, mostri un missatge ocult (Il. 3.1).

És un mètode (com tots els clàssics) de clau privada, i aquesta és el diàmetre de la vara.

Funciona així: dues persones que es vulguin comunicar en secret pacten quin serà el diàmetre de les vares que faran servir. Una vegada decidit, creen dues vares idèntiques, i cadascun es queda una. Llavors, l'emissor enrotlla una tira de paper a voltant del seu cilindre i escriu el missatge que vulgui. El desenrotlla i l'entrega al receptor. Ara, aquest agafa la seva còpia de la vara i el torna a enrotllar i podrà llegir el missatge original.

És un mètode molt senzill de trencar, ja que no cal saber exactament la clau, sinó que només cal anar provant amb cilindres de diferent gruix fins a obtenir un text que tingui sentit.

Com a avantatges, podem destacar la rapidesa amb què es pot xifrar i desxifrar aquest codi, i com a inconvenient, l'evident facilitat amb què es pot trencar.

3.3. El xifratge de Cèsar

El xifrat Cèsar rep el seu nom del dictador romà Juli Cèsar, qui el va inventar per a encriptar missatges de caràcter militar. Es creu que va dir el següent:

"Si tenia quelcom confidencial a dir, ho escrivia xifrat, és a dir, a base de canviar l'ordre de les lletres de l'alfabet, de forma que no hi apareixia ni un paraula. Si algú vol desxifrar-ho, i obtenir el seu significat, ha de substituir la quarta lletra de l'alfabet, la D, per la A, i així amb les altres".

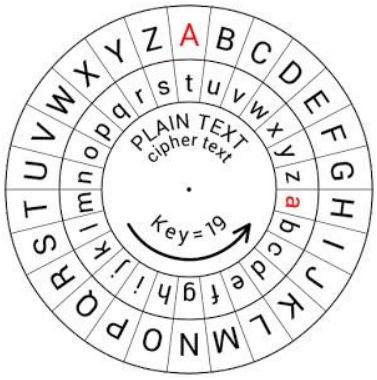
És un mètode de xifrat per substitució[5]. Consisteix a assignar valors numèrics a les lletres i escollir un nombre n (que és la clau privada) i, a través d'aritmètica modular[6], afegir aquest nombre n a totes les lletres d'un missatge que es vulgui xifrar. Per a desxifrar, només cal restar aquest nombre n al text xifrat i fer el seu mòdul, que serà el nombre de lletres de l'alfabet:

4: el mètode de la transposició (o permutació) consisteix a canviar de posició les lletres del text pla amb l'objectiu d'ocultar la seva informació. Exemple: "hola" → "lhoa".
5: la substitució consisteix a alterar el valor numèric de totes les lletres d'un alfabet seguint un mètode constant. Exemple: si li afegim +2 posicions a cada lletra de l'alfabet: "hola" → "jqnc".
6: consultar a l'annex l'apartat sobre aritmètica modular (pàg. 57).

$$C = f(M,k) = M + k\ (mod\ 26^*)$$

$$M = f^{-1}(C,k) = C - k\ (mod\ 26)$$

on M és un missatge de text pla qualsevol i C el resultat de xifrar M amb la clau k.



Il·lustració 3.2., Roda de Cèsar, mètode mecànic molt ràpid i senzill per a xifrar text fent servir el xifratge

És un mètode també bastant senzill de trencar i insegur, ja que només cal provar les 26 possibles claus que es poden haver fet servir fins a desxifrar el missatge (*brute-force solution*), cosa que un ordinador pot fer en una fracció de segon i una persona amb paciència podria fer en menys de cinc minuts. Es podria també desxifrar fent servir l'anàlisi de freqüències (explicat anteriorment).

Com la gran majoria de mètodes clàssics, no té gaire avantatges (només la seva facilitat de portar a terme). Els inconvenients són molts, i el principal és la seva facilitat a ser desxifrat.

No obstant la seva facilitat de desxifrat, aquest mètode ha estat considerat irrompible durant segles i, fins i tot, va ser usat per l'armada russa durant la Primera Guerra Mundial.

3.4. El xifratge de Vigenère

Durant molts segles, es va considerar el xifratge Cèsar com irrompible, però a finals de l'Edat Medieval, al principi del Renaixement, els criptoanalistes de l'època van adonar-se que aquest mètode no era gens segur i, el 1533, el criptologista Giovan Battista Bellaso va inventar un nou xifratge, que va rebre el nom de Vigenère perquè la seva invenció es va atribuir injustament a Blaise de Vigenère al segle XIX. Aquest mètode de xifrat també ha estat considerat irrompible (li deien "le chiffre indéchiffrable", "el xifratge indesxifrable"), i fins i tot criptògrafs americanes el consideraven així a inicis del segle XX.

Es tracta d'un xifratge de clau privada per substitució polialfabètic[7] que consisteix en una successió de xifrats de Cèsar. La clau privada és una paraula formada per n lletres ($n_1n_2...n_{n-1}n_n$), de manera que a cada lletra del text pla se li sumarà el valor de la n_x que li correspongui (fent servir el mateix mètode que al xifrat de Cèsar). Per a desxifrar, només caldrà fer el procés a la inversa. Per exemple, si agafem com a clau privada la paraula SERP, la clau k=(19, 5, 18, 16). Ara, si volguéssim xifrar la frase BON DIA, es faria el següent: a B se li afegiria 19, a O se li afegiria 5, a N se li afegiria 18 i a D se li afegiria 16, i es tornaria a començar amb el valor de la primera lletra de la clau k, és a dir, a I se li afegiria 19 i, finalment, a A se li afegiria 5. Per a

*: es fa servir el número 26 perquè a l'alfabet anglès hi ha només 26 lletres.

7: un mètode de xifrat polialfabètic és un mètode que genera més d'un alfabet. Si el xifratge de Cèsar generava un nou alfabet al sumar-li un valor a totes les lletres, un mètode de xifrat polialfabètic en generarà diversos alfabets, depenent de la mida de la clau.

desxifrar el criptograma, només s'hauria de restar el valor de cada lletra en l'ordre corresponent.

És un mètode bastant més segur que el de Cèsar, i la seva seguretat depèn de la mida de la clau i del fet que l'enemic no sap aquesta mida. No obstant, però, mitjançant un anàlisi del text xifrat, aquest mètode també pot ser trencat. I això mateix va fer Friedrich Kasiski, un criptògraf prussià. Es va donar compte de que, sobre tot a missatges molt llargs, al text xifrat sempre es repetien paraules, el qual vol dir que és molt probables que fossin la mateixa paraula al missatge original. Per tant, la distància entre dues paraules iguals al criptograma ha de ser un múltiple de la mida de la clau. Ara, només cal buscar diferents paraules que es repeteixin i fer el màxim comú divisor entre les seves distàncies. La longitud de la clau serà doncs amb molta seguretat aquest nombre o un dels seus factors primers. Després, només cal dividir el criptograma en tants blocs de text com lletres tingui la clau. Cadascuna de les lletres d'aquest bloc haurà estat xifrada amb la mateixa posició de la clau. Per tant, cada bloc ha estat xifrat fent servir un xifratge de Cèsar. Si la clau té una longitud de 5 lletres, es dividirà el text xifrat de la manera següent: s'agafarà la primera lletra, la cinquena, la desena, etc., i totes aquestes lletres constituiran un bloc. S'agafarà, la segona, la sisena, l'onzena, etc., i així fins a tenir cinc blocs. Finalment, només caldrà aplicar l'anàlisi de freqüències tradicional a cada bloc de text per a obtenir totes les lletres que formen la clau privada.

Aquest xifrat, encara que més segur que els monoalfabètics, no era irrompible. No obstant això, fins els anys 20 va ser emprat per molts exèrcits i alguns criptògrafs americans afirmaven que no es podia trencar (encara que això ja havia estat falsejat per Kasiski).

3.5. La màquina Enigma

3.5.1. Història

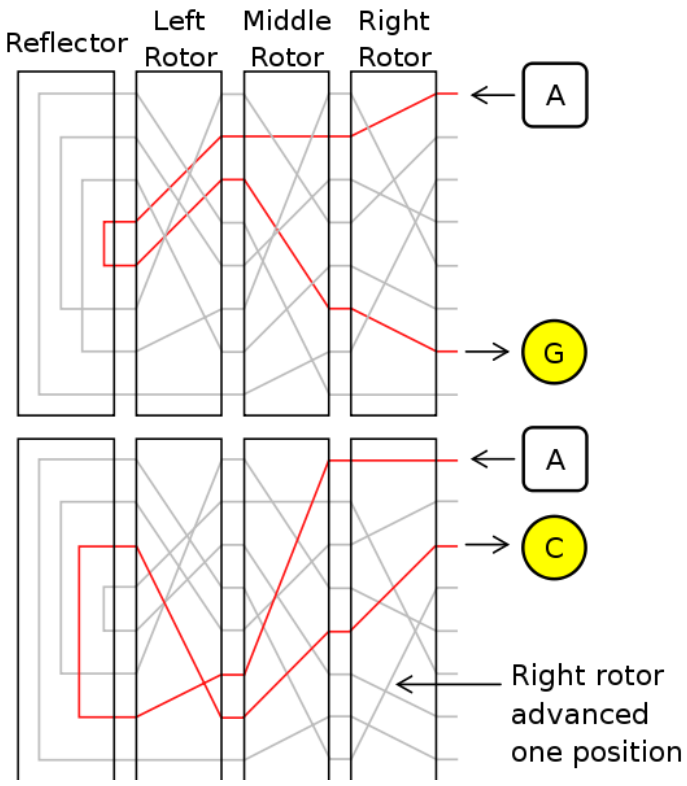
La màquina Enigma va ser inventada per l'enginyer alemany Arthur Scherbius al final de la Primera Guerra Mundial. Durant la dècada dels anys 20, el model i mode de funcionament es van popularitzar i comercialitzar per a ús privat. El fet que aquest tipus de sistemes es fessin tan populars cridà l'atenció dels militars alemanys, els quals van adoptar el model de funcionament d'Enigma i el van modificar, afegint-li molta seguretat extra. L'adopció d'aquesta màquina per part de l'exèrcit alemany va significar la interrupció total dels atacs que els Aliats feien sistemàticament a les comunicacions militars d'Alemanya que, fins aleshores, eren encriptades fent servir mètodes antiquats que els francesos interceptaven i desxifraven sense cap dificultat. Enigma va proporcionar gran seguretat a les comunicacions de l'exèrcit alemany i va jugar un paper molt important al seu favor durant la Segona Guerra Mundial.

3.5.2. Fonaments matemàtics i teòrics del mètode. Xifrat i desxifrat

Enigma és una màquina electromecànica d'encriptació de clau privada amb rotors. Això vol dir que fa servir elements tant elèctrics com mecànics, que encripta fent servir rotors alfanumèrics que, al teclejar una lletra, canvien la seva posició i que la clau tant de xifrat com de desxifrat és la posició inicial dels rotors i de la màquina.

La màquina consisteix en un teclat amb les 26 lletres de l'alfabet i una pantalla amb totes les lletres també. Tenia, inicialment, tres rotors amb 26 posicions possibles cadascun. El conjunt dels rotors, plens de cables i de connexions al seu interior, connectaven les lletres del teclat amb les de la pantalla, fent així que cada vegada que es piqués una lletra al teclat, aquesta enviés un puls elèctric a través dels cables dels rotors i il·luminés una lletra a la pantalla. També hi havia un reflector, element que connectava una sortida amb una altra per a fer que el desxifrat fos el procés invers al xifrat (si, per exemple, amb la mateixa clau es tecleja K i s'obté L, una altra persona amb la mateixa posició als seus rotors podrà teclejar L i obtenir K).

Per a fer-la servir, s'havia d'escollir una posició inicial per als rotors i la màquina. La lletra que s'il·lumina depèn totalment de la posició actual dels rotors. Cada vegada que es pitjava una tecla, el rotor avançava una posició, cosa que feia que es pogués teclejar la mateixa lletra moltes vegades seguides i que a la pantalla s'il·luminés una lletra diferent cada vegada.



Il·lustració 3.3., Esquema del funcionament d'una màquina de xifrat per rotors

Com es pot observar a la il·lustració, en teclejar la lletra A, s'enviaria un puls elèctric a través dels cables dels rotors i, a través d'aquests, s'il·luminaria G, és a dir, A es xifraria i passaria a ser G. Com que una lletra ha estat premuda, el rotor petit giraria una posició i, si es tornés a teclejar A, donat que un rotor està a una posició diferent, les connexions entre aquests serien diferents i, per tant, la mateixa lletra s'hauria xifrat de manera completament diferent.

Per a desxifrar, només s'havia de conèixer la posició inicial de la màquina amb la qual s'havia xifrat un missatge. És a dir, per a desxifrar, la màquina havia de tenir la mateixa configuració que la que tenia la màquina que s'havia fet servir per a xifrar. La part de la màquina que permetia això és el reflector.

La "força" de la màquina Enigma militar depenia del nombre total de possibles estats (configuracions inicials diferents) que aquesta podia tenir (el nombre dels quals pot variar depenent del model d'Enigma i de l'any). Per demostrar matemàticament això, es farà servir combinatòria:

Primer, existien cinc models de rotors de 26 posicions, cadascun d'ells amb connexions internes diferents degut a la seva fabricació, dels quals només es feien servir tres alhora. El màxim nombre de combinacions possibles que es pot fer amb tres rotors de 26 posicions és:

$$a = 26^3 = 17576$$

Com que hi havia cinc i aquests es podien col·locar intercanviablement, això augmenta el nombre possible de combinacions:

$$b = 5 \cdot 4 \cdot 3 = 60$$

Finalment, els models militars d'Enigma tenien una mesura de seguretat extra que els d'ús civil no tenien: un taulell de connexions que permetia connectar dos lletres de manera que aquestes canviaven entre si les seves posicions a l'hora de xifrar i de desxifrar. Per exemple, si es connecta la lletra A amb la G, al teclejar al teclat la tecla A, Enigma ho interpretarà com si s'hagués teclejat una G, i viceversa. Es podien fer deu parelles, és a dir, connectar 20 lletres entre si per parelles, de manera que al teclejar una lletra de la parella s'enviés a la màquina l'altra lletra de la parella i viceversa, tant a l'hora de xifrar com a l'hora de desxifrar. Aquest nombre c és el nombre de diferents maneres possibles de fer 10 parelles amb les 26 lletres de l'alfabet de manera que:

- Només aparegui una lletra una vegada.
- No importi l'ordre en què apareixen les parelles.
- No importi la forma de les parelles (AB=BA).

Aquest nombre de possibles combinacions es pot deduir de la manera següent:

$26!$ és el nombre total de possibles combinacions amb 26 lletres sense importar l'ordre, només tenint en compte que aquestes no es repeteixin. Com que només ens interessa com estan

connectades 20 d'aquestes, és igual com estiguin connectades les últimes sis lletres. Per tant, dividim per sis factorial: $\frac{26!}{6!}$.

Després, també ens és igual si primer s'ha connectat la lletra A amb la lletra C o si primer s'ha connectat la lletra D amb la lletra X. Per tant, dividim pel nombre de connexions l'ordre en el qual apareguin és irrellevant, deu factorial: $\frac{26!}{6!\cdot 10!}$.

Finalment, ara tenim el nombre de diferents parelles que es poden formar fent servir 20 lletres de manera que una lletra només aparegui una vegada i que l'odre en el qual les parelles apareguin sigui irrellevant. Però l'expressió $\frac{26!}{6!\cdot 10!}$ encara té en compte la forma que tenen les parelles. És a dir, diferencia AB de BA, cosa que la màquina Enigma no fa. Per tant, dividirem per dos tantes vegades com parelles hi ha: $\frac{26!}{6!\cdot 10!\cdot 2^{10}}$:

$$c = \frac{26!}{6! \cdot 10! \cdot 2^{10}}$$

Finalment, el nombre total de possibles estats de la màquina Enigma ve donada per l'expressió:

$a \cdot b \cdot c = 158\,962\,555\,217\,826\,360\,000$ possibles combinacions.

Això donava a Enigma un nivell de seguretat colossal, cosa que va portar els alemanys a pensar que havien trobat un xifrat irrompible.

La clau privada és, com ja s'ha dit abans, l'estat inicial d'Enigma (l'ordre dels rotors, la posició dels rotors i les connexions que s'han fet entre les lletres al taulell de connexions) amb què es va codificar un missatge. Per a desxifrar un missatge, només cal posar Enigma en aquest estat inicial i teclejar les lletres del text xifrat, i Enigma retornarà el missatge original.

3.5.3. Creació de claus

La clau privada d'Enigma era la seva configuració inicial. Com és un mètode criptogràfic simètric, tant el destinatari com l'emissor havien de conèixer prèviament la clau. Els alemanys van organitzar aquest sistema per mitjà de llibres de codis mensuals: llibres que es repartien a tots els operaris de màquines Enigma i que els indicava quina configuració inicial havien de fer servir cada dia. Aquests llibres estaven fets de paper molt dèbil i de tinta que s'esborrava amb el contacte amb l'aigua, de manera que poguessin ser fàcilment destruïts.

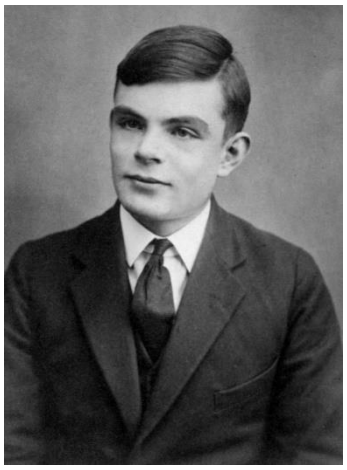
La Kriegsmarine (marina de guerra) feia servir el seu propi sistema, utilitzant els llibres de codis com a ajut. Cada operari configurava la seva màquina en la configuració inicial dictada pel

Si el lector desitja obtenir una justificació més ampliada del nombre total de possibles estats d'Enigma, la pot trobar a l'apartat sobre Enigma a l'annex (pàg. 59).

llibre i, quan volien escriure un missatge, teclejaven dues vegades la configuració inicial amb què escriurien aquest. Per exemple, si volien xifrar un missatge amb la configuració de rotors AAA, ells el teclejaven dues vegades (teclejaven AAAAAA), que es xifrava amb la configuració del llibre (per exemple, assumirem que es xifrava com JHGJLD). Ara, canviaven els rotors a la configuració que ells havien escollit i seguien escrivint. El receptor hauria d'obrir el seu llibre de codis, configurar Enigma amb la configuració adequada per a aquell dia, teclejar els sis primers caràcters (JHGJLD), anotar el resultat (que hauria de ser AAAAAA), i configurar la seva màquina amb aquesta configuració obtinguda (AAA).

3.5.4. Trencar Enigma

Durant la Segona Guerra Mundial, els Aliats van dedicar grans esforços a intentar trencar la màquina Enigma per a poder tenir accés a les comunicacions dels alemanys. Aquests esforços van culminar en la creació de la màquina Bombe que, de manera automàtica, ajudava a obtenir la clau diària d'Enigma. Amb aquesta màquina, els Aliats eren capaços d'interceptar missatges alemanys i desxifrar-los en qüestió de minuts i, en conseqüència, salvar moltes vides i escurçar la guerra.



Il·lustració 3.4., Alan Turing, creador de la màquina Bombe i pare de la computació moderna

La màquina Bombe, dissenyada el 1939 per Alan Turing, s'aprofitava d'un error al disseny d'Enigma. El disseny d'Enigma violava la segona (i més important) llei de Kerckhoffs: el seu disseny donava pistes sobre les lletres del missatge original. Concretament, l'algorisme de la màquina Enigma li impedia que, al teclejar una lletra, aquesta es xifrés en si mateixa. És a dir, si es teclejava la lletra A, es podria il·luminar la lletra B, C, X, J..., però mai la lletra A. Aquesta característica, creada a propòsit pels alemanys (ja que ells creien que era un nivell extra de seguretat), era en realitat una debilitat, debilitat de la qual els Aliats es van aprofitar per a trencar Enigma. Aquest error al disseny dona una pista molt gran: deixa saber on *no* hi pot anar una paraula a un missatge i, consegüentment, si es coneix part del missatge original, es poden eliminar totes les configuracions d'Enigma que *no* són possibles. Aquest procés, encara que infinitament lent de portar a terme a mà, va ser automatitzat i, en qüestió de vint minuts, donat un missatge xifrat amb Enigma del qual es coneixia (o es creia conèixer) una part sense xifrar, descartava totes les configuracions inicials que *no* eren possibles per a aquell missatge, deixant-ne només unes

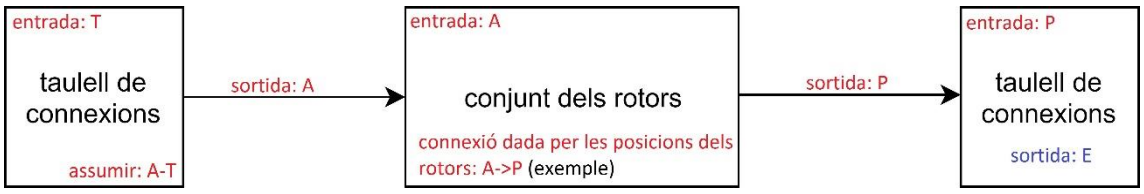
poques vàlides que es podien comprovar a mà per humans. Bombe va ser la màquina que, a través de l'ús de lògica cablada, va automatitzar aquest procés.

Funcionava de la següent manera: s'havia d'escollir un missatge xifrat del qual es coneguès part del que es deia a l'original. Per exemple, cada dia a les 06:00, els alemanys enviaven un informe del temps. Es pot assumir que dins d'aquest missatge sempre hi trobarem encriptada la paraula alemanya "wetterbericht" (informe del temps). Ara, aprofitant-nos d'aquest error de disseny, només cal col·locar la paraula *wetterbericht* a una posició vàlida del missatge xifrat. Per exemple:

text xifrat d'exemple	J	X	A	T	Q	B	G	G	Y	W	C	R	B	D	G	T
paraula d'exemple	W	E	T	T	E	R	B	E	R	I	C	H	T			
paraula d'exemple		W	E	T	T	E	R	B	E	R	I	C	H	T		
paraula d'exemple				W	E	T	T	E	R	B	E	R	I	C	H	T
paraula d'exemple			W	E	T	T	E	R	B	E	R	I	C	H	T	

Il·lustració 3.5., Exemple de missatge xifrat i les posicions on hi podria i no hi podria anar la paraula "wetterbericht"

Ara, assumint que la paraula original és al text xifrat en aquella posició, es dedueix que, amb la configuració inicial correcta, al teclejar ATQBGGYWCR obtindríem WETTERBERICHT com a resultat. El següent a fer és mirar a la nostra taula i agafar una lletra del text xifrat, la primera T, per exemple. La taula ens diu que T (criptograma) és E (missatge original). A partir d'ara, s'han de fer altres assumpcions cegues i anar provant. Assumim que, al taulell de connexions, T està connectada amb A. Ara, s'ha d'escollir una posició dels rotors qualssevol i, a partir de la premissa que A està connectada amb T, deduir-ne altres connexions. Per exemple:

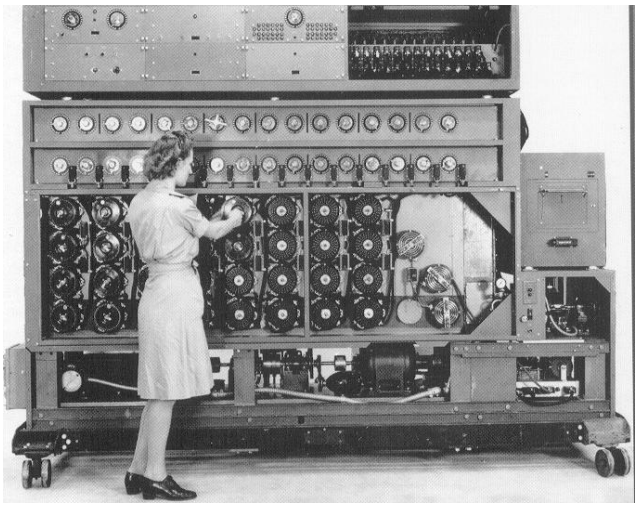


Il·lustració 3.6., Diagrama de flux d'informació de la màquina Enigma (explicat a l'apartat 3.5.2.). Exemple de deducció que P i E estan connectades seguint l'assumpció que A i T estan connectades al text de l'exemple anterior.

Gràcies a assumir que A i T estan connectades, es pot deduir que P i E també ho estan (si la posició dels rotors escollida connecta A amb P). Ara, cal fer el mateix per a totes les lletres de la il·lustració 3.5. Pel camí, ens trobarem contradiccions lògiques (per exemple, que A i T estiguin connectades *i* que T i J estiguin connectades). Si trobem una contradicció, això voldrà dir que la nostra assumpció inicial era incorrecta, és a dir, que A *no* estava connectada amb T.

Llavors, s'haurà d'assumir que T està connectada amb B i repetir el procés. Si trobem una altra contradicció, provarem amb la C, D, E... Si hem provat totes les lletres i cap era una combinació vàlida, haurem de moure el conjunt dels rotors una posició i tornar a provar amb que T està connectada amb A, B, C..., i tornar a girar el conjunt dels rotors... (fins que no es trobés cap contradicció). S'haurà de repetir aquest procés fins que no es trobi cap contradicció. Si no trobem cap contradicció, això voldrà dir que totes les connexions que hem deduït durant el procés i que la posició dels rotors són correctes. Finalment, un operari humà haurà de comprovar a mà que, per al missatge xifrat, aquesta configuració inicial deduïda produeix un missatge amb sentit. De ser així, el missatge ja estaria desxifrat. De no ser-ho, caldria provar una altra combinació la qual no hagués sigut descartada (que no contingui cap contradicció).

Resumidament, Bombe servia per a descartar possibles configuracions inicials (claus privades) d'Enigma i deixar-ne unes poques possibles (sense contradiccions) que un equip d'operaris humans podia comprovar a mà.



Il·lustració 3.7., Màquina Bombe britànica

Aquest és un procés lent (lentíssim) i irritant per a una persona, però mitjançant l'ús de lògica cablada, es torna un procés senzill i relativament ràpid que una màquina pot executar automàticament.

Es creu que Alan Turing i la seva màquina van contribuir enormement a escurçar la guerra i que, fins i tot, van salvar "milions de vides"[8].

8: a un article que va escriure per la BBC, el Professor Jack Copeland de la Universitat de Canterbury, afirma que Turing i la seva màquina van salvar entre 14 i 21 milions de vides a l'escurçar la guerra entre 2 i 3 anys (http://www.bbc.com/news/technology-18419691).

3.6. El mètode RSA

3.6.1. Història

Aquest mètode deu el seu nom als dos informàtics i al matemàtic que el van inventar: Ron **R**ivest, Adi **S**hamir i Leonard **A**dleman. Els tres van formar un equip amb l'objectiu de trobar una funció matemàtica de *one-way*[9]. Els dos informàtics, Rivest i Shamir, eren els encarregats de proposar possibles funcions que ells creien que podrien ser d'aquest tipus, i Adleman, el matemàtic, era l'encarregat de buscar-li debilitats i punts febles a les funcions que els informàtics proposaven. Van fer diversos intents, però no aconseguien trobar la seva funció.

Una nit després d'una festa, Rivest va arribar a casa seva molt tard, i no podia dormir. Es tombà al llit amb un llibre de matemàtiques al damunt, i va començar a pensar en la seva funció de *one-way*. Una brillant idea li va venir al cap: la multiplicació i la factorització de nombres primers! Trucà a Adleman i a aquest li va semblar una molt bona proposta. Es va passar tota la nit formalitzant i escrivint la seva idea. A primera hora del matí, ja tenia quasi bé la totalitat del seu article escrit. Aquesta funció és en la que es fonamenta el mètode RSA, i aquest rep el nom de l'ordre en el qual apareixien les inicials dels cognoms dels seus inventors a l'article.

Avui dia, aquest mètode és el que fan servir la majoria de grans empreses, com ara Twitter, Facebook o Google per a proporcionar seguretat als seus usuaris i encriptar la informació que aquests els confien. De vegades, algunes empreses fan servir mètodes simètrics per a xifrar la informació que volen mantenir en secret i xifren les claus d'aquests mètodes simètrics fent servir el mètode RSA, ja que aquest requereix molt poder computacional.

3.6.2. Fonaments matemàtics i teòrics del mètode

El mètode RSA és un mètode criptogràfic asimètric, és a dir, fa servir dos claus, una de pública (per a encriptar) i una de privada (per a desencriptar).

El mètode es basa en que, a matemàtiques, hi ha operacions que resulten molt senzilles en un sentit, però que són molt més complicades en el sentit oposat. A la vida real també existeixen aquest tipus de fenòmens: és molt fàcil fer cendres a través de foc i fusta, però és pràcticament impossible fer foc i fusta a través de cendres. Doncs bé, la idea en la qual ells van fonamentar el seu mètode és la següent: és molt senzill multiplicar dos nombres primers grans, però és infinitament més difícil obtenir aquests dos factors primers a partir del seu producte. Per exemple, 103801 i 104729 són dos nombres primers. És relativament senzill fer el seu producte, $103801 \cdot 104\,729 = 10\,870974929$, en menys de dos minuts qualsevol

9: vol dir "només d'anada". És a dir, una funció o operació que sigui molt fàcil de realitzar en un sentit però molt difícil de realitzar en l'altre.

podria fer aquesta multiplicació. No obstant, però, si només ens ensenyessin el resultat, 10870974929, quant de temps trigaria una persona a trobar els seus factors primers? Hores, dies, setmanes? I si el producte dels dos primers, en comptes de tenir 12 xifres, en tingués 1400? Un ordinador modern en trigaria més d'un segle a descomposar-lo. Els nombres primers serviran doncs per a generar les claus pública i privada.

Es basa, també, en la idea que hi ha infinits nombres primers[10] i que, per tant, es poden crear infinites claus diferents. Llavors, no hi ha cap perill que dos usuaris diferents tinguin el mateix joc de claus, ja que hi ha infinites per a escollir.

3.6.3. Algorismes. Creació de claus

3.6.3.1. L'algorisme de xifrat i el de desxifrat

La teoria del mètode diu que existeixen uns nombres enters positius e, d i n tals que, per a qualsevol missatge m (representat com un nombre natural), $m = (m^e)^d \ (mod\ n)$*. Això vol dir que m és igual al residu de la divisió entera de $(m^e)^d$ entre n.

- $c = m^e \ (mod\ n)$ és el missatge xifrat. Per tant, la clau pública és (e, n).
- $m = c^d \ (mod\ n) = (m^e)^d \ (mod\ n)$. Per tant, la clau privada és (d, n).

Per a xifrar, llavors, s'elevarà el missatge m a e i es farà el seu mòdul en n. És a dir, el missatge xifrat c és igual al residu que queda al dividir m^e entre n.

Per a desxifrar, s'elevarà el missatge xifrat c a d i es farà el seu mòdul en n. És a dir, el missatge desxifrat m és igual al residu que queda al dividir c^d entre n, o el que és el mateix, el residu que queda al dividir $(m^e)^d$ entre m.

3.6.3.2. Creació de claus[11]

El primer a fer és triar dos nombres primers, p i q, de manera que siguin diferents entre ells mateixos ($p \neq q$). Com a nivell extra de seguretat, haurien de tenir una mida similar i ser relativament grans. No obstant això, i per raons didàctiques, en aquest apartat es faran servir valors petits per a p i q a l'hora de posar exemples.

Després, es calcula $n = p \cdot q$. I n és part de la clau pública i de la clau privada.

Seguidament, es calcula $\varphi(n)$. Si n és un producte de dos nombres primers diferents, com en el nostre cas, $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$.

10: consultar a l'annex l'apartat sobre nombres primers per a trobar la justificació d'aquesta premissa (pàg. 58).
*: consultar a l'annex l'apartat sobre aritmètica modular (pàg. 57).
11: consultar a l'annex l'apartat sobre nombres primers per a trobar informació sobre la funció φ d'Euler (pàg. 58).

Llavors, s'ha d'escollir la clau pública e, que és qualsevol nombre que satisfaci aquestes dues condicions:

- $1 < e < \varphi(n)$
- $mcd(e, \varphi(n)) = 1$, és a dir, que e i $\varphi(n)$ siguin coprimers[12].

Finalment, per a escollir la clau privada d, s'ha de trobar un nombre que compleixi les següents condicions:

- $1 < d < \varphi(n)$
- $e \cdot d \equiv 1\ (mod\ \varphi(n))$, o el que és el mateix, que $(e \cdot d) - 1$ sigui enterament divisible per n (deixant residu 0).

De $e \cdot d \equiv 1\ (mod\ \varphi(n))$, se'n dedueix que $d = \frac{1+k\varphi(n)}{e}, k \in \mathbb{Z}$. D'aquesta manera es pot calcular d de manera ràpida donant diferents valors a k fins a obtenir un nombre enter.

Una vegada escollides les claus i tots els nombres que entren en joc a l'hora d'encriptar en el mètode RSA, només cal aplicar correctament els mètodes explicats a l'apartat 3.6.3.1. per a xifrar i desxifrar un missatge.

3.6.3.3. Exemple de funcionament

Primer, escollim dos valors per a p i q. Aquests valors poden ser 11 i 13, respectivament. De moment tenim que $p = 11$ i que $q = 13$.

Després, calculem $n = p \cdot q = 11 \cdot 13 = 143$.

Seguidament, calculem $\varphi(n) = (p-1) \cdot (q-1) = (11-1) \cdot (13-1) = 10 \cdot 12 = 120$.

Llavors, calcularem e. Com que e ha de ser coprimer amb $\varphi(n)$, el més senzill seria escollir un nombre primer per al valor d'aquesta variable i comprovar si és un dels factors primers de $\varphi(n)$. De no ser-ho, seria un valor correcte per a la variable. 17 és un valor correcte, ja que $mcd\big(17, \varphi(n)\big) = mcd(17, 120) = 1$.

Finalment, per a calcular d, podem fer servir l'expressió deduïda anteriorment:

$d = \frac{1+k\varphi(n)}{e}$, que per a $k = 16 \rightarrow d = \frac{1+16 \cdot \varphi(n)}{e} = \frac{1+16 \cdot 120}{17} = \frac{1921}{17} = 113$.

Ja tenim totes les variables necessàries.

12: dos nombres són coprimers entre si mateixos quan no tenen cap factor primer en comú (per exemple: 81 i 100).

Si el lector desitja entendre per què funciona aquest mètode, pot trobar la seva justificació matemàtica a l'apartat sobre el mètode RSA a l'annex (pàg. 60).

Els valors de les variables del sistema s'han resumit a una taula (Il. 3.8.).

p	11
q	13
n	143
$\varphi(n)$	120
e	17
d	113
Clau pública	(17, 143)
Clau privada	(113, 143)

Il·lustració 3.8., Valors de les variables de l'exemple

Els missatges han de representar valors alfanumèrics. Podem fer servir nombres per a representar lletres de manera que A=2, B=3, etc. (compte! A no pot ser igual a 1, ja que per a xifrar-la l'hem d'elevar a un nombre, i 1^k sempre és igual a 1 per a valors positius del nombre k, per tant, comencem a comptar des del 2). També es pot fer servir el codi ASCII[13]. En aquest exemple, es farà servir la primera manera esmentada. Si l'Alice volgués xifrar el missatge HOLA (9 16 13 2), aplicaria el mètode RSA a cada lletra del missatge, com es farà ara:

- $c_1 = m_1^e \ (mod\ n) = H^e \ (mod\ n) = 9^{17} \ (mod\ 143) = 81.$
- $c_2 = m_2^e \ (mod\ n) = O^e \ (mod\ n) = 16^{17} \ (mod\ 143) = 113.$
- $c_3 = m_3^e \ (mod\ n) = L^e \ (mod\ n) = 13^{17} \ (mod\ 143) = 117.$
- $c_4 = m_4^e \ (mod\ n) = A^e \ (mod\ n) = 2^{17} \ (mod\ 143) = 84.$

En aquest sistema, doncs, la paraula HOLA (8, 15, 12, 1) quedaria xifrada com 81 113 117 84.

L'Alice enviaria aquest missatge a en Bob i aquest hauria de fer el procés de desxifrat, fent servir la clau privada d que només ell coneix:

- $m_1 = c_1^d \ (mod\ n) = 81^{113} \ (mod\ 143) = 9.$
- $m_2 = c_2^d \ (mod\ n) = 113^{113} \ (mod\ 143) = 16.$
- $m_3 = c_3^d \ (mod\ n) = 117^{113} \ (mod\ 143) = 13.$
- $m_4 = c_4^d \ (mod\ n) = 84^{113} \ (mod\ 143) = 2.$

Finalment, en Bob veuria que el missatge original és 9 16 13 2 i, fent servir el sistema que ens diu que A=2, B=3..., en Bob obtindria que el missatge sense xifrar és HOLA.

3.6.4. Trencar el mètode RSA

Per a accedir a la informació xifrada pel mètode RSA, es necessita la clau privada d. Llavors, tota la seguretat del mètode depèn de la dificultat que presenta l'obtenció d'aquesta variable. d no es pot obtenir si es desconeix $\varphi(n)$, i aquesta última variable és extremadament difícil d'obtenir si es desconeixen els factors primers de n, és a dir, p i q. Llavors, el mètode RSA es pot trencar trobant els valors de p i q, i el fet que aquests dos siguin molt grans impossibilita que la seva obtenció sigui possible en un temps real o pràctic. La seguretat d'un sistema

13: consultar a l'annex l'apartat sobre el codi ASCII (pàg. 61).

Si el lector desitja comprovar aquestes operacions, pot fer-ho a pàgines web que realitzen càlculs d'exponenciació modular, com ara a Wolfram Alpha, pàgina web de matemàtiques que ofereix moltes eines de caràcter didàctic en forma de widgets (http://www.wolframalpha.com/widgets/view.jsp?id=570e7445d8bdb334c7128de82b81fc13).

d'encriptació que fa servir el mètode RSA és, doncs, directament proporcional a la mida d'aquests dos nombres.

No obstant, cal destacar que a mesura que la informàtica avança i els computadors són cada vegada més potents, es requeriran cada vegada valors més i més grans per als primers p i q (els quals són la base de la seguretat del mètode RSA).

3.6.5. Avantatges i inconvenients

El gran avantatge d'aquest mètode és que, fins ara, no s'ha trobat cap algorisme per a factoritzar un nombre molt gran en factors primers i, per tant, el mètode RSA roman un dels mètodes d'encriptació més útils, fiables i segurs.

Un clar inconvenient d'aquest mètode és l'enorme capacitat computacional que requereix. A l'exemple, hem emprat valors molt petits per a les variables, però els valors que es fan servir actualment (sobre tot a les grans empreses), per raons de seguretat, són molt grans: la mida d'aquests valors pot arribar a superar les 1400 xifres. Crear sistemes criptogràfics amb variables tan grans requereix molt poder computacional.

Un altre inconvenient és la possible futura invenció d'un ordinador quàntic (ordinador que faria servir principis de física quàntica per a realitzar operacions i que funcionaria de manera diferent a les màquines lògiques que fem servir avui dia) que, per la seva manera de funcionar, seria capaç de realitzar molts més càlculs per unitat de temps i, per tant, podria factoritzar el nombre n ràpidament i trobar així la clau privada d.

4. Apartat pràctic

En aquest apartat, s'aplicaran els conceptes teòrics explicats prèviament i es crearan dos programes per a l'encriptació de missatges, un emulant una màquina d'encriptació que fa servir cilindres rotatoris (com Enigma), i l'altre implementant l'algorisme del mètode RSA. Els objectius principals són crear dos programes per a encriptar i desencriptar missatges i redactar un model o plantilla per a la creació de qualsevol tipus de programa.

4.1. Disseny d'un programa informàtic. Fases

El propòsit d'aquest punt del treball és explicar, en termes generals, tot el procés de creació i desenvolupament d'un programa, tot destacant les més importants.

Aquestes fases seran les que farem servir de model per a documentar la creació dels dos programes proposats anteriorment.

4.1.1. Anàlisi de l'objectiu

En aquesta fase es descriu de la manera més precisa i rigorosa possible, en un llenguatge natural (com ara el català o l'espanyol), què farà el programa a dissenyar i com ho farà. En definitiva, és necessari concretar tres aspectes:

1. La funció del programa.
2. En què es fonamentarà el seu algorisme principal, és a dir, com portarà a terme aquesta funció.
3. L'*input* i l'*output*, és a dir, les dades d'entrada necessàries que rebrà el nostre programa per a funcionar i les dades de sortida que retornarà.

4.1.2. L'algorisme

Com ja vam comentar a l'apartat teòric, els algorismes són la part més important de tot programa. Podem definir un algorisme com un conjunt d'instruccions i/o d'operacions sistemàtiques i ordenades que, tenint en compte circumstàncies variables i diferents possibilitats, són aplicades per a trobar la solució a un tipus determinat de problema i/o realitzar una funció concreta.

Sembla un terme molt específic i abstracte, però tothom utilitza algorismes cada dia. Per exemple, un algorisme per a decidir si se surt de casa amb paraigües seria el següent: si plou/pedrega o si està previst que precipiti → agafar paraigües; a tota la resta de casos → no agafar paraigües.

Un algorisme és (ha de ser), per definició, concís i no pot donar lloc a cap tipus d'ambigüitat. Per tant, utilitzar un llenguatge natural no sembla una bona idea. És millor fer ús d'un *flowchart*, o diagrama de flux en català (alternativament conegut com a ordinograma). Per a crear un bon diagrama de flux, cal definir els diferents símbols que farem servir per a donar les nostres instruccions.

Aquets són els símbols normalment utilitzats en aquest tipus de diagrames, y són els que farem servir, amb el significat indicat en la descripció.

Símbol	**Nom**	**Descripció**
	Terminal	Serveix per a indicar l'inici i el final del diagrama de flux
	Input/output	Serveix per a senyalitzar que el programa llegirà una informació d'entrada (*input*) o retornarà una informació de sortida (*output*)
	Procés	Serveix per a indicar que es realitza una operació matemàtica bàsica
	Decisió	Serveix per a senyalitzar que el programa farà una operació lògica per a prendre una decisió on hi ha diferents possibles resultats
	Funció predefinida	Utilitzat per a simplificar el diagrama de flux. És una altra funció que realitza una tasca complexa que es simplifica per a estalviar espai i fer l'algorisme principal més simple visualment
	Línia de flux	Serveix per a indicar el flux lògic del programa, és a dir, què es fa després de cada acció

Il·lustració 4.1., Taula explicant el significat dels diferents símbols d'un ordinograma

En aquest apartat, també s'indicarà la complexitat asimptòtica[14] dels algorismes principals del programa fent servir la *big-O notation*.

14: consultar a l'annex l'apartat sobre complexitat asimptòtica (pàg. 61).

4.1.3. La programació

4.1.2.1. El llenguatge de programació

Programar és, en essència, donar ordres a un ordinador per a que les porti a terme en lloc del programador. Però aquestes ordres no poden contenir la més mínima ambigüitat i han de ser donades en una idioma que l'ordinador pugui entendre i, fins ara, els ordinadors no entenen el català. És a dir, no podem fer servir llenguatges naturals, plens d'ambigüitats i de dobles sentits, sinó que haurem d'emprar un llenguatge de programació, és a dir, un llenguatge formal, lògic i precís (amb una única possible interpretació) que l'ordinador és capaç d'entendre. Els llenguatges de programació es poden classificar segons el seu nivell:

- De baix nivell: llenguatge de la màquina, que l'ordinador interpreta directament.
- D'alt nivell: llenguatge formal creat per a simplificar la programació per als humans que, una vegada escrit, és compilat i "traduït" al llenguatge de la màquina, que aquesta interpreta.

L'avantatge dels llenguatges de baix nivell és que ens permeten "endinsar-nos" més a la màquina donant-li ordres lògiques que tenen un funcionament molt semblant al dels mateixos ordinadors. És a dir, permeten un major i més "natural" control de l'ordinador en un llenguatge que li és propi i no necessita ser traduït.

Però aquets llenguatges són molt difícils i tediosos degut a la gran complexitat que presenten. Per tant, quan es programa, la immensa majoria de vegades, els programadors utilitzen un llenguatge d'alt nivell que, encara que no els permeti treballar directament amb codi màquina (així s'anomena el llenguatge de l'ordinador al qual es tradueixen els programes escrits en un llenguatge d'alt nivell), els permet molt més fàcilment programar algorismes. Quan es fa servir un llenguatge d'alt nivell, el codi és convertit en un programa i es tradueix a codi màquina a través d'una eina anomenada compilador.

4.1.2.2. El codi

En aquesta part es mostraran imatges amb tots els algorismes importants del programa, senyalitzant quin és l'algorisme o funció principal i explicarem què fa cadascuna de les funcions llistades.

Una part molt important del codi és la documentació o comentaris. El codi ha d'estar acompanyat d'esclariments en forma de comentaris que, en la gran majoria de llenguatges de

programació, poden fer-se emprant // o /* */, tot explicant què fa cada part del codi i com ho fa, amb l'objectiu de:

- Fer el codi llegible per a persones que no siguin el seu programador original.
- Fer que el programador original sigui capaç d'entendre ràpidament el seu propi codi temps després d'haver-lo escrit.

```
1 #include <iostream>
2
3 int main(){//la funció "main" (principal) del nostre programa
4     std::cout << "hola";
5     /*aquesta ordre serveix per a que el programa "imprimeixi"
6     la paraula 'hola' per pantalla*/
7 }
```

Il·lustració 4.2., Exemple de comentaris de codi i les dues maneres de realitzar-los

4.1.4. *Debug* o depuració

Aquesta fase consisteix a provar diferents *inputs* per a comprovar que el programa funciona correctament, és a dir, provar el programa en qüestió. S'inclouran proves fotogràfiques dels resultats obtinguts i es descriurà amb detall el que mostra cada fotografia. També s'ha de destacar qualsevol *bug* o error que el programa pugui contenir i que no s'hagi pogut solucionar.

4.2. Rotex: xifrat simètric de missatges emulant cilindres rotatoris

4.2.1. Anàlisi de l'objectiu

L'objectiu d'aquest punt del treball és la creació d'un programa d'encriptació simètrica de missatges que emuli el funcionament d'una màquina de xifrat per cilindres rotatoris, com la màquina Enigma.

El primer a fer és descriure, de la manera més precisa possible, el seu funcionament.

El programa emularà el funcionament de tres rotors, anomenats rotor gran (RG), rotor mitjà (RM) i rotor petit (RP), amb 26 lletres (posicions) cadascun, amb un nombre de l'1 al 26 en cada posició col·locats tot seguint un ordre aleatori.

S'assignaran valors numèrics a les lletres de tal manera que a=A=1, b=B=2, etc.

Cada vegada que l'usuari premi una lletra de l'alfabet anglès (el programa no acceptarà caràcters especials com ñ, ç o ß), es realitzarà una sèrie d'operacions complexes amb el valor numèric de la lletra introduïda i els tres nombres de les posicions dels rotors.

Addicionalment, cada vegada que l'usuari premi una tecla, el rotor petit avançarà una posició. Quan el rotor petit hagi fet una volta sencera (26 posicions), el rotor mitjà en farà una, i quan aquest n'hagi fet 26, el rotor gran en farà una. Això dóna lloc a un total de 26^3 possibles combinacions diferents, és a dir 17 576 possibles combinacions diferents.

Això vol dir que un mateix missatge (com ara la paraula "hola" o la frase "avui és vint-i-set de juliol") pot ser encriptat de 17 576 possibles maneres diferents.

El funcionament serà el següent: a l'obrir el programa, a l'usuari se li preguntarà si desitja encriptar un missatge o desencriptar-ne un prèviament encriptat amb aquest mètode.

Si l'usuari decideix encriptar un missatge, el programa li demanarà que hi introdueixi quina posició vol per a cadascun dels tres rotors, és a dir, es podrà escollir, com ja hem dit abans, d'entre 17576 posicions diferents.

Una vegada fet això, l'usuari podrà escriure el missatge i el programa l'encriptarà.

Un altre usuari que desitgi desencriptar aquest missatge encriptat només haurà de seleccionar l'opció de desencriptar quan el programa se la ofereixi, introduir la configuració inicial amb la qual el missatge encriptat va ser encriptat, i el programa retornarà el text original. El programa inclourà en tot el procés, per suposat, missatges cap a l'usuari informant-lo en tot moment de les decisions que ha pres (per exemple: "ha escollit l'opció de encriptat un missatge" o "per a tancar el programa, premi Ctrl+C", etc.).

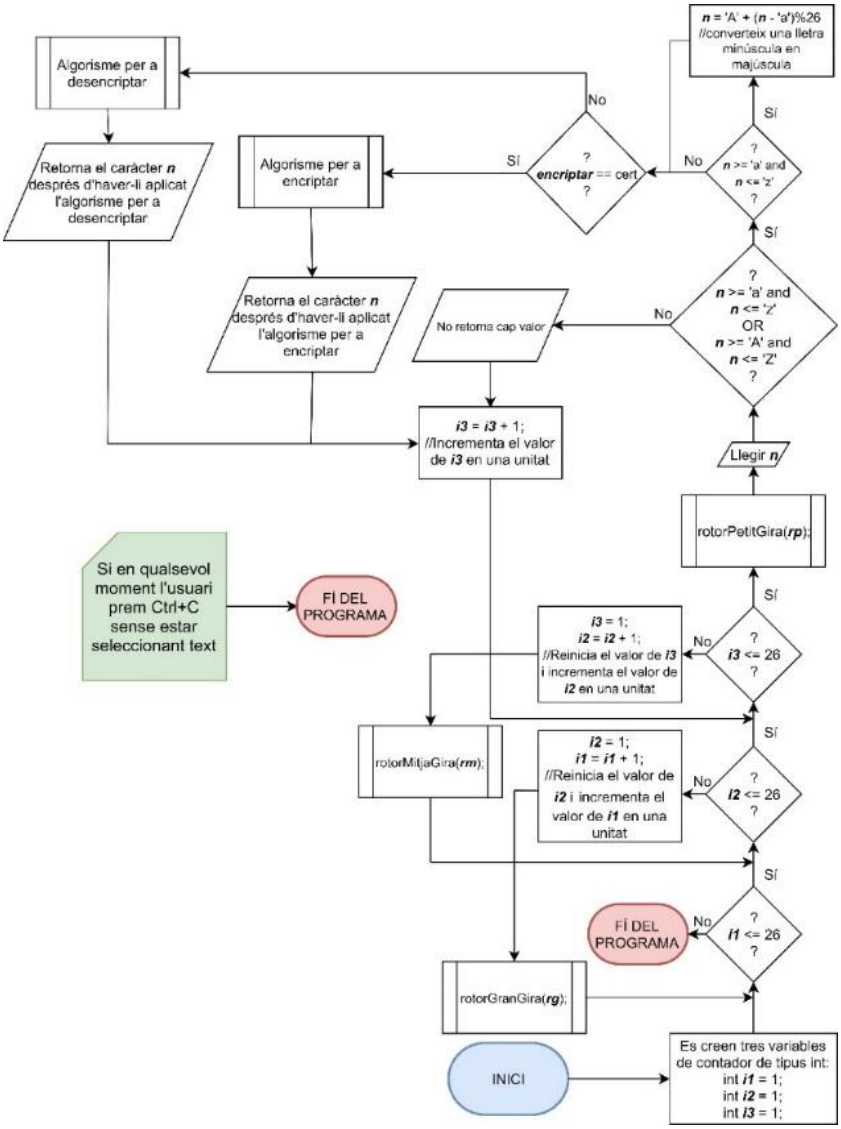
4.2.2. Els algorismes i la programació

Com ja ha estat decidit anteriorment, el funcionament principal d'aquest algorisme es basa en assignar valors numèrics a les lletres de manera que a=A=1, b=B=2, etc. Els rotors tenen 26 posicions cadascun, amb nombres de l'1 al 26 col·locats de manera totalment aleatòria i, cada vegada que l'usuari premi una tecla per a encriptar, es faran una sèrie d'operacions amb els valors numèrics de les posicions del rotor per a encriptar el missatge; per a desencriptar-lo, només s'hauran de fer les mateixes operacions però a la inversa (si, per exemple, per a encriptar es sumava, per a desencriptar, es restarà).

L'algorisme principal funciona de la següent manera: tres variables de nombres enters, i1, i2 i i3 són creades i assignades un valor inicial d'1. De moment, i1=i2=i3=1. Aquestes variables serviran de comptadors per a simular les voltes que fa un rotor. Cada vegada que el rotor petit faci una volta, el valor de i3 incrementarà en una unitat. Quan i3 arribi a 26 i el rotor petit torni a girar, el seu valor tornarà a ser 1, i el valor de i2 serà incrementat una unitat. Passarà el

mateix amb i1 quan i2 passi a ser 26 i aquest torni a girar. Cada vegada que es prem una tecla per a encriptar, es comprovarà que sigui una lletra de l'alfabet (un caràcter vàlid) i, de ser-ho, es realitzarà una sèrie d'operacions complexes que, a l'ordinograma, hem resumit com "Algorisme per a encriptar" o "Algorisme per a desencriptar" i es transformarà la lletra, que el programa retornarà per pantalla.

El programa també inclou missatges per a l'usuari informant-lo en tot moment de la seva posició inicial escollida i guiant-lo a través de les diferents accions que pot fer.



Il·lustració 4.3., Algorisme principal del programa

Ara, es mostrarà aquest algorisme escrit en C++. Després, es trobarà la seva explicació:

```
156     for (int i1 = 1; i1 <= 26; i1++){
157         for (int i2 = 1; i2 <= 26; i2++){
158             for (int i3 = 1; i3 <= 26; i3++){
159                 rotorPetitGira(&rp);
160                 cin >> n;
161                 if ((n >= 'a' && n <= 'z') || (n >= 'A' && n <= 'Z')){
162                     if (n >= 'a' && n <= 'z') n = 'A' + (n - 'a')%26;
163 
164                     int b = n;
165 
166                     if (encriptar){
167                         b = n + (rm + rg + rp);
168                         b = b + (rp*rm);
169                         b = b + (3*rp + 5*rm*rg - 6*rg - rp/2);
170                         b = b + (5*rp*rm - 2*rg*rp);
171                         b = b + (2*rp*rg*rm*rm + 11/rp);
172                         b = b + (29/rm + 23/rg - 6/rp);
173                     }
174                     else{
175                         b = n - (rm + rg + rp);
176                         b = b - (rp*rm);
177                         b = b - (3*rp + 5*rm*rg - 6*rg - rp/2);
178                         b = b - (5*rp*rm - 2*rg*rp);
179                         b = b - (2*rp*rg*rm*rm + 11/rp);
180                         b = b - (29/rm + 23/rg - 6/rp);
181                     }
182                     b = mod26(b) + 65;
183 
184                     char c = b;
185 
186                     cout << c;
187                 }
188                 else cout << "";
189             }
190             rotorMitjaGira(&rm);
191         }
192         rotorGranGira(&rg);
193     }
```

Il·lustració 4.4., Codi de l'algorisme principal del programa

Una vegada explicat el funcionament bàsic d'aquest algorisme, només falta explicar l'algorisme per a encriptar i desencriptar.

Per a encriptar, es fa el següent:

1. Suma els valors de les posicions dels tres rotors al valor numèric de la lletra introduïda.
2. Li afegeix el producte del petit i del mitjà.
3. Li afegeix tres vegades el petit, cinc vegades el producte del mitjà i del gran, li sostrau sis vegades el gran i li resta la meitat del petit.
4. Li afegeix cinc vegades el petit pel mitjà i li resta dues vegades el gran pel petit.
5. Li afegeix el doble del petit pel gran pel quadrat del mitjà i li afegeix onze vegades l'invers del petit.
6. Li afegeix vint-i-nou vegades l'invers del mitjà, vint-i-tres vegades l'invers del gran i li resta sis vegades l'invers del petit.

Aquest algorisme funciona perquè per a xifrar afegeix al missatge m un valor k en funció de les posicions dels rotors i fa mòdul 26 i, per a desxifrar, se'l resta i fa mòdul 26:

$$c = m + k \ (mod\ 26)$$

$$m = c - k \ (mod\ 26) = m + k - k \ (mod\ 26) = m \ (mod\ 26)$$

Per a desencriptar, el que es fa és fer el procés invers, és a dir, sostraure'n els valors afegits i sumar-ne els restats.

La complexitat de computació asimptòtica dels algorismes per a encriptar i desencriptar cada lletra és $O(n^0) = O(1)$, on n és el valor numèric de cada lletra, ja que l'algorisme trigarà el mateix temps a encriptar/desencriptar una lletra sempre, sense importar el valor numèric d'aquesta. Per a una frase sencera o un conjunt de caràcters, serà $O(n)$, on n és el nombre de caràcters de la frase. Aquesta complexitat de temps augmenta proporcionalment amb el nombre de lletres a encriptar/desencriptar, ja que l'algorisme realitza el mateix nombre de càlculs per a cada lletra sense importar la quantitat total d'aquestes. Per tant, el seu creixement és lineal.

Les posicions dels rotors, com ja s'ha esclarit abans, han estat escollides aleatòriament, i aquesta és la seva estructura:

Rotor gran	Rotor mitjà	Rotor petit
20	26	3
12	19	26
8	15	22
10	17	23
2	4	18
5	10	5
14	8	10
25	11	12
13	12	14
11	18	11
3	16	15
9	3	8
22	23	9
18	1	25
15	6	2
26	14	24
1	20	20
4	25	16
16	2	19
7	13	17
6	5	21
17	7	4
19	22	1
21	9	6
24	21	7
23	24	13

Il·lustració 4.5., Ordre aleatori dels valors numèrics dels rotors

Els rotors roten de la següent manera:

```
10  void rotorPetitGira(int *rp){ //fa girar el rotor petit
11      if (*rp == 3) *rp = 26;
12      else if (*rp == 26) *rp = 22;
13      else if (*rp == 22) *rp = 23;
14      else if (*rp == 23) *rp = 18;
15      else if (*rp == 18) *rp = 5;
16      else if (*rp == 5) *rp = 10;
17      else if (*rp == 10) *rp = 12;
18      else if (*rp == 12) *rp = 14;
19      else if (*rp == 14) *rp = 11;
20      else if (*rp == 11) *rp = 15;
21      else if (*rp == 15) *rp = 8;    ..., i segueix així, passant per les
22      else if (*rp == 8) *rp = 9;...  26 posicions del rotor
```

Il·lustració 4.6., Codi de l'algorisme que fa girar el rotor

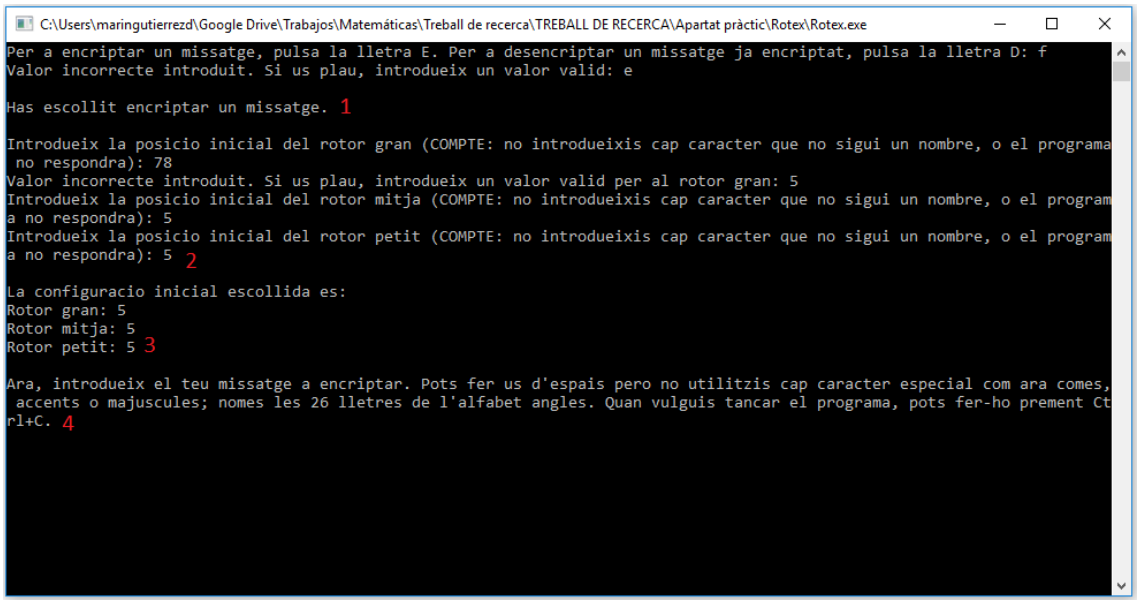
El que fa aquest algorisme (Il. 4.6.) és, una vegada l'usuari ha introduït la posició inicial del rotor, comprovar quin nombre ve després del que ha estat introduït en una estructura que emula la taula de la Il·lustració 4.5 i el canvia pel següent valor a la taula. La següent vegada que es cridi aquesta funció, farà el mateix amb el valor anterior, i així repetidament.

La resta del programa és només codi molt bàsic, fet per a guiar l'usuari a que faci un correcte ús del programa i que hi pugui introduir els valors desitjats. No s'inclou a aquest document, però si el lector ho desitja, pot consular el codi amb comentaris a l'arxiu "Rotex.cpp".

4.2.3. Debug o depuració

Una vegada ja ha estat creat el programa, s'han de provar diferents dades d'entrada per a comprovar que fa el seu treball bé.

Primer, s'analitzarà el comportament del programa quan aquest guia l'usuari a través de la configuració inicial. S'introduiran valors incorrectes (com ara un nombre més gran de 26 a una posició del rotor) per a comprovar la reacció del programa:



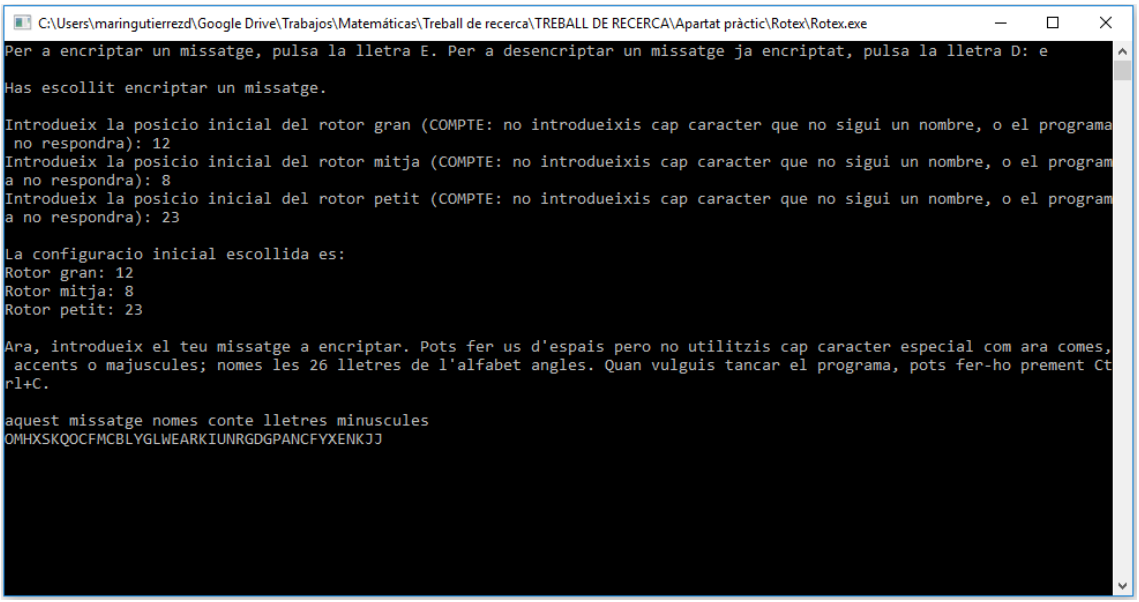
Il·lustració 4.7., Execució del programa: configuració inicial i valors incorrectes

Ara, s'analitzarà el resultat punt per punt:

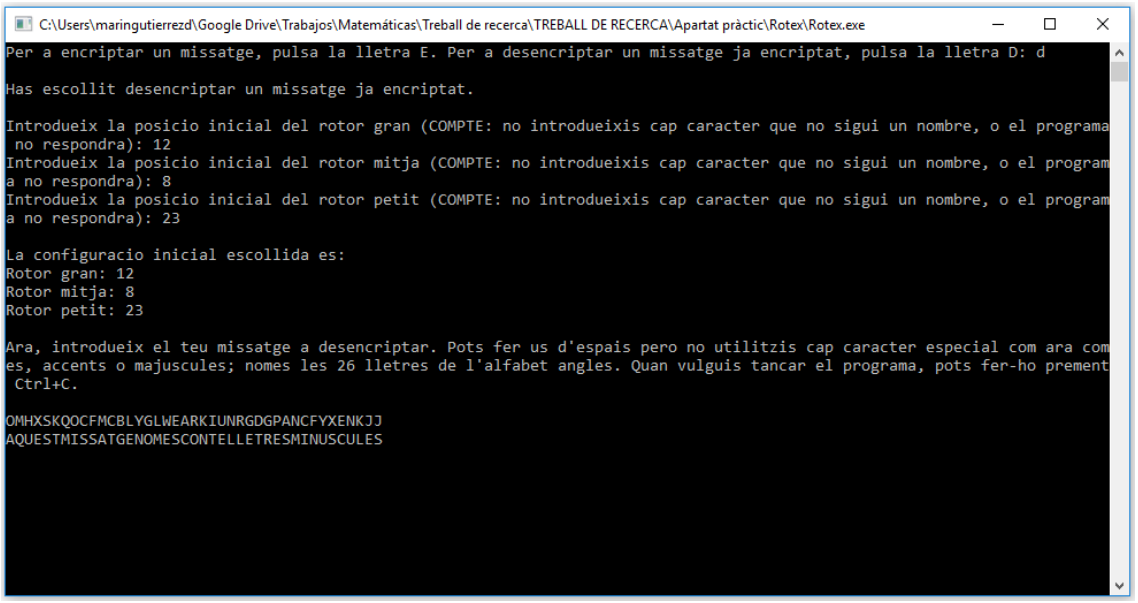
1. El programa dóna dues opcions a l'usuari: introduir una D per a desencriptar o una E per a encriptar. Quan s'introdueix un valor incorrecte, com ara una f, el programa ho notifica i fa saber a l'usuari que ha de tornar a introduir un valor i que, si us plau, sigui un de correcte. Després d'haver introduït un valor correcte, el programa fa saber quina opció ha estat escollida (en aquest cas, s'ha escollit encriptar).
2. El programa demana llavors que l'usuari introdueixi la configuració inicial dels rotors gran, mitjà i petit, en aquest ordre. Si un valor incorrecte és introduït (com ara el 78, ja que $78 \notin [1, 26]$, el programa ho notificarà i tornarà a demanar que un valor correcte sigui introduït.
3. Una vegada els tres rotors tinguin una posició inicial designada, l'usuari serà informat d'aquesta posició abans de començar a encriptar.
4. Finalment, el programa donarà instruccions clares de com introduir text per a ser encriptat i de com tancar-lo quan es desitgi.

El següent pas és provar que el programa encripti i desencripti de manera adequada i admeti l'ús de majúscules i de minúscules.

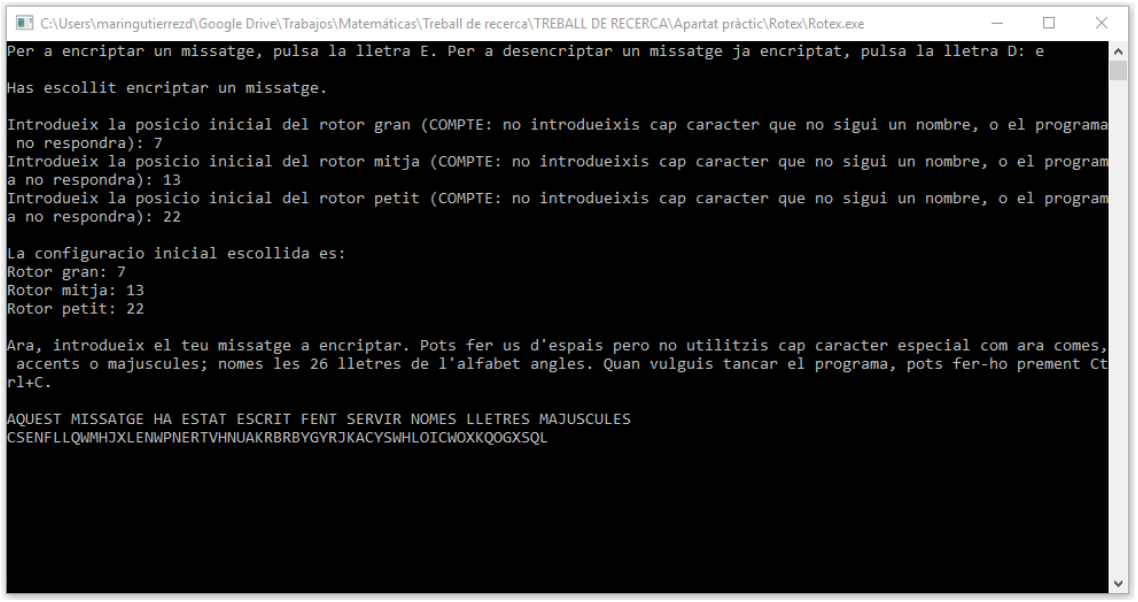
Per a encriptar el missatge amb minúscules, farem servir la configuració inicial 12, 8, 23. Per a encriptar el missatge amb majúscules, farem servir la configuració inicial 7, 13, 22:



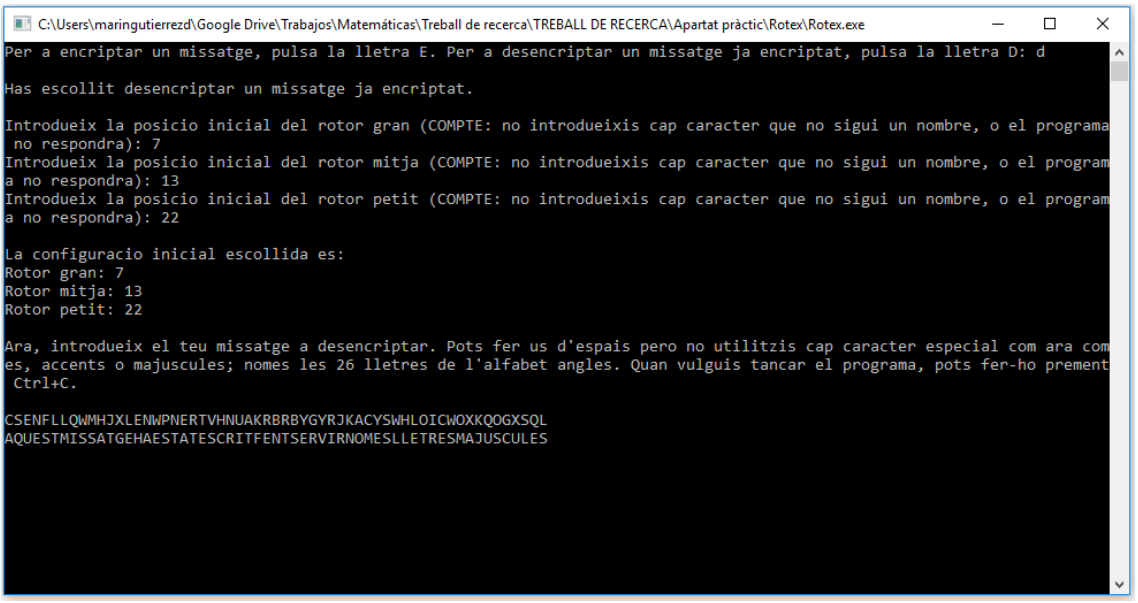
Il·lustració 4.8., Prova: encriptació d'un missatge amb lletres minúscules



Il·lustració 4.9., Prova: desencriptació d'un missatge amb lletres minúscules



Il·lustració 4.10., Prova: encriptació d'un missatge amb lletres majúscules

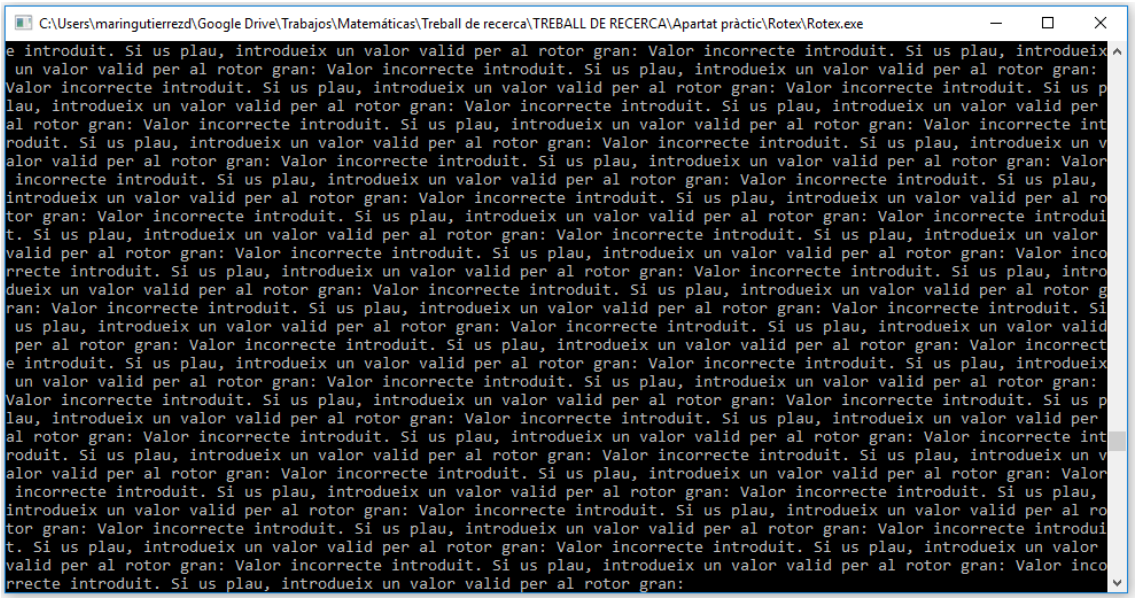


Il·lustració 4.11., Prova: desencriptació d'un missatge amb lletres majúscules

Com es pot comprovar a les quatre il·lustracions anteriors d'exemple, les proves d'encriptat i desencriptar han estat exitoses, la qual cosa permet estar-ne més segurs del bon funcionament del programa.

Cal destacar el següent funcionament incorrecte:

A l'hora d'introduir els nombres dels rotors, si l'usuari introdueix qualsevol caràcter que no sigui un nombre (0-9), el programa es penjarà i no funcionarà de la següent manera:



Il·lustració 4.12., Error produït per introduir un caràcter que no sigui un nombre com a posició d'un rotor

4.3. RSA: implementació d'un algorisme de xifrat asimètric

4.3.1. Anàlisi de l'objectiu

L'objectiu d'aquest últim punt és la implementació de l'algorisme RSA. Resumidament, és un mètode de xifrat asimètric que fa servir un sistema de clau pública i clau privada per a xifrar/desxifrar text.

A l'iniciar-se, el programa demanarà a l'usuari que li especifiqui si desitja encriptar o desencriptar un missatge fent servir un joc de claus creat anteriorment o si vol crear-ne un de nou per a rebre missatges. A l'escollir la primera opció, se li demanarà a l'usuari que esculli si vol encriptar un missatge o si vol desencriptar-ne un. A l'escollir la primera opció d'aquestes, se li demanarà llavors que introdueixi la clau pública (e, n). A l'escollir la segona, haurà d'introduir la clau privada (d, n). Si escull crear un nou joc de claus, el programa li demanarà que introdueixi totes les variables necessàries (p, q, e, etc.) per a crear joc nou. Si l'usuari no introdueix nombres primers quan és necessari, el programa li notificarà i li tornarà a demanar que introdueixi el nombre fins que n'introdueixi un de correcte. Les variables que són difícils de determinar, e i d, seran ajustades automàticament pel programa.

Una vegada totes les variables tinguin un valor correcte assignat, es llistaran i el programa finalitzarà la seva execució.

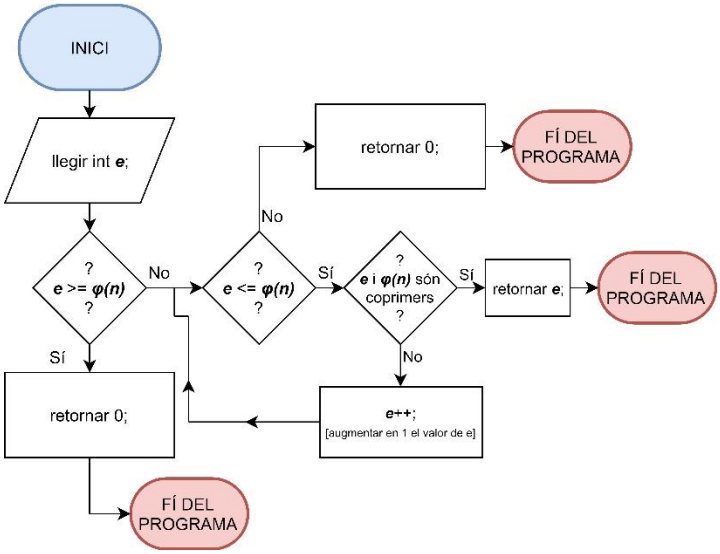
Aquest programa també inclourà missatges per a guiar els usuaris a través del procés d'encriptar, desencriptar o d'escollir un joc de claus per a rebre missatges.

4.3.2. Els algorismes i la programació

Aquest programa farà servir més d'un algorisme que cal destacar: un per a elevar nombres a potències molt grans i poder realitzar operacions d'aritmètica modular sense cap problema, per a ajustar les variables, un altre per a ajustar el valor de la variable d, etc. Es llisten a continuació:

4.3.2.1. Algorisme per a ajustar la variable e

Aquest algorisme funcionarà de la manera següent: primer, l'usuari haurà introduït un valor inicial per a e. L'algorisme llegirà aquest valor, i comprovarà si és un valor vàlid. De ser-ho, e es quedarà amb aquest valor. De no ser-ho, anirà provant d'un en un, diferents valors per a e fins a trobar el més petit i vàlid.



Il·lustració 4.13., Diagrama de flux de l'algorisme per a ajustar el valor de e

```
27 long long int ajustar_e(long long int e, long long int phi){
28     if(e >= phi) return 0;
29     while(e <= phi){
30         if(__gcd(e, phi) == 1) return e;
31         else e++;
32     }
33     return 0;
34 }
```

Il·lustració 4.14., Codi de l'algorisme per a ajustar el valor de e

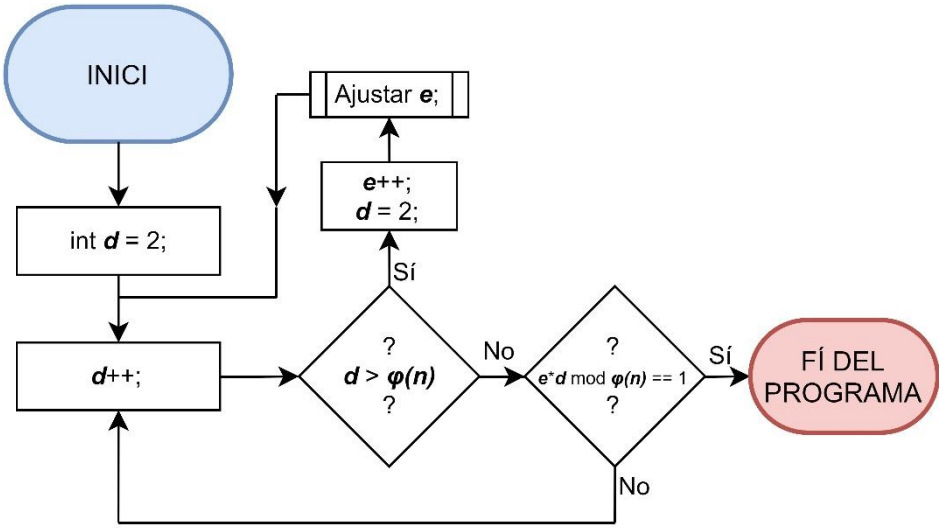
Funciona així:

1. Llegeix el valor de e.
2. Comprova que e sigui més gran que $\varphi(n)$. De ser-ho, retorna el valor 0 i finalitza l'execució. De no ser-ho, continua l'execució.
3. Mentre e sigui més petit que $\varphi(n)$, comprova que e i $\varphi(n)$ siguin coprimers. De ser-ho, retorna el valor actual de e. De no ser-ho, afegeix una unitat al valor de e i torna a executar la mateixa comprovació. Si en qualsevol moment e supera el valor de $\varphi(n)$, retorna 0 i finalitza l'execució.

La seva complexitat asimptòtica depèn totalment del valor de $\varphi(n)$, i és lineal. Aquest algorisme té una complexitat de $O(\varphi(n))$, és a dir, una recta, ja que, en el pitjor cas, ha de fer $\varphi(n)$ comprovacions per a donar un valor correcte de e.

4.3.2.2. Algorisme per a ajustar la variable d

Aquest algorisme serveix per a donar un valor correcte a la clau privada, és a dir, a la variable d. El procediment ha de fer dues combinacions, comprovar que la variable d compleixi les seves propietats, i després, ajustar el valor de e a un valor apropiat una altra vegada.



Il·lustració 4.15., Diagrama de flux de l'algorisme per a ajustar el valor de e

```
130     d = 2;
131     long long int compt = d;
132     do{
133         d++;
134         compt = expmod(e*d, 1, phi);
135         if(d > phi){
136             e++;
137             e = ajustar_e(e, phi);
138             d = 2;
139         }
140     }while(compt != 1);
```

Il·lustració 4.16., Codi de l'algorisme per a ajustar el valor de d

Aquest algorisme fa servir la funció que ajusta la variable e en el seu codi, i funciona així:

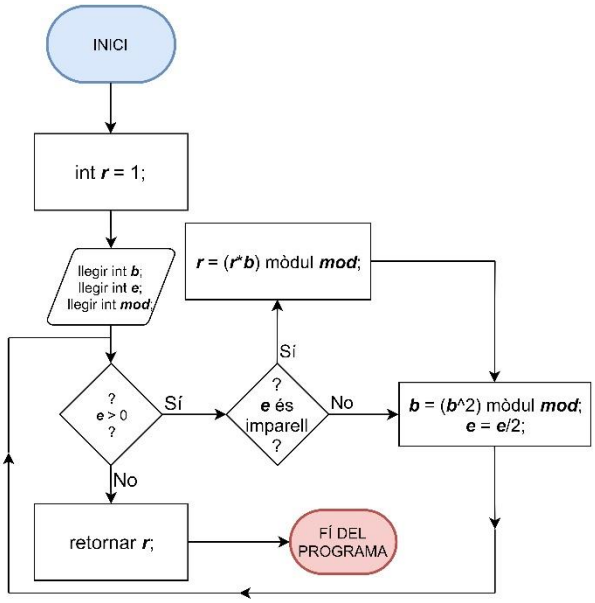
1. Afegeix una unitat al valor de d (de manera que $d = 3$).
2. Mentre el producte de e i d no sigui congruent amb 1 mòdul $\varphi(n)$, augmenta en una unitat el valor de d. Si el valor de d supera el de $\varphi(n)$, reinicia el valor de d a 2, augmenta en una unitat el valor de e, i l'ajusta fent servir l'anterior algorisme.
3. Si el producte de e i d és congruent amb 1 mòdul $\varphi(n)$, surt del bucle i deixa aquest valor de d.

Presenta una complexitat asimptòtica polinòmica, quadràtica exactament, i depèn totalment de la variable $\varphi(n)$. La complexitat augmenta amb el quadrat de $\varphi(n)$ ja que, en el pitjor cas ha de fer $\varphi(n)^2$ comprovacions fins a trobar el valor correcte, ja que l'algorisme és un bucle amb complexitat $O(\varphi(n))$ dins d'un altre bucle d'igual complexitat. La complexitat d'un algorisme que consta d'un bucle dins d'un altre és el producte de les complexitats individuals dels dos bucles. Per tant, la complexitat d'aquest algorisme és $O(\varphi(n)^2)$.

4.3.2.3. Algorisme d'exponenciació modular

A l'hora de programar, va sorgir un problema. A les *libraries* (biblioteques) de C++ hi ha una funció anomenada pow, que fa el següent: $pow(a, b) = a^b$. Aquesta funció presenta un inconvenient, a l'hora de fer $pow(c, d)\ (mod\ n)$, si d era un nombre molt gran (de més de nou xifres), retornava un valor incorrecte.

Degut a aquest inconvenient, hem hagut de definir una altra funció, més convenient per al nostre programa. La nova funció es diu $expmod$, i equival a: $expmod(b, e, m) = b^e\ (mod\ m)$.



Il·lustració 4.17., Diagrama de flux de la funció expmod

```
6  long long int expmod(long long int b, long long int e, long long int mod){
7      long long int r = 1;
8      while(e > 0){
9          if(e%2){
10             r = (r*b)%mod;
11         }
12         b = (b*b)%mod;
13         e = e/2;
14     }
15     return r;
16 }
```

Il·lustració 4.18., Codi de la funció expmod

Aquest algorisme/funció fa el següent:

Sempre que el valor de l'exponent e sigui més gran que 0:

1. Actualitza el valor de la base b a b^2 mòdul mod i redueix el valor de e a la meitat.
2. Comprova si e és imparell. De ser-ho, actualitza el valor del resultat r (que té l'1 com a valor inicial) al producte de r per b mòdul mod. De no ser-ho, no fa res.
3. Torna a l'acció nº 1.

Quan l'exponent e pren un valor que no sigui superior a 0, retorna el valor de r.

Per a entendre més bé el funcionament d'aquest programa, farem servir un exemple (per a fer-ho més senzill, el valor de l'exponent de l'exemple és molt petit):

Volem saber el valor de 217^{12} mod 463.

Partim de la base podem saber el valor de 217^2 mod 463 i de qualsevol nombre elevat a 2 (que és la base del funcionament del nostre programa).

Llavors, podem escriure 217^{12} com:

$217^{12} = 217^8 \cdot 217^4 = ((217^2)^2)^2 \cdot (217^2)^2$,

Pas a pas, el comportament del programa és el següent (és recomanable llegir-ho seguint el diagrama de flux):

r=1 , b=217, e=12, m=463.

1. <u>e > 0 i e és parell. Per tant, fer:</u>
   b=217^2 mod 463.
   e=12/2=6

2. <u>e > 0 i e és parell. Per tant, fer:</u>
   b=$(217^2)^2$ mod 463=217^4 mod 463.
   e=6/2=3.

3. <u>e > 0 i e és imparell. Per tant, fer:</u>
   r=1*217^4 mod 463.
   b=$(217^4)^2$ mod 463=217^8 mod 463.
   e=3/2=1 (en informàtica sempre s'arrodoneix per defecte a la part sencera).

4. <u>e > 0 i e és imparell. Per tant, fer:</u>
   r=$217^4 \cdot 217^8$ mod 463=217^{12} mod 463.
   b=no és important per a l'exemple.
   e=1/2=0.

5. <u>e > 0 no es compleix. Per tant:</u>
   retornar r.

$r = 217^{12}\ (mod\ 463)$.

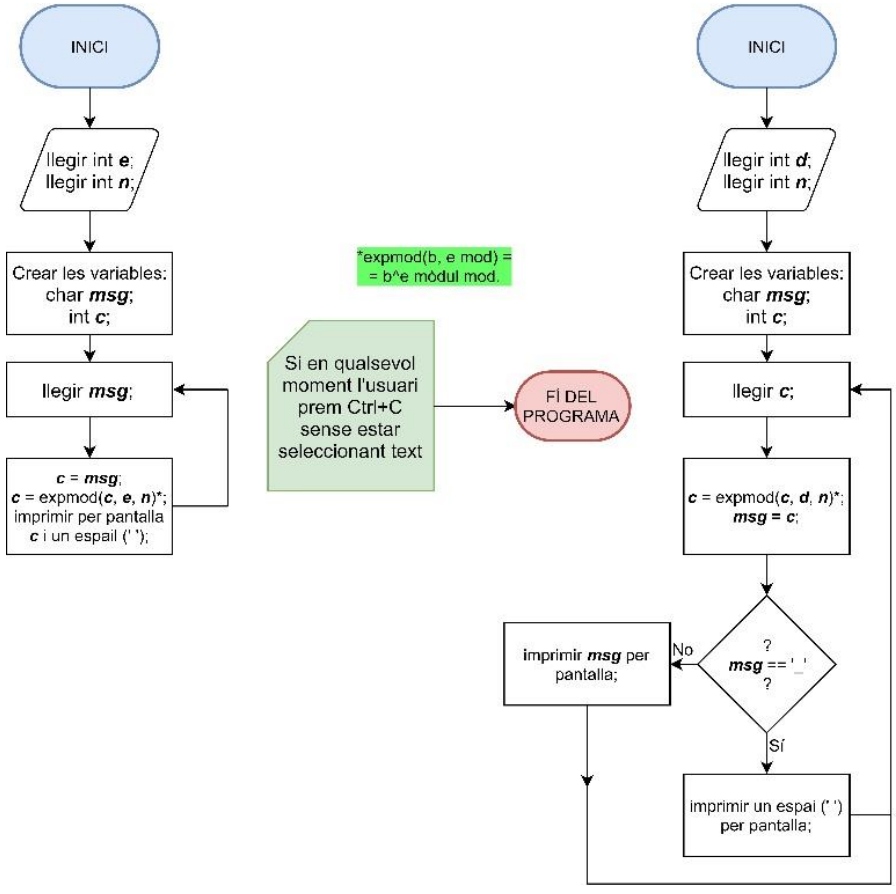
Es pot comprovar que $217^{12}\ (mod\ 463) = 217^8 \cdot 217^4\ (mod\ 463) = 132$.

Cal destacar que aquest exemple fa servir valors sense simplificar com ara $217^8 \ (mod\ 463)$ per a que el lector el pugui entendre, però l'ordinador els simplifica a l'hora de fer els càlculs. La principal diferència entre la funció predeterminada de C++ i aquesta és que la predeterminada eleva el nombre a la potència de manera directa, d'un sol cop, i l'ordinador no pot fer un càlcul tan gran d'una vegada. La nostra funció, no obstant, descompon el nombre en potències de 2, i fa els càlculs necessaris pas a pas, la qual cosa permet a l'ordinador fer les operacions correctament, ja que treballa amb nombres molt més petits.

Com que l'algorisme és un bucle que s'executa sempre i quan e > 0 i cada vegada que entra al bucle redueix el valor de e a la seva meitat, la complexitat d'aquesta funció és $O(\log_2 e + 1) = \log_2 2e$. Per exemple, per a e=64, l'algorisme farà el bucle set vegades, i per a e=128, vuit vegades. Seguint les normes de nomenclatura per a determinar la complexitat asimptòtica, les constants s'eliminen. Per tant, la seva complexitat és $O(\log_2 e)$.

4.3.2.4. Algorismes per a xifrar i desxifrar

Aquests algorismes, que són els principals del programa, serveixen per a fer la seva funció més important: encriptar i desencriptar missatges.



Il·lustració 4.19., Diagrames de flux dels algorismes per a xifrar (esquerra) i desxifrar (dreta)

```
185         while(cin >> msg){
186             c = msg;
187             c = expmod(c, e, n);
188             cout << c << " ";
189         }
```

Il·lustració 4.20., Codi de l'algorisme per a xifrar

```
193         while(cin >> c){
194             c = expmod(c, d, n);
195             msg = c;
196 
197             if(c == '_') cout << " ";
198             else cout << msg;
199         }
```

Il·lustració 4.21., Codi de l'algorisme per a desxifrar

L'algorisme per a encriptar fa el següent:

1. Llegeix msg, el missatge (un caràcter).
2. Li assigna el seu valor numèric a una variable c.
3. Eleva c a la potència e i li fa mòdul n.
4. Retorna aquest resultat per pantalla.

L'algorisme per a desxifrar fa el procés invers:

1. Llegeix c, el missatge xifrat.
2. L'eleva a d i li fa mòdul n.
3. Li assigna el valor numèric de c a una variable de tipus char, msg.
4. Si el caràcter és una barra baixa, retorna un espai. Si no ho és, retorna el caràcter.

   Aquest últim punt ha estat afegit per a que a l'hora d'encriptar, l'usuari pugui fer ús de barres baixes per a escriure espais, que més tard el receptor, quan desxifri el missatge, les veurà en forma d'espais. Aquesta part ha estat implementada degut a que el programa no pot llegir un espai de manera correcta.

Aquest algorisme funciona perquè, si els valors de e i de d han estat escollits correctament, sempre es compleix que $(m^e)^d(mod\ n) = c^d\ (mod\ n) = m\ (mod\ n)$, com ja va ser enunciat a la part teòrica i demostrat matemàticament a l'annex.

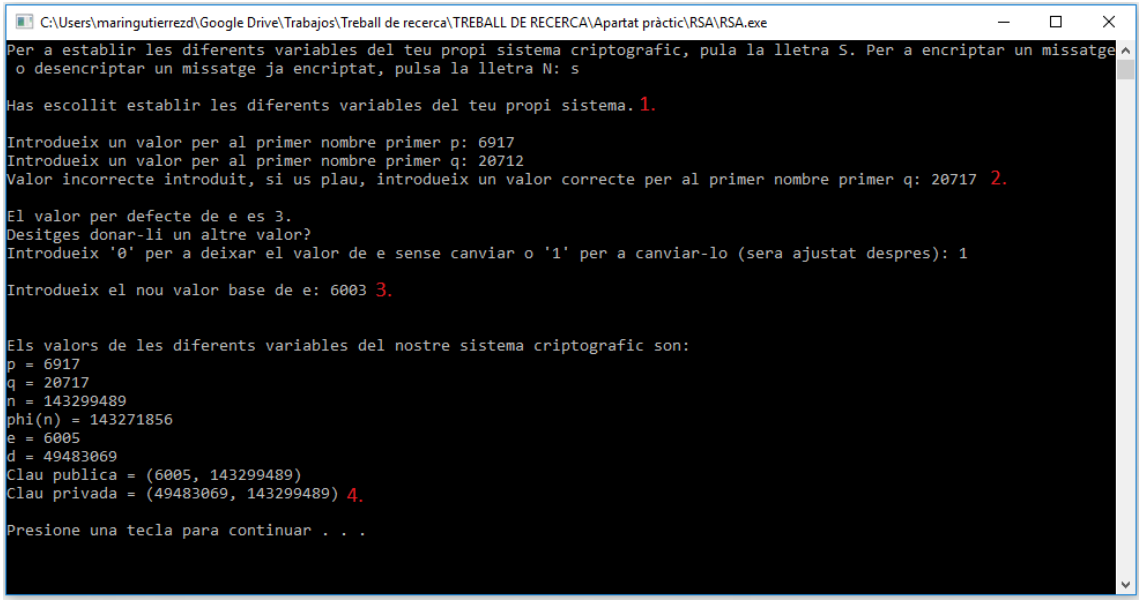
La manera d'escollir les altres variables és trivial i està explicada a la part teòrica, i per això no s'inclou aquí.

Si el lector desitja llegir el codi sencer amb comentaris, el pot trobar a l'arxiu "RSA.cpp".

4.3.3. Debug o depuració

Finalment, es provarà el correcte funcionament dels diferents aspectes del programa amb proves fotogràfiques i s'explicarà, punt per punt, el significat de cada resultat obtingut.

Primer, es provarà el correcte funcionament de la interfície per a establir un joc de claus i, després, provarem a xifrar i desxifrar un missatge fe servir el joc de claus generat anteriorment, i s'analitzaran els resultats punt per punt:

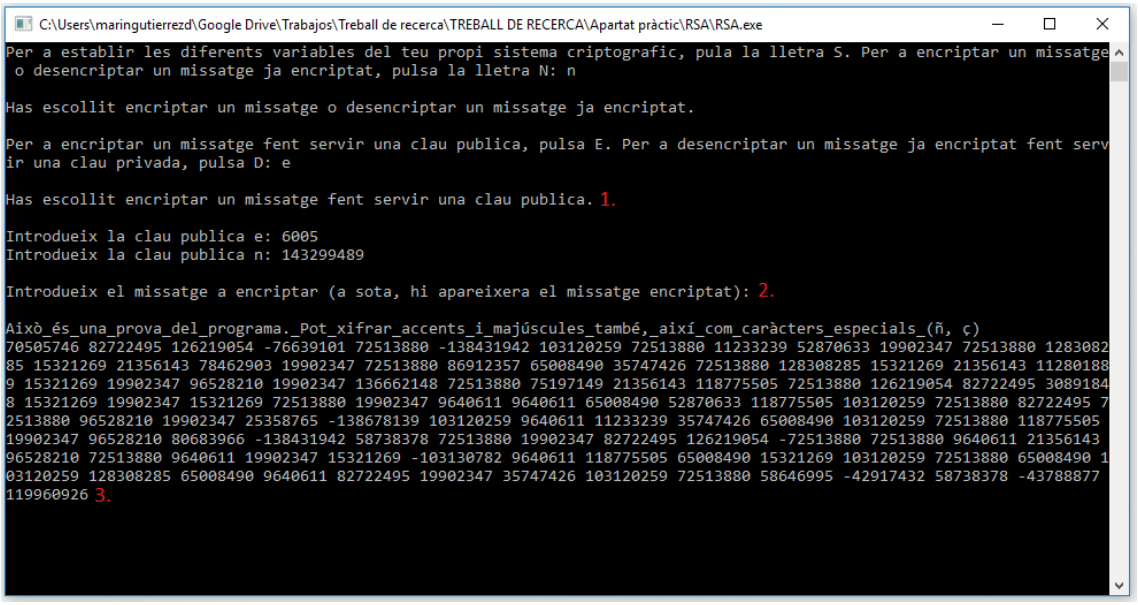


Il·lustració 4.22., Execució del programa: configuració inicial i valors incorrectes

1. El programa demana a l'usuari que introdueixi una S per a crear un joc de claus o una N per a procedir a xifrar o desxifrar un missatge. Una vegada introduïda la lletra S (que també pot ser minúscula), el programa informa a l'usuari de la seva elecció.
2. Ara, el programa procedeix a demanar dos nombres primers per a començar a construir el criptosistema. Quan un nombre no primer és introduït, el programa ho notifica i torna a demanar-ne un fins que n'obtingui un de primer.

3. L'últim element que queda encara per a definir és e, la clau pública. El programa diu a l'usuari que el valor predeterminat d'e és 3, i li pregunta que si desitjaria canviar-lo. Després d'una resposta afirmativa, se li demana que li doni un nou valor, que pot ser sigui ajustat després.
4. Finalment, el programa ajusta el valor de les diferents variables i fa els càlculs necessaris seguint els diferents algorismes llistats a l'apartat 4.3.2. i imprimeix per pantalla tots els valors de totes les variables.

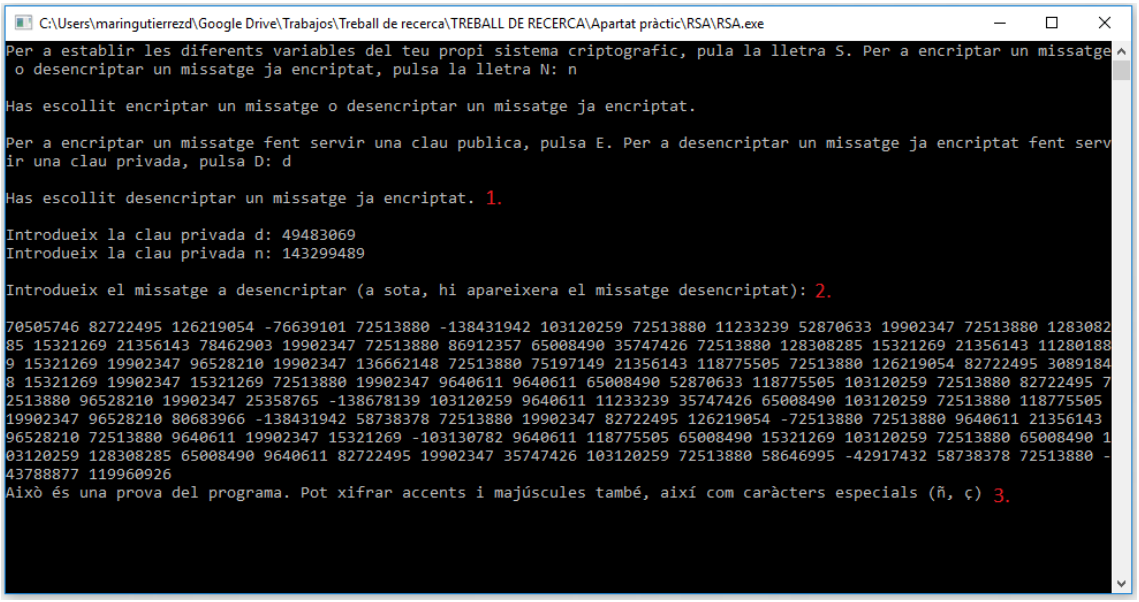
Ara, es provarà a xifrar un missatge pel mètode RSA fent servir les variables de l'exemple anterior. El missatge serà "Això és una prova del programa. Pot xifrar accents i majúscules també, així com caràcters especials (ñ, ç)", fent servir barres baixes com a espais:



Il·lustració 4.23., Prova: encriptació d'un missatge amb caràcters especials, accents, signes i majúscules

1. Es torna a formular la mateixa pregunta. Ara, s'ha introduït la lletra N. El programa llavors pregunta si es desitja xifrar o desxifrar un missatge. Quan s'introdueix E, el programa enregistra aquesta entrada i sap que ha d'executar l'algorisme per a encriptar un missatge.
2. Llavors, demana a l'usuari el conjunt de nombres que formen la clau pública, (e, n). Quan aquests són introduïts, es guarden i seran utilitzats per l'algorisme de xifrat. Finalment, es demana a l'usuari que introdueixi el missatge que vulgui encriptar.
3. Quan el programa llegeix aquest missatge, aplica l'algorisme de xifrar terme a terme i retorna per pantalla el resultat d'haver xifrat tots els caràcters del missatge. Tots els nombres que hi ha sota el missatge original són el missatge xifrat, i cadascun d'ells

correspon a una lletra o caràcter del missatge sense xifrar. Per exemple, el primer nombre, 70505746 correspon a la primera lletra del missatge, la A majúscula.



Il·lustració 4.24., Prova: desxifrat del missatge anterior

1. S'hi introdueix N i després D per a indicar que es vol desxifrar un missatge.
2. S'hi introdueix la clau privada $(d,\ n)$, en aquest ordre, que és enregistrada pel programa. Es notifica a l'usuari que ja pot procedir a escriure el missatge a desencriptar.
3. El programa ha desxifrat correctament el missatge anterior fent servir les claus públiques i privades creades en aquest exemple.

En quant a errors o mal funcionaments, cal destacar que si s'introdueixen nombres on s'hi haurien d'introduir lletres o viceversa, el programa no funcionarà de la manera esperada ni realitzarà cap de les seves funcions correctament.

5. Conclusions del treball

Avui dia, la criptografia juga un paper molt important a les nostres vides. És a tot arreu: quan comprem online, quan iniciem sessió a qualsevol web, quan una web emmagatzema de manera segura la nostra informació sensible (com ara el nombre del compte bancari), quan enviem missatges privats, etc. És el camp de treball que ens proporciona dos dels elements més importants de la vida moderna: seguretat i confidencialitat (privacitat). Seguretat perquè permet la realització d'accions com ara comprar online, on hem d'enviar la informació de la nostra targeta de crèdit a una web i, pel camí, la criptografia assegura que ningú no pugui interceptar la nostra informació i donar-li un mal ús; confidencialitat perquè ens permet poder enviar missatges xifrats d'extrem a extrem de manera que només les persones autoritzades puguin accedir a ells.

A causa de la ràpida evolució de la informàtica i de la creació d'ordinadors cada vegada més potents a una velocitat vertiginosa, els mètodes criptogràfics que són útils i moderns avui dia estan en perill de tornar-se obsolets en qüestió de dècades. A més, la possible futura invenció d'un ordinador quàntic amenaça tots els mètodes d'encriptació moderns, que basen la seva seguretat en la dificultat que presenten a l'hora de ser atacats per força bruta. La dificultat d'aquests mètodes moderns es basa en que són problemes que semblen fàcils de realitzar en un sentit però difícils de resoldre en l'altre sentit. Això no vol dir que siguin difícils de resoldre de manera objectiva, sinó que encara no hem sigut capaços de trobar un algorisme que els resolgui eficaçment en un temps real. Dit amb altres paraules: pot arribar el dia que el nostre coneixement de matemàtiques avanci fins a tal punt que trobem un algorisme eficient per a factoritzar el producte de dos primers molt grans en un temps breu, per exemple.

L'aparent existència de problemes la solució als quals pot ser comprovada en un temps raonable però *no* pot ser trobada en un temps raonable constitueix un dels set problemes matemàtics del mil·lenni: si podem *comprovar* una solució concreta a un problema en un temps raonable (polinòmic), implica això que existeix (independentment de si la coneixem o no) una manera de *trobar* una solució a aquest problema en un temps polinòmic també? Aquesta pregunta resumeix el problema $P = NP$*. Si es demostrés que sí, tots els mètodes criptogràfics moderns quedarien automàticament obsolets.

Per a combatre aquestes amenaces, crec que s'hauria d'investigar en criptografia quàntica, l'ús no només de principis matemàtics sinó també de principis de física quàntica per a xifrar missatges generant claus d'un sol ús (que tenen, per tant, secret perfecte) a través de l'ús de la

*: s'anima al lector que investigui sobre el problema $P = NP$. La solució a aquest problema té un premi d'un milió de dòlars!

polarització de fotons per a crear sistemes d'encriptació, ja que aquest tipus de criptografia, al fer servir claus d'un sol ús, seria irrompible. El principal problema de les claus d'un sol ús és que la seva creació és molt pesada, llarga i ineficient. Si aquestes es creessin fent servir fotons i la seva polarització, es podria crear un mètode criptogràfic amb secret perfecte (criptografia quàntica). No obstant, pel que he llegit, el principal problema que la creació d'un criptosistema quàntic presenta és la dificultat d'enviar fotons a llargues distàncies sense alterar la seva polarització i de poder enviar diferents grups de molts fotons sense que els uns interfereixin i modifiquin la trajectòria dels altres.

No obstant, actualment, els mètodes criptogràfics com l'RSA són encara segurs, fiables i eficients. Per tant, crec també que s'hauria d'investigar en la creació d'ordinadors potents amb molt poder computacional, capaços de portar a terme el mètode RSA per a variables cada vegada més i més gran i, al mateix temps, preparar-nos per a quan aquests mètodes deixin de ser útils, és a dir, preparar-nos per a la creació d'un ordinador quàntic: tenir les bases de la criptografia quàntica esclarides i assentades, i un pla o model a seguir per a modernitzar la infraestructura i la maquinària necessària per a fer servir els computadors quàntics i generalitzar el seu ús de la manera més ràpida possible (encara que en un primer moment només els puguin fer servir els governs i les grans empreses).

A l'àmbit personal, la realització d'aquest treball ha ampliat enormement els meus coneixements de matemàtiques aplicades a la computació. No només he après moltes matemàtiques, sinó que també he pogut apreciar la seva importància i aplicacions a un camp que m'agrada molt i, el que és més important, al món real. Les matemàtiques de la criptografia juguen un paper tan important a les nostres vides i al nostre dia a dia, que ens seria impossible imaginar l'estil de vida modern sense elles. El més important que m'ha aportat aquest treball de recerca és veure aplicacions de les matemàtiques al món real i l'ampliació dels meus coneixements en un camp que em fascina.

6. Annex

6.1. Aritmètica modular. Congruències

L'aritmètica modular és un sistema aritmètic, un tipus d'aritmètica que es crea a través de la creació de la relació de congruències entre nombres enters.

Dos enters a i b són congruents (en) mòdul n, si al ser dividits enterament per n, deixen el mateix residu.

Formalitzada, aquesta relació s'expressa per:

$$a \equiv b \ (mod\ n)$$

L'exemple més clar i quotidià de l'aritmètica modular és el sistema d'hores, que és mòdul 12. Per això, les 13 h equivalen a les 1 h, les 17 a les 5, etc.:

$13 \equiv 1 \ (mod\ 12)$, $17 \equiv 5 \ (mod\ 12)$

També passa el mateix al canviar de dia. Per exemple, si són les 22:00 h, i passen 5 h, no són les 27:00 h, sinó les 03:00 h. Això s'explica perquè el dia té 24 hores i, per tant, el mòdul és 24:

$$27 \equiv 3 \ (mod\ 24)$$

Encara que representen relacions diferents, la congruència té moltes similituds amb la igualtat. Per exemple, ambdues tenen la propietat de la reflexivitat: $a \equiv a \ (mod\ n)$, la simetria: $a \equiv b \ (mod\ n) \leftrightarrow b \equiv a \ (mod\ n)$, etc.

Per la definició de congruència, es dedueix trivialment que:

Si $a \equiv b \ (mod\ n) \rightarrow a + k \equiv b + k \ (mod\ n), k \in \mathbb{Z}$

També serveix si k resta, multiplica, divideix o eleva ambdós costats.

En aquest senzill principi es basa el programa Rotex que hem creat a l'apartat 4.2. de la part pràctica.

6.2. Nombres primers

Un nombre primer és un nombre natural major que 1 que no té més possibles divisors enters que 1 i si mateix. L'existència d'aquests nombres fa possible l'existència de mètodes com el RSA, i no només juguen un paper protagonista a l'encriptació moderna, sinó també a les

matemàtiques i a la teoria dels nombres. Són molt importants, també, pel fet que qualsevol nombre enter pot ser expressat com un producte de nombres primers.

6.2.1 Hi ha infinits nombres primers

El fet que hi hagi infinits nombres primers constitueix una propietat important per al mètode RSA. Ara, es demostrarà aquesta propietat a través d'una demostració per reducció a l'absurd, com va fer Euclides d'Alexandria fa més de dos mil·lennis. La reducció a l'absurd consisteix a considerar com a falsa una premissa que volem provar com a certa i, a través de deduccions lògiques, arribar a la conclusió que el fet que aquesta sigui falsa és una contradicció, per tant ha de ser certa per necessitat.

Assumim que *no* hi ha infinits nombres primers. És a dir, existeix un conjunt de nombres primers $p_1, p_2, \dots, p_{n-1}, p_n$ tals que p_n és el primer més gran de tots.

Ara, multipliquem-los tots i els sumem 1: $p_1 \cdot p_2 \cdot \dots \cdot p_{n-1} \cdot p_n = Q$. Aquest nombre Q, com tota la resta de nombres, pot ser primer o no-primer.

Si és primer, ja hem trobat la contradicció i, per tant, demostrat que hi ha infinits nombres primers.

Si no és primer, això vol dir que pot ser expressat com a producte d'altres primers de manera que $Q = q_1 \cdot q_2 \cdot \dots \cdot q_{s-1} \cdot q_s$. Però Q *no* és divisible de manera entera per cap p_x, ja que sempre hi quedaria 1 com a residu. Per tant, si Q no és primer, pot ser expressat com a producte de nombres primers, i aquest factors de Q són diferents de $p_1, p_2, \dots, p_{n-1}, p_n$, ja que cap d'aquests pot dividir enterament Q. Encara no hem demostrat que hi hagi infinits primers, només hem demostrat que n'hi ha més. Per tant, el conjunt de primers s'ha ampliat al conjunt format pels p_x i pels q_x. Ara, podem multiplicar una vegada tots els primers coneguts i trobar un altre nombre, T, que tampoc podrà ser expressat com a producte dels primers coneguts. Això s'allargaria fins a l'infinit, per tant, hi ha infinits nombres primers.

6.2.2 Funció $\varphi(n)$ d'Euler

La funció $\varphi(n)$ d'Euler, donat un nombre natural n, retorna la quantitat de nombres coprimers amb n que hi ha des de l'1 fins a arribar a n. Per exemple, $\varphi(12)$ és igual al nombre de números coprimers que hi ha amb 12 des de l'1 fins al 12. Aquests són l'1, el 5, el 7 i l'11, és a dir, 4 nombres → $\varphi(12) = 4$.

Aquesta funció té dues propietats molt especials:

Per a un primer p, $\varphi(p) = p - 1$, ja que p és coprimer amb tots els nombres menys amb ell mateix. Per tant, el nombre de números coprimers amb ell fins a arribar a ell mateix és ell mateix menys 1.

SI n és un producte de *dos* primers (p i q, per exemple), la funció es torna multiplicativa, és a dir, $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)$.

6.3. Ampliació de la justificació del nombre de combinacions possibles a Enigma

Un pot imaginar-se l'alfabet de manera que les deu parelles vinguin primer i les sis lletres que no importen siguin les últimes i separades per colors. Per exemple:

AB CD XJ ... LM TEKSUO.

1. Com que no importa l'odre en el qual apareguin les sis últimes lletres, l'alfabet de l'exemple equival a aquest (són el mateix cas):

AB CD XJ ... LM OSUTEK.

2. Com que tampoc importa l'ordre en el qual apareixen les parelles, els dos alfabets anteriors també equivalen al següent:

LM XJ CD ... AB TEKSUO.

3. Finalment, tampoc importa l'ordre de les lletres d'una parella, és a dir, no importa si una parella és AB o BA. Per tant, el següent alfabet també és el mateix cas que tots els anteriors:

ML BA DC ... JX TEKSUO.

1. Ens permet dividir pel nombre de possibles maneres de combinar sis lletres, ja que tots aquests possibles estats no ens importen. Dividim per $6!$
2. Ens permet dividir pel nombre de diferents maneres de col·locar deu elements (parelles), ja que no ens importa l'ordre en el qual aquests apareguin. Dividim per $10!$
3. Ens permet reduir el nombre possible de combinacions a la seva meitat tantes vegades com parelles l'odre de les quals no ens importi tinguem. Per tant, dividim per 2^{10}. És a dir, estem reduint el nombre total de possibles combinacions a la seva meitat (i a la meitat de l'anterior) deu vegades per a compensar que ens és igual si A es connecta

4. amb B o B es connecta amb A i si C es connecta amb D o si D es connecta amb D... i així per a deu cassos diferents alhora.

6.4. El mètode RSA. Justificació matemàtica

Per a fer aquesta justificació, ens ajudarem del Teorema d'Euler, enunciat per Leonhard Euler l'any 1734. Com que és un *teorema*, no cal que el demostrem nosaltres, i el podem considerar com a cert a l'hora de fer qualsevol demostració. El teorema diu el següent:

"Si a i n són nombres coprimers, $a^{\varphi(n)} \equiv 1 \ (mod\ n)$".*

Partint del teorema d'Euler, i les propietats podem de les congruències elevar ambdós costats a qualsevol constant entera positiva k:

$$(a^{\varphi(n)})^k \equiv 1^k \ (mod\ n) \rightarrow a^{\varphi(n)\cdot k} \equiv 1 \ (mod\ n)$$

Podem també multiplicar ambdós costats per a:

$$a \cdot a^{\varphi(n)\cdot k} \equiv a \cdot 1 \ (mod\ n) \rightarrow a^{\varphi(n)\cdot k+1} \equiv a \ (mod\ n)$$

Una de les propietats del mètode RSA era que $e \cdot d \equiv 1 \ (mod\ \varphi(n))$. D'aquí es dedueix la següent expressió: $e \cdot d = k \cdot \varphi(n) + 1$, on k és qualsevol constant entera positiva. Gràcies a la deducció de la igualtat anterior es pot justificar la correcció del mètode RSA.

La hipòtesi que volem demostrar és $m \equiv (m^e)^d \ (mod\ n)$.

$(m^e)^d = m^{e\cdot d}$, que gràcies a la igualtat anterior, pot ser reescrit així: $m^{k\cdot\varphi(n)+1}$.

$$m^{k\cdot\varphi(n)+1} = m^{k\cdot\varphi(n)} \cdot m = \left(m^{\varphi(n)}\right)^k \cdot m$$

*Ara, aplicant el teorema d'Euler a $\left(m^{\varphi(n)}\right)^k \cdot m$,

$\left(m^{\varphi(n)}\right)^k \cdot m \equiv 1^k \cdot m \ (mod\ n) \rightarrow \underline{(m^e)^d \equiv 1^k \cdot m \ (mod\ n)}$

I, finalment, com que $1^k \cdot m = m$, la congruència ressaltada en vermell es reescriu com:

$$m \equiv (m^e)^d \ (mod\ n)$$

Per tant, la nostra hipòtesi inicial ha estat demostrada.

*: podria donar-se el cas que m i n no fossin coprimers, però mai es donarà, ja que m sempre rebrà valors petits (mai superarà les tres xifres) i els dos únics factors del nombre n seran nombres molt més grans que els factors del nombre m. Per tant, no hi ha cap perill, ja que m i n, a la pràctica sempre seran coprimers.

6.5. El codi ASCII

El codi ASCII (codi estàndard americà per a l'intercanvi d'informació) és un codi basat en l'alfabet anglès (llatí) que fa servir 7 bits per a representar caràcters. És un dels codis més emprats a l'hora d'assignar valors numèrics a lletres.

El código ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

Caracteres de control ASCII

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

ASCII extendido

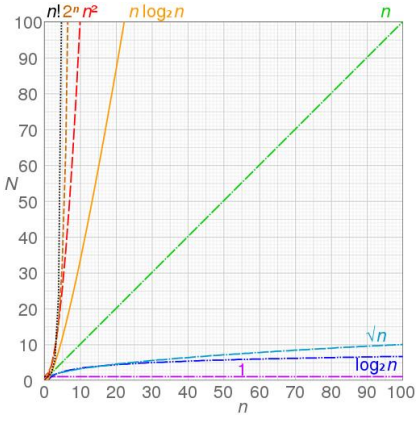
DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Il·lustració 6.1., Taula d'equivalències número-símbol del codi ASCII

6.6. Complexitat asimptòtica. *Big-O notation*

La teoria de la complexitat estudia com creix el cost computacional (principalment en termes de memòria i temps, encara que també en el nombre d'accions que s'han de fer) de resoldre un determinat problema en funció de la mida del problema i a mesura que aquesta augmenta.

La complexitat asimptòtica d'un algorisme ve donada per la funció que relaciona el cost computacional que li costa a un algorisme resoldre un determinat problema amb la mida d'aquest mateix problema. Aquesta complexitat pot ser constant, lineal, polinòmica, logarítmica, exponencial, etc.



Il·lustració 6.2., gràfica de les diferents tipus de complexitats computacionals

Per a representar-la se sol fe servir la *big-O notation* que, de manera molt resumida, indica el tipus de funció amb la qual el cost computacional creix a mesura que la mida del problema creix, és a dir, si creix a un ritme exponencial, quadràtic, factorial, etc.

Per raons de nomenclatura, s'eliminen els valors constants, tots els termes de menor grau que estiguin sumant o restant, etc. Bàsicament, es deixa només la part que indica quin tipus de funció és. Per exemple:

$O(12x^3 + 7x^2 - 98x + 1) = O(x^3)$,

$O\left(\frac{x^5 - 8x^4}{200x^4 + 76x}\right) = O\left(\frac{x^5}{x^4}\right) = O(x)$, etc.

En definitiva, $O(f(x))$ simbolitza la forma conceptual que $f(x)$ té quan $x \to \infty$.

7. Bibliografia i webgrafia. Recursos

7.1. Recursos bibliogràfics

- SMART, Nigel: *Cyrptography: an introduction*. McGraw-Hill College, 2004. 3ª edició. ISBN-13: 978-0077099879.
- ROSEN, Kenneth H.: *Discrete mathematics and its applications*. McGraw-Hill College, 2012. 7ª edició. ISBN-13: 9780073383095.
- RUOHONEN, Keijo: *Mathematical cryptology*. Universitat de Tecnologia de Tampere, 2014. 1ª edició. ISBN-13: 9780073383095.
- KOSHY, Thomas. *Elementary number theory with applications*. Academic Press (Elsevier), 2007. 2ª edició. ISBN-13: 9780123724878.
- PAAR, Christof; PELZL, Jan. *Understanding cryptography*. Springer, 2010. 1ª edició. ISBN-13: 978-3642041006

7.2. Recursos en la web

La majoria d'informació del treball ha estat extreta dels llibres llistats anteriorment, o de ponències d'universitats (contingut audiovisual). Les úniques webs que he visitat, i només amb l'objectiu de contrastar la informació dels llibres i dels oradors de les universitats han estat les versions anglesa i espanyola de la viquipèdia (https://ca.wikipedia.org).

De vegades, però, quan em sorgia un dubte, el buscava a Internet en forma de pregunta i consultava Stack Exchange: pàgina web que permet a estudiants i professionals fer preguntes i que altres estudiants o professionals les responguin (les respostes a aquestes preguntes són verificades per experts). Mai he fet una pregunta jo mateix, però sí que he consultat les respostes (només les verificades) a preguntes que havien fet altres estudiants. Les comunitats d'Stack Exchange que he visitat són la comunitat anglo-parlant de matemàtiques (https://math.stackexchange.com) i la de criptografia (https://crypto.stackexchange.com) . A l'hora de programar, també he consultat preguntes fetes a la comunitat de programadors Stack Overflow (https://stackoverflow.com), també part d'Stack Exchange.

7.3. Recursos audiovisuals

- PAAR, Christof. *Introduction to cryptography* [vídeo online]. Bochum: el seu propi canal de YouTube, 2014.
  < https://www.youtube.com/watch?v=2aHkqB2-46k>.

- PAAR, Christof. *Modular arithmetic and historical ciphers* [vídeo online]. Bochum: el seu propi canal de YouTube, 2014. <https://www.youtube.com/watch?v=W1SY6qKZrUk>.
- PAAR, Christof. *Stream ciphers, random numbers and the one time pad* [vídeo online]. Bochum: el seu propi canal de YouTube, 2014. <https://www.youtube.com/watch?v=AELVJL0axRs>.
- PAAR, Christof. *Multiple encryption and brute-force* attacks [vídeo online]. Bochum: el seu propi canal de YouTube, 2014. <https://www.youtube.com/watch?v=M10BVpTCzGg>.
- PAAR, Christof. *Number theory for PKC: Euclidean algorithm, Euler's phi function & Euler's theorem* [vídeo online]. Bochum: el seu propi canal de YouTube, 2014. <https://www.youtube.com/watch?v=fq6SXByItUI>.
- PAAR, Christof. *The RSA cryptosystem and efficient exponentiation* [vídeo online]. Bochum: el seu propi canal de YouTube, 2014. <https://www.youtube.com/watch?v=QSlWzKNbKrU>.
- PERRY, David. *The Enigma code* [vídeo online]. Califòrnia: canal de YouTube de l'UC Davis, 2010. < https://www.youtube.com/watch?v=ncL2Fl6prH8>.
- MAKAROV, Vadim. *Introduction to quantum cryptography* [vídeo online]. Waterloo: canal de YouTube de l'Institute for Quantum Computing (Universitat de Waterloo), 2014. < https://www.youtube.com/watch?v=ToOLbdrWst4>.