USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Criptografía simétrica vs. asimétrica

SARA RODRÍGUEZ GÓMEZ

2020/2021

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Criptografía simétrica vs. asimétrica

SARA RODRÍGUEZ GÓMEZ
TITOR: FELIPE GAGO COUSO
COTITORA: MARÍA PILAR PÁEZ GUILLÁN

Xullo, 2021

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Traballo proposto

Área de Coñecemento: Álxebra
Título: Criptografía simétrica vs. asimétrica
Breve descrición do contido
Anque os primeiros usos da criptografía datan de fai máis de 4000 anos, no Antigo Exipto, e as matemáticas aínda son máis antigas, non vai moito tempo que estas dúas disciplinas converxeron. Este encontro significou un cambio profundo para a criptografía, que ata entón se limitaba a comunicarse en segredo e que hoxe é, en boa medida, unha disciplina matemática. O novo paradigma de funcións de vía única, procesos doados de levar a cabo pero difíciles de invertir, está detrás da criptografía asimétrica, tamén chamada de chave publica, que en vez de usar unha única chave, que era preciso intercambiar previamente para despois poder encriptar, usa unha chave pública, coñecida por todos, e unha chave privada, só coñecida polo propietario. Este traballo pretende facer unha introdución á criptografía, dende a clásica ata chegar a moderna, onde falaremos da simétrica e a asimétrica, compararémolas, e revisaremos algúns dos seus algoritmos dende o punto de vista matemático e computacional para entender en que se basea o funcionamento e a seguridade presente e futura dos mesmos.
Recomendacións
Cursar a materia *Códigos Correctores e Criptografía.*
Outras observacións

Índice xeral

Resumo

A criptografía clásica xorde no século V a.C. ligada ao mundo militar e co único fin de ocultar mensaxes dun xeito rudimentario. Non é ata arredor do 1948, cando esta se sitúa nun contexto computacional e matemático grazas ao científico Claude Shannon, nacendo así a criptografía moderna.

Esta última clasifícase en simétrica e asimétrica. Na primeira tanto quen emite como quen recibe empregan unha mesma e única clave previamente compartida por unha canle segura, para cifrar e descifrar a mensaxe, con algoritmos como o DES ou o AES. A necesidade de que exista unha canle segura pode supor unha desvantaxe, á cal veu dar resposta a criptografía asimétrica, proposta por Diffie e Hellman. Nela, cada usuario posúe dúas claves, unha pública, coñecida por todos e que o emisor utilizará para ocultar a mensaxe, e outra privada que permitirá soamente a dito usuario recuperar o texto orixinal, con cifrados como o RSA ou ElGamal.

A única criptografía que aporta confidencialidade e non repudio as mensaxes é a asimétrica, facendo uso da sinatura dixital. Non obstante, a moi superior velocidade de cifra da simétrica fronte a esta fai que na maioría das aplicacións resulten necesarios e útiles os cifrados híbridos. Estes empregan a asimétrica para levar a cabo o intercambio da clave, co algoritmo de Diffie-Hellman, entre outros, e logo farase uso desa clave para cifrar a mensaxe con métodos simétricos. Un exemplo é o protocolo SSL.

Palabras chave: criptografía, simétrica, asimétrica, cifrado, algoritmo, clave, mensaxe.

Abstract

Classical cryptography arises in the 5th century B.C. linked to the military world and with the sole purpose of hiding messages in a rudimentary way. It was not until around 1948, when it was placed in a computational and mathematical context thanks to scientist Claude Shannon, that modern cryptography was born.

The latter is classified as symmetric and asymmetric. In the first, both the sender and the receiver use the same and unique key previously shared through a secure channel, to encrypt and decrypt the message, with algorithms such as DES or AES. The need for a secure channel can be a disadvantage, to which the asymmetric approach proposed by Diffie and Hellman offers a solution. In it, each user has two keys, a public one, known by everyone and that the sender will use to hide the message, and a private one that will allow only that user to retrieve the original text, with encryption systems such as RSA or ElGamal.

The only type of cryptography that provides confidentiality and non-repudiation messages is asymmetric, making use of the digital signature. However, the much higher encryption speed of the symmetric makes hybrid encryption necessary and useful in most applications. It uses the asymmetric to share the keys, precisely with the Diffie-Hellman algorithm, or any other, and then this key is used to encrypt the message with symmetric methods. An example is the SSL protocol.

Keywords: cryptography, symmetric, asymmetric, encryption, algorithm, key, message.

Introdución

Ocultar mensaxes escritas de xeito que se caen en mans equivocadas non poidan ser lidas, ou ata modificadas por un terceiro, é unha necesidade xurdida fai miles de anos. Téñense probas de métodos ideados con tal fin que datan de cincocentos anos antes de Cristo.

As primeiras formas de ocultar o texto que se coñecen só consistían en agochalo sen modificalo, isto é do que se encarga a ciencia coñecida como esteganografía, que se atopa na antigüidade en situacións como as seguintes –que podemos ver, por exemplo, en [13]–. Aos escravos cortábaselles o pelo e escribíaselles o texto a ocultar na cabeza, unha vez que o pelo volvía medrar enviábanse coas mensaxes aos destinos. Outro caso dábase na Italia do século XV, onde se empregaba unha tinta especial para escribir sobre un ovo cocido, esta era capaz de penetrar a casca e plasmarse na albumina ocultando alí o escrito. Ou na actualidade, esconder un documento de texto nunha imaxe ou nun arquivo de audio MP3.

A criptografía xorde con este mesmo obxectivo, pero os métodos que emprega son totalmente diferentes. Neste caso, a seguridade non recaerá en esconder a mensaxe e si en modificala, é dicir, en ecriptala de tal xeito que só o receptor será capaz de recuperar a enviada nun inicio.

Algúns destes primeiros cifrados son a escítala, unha vara dun determinado grosor arredor da que se envolve o papiro para escribir e ler mensaxes, ideada no século V a.C., os xeróglifos nas tumbas exipcias, do terceiro a.C., ou cifrados indios que empregaban substitucións no alfabeto. Precisamente os indios foron os que introduciron a criptografía no goberno para o intercambio de información sensible. A idea tomou forza durante a Idade Media ata acabar empregándose en todos os gobernos de Europa. Xa no 1860 se atopaba na maioría das comunicacións diplomáticas e, tempo despois, cobraría moita importancia nas militares. Un claro exemplo desto último é a máquina Enigma empregada na Segunda Guerra Mundial, que permitía aos alemáns enviarse mensaxes encriptadas que só podían entender eles, á vista de todos.

A orixe da criptografía pode situarse, polo tanto, hai máis de catro mil anos co nacemento de sinxelos e rudimentarios métodos de encriptación ligados ao mundo militar pero, non é ata o ano 1948 cando o científico Claude Shannon a enmarca nun contexto matemático. A partir dese momento a importancia desta foi medrando ata chegar a ser indispensable no día a día polas súas aplicacións no comercio electrónico, na identificación online e ata na seguridade a través de Internet.

Este traballo consta de cinco capítulos ao longo dos que se introducirán interesantes formas de encriptar dende un punto de vista matemático e computacional. A narración faise cronoloxicamente deixando así ver, dun xeito moi intuitivo, como van aparecendo novos algoritmos de cifrado como resposta as limitacións que se van atopando nos existentes.

No primeiro capítulo verase a orixe da criptografía e estudaranse dende un punto de vista actual as matemáticas que hai detrás deses primeiros cifrados clásicos.

No segundo introdúcese a criptografía moderna que pode clasificarse na simétrica e na asimétrica, e, en función de como o cifrado encripte o escrito, nos que o dividen en bloques ou nos que o cifran en fluxo. Defínese tamén a criptoanálise como a ciencia que estuda a forma de descubrir a mensaxe que se oculta no texto encriptado.

O terceiro capítulo estuda a criptografía simétrica, as ventaxas e limitacións coas que conta, e detalla algúns dos algoritmos simétricos máis importantes, como son o DES e AES, dos que poderán atoparse posibles implementacións en `SageMath`.

O cuarto capítulo adícase á criptografía asimétrica, explícase dun xeito divulgativo empregando unha analoxía cos cadeados, para xa logo introducir o protocolo de intercambio de chaves de Diffie e Hellman, e o algoritmo máis importante na actualidade, o RSA. Estudaranse máis a fondo as matemáticas que esconde dito algoritmo así como as que demostran o seu bo funcionamento, e apórtanse interesantes implementacións en `SageMath`, tando del como de tales matemáticas das que fai uso.

O último capítulo resume algunhas das aplicacións actuais da criptografía, que a fan indispensable no día a día, e que requiren tanto da simétrica como da asimétrica. Recálcase a gran diferencia na velocidade de cifrado que hai entre ambas, e a capacidade dos algoritmos asimétricos para obter unha firma dixital.

Capítulo 1

Orixe da criptografía: a criptografía clásica

A criptografía, do grego "kryptos" oculto, e "graphia" escritura, xorde en épocas arcaicas co obxectivo, que podemos intuír do significado etimolóxico da palabra, de codificar mensaxes facéndoas inintelixibles para aqueles a quen non van dirixidas. Esta primeira criptografía é a que se coñece como criptografía clásica.

A criptografía clásica abarca as técnicas máis primitivas e intuitivas que permiten ocultar unha mensaxe –texto claro– cifrándoa e converténdoa no que se coñece como criptotexto. Pode clasificarse, segundo as técnicas de encriptación que se empreguen, en cifrados de transposición ou de substitución [6]:

Definición 1.1 (Cifrado de transposición)**.** O **cifrado de transposición** é unha técnica criptográfica que consiste en intercambiar a posición das letras dunha palabra ou frase seguindo sempre un esquema ben definido. Un exemplo sinxelo é encriptar unha mensaxe escribíndoa ao revés.

Definición 1.2 (Cifrado de substitución)**.** O **cifrado de substitución** é unha técnica de encriptación onde a cada letra, ou conxunto de letras, da mensaxe clara, se lle fai corresponder unha determinada grafía, que será polo que se substitúa para cifrala, dando lugar ao criptotexto. Se para cifrar se emprega só un alfabeto, diremos que é un cifrado de substitución **monoalfabético**, noutro caso, coñecerase como **polialfabético**.

A aparición de novas técnicas de encriptación débese a que as existentes ata o momento deixan de ser seguras: unha técnica de encriptación deixa de ser segura cando se atopa a forma de ler a mensaxe clara que oculta o criptotexto xerado con tal técnica, dise que o

cifrado sufre un ataque capaz de rompelo, e como resposta a dito ataque xurdirán novos métodos para cifrar. Así vai evolucionando a criptografía. Neste capítulo estudaremos a evolución dos cifrados clásicos [7]; comezando pola Escítala, o primeiro cifrado (de transposición) que se coñece, e que data de antes de Cristo; pasando por cifrados de substitución monoalfabéticos, como é a Cifra de César; ata chegar a necesitar os polialfabéticos, como Alberti ou Vigenère. Describiremos as distintas técnicas dende un punto de vista máis moderno e analizaremos os ataques que sufriron –introducindo así o concepto de criptoanálise, considerado na actualidade como unha ciencia, que se encarga de buscar formas de romper cifrados.

Pode situarse a orixe da criptografía no século V a.C. durante a guerra entre Atenas e Esparta, época da que data o primeiro método de encriptación coñecido, a **Escítala**: os espartanos empregaban unha vara dun determinado grosor, ao redor da cal se envolve unha tira de papiro, sobre a que logo se escribe a mensaxe lonxitudinalmente, como pode observarse na Figura 1.1; así, ao desenvolver o pergamiño só se ten unha sucesión de letras desordenadas, e a mensaxe orixinal non será entendible ata envolver de novo dita tira nunha vara do mesmo grosor que a inicial.

Figura 1.1: Escítala.
(Fonte: Wikipedia Commons)

Podemos ver a Escítala como un cifrado de transposición de grupos, pois a esencia do algoritmo de encriptación que emprega radica en modificar a posición das letras do comunicado seguindo un patrón. En termos matemáticos diríase que o cifrado emprega permutacións para descolocar as grafemas da mensaxe.

No século II a.C., xorde un cifrado ideado polo historiador grego **Polibio**, e que leva por tanto o seu nome. Consiste nunha matriz de 5 x 5 onde se distribúen as letras do alfabeto, a isto engádese unha columna e unha fila con 5 caracteres, como poden ser os números do 1 ao 5. Así, calquera letra da matriz será cifrada co par formado polos elementos da columna e fila correspondentes. Un exemplo de táboa deste cifrado é o seguinte:

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N/Ñ	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Táboa 1.1: Táboa de Polibio.

Exemplo 1.3. Empregando a Táboa 1.1, pode observarse que a palabra POLIBIO cifraríase como 35343124122434. En canto ao descifrado, en ocasións terase máis de unha opción debido a colocación da I xunto ca J, e a N coa Ñ, pero é claro ver que só unha desas opcións será lóxica, e será polo que descifremos. No exemplo téñense as opcións POLJBJO, POLIBJO, POLJBIO ou, claramente a correcta, POLIBIO.

Podemos observar que ao numerar as filas e as columnas da matriz, este cifrado converte as letras en números, moito máis prácticos na criptografía e que permitiron ademais nesa época a comunicación secreta a distancia: a idea de Polibio consistía en empregar palluzos para enviar sinais luminosos, os da man esquerda sinalaban o número da fila, e os da dereita o da columna nas que estaba a letra cifrada. Ao lonxe era visible a luz dos palluzos e permitía ir lendo o cifrado sen necesidade de ningún tipo de acercamento ou mensaxeiro.

Na época de Julio César, século I a.C., comezou a empregarse o cifrado de substitución coñecido, precisamente, como a **cifra de César**, que encripta carácter a carácter cambiándoo polo que está un determinado número de posicións a continuación del no alfabeto. A clave secreta, k, será precisamente, o número de posicións que se acordan avanzar, $k = 3$ era a que empregaba César.

Na Táboa 1.2 móstrase o alfabeto galego e de novo o mesmo alfabeto desprazado 3 posicións, permitindo así observar a correspondencia de cada letra co seu cifrado de César.

Alfabeto orixinal	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
Alfabeto cifrado	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C

Táboa 1.2: Correspondencia entre os alfabetos orixinal e cifrado.

Actualmente podemos describir o algoritmo de encriptación deste método empregando aritmética modular. Cando falamos de avanzar 3 caracteres facémolo de forma cíclica, é dicir, para cifrar Z avánzase (comezando de novo por A) ata chegar a C. Estamos nun conxunto finito formado polas 23 letras do alfabeto, ao que lle chamamos en matemáticas traballar en aritmética modular con módulo 23. Numerando entón as grafías da A a Z, denotando A por 0, B por 1, e así ata Z que será 22, diremos que o alfabeto está en $\mathbb{Z}_{23}$, e obterase o cifrado c como $c = l + 3 \mod 23$ sendo l a letra a cifrar.

Exemplo 1.4. Dada a palabra CINE obtense facilmente o cifrado con César como se mostra na seguinte Táboa 1.3:

Mensaxe clara	C	I	N	E
l	2	8	11	4
$c = l + 3$	5	11	14	7
Criptotexto	F	N	P	H

Táboa 1.3: Cifrado de César da palabra CINE.

No método de encriptación da cifra de César, os alfabetos de orixe e de cifrado son o mesmo. Cada letra da mensaxe ocúltase empregando outra letra distinta deste alfabeto, pero sempre é a mesma outra letra. Clasifícase entón como un **cifrado de substitución monoalfabético**.

Encriptar un escrito cambiando cada letra pola súa respectiva grafía cifrada non é unha técnica segura, pero este feito non se descobre ata mediados do século IX coa publicación de Al-Kindi, un filósofo de Bagdad que se interesou polas matemáticas, entre outras ciencias. Nela explícase como o estudo da análise de frecuencias das letras en cada linguaxe permite romper os cifrados de substitución monoalfabéticos.

A **análise de frecuencias** consiste en analizar en cada lingua cales son as letras que máis se repiten. Unha vez coñecida esa información, será posible desencriptar un texto cifrado facendo corresponder os caracteres que máis aparecen neste, coas letras máis frecuentes no idioma co que se traballe.

É lóxico ver que se coñecemos a linguaxe do texto claro ou plano, que é como chamamos a mensaxe orixinal que se quere ocultar, dará igual o carácter que se empregue para cifrar

cada unha das grafías, pois ben, sexa unha letra doutro alfabeto, un símbolo ou calquera outro elemento, o procedemento de ataque non se modifica, seguirá a ser suficiente identificar a compoñente que máis se repite no cifrado para ir facéndoa corresponder coa letra máis empregada na lingua.

Para evitar este ataque non basta entón con valerse dun alfabeto distinto ao da mensaxe en claro para encriptar, senón que será necesario utilizar varios deles nun mesmo proceso, é dicir, non asociar a cada letra un cifrado fixo, aí está o erro que facilita a rotura do encriptamento. Este fallo é descuberto no século XV polo erudito **Alberti**, cando, dende o Vaticano, lle piden idear un novo método para ocultar mensaxes, e deu lugar a construción de **técnicas de substitución polialfabética**. Nestes cifrados, distintas letras do texto plano poden ocultarse polo mesmo carácter, así como unha mesma letra se encriptará de formas diferentes, facendo que a frecuencia das grafías no texto encriptado non se corresponda coa realidade.

Como o método de cifrado que emprega Alberti xa non é fixo, ideou un mecanismo para facilitar o proceso de encriptación. Consiste en dous discos concéntricos, un máis pequeno, e móbil, sobre outro, representados na Figura 1.2. As letras do alfabeto empregado no texto claro escríbense en minúscula no pequeno. En maiúscula están escritos, no disco base, os caracteres que se utilizan para ir cifrando a mensaxe. A posición inicial destes será indicada ao comezo do criptotexto nomeando un par de letras que se enfrontarán, por exemplo na imaxe enfróntanse A e g, logo o texto encriptado podería comezar con gA –a letra g da mensaxe encriptouse, nun comezo, por un A. No momento en que volva atoparse outra maiúscula no cifrado modificarase a posición do disco interior enfrontando o g con ela. Cambiar a disposición dos discos significa empregar un novo alfabeto para cifrar.

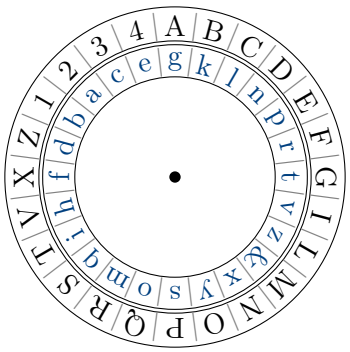

Figura 1.2: Disco de Alberti.

Exemplo 1.5. Unha posible encriptación, empregando a cifra de Alberti, da mensaxe XA PASOU UN MES, sería a seguinte: gAN2E2PQNNDR4P, onde non se teñen en conta os espazos, e ademais, as letras que non aparecen no alfabeto en claro de Alberti (como o u) considéranse xes. Neste exemplo estase a empregar a disposición dos discos da imaxe, non obstante, outra opción modificando esta, podría ser: gAN2E2PQNmTPFTBR. É dicir, a disposición inicial dos discos é a da imaxe, enfrontando gA, con ela cifamos XA PASOU; ata que nos atopamos con mT e se modifica a posición do disco para cifrar UN MES.

Outro encriptado de substitución polialfabético é o publicado por **Vigènere** na súa obra *Traicté des chiffres* a finais do século XVI. Xorde co mesmo obxectivo que Alberti, o de resistir o ataque por análise de frecuencias. A xa mencionada clasificación deste método deixa claro que tamén empregará máis de un alfabeto cifrado.

Carácter da mensaxe

Carácter da clave	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
A	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
B	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A
C	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B
D	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C
E	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D
F	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E
G	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F
H	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G
I	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H
L	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I
M	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L
N	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M
Ñ	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N
O	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ
P	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O
Q	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P
R	R	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q
S	S	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R
T	T	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S
U	U	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T
V	V	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U
X	X	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V
Z	Z	A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X

Táboa 1.4: *Tabula* de Tritemio.

O procedemento empregado por Vigenère para ocultar a información pode describirse empregando a Táboa 1.4, elaborada un século antes por Tritemio, que compila o alfabeto galego ordenado de forma usual, e todos os posibles alfabetos cifrados que poden obterse desprazando o primeiro. No seguinte exemplo veremos como.

Exemplo 1.6. Vexamos como se encripta a mensaxe ESTUDANTE-DE-CIENCIAS-DA-USC empregando o cifrado de Vigénere, e tomando como palabra secreta LIBRO:

Rómpese o texto claro, no que non consideraremos os espazos en branco, en subtextos de lonxitude igual que a da clave, 5, se a lonxitude da mensaxe non é múltiplo desta, ao último subtexto engadiránselle xes ata que teña igual lonxitude que os demáis: ESTUD ANTED ECIEN CIASD AUSCX. Unha vez definidos estes subgrupos, basta con detallar como se encriptaría o primeiro, pois en todos se seguirá o mesmo procedemento.
Cada unha das letras da clave, c_i, indica cal é o alfabeto cifrado que hai de empregarse para encriptar cada unha das letras dos subtextos, l_i. Ditos alfabetos cifrados serán os organizados de tal xeito que comencen pola letra de dita clave; se esta é LIBRO, na primeira letra, de cada un dos grupos de texto, empregarase para cifrar o alfabeto que comece por $c_1 = \mathrm{L}$, na segunda letra o que comece por $c_2 = \mathrm{I}$, deste xeito chegaranse a empregar ata 5 alfabetos. O paso final será tan sinxelo como atopar, en dito alfabeto cifrado, o carácter que ocupe a mesma columna que a letra do texto claro no alfabeto orixinal (o ordenado de forma usual). É dicir, para cifrar a primeira E, que está na quinta posición do alfabeto orixinal, buscarase no alfabeto cifrado, neste caso o que comeza por L, a letra que estea nesa mesma columna: O. Repetindo este proceso en cada letra, de cada subgrupo e texto, obtense a mensaxe cifrada, como pode observarse na Táboa 1.5.

Mensaxe clara	E	S	T	U	D	A	N	T	E	D	E	C	I	E	N	C	I	A	S	D	A	U	S	C	X
Clave	L	I	B	R	O	L	I	B	R	O	L	I	B	R	O	L	I	B	R	O	L	I	B	R	O
Texto cifrado	O	C	U	Ñ	R	L	U	U	V	R	O	M	L	V	B	N	R	B	M	R	L	E	T	T	N

Táboa 1.5: Exemplo do cifrado de Vigenère.

Noutras palabras, a técnica de Vigenère consiste en ir aplicando distintos cifrados de César ás letras da mensaxe. Neste método a clave non será un só díxito, senón que será unha palabra, ou ata unha frase, de tamaño t. Cada un dos caracteres desta clave definirán o número secreto de César que se empregará para cifrar cada un dos grafemas da mensaxe. Ditos números virán dados polas posicións que ocupen cada unha das letras da clave no alfabeto –se a palabra secreta é ABC, cifrarase a primeira letra do comunicado (e a primeira letra de todos os subtextos) empregando César con $k_1 = 0$, a segunda con $k_2 = 1$, ... Así, dividirase o texto claro en grupos de lonxitude t, facendo corresponder cada unha das letras de ditos conxuntos, l_i, coas da clave, c_i, sendo esta última a que indica o número a empregar no cifrado de César, como xa se explicou.

Empregando aritmética modular, e traballando co alfabeto galego de 23 letras, definimos o algoritmo de encriptación, de cada subgrupo de texto, como segue:

$$(c_1, \ldots, c_t) \coloneqq l_1 + k_1, \ldots, l_t + k_t \mod 23$$

onde $l_1 + k_1, \ldots, l_t + k_t \mod 23$ denota $(l_1 + k_1 \mod 23, \ldots, l_t + k_t \mod 23)$.

Cando falamos de como quebrantar un cifrado, referímonos a estudar as técnicas que poden empregarse para descubrir o texto en clave, obtido con dito cifrado. Vimos que a cifra de Vigènere xorde como resposta ao ataque por análise de frecuencias, de feito, durante centurias foi chamada a *chiffre indéchiffrable* ata que finalmente no século XIX se idea un método para rompela [9].

O punto débil da cifra de Vigenère está na lonxitude da clave empregada, no momento en que esta se descobre, o encriptado deixa de ser seguro. Unha vez se sabe que o tamaño da palabra secreta é t, é dicir, coñécese precisamente o tamaño dos subtextos que se empregan para cifrar a mensaxe, dedúcese que todas as grafías congruentes con $1 \mod t$ se ocultan coa mesma clave de César, k_1; así como as congruentes con $2 \mod t$, e demais. Unha vez atopados os elementos que se cifren con claves iguais, bastará con agrupalos e logo empregar a análise de frecuencias para desencriptar a mensaxe.

O **método de Kasiski**, publicado no 1863, é a primeira intuición que xorde para determinar a lonxitude da clave. A idea baséase en buscar no texto cifrado cadeas de 3 caracteres que se repitan, e conxecturar que veñen dadas de encriptar a mesma mensaxe coa mesma clave (pois menos probable é obter a mesma cadea de texto cifrado se vén dada de mensaxes distintas cifradas con claves distintas). Así, Kasiski chega á conclusión, empregando o seu método homónimo, de que a distancia entre as cadeas será un múltiplo do buscado tamaño da clave.

Outro método máis preciso, para romper Vigenère, é desenvolvido polo matemático Friedman na década dos 1920, empregando o que chamará índice de coincidencia. O **índice de coincidencia** (I_c) defínese como a probabilidade de que dous elementos de un texto, tomados aleatoriamente, coincidan. É claro ver que, se o número de veces que aparece cada carácter nunha mensaxe se aproxima a unha distribución uniforme o I_c será menor que se dita mensaxe contén só unha letra repetida constantemente.

Pode deducirse a fórmula para obter o índice de coincidencia de un texto cifrado como segue:
consideremos un texto de lonxitude n, no que se emprega un alfabeto con r caracteres, $\sigma = a_1, a_2, \ldots, a_r$; denotaremos por $f_1, f_2, \ldots, f_r$ o número de veces que cada elemento do alfabeto (cada a_i) aparece no texto.

Dado que a_1 se atopa f_1 veces no texto, a probabilidade de que tomadas ao azar dúas grafías de dito texto, ambas sexan a_1 virá dada polo binomio $\binom{f_1}{2}$. Por tanto, a probabilidade de que se repitan calquera dous caracteres aleatorios será

$$\binom{f_1}{2} + \binom{f_2}{2} + \cdots + \binom{f_r}{2} = \sum_{i=1}^{r} \binom{f_i}{2}$$

E dita probabilidade nun texto de lonxitude n redúcese a $\frac{\sum_{i=1}^{r}\binom{f_i}{2}}{\binom{n}{2}}$.

Desenvolvendo a expresión do coeficiente binomial obtense:

$$I_c(texto) = \frac{\sum_{i=1}^{r}\binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=1}^{r} f_i(f_i-1)}{n(n-1)} = \frac{\sum_{i=1}^{r}(f_i^2 - f_i)}{n^2-n}.$$

E tendo en conta que $\sum_{i=1}^{r} f_i = n$,

$$I_c(texto) = \frac{\sum_{i=1}^{r} f_i^2 - n}{n^2-n}.$$

Esta fórmula verase modificada no caso de traballar en contextos que aporten nova información, como pode ser identificar o alfabeto cun idioma, ou estudar un texto onde a probabilidade de que apareza un carácter ou outro é a mesma (un texto aleatorio):

$$I_c(idioma) = \sum_{i=1}^{r} \left(\frac{f_i}{n}\right)^2;$$
$$I_c(aleatorio) = \frac{n}{f_i}.$$

É claro que o índice de coincidencia da mensaxe clara coincide co do seu cifrado, por tanto, para descubrir a lonxitude da clave bastará con conxecturar distintos tamaños desta, t, dividir a mensaxe subtextos dise tamaño t e calcular a media dos I_c de cada un deles. No momento en que esta se aproxime á do idioma, a conxectura estará ben feita, se pola contra se aproxima ao dun texto aleatorio, a suposición sobre t será incorrecta.

Dende a súa orixe, no século V a.C., a criptografía estivo estreitamente ligada ao mundo militar ata chegar a ser unha das armas máis poderosas nas guerras. Un claro e popular exemplo é a **máquina Enigma** empregada polos exércitos alemáns para comunicarse durante a II Guerra Mundial [10]. A idea de empregar máquinas que faciliten o proceso de encriptación xorde xa no 1918. Nese ano créanse as primeiras, de rotores, artificios para cifrar que empregan discos móbiles que van modificando a súa posición durante o cifrado. Enigma é unha destas, que dende que foi patentada no 1923 sufriu actualizacións ata converterse no que podemos observar na Figura 1.3.

O aparato aseméllase a unha máquina de escribir: conta cun teclado no que se escribirá (letra a letra) o comunicado que quere ocultarse, e cun panel luminoso, que se iluminará indicando cal é a letra cifrada correspondente a cada unha da mensaxe. Dito cifrado virá dado pola posición dos discos ou rotores, sendo a colocación inicial destes a clave secreta. O movemento dos discos é tal que o primeiro avanza unha posición por cada tecleado, e no momento en que este dá unha volta completa, é cando avanza un posto o segundo, e así sucesivamente. Ademais, introduciuse un panel de conexións no que, antes de comezar o encriptado, se indicaban pares de letras que se intercambiarían.

Figura 1.3: Máquina Enigma.
(Fonte: Bob Lord's Crypto Museum)

Enigma tamén emprega un cifrado de substitución polialfabético pero o gran tamaño do espazo de claves, e o feito de empregar unha permutación distinta en cada letra, converten á máquina en potencialmente inquebrantable. Creáronse potentes enxeños para atacar Enigma, pero ningún foi o suficientemente rápido e efectivo ata que Alan Turing constrúe a coñecida como maquina de Turing, e tras o estudo de numerosos textos cifrados interceptados se descobre que as mensaxes que envían os alemáns comezan sempre coa mesma frase. Esta información adicional permitiu despois de moito esforzo e intentos, no 1942, descifrar Enigma.

Outro método de encriptación que pode considerarse clásico xorde no 1929, a **cifra de Hill**. Hill destaca por ser o primeiro cifrado de substitución por bloques.

Definición 1.7 (Cifrado por bloques)**.** O **cifrado por bloques** baséase na idea de dividir o texto en bloques dunha determinada lonxitude, e encriptar cada un dos bloques por separado, como se foran novos textos independentes, e empregando en cada un deles a mesma técnica de cifrado.

Ademais cómpre adicar unhas liñas a explicar este cifrado que servirá de introdución aos conceptos de difusión e confusión, dúas interesantes propiedades que posúe. Podemos describir o cifrado empregando como alfabeto o do galego, traballando en aritmética modular, $\mathbb{Z}_{23}$; neste caso tamén se dividirá a mensaxe en bloques, que serán de tamaño t, por tanto o criptotexto virá dado como un elemento de $\mathbb{Z}_{23}^t$. O algoritmo de encriptación basearase en permutacións dadas pola seguinte aplicación linear bixectiva:

$$\begin{aligned} f\colon \mathbb{Z}_{23}^t &\longrightarrow \mathbb{Z}_{23}^t \\ m &\longmapsto m \times A \end{aligned}$$

donde m é a mensaxe a cifrar é $A \in \mathcal{M}_{t\times t}(\mathbb{Z}_{23})$ a clave expresada en forma de matriz. A será unha matriz invertible.

Observación 1.8. Cando falamos dunha matriz A invertible con coeficientes nun corpo, referímonos a aquela cadrada que verifique $\det(A) \neq 0$. Non obstante, cabe mencionar que cando non traballamos sobre un corpo, unha matriz $A \in \mathcal{M}(\mathbb{Z}_q)$ é invertible se $\text{m.c.d}(\det(A), q) = 1$.

O procedemento de cifrado entenderase facilmente co Exemplo 1.9.

Exemplo 1.9. Traballaremos co alfabeto galego en aritmética modular $\mathbb{Z}_{23}$ para encriptar a mensaxe MATEMÁTICAS. Consideraremos bloques de lonxitude 3 (completaremos o último bloque para que teña este tamaño cun X) e a seguinte clave

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Táboa 1.6: Correspondencia entre as letras do alfabeto e a súa posición.

Facendo corresponder cada unha das letras do alfabeto co lugar que ocupan neste, como se amosa na Táboa 1.6, podemos reescribir a mensaxe como 10-0-18-4-10-0-10-8-2-0-17-21, para logo distribuír estes díxitos nunha matriz onde cada unha das filas sexa un bloque. Así o criptotexto virá dado polo produto de ambas matrices:

$$\begin{pmatrix} 10 & 0 & 18 \\ 4 & 10 & 0 \\ 10 & 8 & 2 \\ 0 & 17 & 21 \end{pmatrix} \times \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 0 \\ 1 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 28 & 66 & 68 \\ 74 & 102 & 20 \\ 68 & 106 & 52 \\ 140 & 195 & 21 \end{pmatrix} \underset{\text{en } \mathbb{Z}_{23}}{=} \begin{pmatrix} 5 & 20 & 22 \\ 5 & 10 & 20 \\ 22 & 14 & 6 \\ 2 & 11 & 21 \end{pmatrix} \longrightarrow \begin{pmatrix} F & V & Z \\ F & M & V \\ Z & P & G \\ C & N & X \end{pmatrix}$$

Por tanto, a mensaxe cifrada será FVZFMVZPGCNX.

Ao traballar co produto de matrices, a cifra de Hill presenta as seguintes propiedades: **difusión**, é dicir, a modificación de só un carácter no texto plano repercute en máis de un carácter do cifrado, como se mostra en azul; e **confusión**, cada elemento do criptotexto depende de varias cifras da clave.

Esta técnica de encriptación non se ve comprometida por descubrirse a lonxitude dos bloques en que se divide a mensaxe, o que romperá o cifrado será coñecer un par (m, c), onde m é un texto plano coñecido e c o seu cifrado. Bastará con atopar unha base nese texto, $m_1, \ldots, m_s$, e denotando por K á matriz na que se distribúe a clave, saberase que:

$$\begin{pmatrix} c_1 \\ \vdots \\ c_s \end{pmatrix} = \begin{pmatrix} m_1 K \\ \vdots \\ m_s K \end{pmatrix} = \begin{pmatrix} m_1 \\ \vdots \\ m_s \end{pmatrix} K \Longrightarrow K = \begin{pmatrix} m_1 \\ \vdots \\ m_s \end{pmatrix}^{-1} \cdot \begin{pmatrix} c_1 \\ \vdots \\ c_s \end{pmatrix}$$

Deste xeito obtense a clave do cifrado, K.

Capítulo 2

Criptografía moderna

O momento en que comeza a falarse de criptografía moderna non está ben definido. A introdución da computación nesta ciencia pode considerarse como un dos feitos máis relevantes entre os que marcan este antes e despois.

Mentres que a criptografía clásica manipulaba caracteres tradicionais, é dicir, letras e díxitos directamente –aínda que agora lle podemos dar unha interpretación aritmética, e estudala dende un punto de vista moderno, como fixemos–, a criptografía moderna opera en secuencias de bits e depende de algoritmos matemáticos coñecidos publicamente para codificar a información.

Foi **Claude Shannon**, un matemático e criptógrafo estadounidense, quen no ano 1948, publicou o tratado "Unha teoría matemática da comunicación" no que comeza a darlle un contexto matemático a criptografía ata enmarcala como unha rama da teoría da información, que xunto coa teoría de códigos e de compresión da información, estudia a transmisión desta a través dunha canle pola que se comunican un emisor e un receptor.

Destaca tamén nesta transición que, ata este momento, a criptografía estaba máis dirixida a usos militares e se empregaba unicamente no cifrado das comunicacións. Non obstante, na actualidade, faise uso dela no día a día, conta con infinidade de aplicacións, como no comercio ou na identificación e autenticación online, e é ata indispensable na seguridade vía internet.

A criptografía moderna xorde, por tanto, para facer fronte a novos obxectivos, que definiremos neste capítulo. Veremos tamén como co crecemento desta e a creación de algoritmos cada vez máis complexos, a criptoanálise, que presentaramos ata agora como

unha arte, pasa a considerarse unha ciencia. Finalmente, introduciremos os algoritmos modernos, que podemos clasificar nos simétricos –entre os que podemos atopar os que cifran en fluxo, e os que o fan en bloques– e nos asimétricos.

2.1. Obxectivos da criptografía

Inicialmente, a criptografía empregábase para compartir mensaxes secretas, queda entón claro que a privacidade e a confidencialidade eran dos primeiros obxectivos desta ciencia. Non obtante, na actualidade, debido especialmente ao crecemento das comunicacións vía internet, cabe destacar os seguintes [5]:

- **Autenticación**: evitar que unha terceira persoa poida suplantar a identidade da outra parte.
- **Integridade**: evitar que un terceiro poida modificar a información.
- **Non repudio**: evitar que calquera das dúas partes poida negar ter participado na comunicación.

Así, podemos definir a **criptografía** como a encargada de estudar a comunicación secreta garantindo a autenticación, integridade e non repudio.

O principio básico principal da criptografía clásica era o de "seguridade a través da ocultación". As técnicas empregadas para a encriptación mantíñanse en secreto e só as partes implicadas na comunicación sabían delas, polo tanto a confidencialidade precisa de todo o criptosistema. Na moderna, o secreto obtense a través dunha clave secreta que se usa como semente dos algoritmos. A dificultade computacional dos algoritmos, a ausencia de clave secreta, etc., fan imposible que un atacante obteña a información orixinal aínda que coñeza o algoritmo empregado para cifrar; só se require que as partes interesadas nunha comunicación segura posúan dita clave secreta.

A afirmación de que a seguridade dun criptosistema moderno reside na ocultación da clave que emprega, non no secreto do algoritmo, é á que xa chegou Auguste Kerckhoffs von Nieuwenfof na súa obra *La cryptographie militaire*, da que podemos deducir os coñecidos **Principios de Kerckhoffs** que se presentan a continuación, nas verbas de Andrea Sgarro e traducidos ao galego, e que poden atoparse no capítulo 8: *El principio de Auguste Kerckhoffs* de [15]:

- O sistema de cifrado debe ser impenetrable, se non na teoría, polo menos na práctica.
- No caso de que o sistema se vexa comprometido, os correspondentes deben quedar regardados de calquera sospeita.
- A clave debe ser sinxela de memorizar e, á vez, fácil de substituír.
- Tanto o instrumento, coma o utensilio cifrador, coma os documentos necesarios para o encriptado deben ser manexables para o seu transporte.
- É necesario e recomendable que a operación de cifrar a poida realizar unha soa persoa.
- O sistema debe ser comprensible, polo que non se debe basear no coñecemento de longas listas de normas nin requirir de esforzos mentais excesivos.

No desenvolvemento destes algoritmos públicos téñense en conta as posibles vías de ataque que un estudo por parte da comunidade poida desenvolver. De feito, o interese por analizar estas debilidades para reforzar os métodos de encriptado deu orixe a unha nova disciplina que se coñece co nome de criptoanálise e que traballa en paralelo coa criptografía, aínda que algunhas veces con fins diferentes.

2.2. Criptoanálise. Tipos de ataques

Definición 2.1 (Criptoanálise)**.** A **criptoanálise** pode considerarse como a disciplina que complementa á criptografía, pois encárgase de estudar como romper os cifrados. Podemos diferenciar dous tipos de criptoanálise: a pasiva, aquela que rompe o cifrado co único obxectivo de espiar a información; e a activa, que ademais modifica o texto interceptado.

Fálase de atacar un cifrado empregando *forza bruta* cando o que se fai é ir probando, un a un, os elementos do conxunto de claves posibles, tratando de atopar a que devolve unha mensaxe coherente. Diferéncianse catro tipos de posibles ataques para romper o cifrado [5]:

- Só criptotexto: o único ao que se ten acceso é a texto cifrado. É o ataque máis difícil.
- Texto plano coñecido: coñécense algúns fragmentos do texto plano e o seu cifrado, é dicir, coñécense pares (m, c).
- Texto plano elixido: no caso de ter acceso temporal a máquina de cifrar, o que nos permitirá facer conxecturas.
- Criptotexto coñecido: ter acceso temporal á máquina de descifrar.

Observación 2.2. A ciencia que engloba a criptografía e a criptoanálise recibe o nome de **criptoloxía**.

2.3. Clasificación dos algoritmos de criptografía

Dentro da criptografía moderna podemos falar da simétrica e da asimétrica, e clasificala como se mostra na Figura 2.1.

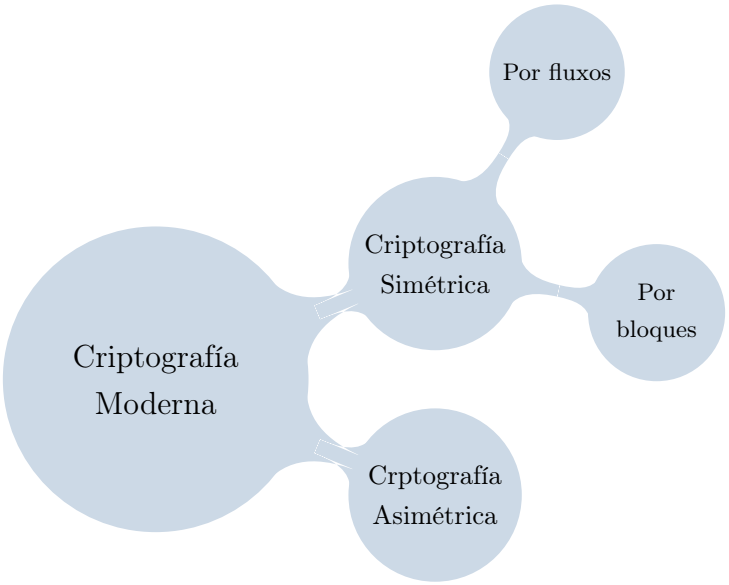

Figura 2.1: Clasificación da criptografía moderna.

Os **cifrados simétricos**, ou de clave secreta, empregan só unha clave que comparten un único emisor e receptor, tanto para cifrar como para descifrar. En canto aos inconvenientes desta criptografía destacan que require dunha canle segura pola que, previamente á comunicación, compartir dita clave; así como o feito de que nun sistema criptográfico de n usuarios empréganse $\binom{n}{2}$ canles totais, cada usuario debería memorizar $n-1$ claves, para garantir a integridade.

Nos **cifrados asimétricos**, ou de clave pública, cada usuario conta cun par de claves, unha pública, que se dá a coñecer, e outra privada, só coñecida polo usuario. Así, se o emisor desexa mandar unha mensaxe, empregará a clase pública do receptor para cifrala, e soamente este, coa súa clave privada, será capaz de desencriptala. Estes sistemas non requiren dunha canle segura e a única clave que hai de memorizar cada usuario é a súa privada. A dificultade neste caso débese á gran potencia de cálculo que necesitan.

Nas situacións reais combínanse ambos métodos facendo uso do que se coñecen como **cifrados híbridos**. Coa criptografía asimétrica créase unha canle segura que se emprega para enviar unha clave, que será a clave secreta que emprega o cifrado simétrico, e coa que se poderá empezar a traballar para axilizar a comunicación.

Observación 2.3. Tanto na criptografía simétrica como na asimétrica existe unha clave que o usuario terá que ocultar. Cando traballemos con esta primeira criptografía referirémonos a ela como clave secreta, mentres que no contexto da asimétrica chamarémoslle clave privada.

2.3.1. Introdución ao cifrado en fluxo e en bloques

Segundo o tratamento da mensaxe, podemos dividir os cifrados modernos simétricos nos que cifran por fluxo e nos que o fan por bloques [8]. Os asimétricos fano sempre da segunda maneira, que será na que nos centraremos e intentaremos explicar dun xeito máis detallado.

Os **cifrados por fluxo** cifran a mensaxe clara bit a bit. Parten da clave secreta, á que chamaremos semente, e que permitirá construír unha secuencia cifrante de aparencia pseudoaleatoria. Unha vez obtida a secuencia, o criptotexto virá dado de aplicar unha operación tipo XOR entre es bits desta e os da mensaxe. O esquema dun cifrado de fluxo pode esquematizarse como se mostra na Figura 2.2.

Observación 2.4. O operador XOR ou OR exclusivo, que se denota por $\oplus$, ten como saída 1 sempre e cando as entradas non coincidan, noutro caso será 0, é dicir, non é máis que unha suma (módulo 2).

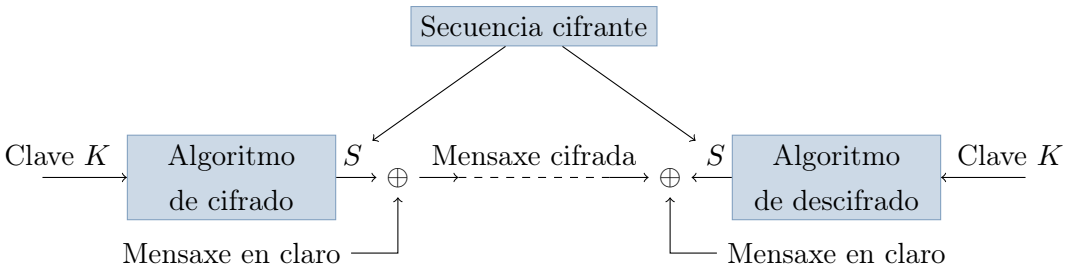

Figura 2.2: Esquema de cifrado por fluxo.

Coñécense como **cifrados en bloque** aqueles que dividen a mensaxe clara nun determinado número de bytes, antes de proceder co cifrado. En canto ao tamaño dos bloques, búscase que faga fronte ao ataque por análise de frecuencias. Nos primeiros cifrados, como IDEA ou DES, traballabase con bloques de 8 bytes, e xa nos máis actuais, como o AES, 16 bytes. Se ao formar bloques faltan bits, para que o último teña o mesmo tamaño engádese un conxunto deles facendo uso de esquemas de recheo (*padding schemes*).

Unha estructura xeral dos pasos que se levan a cabo en cada un dos bloques da mensaxe para obter cada un dos bloques de criptotexto é a seguinte.

1. Transformación inicial: Non todos os cifrados simétricos en bloque contan con esta etapa, dependerá do algoritmo. Nela realízanse permutacións ou transposicións de bits co obxectivo de modificar a posición de letras para evitar así a aparición de repetitivos grupos de texto, como poden ser os artigos ou a palabra *que*.

2. Algoritmo de expansión de clave: dando lugar a un conxunto de novas subclaves distintas que serán as que se empreguen, para cifrar o texto, en cada unha das roldas definidas na seguinte etapa.

3. Roldas do algoritmo: consiste nun determinado número de operación que se lle realizan ao bloque –ou unha parte del– repetidamente. O algoritmo contará con tantas roldas como número de veces se repiten esas mesmas operacións.

Así, a mensaxe encriptada virá dada de concatenar os bloques de criptotexto obtidos, ou ben, de cifrar de forma independente cada un nos que inicialmente se dividiu dita mensaxe; ou ben, de combinar dalgunha maneira a saída obtida de cifrar un bloque coas entradas para cifrar outros. Estes distintos xeitos de obter o texto completo encriptado coñécense como **modos de cifra**.

A continuación explicaranse un par de modos de cifra para entender como funcionan e a mellora que poden chegar a aportar, ou non, á seguridade de todo o criptosistema.

O modo de cifra **ECB** (Electronic Code Book) é o máis sixelo, cífranse todos os bloques facendo uso da mesma clave secreta. Procura non empregarse xa que é vulnerable a ataques por repeticións de bloques elixidos e/ou modificados, entre outros.

O modo de cifra **CBC** (Cipher Block Chaining) é un dos máis utilizados. Neste caso existe unha segunda clave, chamada vector inicial IV, e antes de comezar o cifrado realízase

un XOR entre este vector e o primeiro bloque de texto en claro. Ao resultado disto é ao que se lle aplica a clave secreta e, logo de recorrer os tres pasos definidos anteriormente, dá lugar ao primeiro bloque de criptotexto, que será o que se empregue como vector no novo bloque de texto a cifrar. A Figura 2.3 ilustra o proceder deste modo de cifra. Unha modificación nalgún dos bloques do texto en claro verase reflectida en dous do criptotexto, polo que este modo evita que un atacante reordene, elimine ou inserte un bloque á mensaxe.

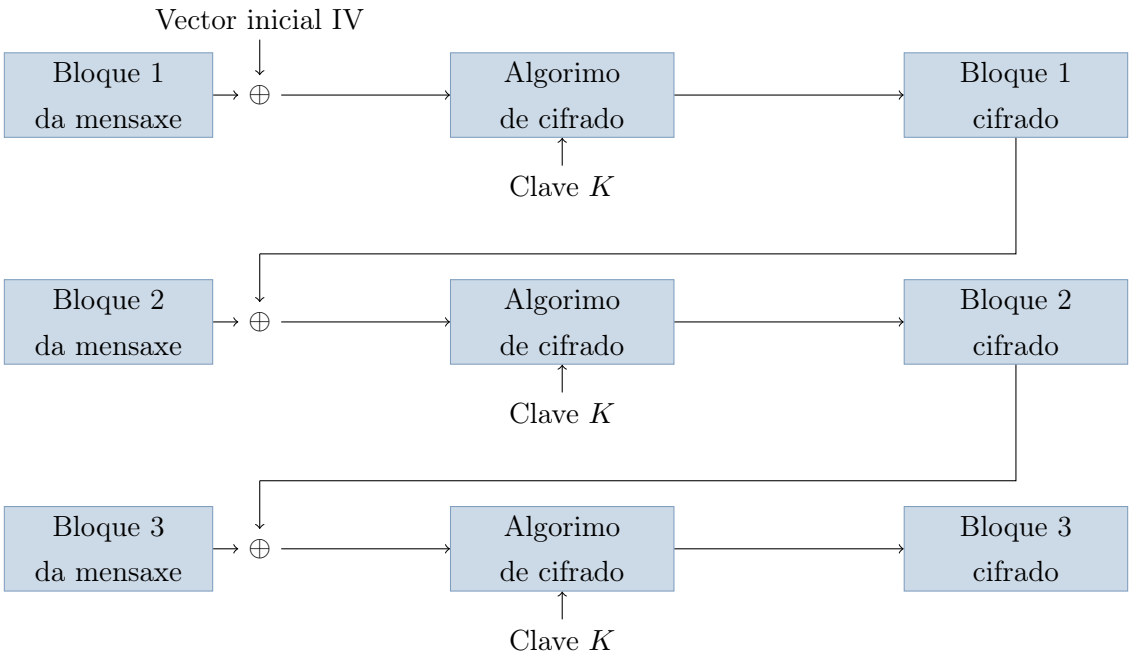

Figura 2.3: Esquema do modo de cifra CBC.

Capítulo 3

Criptografía simétrica

A criptografía simétrica, ou de clave secreta, é a primeira forma de encriptar mensaxes coa que xorde a criptografía moderna. Baséase na utilización dunha única clave, que compartirán e ocultarán emisor e receptor, e será coa que se cifre e descifre.

Neste capítulo falarase da cifra de Vernam, por poder considerarse un dos primeiros métodos de encriptación simétricos, e estudaranse os dous algoritmos de cifrado simétrico en bloques máis importantes, DES e AES. Remataremos falando das vantaxes desta criptografía, que a fan indispensable aínda a día de hoxe, así como das súas desvantaxes, que fomentarán a aparición de novas formas de encriptar.

3.1. Vernam

No 1917, Gilbert Standford Vernam, enxeñeiro dos coñecidos laboratorios Bell, posteriomente convertidos na multinacional estadounidense AT&T, Inc. (American Telephone & Telegraph), a compañía de telecomunicacións máis grande do mundo; xunto con Joseph Oswald Mauborgne, capitán no Corpo de Sinais do Exército de EE.UU., idearon o sistema de cifrado coñecido co nome de Vernam.

O método xorde como resposta aos ataques que foran capaces de romper a cifra de Vigenère.[1] Por tanto, Vernam implementou o que se coñece como *clave dun só uso e infinitamente longa*, é dicir, a clave secreta que se emprega no cifrado será aleatoria, contará

[1]Explicárase no capítulo 1 que, unha vez coñecida a lonxitude da clave secreta –que é o tamaño dos subtextos en que se divide a mensaxe, para logo cifrala con Vigenère– dito encriptado deixa de ser seguro, pois esto significaría que se atoparon os elementos que se cifran con claves iguais. Bastaría con agrupalos para logo empregar a análise de frecuencias, e xa estaría o cifrado roto.

cunha lonxitude non inferior á da mensaxe clara e non se utilizará en máis dunha ocasión, por isto último, recibe tamén o nome de caderno de uso único.

Pode considerarse un dos primeiros cifrados modernos, pois, aínda que segue a empregar sinxelos algoritmos, xa fai uso da computación. Clasifícase ademais como un cifrado simétrico, xa que require dunha única clave secreta, que comparten emisor e receptor; e cifra por bloques, aínda que neste caso, como a lonxitude destes será igual a de toda a mensaxe, «existirá un único bloque», no que cada carácter se cifra cunha clave diferente, facendo imposible dito ataque por análise de frecuencias.

Décadas máis tarde, no 1940, Claude Shannon, presentado como o pai da teoría da información, demostrou que este método criptográfico verifica o que él mesmo definiu como a propiedade de segredo perfecto [5].

Propiedade. 3.1 (O segredo perfecto de Shannon)**.** *Un criptosistema conta co* ***segredo perfecto****, o que tamén se coñece como ser incondicionalmente seguro, se para calquera mensaxe m, e calquera texto cifrado c, $P(m \mid c) = P(m)$, onde P denota a probabilidade. É dicir, a probabilidade de que unha mensaxe sexa m, suposto que se interceptou o criptotexto c, é exactamente igual a probabilidade de que o texto plano sexa m, ou como Shannon explicaba, «despois de interceptar una cantidade determinada de material, o inimigo non está nunha posición máis vantaxosa ca antes».*

3.2. DES

O DES (*Data Encryption Standard*) é un método de cifrado simétrico de bloques que foi introducido nos anos setenta [16]. Ata entón, a criptografía, usada principalmente polos gobernos, era prácticamente sinónimo de secretismo, pero as novas necesidades comerciais animaron ao que hoxe é o NIST (*National Institute of Standard and Technology*) a facer un chamamento para crear un cifrado que puidese convertirse en estándar e facerse público, o que representaba o recoñecemento público dos principios de Kerckhoffs. Ao segundo intento, un grupo de enxeñeiros da IBM (*International Business Machines Corporation*, empresa multinacional de tecnoloxía) propuxeron unha cifra baseada nun algoritmo xa coñecido como *Lucifer*, un método iterativo de cifrado cunha función das que se coñecen como *funcións de Feistel* –nome do líder do equipo de IBM– e traballan seguindo o esquema que se presenta na Figura 3.1, ditas funcións irán quedando máis claras ao longo do capítulo. As iteracións con este tipo de funcións para pasar do bloque de texto claro ao de criptotexto favorecen especialmente a confusión e a difusión.

Nun comezo, DES só se empregaba en aplicacións financeiras pero máis tarde foi aceptado como algoritmo criptográfico por outras organizacións.

A proposta inicial usaba bloques de texto de 64 bits (8 bytes) e unha clave de 128 bits, dos cales 16 son de control, un por cada byte. A Axencia de Seguridade Nacional, NSA, introduciu algunhas modificacións, entre elas a redución da clave a 56 bits máis 8 de control.

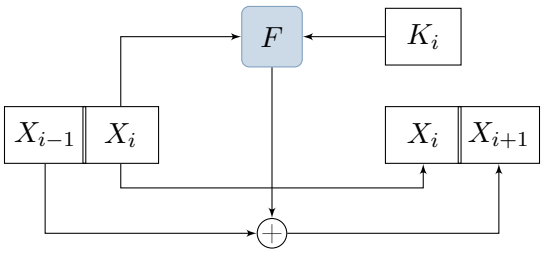

Figura 3.1: Esquema dunha función Feistel.

DES toma bloques de 64 bits de texto plano, usa unha clave de 64 bits –que, como xa se mencionou, son en realidade 56, máis 8 bits de control– e devolve un texto cifrado de 64 bits. O diagrama da Figura 3.2 ilustra o funcionamento deste algoritmo, que se pode describir brevemente, como segue.

Paso 1. Ao texto plano de 64 bits aplícaselle unha **permutación inicial** (PI) que reordena os bits para producir dous bloques permutados de 32 bits, E_0, a parte esquerda, e D_0, a parte dereita. Tamén é o momento en que se realiza a primeira modificación da clave orixinal, que pasa de 64 bits a 56 eliminando os de control, e tras o cal se lle realiza unha permutación e se divide en dous bloques de 28 bits cada un.

Paso 2. A continuación lévanse a cabo **16 roldas de cifrado DES** sobre os dous bloques do texto plano, E_{i-1} o esquerdo e D_{i-1} o dereito, coa clave de 56 bits, que se van transformando mediante unha función Feistel, F:

$$\begin{aligned} \mathrm{E}_i &= \mathrm{D}_{i-1} \\ \mathrm{D}_i &= \mathrm{E}_{i-1} \oplus F(\mathrm{D}_{i-1}, K_i) \qquad \text{con } i \geq 1 \end{aligned}$$

Paso 3. Remata o proceso aplicándolle a **permutación inversa** (PI^{-1}) da inicial, co único obxectivo de usar o mesmo algoritmo para descifrar (invertendo a orde e as transformacións da claves).

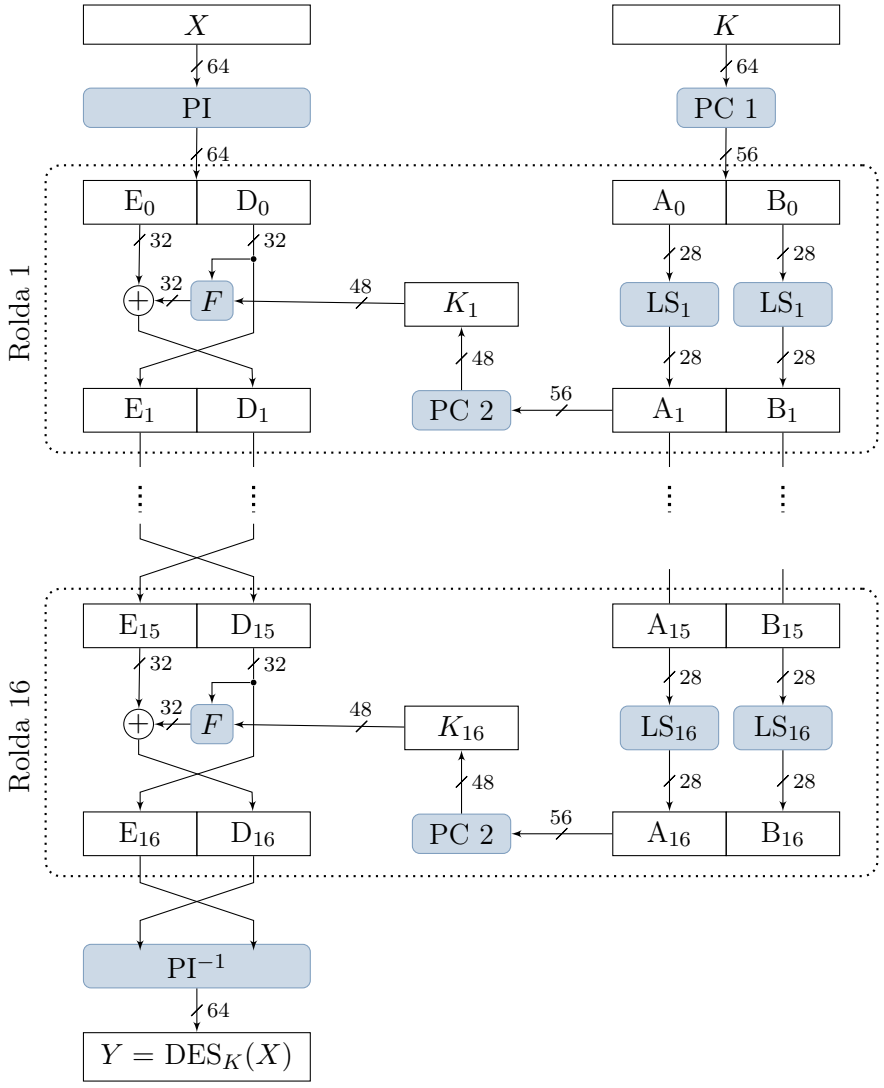

Figura 3.2: DES (Data Encryption Standard).

En cada rolda i de DES realízanse as mesmas operacións:

- **Transformación da clave**: A función de transformación da clave comeza ignorando os bits de control e realizando unha permutación dos restantes para formar dous bloques de 28 bits, que denotamos por A e B, é dicir, comeza tras a primeira modificación da clave (PC 1, primeira Permutación e Selección, Choice). En cada rolda a cada un destes bloques aplícaselle un desprazamento cíclico á esquerda (LS_i, *Left Shift*), por un ou dous bits, dependendo do número de rolda e, finalmente, faise unha selección permutada de 48 bits (PC 2) –24 de cada un dos dous bloques de clave. Así, cada unha das claves que se empregan para cifrar os bloques de texto é distinta.

- A **función de Feistel** opera sobre o bloque D_{i-1} de 32 bits do texto plano, como se mostra na Figura 3.3, realizando as funcións que se explican a continuación:

  Permutación de expansión (*Expansion Permutation*): o bloque de 32 bits pasará a ter 48, duplicando algúns dos bits.

  Caixas de substitución (S-caixas): divídense os 48 bits en 8 caixas de 6 bits cada unha. Cada caixa contará con 6 bits de entrada que convertirá, mediante unha transformación non linear, en 4 de saída.

  Caixas de permutación (P-caixas): reordénanse os 32 bits de saída.

- **XOR e intercambio de bloques**: ao bloque esquerdo E_{i-1} realízaselle un XOR coa saída da función de Feistel, o bloque dereito D_{i-1} e a clave K_i obtida para tal rolda. Isto dá lugar ao que, en principio, será o bloque esquerdo de texto da seguinte rolda, E_i. D_{i-1} non se modifica e tomarase como o bloque dereito, tamén en principio, da seguinte rolda. Finalmente lévase a cabo o intercambio destes obtendo así os que verdadeiramente serán os bloques esquerdo e dereito cos que se traballará na rolda seguinte, e que xa se definiran anteriormente:

$$\begin{aligned} \mathrm{E}_i &= \mathrm{D}_{i-1} \\ \mathrm{D}_i &= \mathrm{E}_{i-1} \oplus F(\mathrm{D}_{i-1}, K_i) \qquad \text{con } i \geq 1 \end{aligned}$$

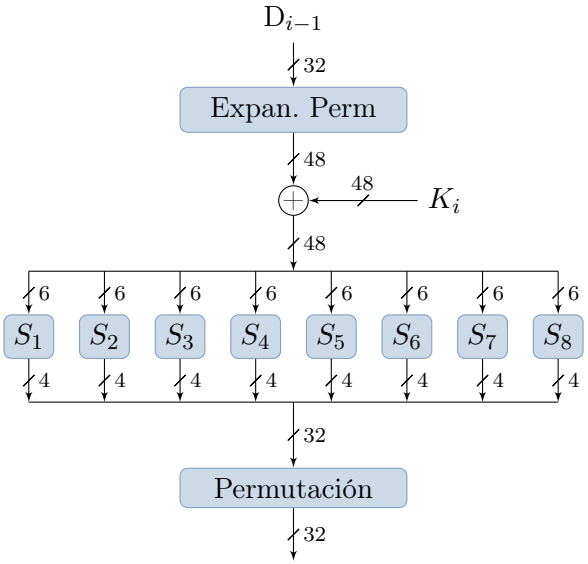

Figura 3.3: Esquema dunha rolda do DES.

As S-caixas, en realidade as 8 S-caixas seguidas dunha permutación que introducen difusión, é onde reside a fortaleza desta cifra e é o núcleo de DES en termos de forza criptográfica. Son o único elemento non linear do algoritmo e proporcionan confusión.

Unha descrición detallada das permutacións e transformacións que emprega este algoritmo pode verse no Apéndice A, onde se adxunta unha implementación en `SageMath` de S-DES, unha versión simplificada de DES, con bloques de 8 bits e só 2 roldas.

Malia que NIST publicou toda a especificación de DES en 1977, non se desvelou a motivación para a elección das táboas das S-caixas, o que deu lugar a especulacións sobre o risco, en particular con respecto á posible existencia dunha porta traseira secreta ou algunha outra debilidade construída intencionadamente, que podería ser explotada pola NSA. A pesar do intenso labor de criptoanálise levado a cabo pola comunidade científica, DES semella resistir ben os ataques analíticos, coma a *criptoanálise diferencial*, un método proposto por Eli Biham e Adi Shamir a finais dos anos 80 ou a *criptoanálise linear*, proposto en 1993 por Misuro Matsui. Tras saberse da criptoanálise diferencial, a NSA revelou os principios que motivaron a elección das S-caixas, ao tempo que o equipo da IBM afirmaba que eran coñecedores deste tipo de ataques e que DES fora deseñado para resistilos.

Non correu a mesma sorte fronte ós ataques de forza bruta, e o reto lanzado pola RSA Company en 1997 logrou que unha cooperación entre *DES cracker*, unha máquina especialmente deseñada para o ataque, e unha rede de máis de cen mil ordenadores foran quen de romper DES en menos de 24 horas, buscando a razón de $245 \cdot 10^9$ claves por segundo. Parece que o criterio de deseño para DES foi a súa eficiencia no hardware –definido polo *Oxford Languages* como o conxunto de elementos físicos ou materiais que constitúen unha computadora ou un sistema informático. Permutacións como E, P, IP e IP^{-1} son moi fáciles de implementar en hardware xa que non requiren lóxica, só cables. As S-caixas pequenas tamén son relativamente fáciles de realizar en hardware con portas lóxicas de Boole (AND, OR, XOR, entre outros).

Xa que o DES foi irrompible durante vinte anos e non mostrou debilidades no seu deseño tomouse a iniciativa de propoñer variantes que, ademáis de evitar o risco de ter que confiar en novos algoritmos, permiten aproveitar moitas das implementacións por hardware existentes do DES: o DES Múltiple, entre as que destaca o Tripe DES (3DES), consiste en aplicar varias veces o algoritmo DES con diferentes claves á mensaxe orixinal; o DES con Subclaves Independentes, que emprega subclaves diferentes e independentes en cada unha

das 16 roldas ou o DES Xeneralizado que, en lugar de utilizar dous bloques de 32 bits en cada rolda, como facía o DES, empregará n, permitindo así aumentar o tamaño do texto a cifrar como a lonxitude da clave.

3.3. AES

Á vista das debilidades de DES, en 1997 o NIST fixo unha nova chamada para crear un novo estándar de cifrado e, desta vez, ao contrario que no desenvolvemento DES, a selección do algoritmo sería un proceso aberto administrado polo propio instituto. En tres roldas posteriores de avaliación, NIST e a comunidade científica internacional discutiron as vantaxes e desvantaxes das propostas enviadas e foron rexeitando aos posibles candidatos, ata que en 2001 o cifrado de bloques chamado *Rijndael* con bloques de 128 bits foi declarado como o novo AES (*Advanced Encryption Standard*) [8].

Rijndael foi deseñado polos criptógrafos belgas, Joan Daemen e Vincent Rijmen e, en contraste co DES, non ten unha estrutura Feistel. Como vimos no apartado anterior e representamos na Figura 3.2, cando falamos de DES, as funcións Feistel non cifran un bloque enteiro en cada rolda, senón a metade –por exemplo, en DES, cífranse $64/2 = 32$ bits. Pola contra, AES cifra os 128 bits en cada iteración e esta é unha das razóns polas que o número de roldas é relativamente pequeno. AES consta das chamadas **capas** e cada unha delas manipula os 128 bits do bloque.

1. **A capa de *substitución de bytes***, unha S-caixa que transforma de xeito non linear cada elemento co fin de introducir *confusión*.
2. **A capa de *difusión***, que en realidade son dúas subcapas en todas as roldas, agás a última que só usa a primeira para que o descifrado resulte simétrico ao cifrado:
   a) A subcapa de *desprazamento de filas*.
   b) A subcapa de *mestura de columnas*.
3. **A capa de *suma da clave* de rolda**.

Na Figura 3.4, móstrase o esquema para a versión de AES con clave de 128 bits, que usa 10 roldas. Hai versións para claves de 192 e 256 bits, con 12 e 14 roldas, respectivamente. Para entender a nomenclatura, convén visualizar os 128 bits formando nunha matriz 4×4, na que os 16 bytes se reparten, por orde, nas 4 columnas.

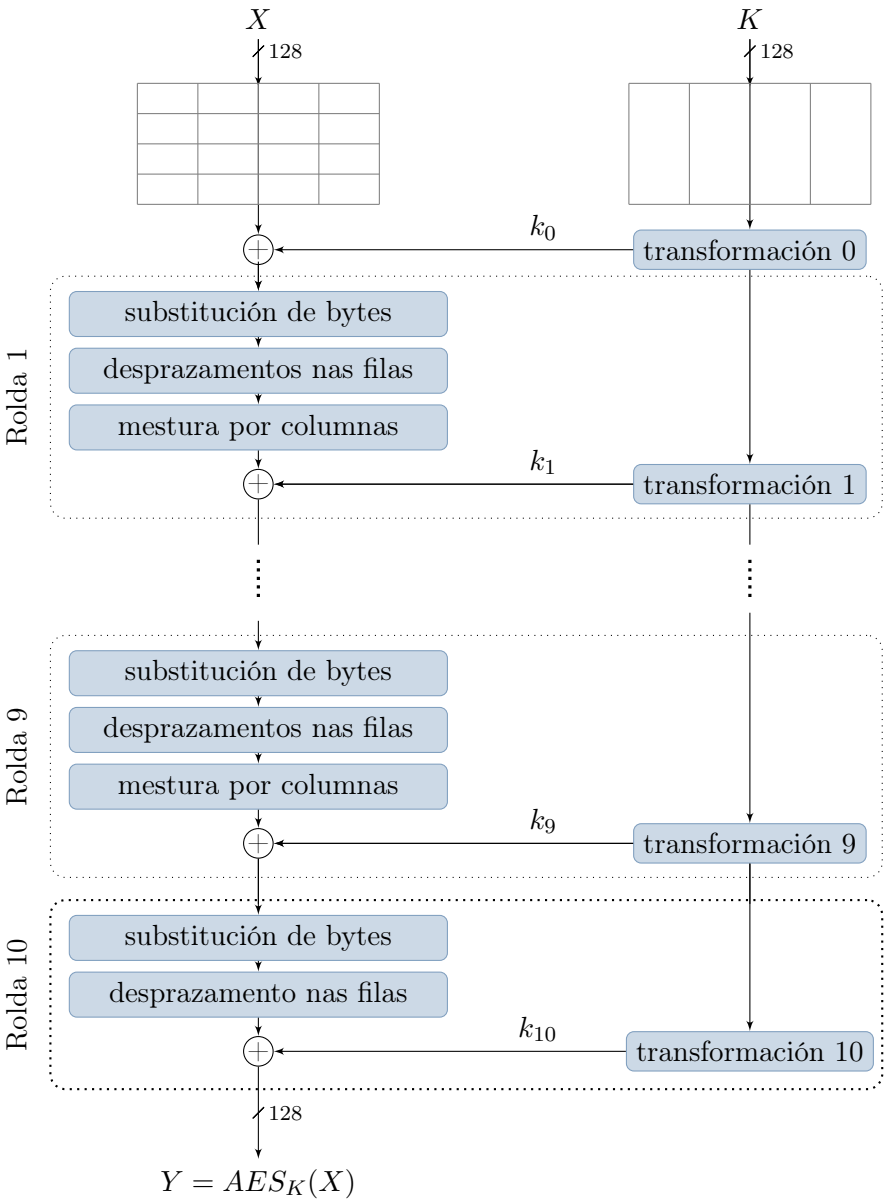

Figura 3.4: AES (Avanced Encryption Standard).

Unha descrición máis minuciosa destas capas móstrase a continuación.

Substitución de bytes

Consiste nunha batería de 16 S-caixas idénticas que constitúen a única parte non linear do algoritmo. A diferenza das S-caixas de DES, que son esencialmente táboas aleatorias que cumpren certas propiedades, as S-caixas do AES teñen unha forte estrutura alxébrica. Pensando cada byte, A_i, coma un elemento do corpo finito $\mathbb{F}_{2^8}$, a S-caixa AES consiste

en aplicarlle unha transformación afín ao inverso do elemento A_i (no caso de ser A_i o 0, considérase como inverso o propio 0).

$$S(A_i) = (T \circ I)(A_i),$$

onde $I : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ está definida por

$$I(a) = \begin{cases} a^{-1} & se \quad a \neq 0, \\ \\ 0 & se \quad a = 0, \end{cases}$$

e $T\colon \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ está definida como

$$T(b) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \end{pmatrix} \oplus \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

Desprazamentos nas filas

Numerando as filas dende 0 ata 3, os bits de cada fila desprázanse circularmente cara á esquerda unha cantidade de posicións igual ao índice da fila como se mostra na Figura 3.5, isto é, se $(B_{i,0}, B_{i,1}, B_{i,2}, B_{i,3})$ son os bytes da fila i-ésima, e $(B'_{i,0}, B'_{i,1}, B'_{i,2}, B'_{i,3})$ os resultantes da transformación de desprazamentos nas filas, teremos que

$$B'_{i,j} = B_{i,j+i} \mod 4$$

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

$\longrightarrow$

B_0	B_4	B_8	B_{12}	$\longleftarrow$ 0 posicións
B_5	B_9	B_{13}	B_1	$\longleftarrow$ 1 posición
B_{10}	B_{14}	B_2	B_6	$\longleftarrow$ 2 posicións
B_{15}	B_3	B_7	B_{11}	$\longleftarrow$ 3 posicións

Figura 3.5: Desprazamentos nas filas (ShiftRows).

Mestura nas columnas

A mestura nas columnas lévase a cabo mediante unha transformación linear que combina os bytes de cada columna da matriz de estado. Para iso, interpretamos cada columna C_i coma un vector do $\mathbb{F}_{2^8}$-espazo vectorial $(\mathbb{F}_{2^8})^4$ e usamos unha matriz, a mesma para todas as columnas, $M \in \mathcal{M}_{4\times 4}(\mathbb{F}_{2^8})$, e facemos $M \cdot C_i$, cos produtos e sumas en $\mathbb{F}_{2^8}$.

No Apéndice B daremos a implementación en `SageMath` do algoritmo Mini-AES, unha versión reducida con bloques de 16 bits (4 cuartetos), clave de 16 bits e 2 roldas. Faremos explícita a identificación do corpo finito $\mathbb{F}_{2^4}$ a través de

$$\mathbb{F}_{2^4} \longrightarrow (\mathbb{F}_2)^4 \longrightarrow \mathbb{F}_2[x]/_{\langle x^4+x+1\rangle},$$

onde a primeira é simplemente unha enumeración dos 16 elementos de $\mathbb{F}_{2^4}$ mediante as cifras en base 2 (completadas ata 4) dos números 0 a 15. A segunda asócialle a cada polinomio a súa clase módulo o ideal maximal xenerado por x^4+x+1 (polinomio irreducible), cun representante canónico, un polinomio de grao menor ou igual que 3.

Dado que cada byte de entrada inflúe en catro bytes de saída, a operación de mestura nas columnas é o principal elemento de difusión en AES. A combinación das transformación de desprazamentos nas filas e de mestura as columnas fai posible que despois de só tres roldas cada byte da matriz de estado dependa de todos os 16 bytes de texto plano.

Aínda que nas implementacións de AES as transformacións fanse por medio de táboas de busca, daremos a continuación a matriz M, onde con 1 denotamos o byte que lle corresponde ao polinomio 1, con 2 o correspondente ao polinomio x, e con 3 o correspondente ao polinomio $x+1$, isto é, `00000001`, `00000010` e `00000011`, respectivamente.

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Expansión da clave

Xa para rematar, quédanos describir como, a partir dos 128 bits (16 bytes) iniciais da clave se van xerando as sucesivas claves de rolda que se usan na capa de suma destas. Posto que estas claves de rolda son as mesmas no cifrado de todos os bloques nos que se rompe o texto plano, AES xera os 11×128 bits (de aí o nome *expansión de claves*) coa novidade

de que, en vez de bytes, como unidade de transformación utiliza palabras de 32 bits (4 bytes), $W_0, \ldots, W_{43}$. Se denotamos con $K_1, \ldots, K_{15}$ os 16 bytes da claves, a formación das palabras faise en 11 grupos de 4, W_{i+j}, con $j = 0, \ldots, 3$ e $i = 0, \ldots, 10$, do seguinte xeito, que procura esquematizarse na Figura 3.6:

$$\begin{aligned} W_{0+j} &= K_{0+j}|K_{1+j}|K_{2+j}|K_{3+j} && j = 0, 1, 2, 3, \\ W_{4i} &= W_{4(i-1)} + g_i(W_{4i-1}), && i = 1, \ldots, 10, \\ W_{4i+j} &= W_{4(i-1)+j} + (W_{4i+j-1}) && j = 1, 2, 3; \quad i = 1, \ldots, 10. \end{aligned}$$

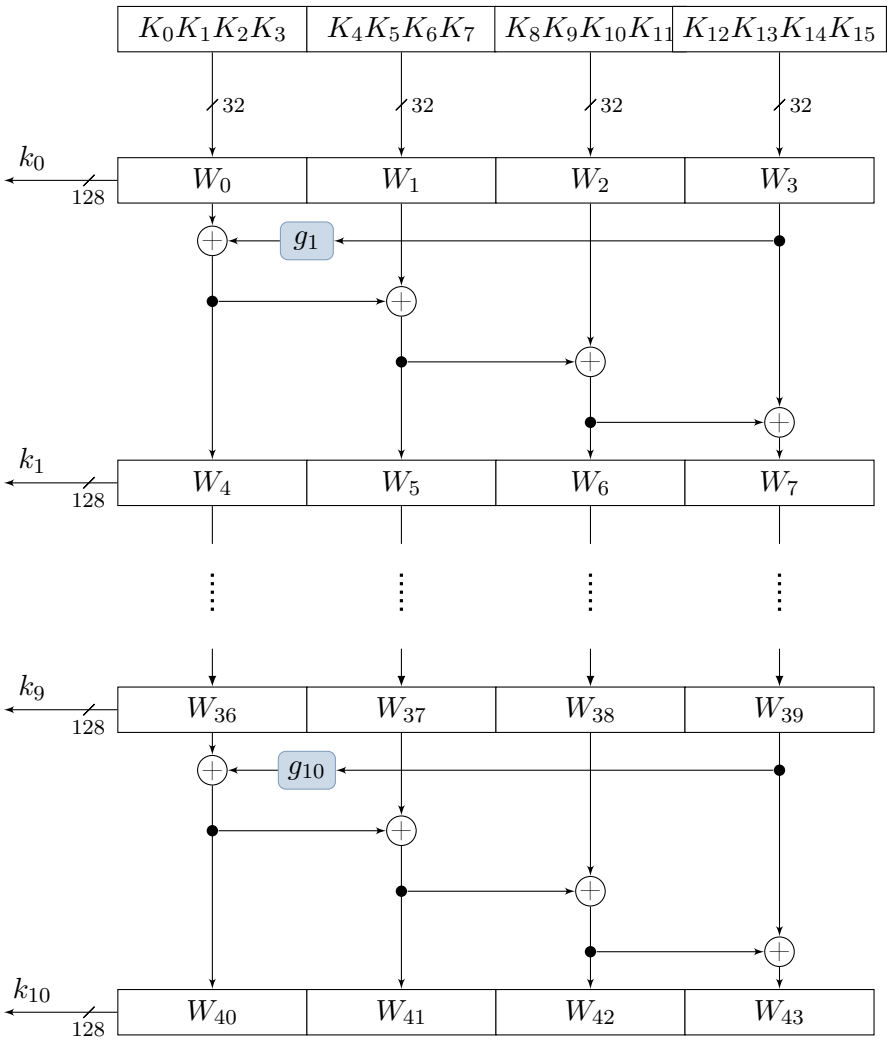

Figura 3.6: AES: Expansión da clave.

Na explicación anterior, g é unha función para engadir non linearidade á expansión da claves e que actúa da seguinte maneira –na Figura 3.7 detállase dun xeito máis visual o que se explica a continuación–:

1. despraza circularmente cara á esquerda unha posición cada un dos 4 bytes,
2. aplícalle unha S-caixa a cada byte (a mesma para os catro),
3. ao primeiro byte na rolda i-ésima engádelle o byte correspondente ao polinomio x^{i-1}, con $i = 1, \ldots, 10$.

É dicir, para $i = 1, \ldots, 10$,

$$g_i(B_0, B_1, B_2, B_3) = (x^{i-1} \oplus S(B_1), S(B_2), S(B_3), S(B_0))).$$

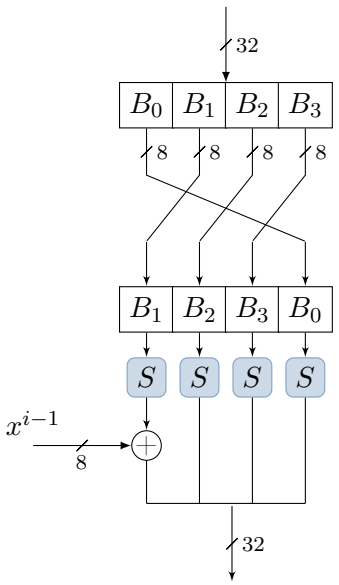

Figura 3.7: AES: Función g_i na expansión da claves.

A diferenza de DES, AES foi deseñado de xeito que é posible unha implementación eficiente en software. De feito, unha implementación sinxela de AES seguindo directamente a descrición aquí presentada, é axeitada para procesadores de 8 bits como os que se atopan nas tarxetas intelixentes, se ben non sería especialmente eficiente nas máquinas de 32 ou 64 bits, comúns nos ordenadores actuais: procesar 1 byte por instrución (todas as operacións que consumen tempo operan en bytes individuais) é ineficiente nos procesadores modernos de 32 ou 64 bits.

3.4. Vantaxes e limitacións da criptografía simétrica

A criptografía simétrica leva empregándose polo menos 4000 anos e os algoritmos simétricos modernos como AES ou a variante 3DES dos que falamos nos apartados 3.2 e 3.3 son moi seguros e moi rápidos e están en uso xeneralizado. Pese a isto, hai varias deficiencias inherentes a este tipo de algoritmos, como as seguintes [11].

1. **Problema de distribución de claves**.

   A clave debe establecerse entre emisor e receptor usando unha canle segura. Ahora ben, a canle pola que se comunica a mensaxe non o é, polo que non se pode enviar a clave directamente por ela –que sería o xeito máis cómodo de intercambiala.

2. **Elevado número de claves necesarias**.

   Aínda que resolvésemos o problema de distribución de claves, potencialmente haise de tratar cun número moi grande de claves. Isto queda reflictido do seguinte xeito: se cada parella de usuarios precisa unha clave distinta, nunha rede con n usuarios hai

   $$\binom{n}{2} = \frac{n(n-1)}{2}$$

   parellas e cada usuario ten que almacenar $n-1$ claves de forma segura. Mesmo para redes de tamaño medio, digamos unha corporación con 2000 persoas, isto require máis de 4 millóns de pares de claves (a mesma para cada membro da parella) que deben xerarse e transportarse a través de canles seguras. Aínda máis, para cada novo membro, será preciso xerar 2000 novas claves e establecer 2000 canles seguras para distribuílas aos membros anteriores, co risco que hai de suplantación se se filtrase algunha delas.

3. **Sen protección contra as trampas** que poidan cometer o emisor ou o receptor.

   O emisor e o receptor teñen as mesmas capacidades, xa que posúen a mesma clave. Como consecuencia, a criptografía simétrica non se pode empregar para aplicacións nas que nos gustaría evitar as trampas, ben que cometa dito emisor ou dito receptor, caso distinto a evitar as trampas de terceiras persoas. O seguinte exemplo explica a que nos referimos con isto. Nas aplicacións de comercio electrónico a miúdo é importante demostrar que o emisor foi quen realmente enviou unha determinada mensaxe, digamos, unha orde de compra en liña. Se só usamos criptografía simétrica e dito emisor cambia de opinión máis tarde, sempre pode afirmar que o receptor –o vendedor– xerou falsamente a orde de compra electrónica. Evitar isto chámase non repudio e pódese conseguir cunha criptografía asimétrica, coas sinaturas dixitais.

Agora ben, unha das vantaxes que destacan desta criptografía, e que fai que siga a ser indispensable a pesar de todas as limitación recentemente descritas, é a **velocidade de cifra**. A criptografía simétrica é capaz de cifrar centos de MB por segundo mentres que a asimétrica só será capaz de cifrar centos de KB por segundo, a primeira é mil veces máis rápida que a segunda, esta diferencia na velocidade de cifra fai a criptografía simétrica indispensable para cifrar grandes cantidades de texto.

Capítulo 4

Criptografía asimétrica

A criptografía asóciase á procura da seguridade na comunicación secreta a través dunha canle insegura. Vimos que, para isto, nos cifrados simétricos se emprega unha única clave, e require, por tanto, dunha canle segura a través da cal emisor e receptor acorden, previamente á comunicación, dita clave, que podería chegar a ser interceptada poñendo en perigo todo o criptosistema. Isto, xunto coa necesidade dunha nova clave diferente por cada novo usuario co que se desexe comunicar –que é o único xeito de garantir a autenticación e a integridade–, son dous dos maiores inconvenientes aos que vén dar resposta a criptografía asimétrica.

Nos cifrados asimétricos **cada usuario conta con dúas claves**, unha pública P_k, e unha privada S_k, e a comunicación pode levarse a cabo dos dous xeitos presentados a continuación, e nos que poderá observarse que cada persoa necesitará memorizar soamente a súa clave privada. E non se require, en ningún momento, dunha canle segura para poder iniciar a comunicación, como sí sucedía na simétrica.

Para o emisor enviar unha mensaxe ao receptor garantindo a **confidencialidade**, o que fará será cifrar a mensaxe coa clave pública, coñecida por todos, de dito receptor, e así, soamente este poderá descifrar o texto en claro coa súa clave privada. Agora ben, para lograr a **integridade** e a **autenticidade**, o emisor cifrará coa súa clave privada e o receptor empregará a P_k tamén do emisor, para descifrar. É dicir, na criptografía asimétrica, a confidencialidade e a integridade obtéñense por separado.

Os algoritmos que se empregan nesta criptografía fan uso das coñecidas **funcións de sentido único**, que son as que imposibilitan descubrir o texto en claro, cifrado cunha clave pública, coñecendo soamente dita clave pública [2]. É dicir, funcións facilmente computables

–dada a mensaxe e a clave pública, o cifrado obtense nun tempo polinómico–, pero que son moi difíciles de inverter –obter o texto en claro coñecidos o criptotexto e a clave pública, requirirá dun tempo exponencial.

Definición 4.1 (Función de sentido único)**.** Unha función f **dise de sentido único**, ou unidirecional, se

1. Calcular $y = f(x)$ é computacionalmente doado e
2. Calcular $x = f^{-1}(y)$ é computacionalmente inviable.

Podemos concretar esta definición dicindo que, en termos matemáticos, unha función é fácil de calcular se pode avaliarse en tempo polinómico, é dicir, o seu tempo de execución é unha expresión polinómica. Para ser útil en esquemas criptográficos prácticos, o cálculo $y = f(x)$ debería ser o suficientemente rápido como para que non leve a tempos de execución inaceptablemente lentos nunha aplicación. O cálculo inverso $x = f^{-1}(y)$ debería ser tan computacionalmente intenso que non sexa factible avalialo nun período de tempo razoable, digamos, 10000 anos, cando se use o mellor dos algoritmos coñecidos.

Na actualidade, xa non se busca que o cifrado sexa inquebrantable independentemente da capacidade computacional, presente e futura, o que se coñecía como *seguridade incondicional* –seguridade eterna. Senón que se pretende todo o contrario, a seguridade do cifrado dependerá da gran cantidade de tempo necesaria para vulnerar dito cifrado, é dicir, búscase a coñecida como **seguridade computacional** –que caracteriza a toda a criptografía moderna.

Ao longo do capítulo explicaremos, dun xeito sinxelo e visual –facendo referencias a instrumentos ben coñecidos, como son os cadeados–, que é o que chamamos cifrados asimétricos, veremos ademáis como estes xorden a partir da idea duns matemáticos, Diffie e Hellman, que elaboran un protocolo de intercambio de claves. Finalmente estudaremos o cifrado máis relevante na actualidade, o RSA, base fundamental da seguridade na maioría de comunicacións e que é precisamente un algoritmo de encriptación asimétrico.

4.1. Analoxía dos cadeados

Unha boa forma de visualizar o cifrado asimétrico é explicalo coa axuda de ferramentas coñecidas, resulta interesante facer uso da analoxía dos cadeados [18].

Considérase a acción de pechar o cadeado como a operación pública –a operación de cifrar–, que pode ser levada a cabo por calquera, mentres que abrir dito cadeado será a operación privada –de descifrar– e que só poderá realizar o usuario posuidor da claves específica del.

Para intercambiar a mensaxe a través dunha canle segura, requirirase dunha caixa irrompible e impenetrable e cadeados. No caso de empregar un só cadeado, o procedemento a levar a cabo sería o seguinte:

Paso 1. O receptor posúe unha caixa e un cadeado coa súa determinada chave.

Paso 2. O receptor envía ao emisor dita caixa e o cadeado, ambos abertos.

Paso 3. Así, o emisor xa pode introducir a mensaxe na caixa, pechala co cadeado, e volver a enviala.

Paso 4. O receptor recibe a caixa que pode abrir ca chave que posúe para obter a mensaxe.

Neste caso, no que estamos a falar dun único cadeado, referímonos, en palabras do método criptográfico, a que se emprega un único alfabeto de cifrado. O alfabeto de cifrado, que é o do emisor –o que se encarga de pechar o cadeado, cifrar a mensaxe– hai de ser coñecido polos dous integrantes na comuncicación para así, o emisor poder cifrar o texto con dito alfabeto, e o receptor descifralo con operacións inversas deste.

Na criptografía asimétrica non se require dunha comunicación previa ao cifrado, é dicir, o receptor non tería por que coñecer o alfabeto de cifrado do emisor. Como solución a este problema basta con empregar dous cadeados e proceder da seguinte forma.

Paso 1. O emisor posúe unha caixa e un cadeado coa súa determinada chave, introduce a mensaxe en dita caixa e péchaa, para logo enviarlla ao receptor.

Paso 2. O receptor recibe a caixa pechada, con un cadeado pechado, e pecharaa doblemente engadindo outro cadeado do cal él posue a clave. Logo reenvía a caixa ao emisor.

Paso 3. O emisor abrirá o seu cadeado e volve a mandala.

Paso 4. Finalmente, o receptor recibe a caixa, que xa só pecha o seu cadeado e poderá, polo tanto, abrila ca chave que posúe e obter a mensaxe.

Con este método, estaríamos a falar de dous alfabetos de cifrado, o do emisor e o do receptor –pois tanto un coma outro se encargan de pechar cadeados ao longo da comunicación. Agora ben, se o encriptado se leva a cabo con cifras de *desprazamento puro* ou de

multiplicación pura, que contan con inversos aditivos e multiplicativos, respectivamente, na aritmética modular, este esquema non ten falla algunha. Non obstante, se a cifra é afín, ou asimétrica, a mensaxe desencriptada non coincidirá coa orixinal. No Exemplo 4.3 móstranse estas situacións.

Observación 4.2. Sexa m_i o grafema da mensaxe plana, e c_i o seu cifrado, que ven dado da operación en aritmética modular $c_i = a \cdot m_i + b \mod n$, dirase que:

- Se $a = 1$ é un cifrado por desprazamento puro.
- Se $b = 0$ é un cifrado por multiplicación pura.
- Se $a \neq 1$ e $b \neq 0$ é un cifrado por substitución afín.

Exemplo 4.3. Se definimos os cifrados do receptor e do emisor por desprazamento ou multiplicación pura, por exemplo, no galego tomamos o alfabeto de cifrado do emisor dado por $c_i = m_i + 2 \mod 23$, e o do receptor por $c_i = m_i + 6 \mod 23$. Na Táboa 4.1 móstranse as correspondencias entre as letras dos alfabetos en claro, e as dos cifrados.

A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B

Alfabeto de cifra do emisor.

A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z	A	B	C	D	E	F

Alfabeto de cifra do receptor.

Táboa 4.1: Alfabeto cifrado por desprazamento puro.

A Táboa 4.2 mostra o proceso de cifrado e descifrado, paso a paso, considerando estes alfabetos. Pode observarse que a mensaxe desencriptada si se corresponde coa enviada.

Mensaxe clara	R	O	M	P	E	O	C	A	D	E	A	D	O
Paso 1. O emisor pecha o cadeado (cifra)	T	Q	Ñ	R	G	Q	E	C	F	G	C	F	Q
Paso 2. O receptor pecha o cadeado (cifra)	B	X	T	Z	Ñ	X	M	I	N	Ñ	I	N	X
Paso 3. O emisor abre o cadeado (descifra)	Z	U	R	V	M	U	I	G	L	M	G	L	U
Paso 4. O emisor abre o cadeado (descifra)	R	O	M	P	E	O	C	A	D	E	A	D	O

Táboa 4.2: Obtense a mensaxe esperada.

Non obstante, se os alfabetos empregados se obteñen de cifrados afíns, por exemplo, os seguintes: o alfabeto de cifrado do emisor dado por $c_i = 3 \cdot m_i + 2 \mod 23$, e o do receptor por $c_i = 2 \cdot m_i + 6 \mod 23$, a mensaxe desencriptada e a enviada non coincidirán. Na Táboa 4.3 móstranse as correspondencias entre as letras dos alfabetos en claro, e as dos cifrados, do mesmo xeito que o fixemos anteriormente.

A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
C	F	I	N	P	S	V	A	D	G	L	Ñ	Q	T	X	B	E	H	M	O	R	U	Z

Alfabeto de cifra do emisor.

A	B	C	D	E	F	G	H	I	L	M	N	Ñ	O	P	Q	R	S	T	U	V	X	Z
R	T	V	Z	B	D	F	H	L	N	O	Q	S	V	X	A	C	E	G	I	M	Ñ	P

Alfabeto de cifra do receptor.

Táboa 4.3: Alfabeto cifrado por substitución afín.

Ao proceder deste xeito –cifrando emisor, logo receptor, e finalmente descifrando cada un deles– nos alfabetos deste tipo, finalmente a mensaxe que se obtén de desencriptar o criptotexto, non é a que se enviara. Podemos velo na Táboa 4.4, traballando co exemplo anterior –*rompe o cadeado*.

Mensaxe clara	R	O	M	P	E	O	C	A	D	E	A	D	O
Paso 1. O emisor pecha o cadeado (cifra)	E	T	L	X	P	T	I	C	N	P	C	N	T
Paso 2. O receptor pecha o cadeado (cifra)	B	G	N	Ñ	X	G	L	V	Q	X	V	Q	G
Paso 3. O emisor abre o cadeado (descifra)	Q	L	D	N	P	L	M	G	Ñ	P	G	Ñ	L
Paso 4. O emisor abre o cadeado (descifra)	N	I	F	L	Z	I	V	T	X	Z	T	X	I

Táboa 4.4: Non se obtén a mensaxe esperada.

Da falla deste procedemento ao traballar con cifras afíns, ou asimétricas, que se mostra de forma clara ao final do exemplo anterior, dedúcese a necesidade das funcións dun só sentido –que son sinxelas de computar pero difíciles de inverter, como o problema do logaritmo discreto que se emprega no protocolo de intercambio de claves de Diffie e Hellman (que veremos a continuación)– e que permiten resolver o problema do seguinte xeito: cada usuario contará coa súa clave que se dividirá en dúas, unha parte –a máis pequena– será a clave pública, un cadeado aberto; e a outra parte –a grande– será a chave de dito cadeado, o que coñecemos como a clave privada. É dicir, existirán tantos cadeados como persoas interveñan na comunicación. E, ademais, tras un emisor ter enviada unha mensaxe dirixida a un receptor –ocultar a mensaxe na caixa pechada coa chave de dito receptor–, o único paso posible para que o criptosistema funcione, será que dito receptor desencripte a mensaxe.

No 1976, Whitfield Diffie e Martin Hellman, dous criptógrafos da Universidade de Stanford, publican un protocolo que permite intercambiar claves secretas entre emisor e receptor de forma segura, idea na que tamén estaba traballando un investigador da universidade de Berkeley, Ralph Merkle, e que se coñece co nome de Diffie-Hellman ou Diffie-Hellman-Merkle. Aínda que non pode considerarse como un algoritmo de cifrado asimétrico, sí que foi a semente a partir da cal se orixinou a criptografía de clave pública.

4.2. Diffie-Hellman

O esquema do protocolo de intercambio de claves de Diffie-Hellman parte de definir a seguinte función de sentido único, onde $G = \langle g \rangle$ é un grupo cíclico e g un xenerador:

$$\begin{aligned} \mathbb{Z}_p^* &\longrightarrow G \\ x &\longmapsto g^x \end{aligned}$$

Coñecido x é facil calcular g^x (g fixado), resólvese en tempo polinómico. Non obstante,

$$\begin{aligned} G &\longrightarrow \mathbb{Z}_p^* \\ g' &\longmapsto t = \log_g(g') \end{aligned}$$

resólvese en tempo exponencial.

O proceso de encontrar t tal que $g^t = g'$ recibe o nome de **problema do logaritmo discreto**.

Tendo en conta que a canle pola que se fan públicos os datos debe estar autentificada para ter a seguridade de que estamos enviando os datos á persoa correcta, levaranse a cabo as seguintes accións que permiten o intercambio dunha clave secreta.

- Faise público o grupo cíclico G e un xerador g;
- o emisor escolle un enteiro ao azar, x, que será a súa clave privada. Calcula $g^x = u$ e envía ao receptor, de forma pública, o valor de dito u;
- o receptor escolle un número, que só coñecerá él, a clave privada y. Determina o elemento do grupo $g^y = v$ e envía v ao emisor;
- así, o emisor coñece x e $v = g^y$, e calcula v^x;
- o receptor coñece y e $u = g^x$, e calcula u^y;
- entón a clave que se intercambiarán será o valor común de v^x e u^y:

$$v^x = (g^y)^x = g^{xy} = (g^x)^y = u^y,$$

$k = g^{xy}$.

Así, fabricar a clave consiste en aplicar o algoritmo de exponenciación discreta, que se executa en tempo polinómico.

A seguridade deste protocolo baséase no feito de que unha persoa que intercepte a mensaxe coñecerá g, n, u, e v, pero para obter a clave privada (x, y) necesita resolver os problemas do logaritmo discreto $x = \log_g u$, $y = \log_g v$, os cales se executan en tempo exponencial.

En canto á criptoanálise do protocolo de Diffie-Hellman destaca o **ataque do intermediario**, tamén coñecido como *Man in the Middle* (MitM), que como ben pode intuirse, consiste na intromisión dunha terceira persona na comunicación, que ben pode interceptala para espiala ou incluso para alterala. Un exemplo deste ataque é o seguinte: o atacante interponse no medio dunha comunicación entre a vítima e unha aplicación bancaria para facerse coas credenciais de acceso e operar no nome del no futuro.

4.3. RSA

RSA é o algoritmo de encriptación asimétrico máis coñecido e empregado na actualidade. No 1977, baseándose no artigo de Diffie-Hellman sobre sistemas de clave pública, os criptógrafos e científicos Ron Rivest, Adi Shamir e Leonard Adleman, que formaban parte do MIT, publicaron a descrición do cifrado que leva por nome as inicias dos apelidos de ditos tres matemáticos. A seguridade do método recae na idea de que a descomposición dun número grande en factores primos é un problema difícil de solucionar.

O algoritmo RSA desenvólvese en tres pasos: unha primeira xeración de claves, para logo pasar ao cifrado, e finalmente, o descifrado. A continuación explícanse os procedementos que se levan a cabo en cada unha das fases e que fan do RSA un algoritmo sinxelo, eficiente e seguro [12].

Paso 1. **Xeración das claves** pública e privada de cada un dos usuarios participantes na comunicación.

O usuario escolle, de forma aleatoria e secreta, dous números primos distintos p, q de tamaño parecido (orde aproximada de 10^{300}). Calcula $n = p \cdot q$, que será o módulo que se empregue tanto na clave pública como na privada –nas que se traballa con aritmética modular–, e faino público. Non será tarefa difícil atopar ditos

números primos empregando test de primalidade, algoritmos que permiten decidir se un número dado é efectivamente primo ou pola contra, composto.

Logo, elixirá un enteiro coprimo con $(p-1)$ e con $(q-1)$, será, polo tanto, coprimo tamén co produto distes, $(p-1)\cdot(q-1)$; e que sexa menor ca el –tampouco demasiado pequeno para evitar comprometer a seguridade–, é dicir, e tal que

$$\text{mcd } (e,(p-1)(q-1)) = 1.$$

Unha vez fixado o enteiro e buscarase o seu inverso multiplicativo, traballando en aritmética modular (como vimos facendo) con módulo $(p-1)\cdot(q-1)$, e que denotaremos por d. Dito d será tal que verifique:

$$1 \equiv d \cdot e \mod (p-1)(q-1),$$

e despexarase empregando o Algoritmo de Euclides estendido.
Deste xeito atopáronse todos os elementos necesarios para definir as claves pública e privada do usuario. A primeira será (n,e), onde n é o módulo xa definido ao principio, e e o enteiro calculado, que será o expoñente de cifrado, como veremos. E a segunda (n,d), con n de novo o módulo, e d, que é o inverso de e, o expoñente de descifrado. A clave pública (n,e) dase a coñecer, e a clave privada (n,d) hai de manterse oculta.

Paso 2. **Cifrado da mensaxe** en claro, M, que envía un emisor a un receptor.
O emisor coñece a clave pública do receptor ao que quere enviarlle dita mensaxe, esta é (n,e). O primeiro que fará será convertir M nun número enteiro m que sexa menor que n. Para isto empregaranse esquemas de recheo ou *padding*. Unha vez obtido m e coa aplicación da exponenciación binaria, rapidamente se obtén o cifrado c do seguinte xeito:

$$c = m^e \mod n,$$

onde c é a mensaxe encriptada, que o emisor envía ao receptor.

Paso 3. **Descifrado do criptotexto** para poder ler a mensaxe orixinal.
O receptor conta coa súa clave privada, (n,d), e esta, unha vez recibido c, permitirá obter m realizando o seguinte cálculo:

$$m = c^d \mod n.$$

Finalmente, recupérase a mensaxe plana M invertindo o esquema de recheo.

4.3.1. Fundamentos matemáticos do RSA

O axeitado funcionamento no descifrado do RSA débese a que $m^{e \cdot d} \equiv m \pmod{n}$. Este resultado verifícase baixo as hipótesis nas que se desenvolve o RSA –que se volverán a enunciar a continuación– e pode demostrarse facilmente, como imos ver. Un exemplo disto apórtao José Florentino Abarca na súa tese, que pode atoparse en [1].

Lema 4.4. *O algoritmo RSA parte de dous primos, p e q, obtén o produto $n = pq$ e busca un número e que non teña divisores en común con $p-1$ nin con $q-1$. Nestas circunstancias, e non ten divisores en común con $(p-1)(q-1)$ e pode, polo tanto, obterse d, o inverso de e módulo $(p-1)(q-1)$. Baixo estas hipóteses, tense o seguinte resultado:*

$$m^{ed} \equiv m \pmod{n}.$$

Para demostrar este enunciado farase uso dos seguintes teoremas:

Teorema 4.5 (Teorema Pequeno de Fermat)**.** *Dados dous enteiros, a e p, tales que p é primo e $mcd(a,p) = 1$, entón $a^{p-1} \equiv 1 \pmod{p}$.*

Teorema 4.6 (Teorema Chinés dos Restos)**.** *Se $m_1, \ldots, m_s$ son n úmeros enteiros positivos sen divisores en común, e $a_1, \ldots, a_s$ son números enteiros calquera, entón o sistema de congruencias*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ &\vdots \\ x &\equiv a_s \pmod{m_s} \end{aligned}$$

ten unha única solución módulo $m_1 \cdot \cdots \cdot m_s$.

Demostración (do Lema 4.4). Polo Teorema 4.6 (Teorema Chinés dos Restos), bastará demostrar que $m^{de} \equiv m \pmod{p}$ e $m^{de} \equiv m \pmod{q}$.
Se m é un múltiplo de p, entón $m \equiv 0 \equiv m^{ed} \pmod{p}$. Se p non divide a n, dado que $de \equiv 1 \pmod{(p-1)(q-1)}$, entón $de = 1 + k(p-1)(q-1)$, para algún k, e tense

$$m^{de} = m^{1+k(p-1)(q-1)} = m \cdot m^{k(p-1)(q-1)},$$

e polo Teorema 4.5 (Teorema Pequeno de Fermat),

$$m^{k(p-1)(q-1)} = \left(m^{(p-1)}\right)^{k(q-1)} \equiv 1^{k(q-1)} \equiv 1 \pmod{p}.$$

Analogamente, $m^{de} \equiv m \pmod{q}$, co que quedaría probado o resultado.

Unha vez demostrado isto, e tendo en conta a definición de $c = m^e \mod n$, queda claro que a mensaxe orixinal convertida, m, si que se recuperará como propón o algoritmo, $m \equiv c^d \pmod{n}$, de feito, por ser m menor que n terase a igualdade $m = c^d \mod n$:

$$c^d = (m^e)^d = m^{e \cdot d} \equiv m \pmod{n}.$$

Unha posible programación do RSA en `SageMath` pode consultarse no Apéndice D, onde tamén se deixa máis claro o funcionamento dos esquemas de recheo que se presentaron na explicación do algoritmo. Cabe mencionar que dependendo do *padding* empregado existen distintas variantes do RSA, como son RSA-OAEP, RSA-REACT ou RSA-PSS, entre outras. En dito apéndice móstrase unha versión escolar do RSA onde o recheo se reduce a engadir un determinado número de bits.

En canto á seguridade do RSA, tamén xogará nela un papel importante a lonxitude das claves, este sistema permite distintas lonxitudes, sendo actualmente aconsellable o uso de claves de polo menos 1024 bits pois, aínda que se necesitou casi de medio ano e de uns 300 ordenadores traballando xuntos, xa se conseguiron romper aquelas de ata 512 bits.

4.3.2. Aritmética modular: as matemáticas do RSA

En toda a criptografía está presente a aritmética modular, xa se empregou cando se presentaron cifrados clásicos, como o César ou Vigenère, e igual de importante segue a ser nos cifrados modernos, pois permite realizar complexas e interesantes operacións nun conxunto finito de elementos. Un claro exemplo é o algoritmo RSA, recentemente explicado, que fai uso, tanto explícito como implícito, de relevantes operacións e teoremas, enmarcados neste sistema aritmético [8], que se enuncian a continuación e se explican con máis detalle no Apéndice C, onde tamén poderán atoparse posibles implementacións en `SageMath` delas.

O RSA comeza coa xeración das claves que se empregarán ao largo de todo algoritmo, co fin de obter ditas claves parte escollendo números primos de gran tamaño, para o que se farán uso de **test de primalidade**, como xa se mencionou. Entre os métodos existentes destaca o de Rabin-Miller por ser o máis eficiente, preciso e fácil de implementar. Os **algoritmos de Euclides e de Euclides estendido** requírense pola súa aplicación a hora de obter o máximo común divisor entre dous números ou inversos en aritmética modular, respectivamente. Outra operación moi interesante, da que se fai uso, é a **exponenciación binaria** que se emprega no momento de cifrar.

Ademáis destes cálculos, dos que se fai directamente uso á hora de implementar o algoritmo, viuse na sección 4.3.1 que o RSA tamén require, dun xeito máis implícito, de teoremas como o **Teorema Pequeno de Fermat ou o Chinés dos Restos**, que tamén traballan en aritmética modular, para demostrar o seu bo funcionamento.

Capítulo 5

Aplicacións da criptografía: a sinatura dixital

No capítulo anterior vimos que, para superar os inconvenientes da criptografía simétrica, que resumimos na Sección 3.4, Diffie, Hellman e Merkle fixeron unha proposta revolucionaria baseada na seguinte idea: non é necesario que a clave que posúe o emisor da mensaxe sexa secreta. A parte crucial é que só o receptor a pode descifrar usando unha clave secreta. A tal fin, dito receptor dá a coñecer unha clave de cifrado, a pública, pero ademais, el contará tamén cunha clave secreta correspondente, que se usa para descifrar. Así, dita clave deste, k, consta de dúas partes, unha parte pública, P_k e outra privada, S_k.

Este protocolo para cifrar sen necesidade dunha canle segura, pode modificarse facilmente para cifrar unha clave simétrica usando algoritmos asimétricos, como pode ser unha clave AES ou DES entre outras. Así, unha vez que o receptor desencripta dita clave simétrica, ambas as partes poderán usala para cifrar e descifrar mensaxes usando criptografía simétrica.

Polo estudado ata agora, parece que a criptografía asimétrica é unha ferramenta desexable para aplicacións de seguridade [4]. A cuestión recaía en como poñer en práctica este método de Diffie, Hellman e Merkle, é dicir, en como se poderían construír algoritmos de clave pública, e para dar resposta a isto viuse que a idea central atópase no que se definiu como *función de sentido único.*

Hai dúas funcións de sentido único populares que se usan en esquemas prácticos de clave pública. O primeiro é o **problema de factorización enteira**, no que se basea RSA.

Dados dous números primos grandes, é doado calcular o seu produto; non obstante, é moi difícil recuperar os factores a partir do produto. De feito, se cada un dos números primos ten 150 ou máis díxitos decimais, o produto resultante non se pode factorizar, nin sequera con miles de ordenadores funcionando durante moitos anos. A outra función unidireccional que se usa amplamente é o **problema do logaritmo discreto**. Este problema consiste en, dado un grupo cíclico finito, $\mathbb{Z}_p^*$, un elemento primitivo, $\alpha \in \mathbb{Z}_p^*$ e outro elemento, $\beta \in \mathbb{Z}_p^*$, determinar o enteiro $1 \leq x < p$ tal que $\alpha^x \equiv \beta \mod p$, nesta función baséanse outros algoritmos asimétricos coñecidos como, por exemplo, ElGamal ou DSA –que se introducirán ao final deste capítulo.

5.1. Mecanismos de seguridade

Como xa dixemos, además do cifrado de datos, estes métodos de encriptación poden usarse para facer moitas outras cousas, antes inimaxinables, como as seguintes:

1. **Establecemento de claves**

   Hai protocolos para establecer claves secretas nunha canle insegura. Exemplos para estes protocolos inclúen o intercambio de claves Diffie-Hellman (DHKE) ou RSA.

2. **Non rexeitamento**

   Proporcionar non rexeitamento e integridade da mensaxe pódese realizar con algoritmos de sinatura dixital, por exemplo, RSA, DSA ou ECDSA –estes últimos son outros dous algoritmos asimétricos de sinatura dixital e sinatura dixital con curvas elípticas, respectivamente.

3. **Identificación**

   Podemos identificar entidades mediante o que se coñecen como protocolos de resposta e desafío xunto con sinaturas dixitais, por exemplo, en aplicacións como tarxetas intelixentes para a banca ou para teléfonos móbiles.

4. **Cifrado**

   Podemos cifrar mensaxes utilizando algoritmos como RSA ou ElGamal.

Hai que dicir que a identificación e máis o cifrado tamén se poden conseguir con algoritmos de encriptación simétricos, pero normalmente requiren moito máis esforzo coa xestión de claves. Parece que os esquemas de clave pública poden proporcionar todas as funcións requiridas polos protocolos de seguridade de hoxe en día. Aínda que isto é certo, o principal inconveniente na práctica é que o cifrado de datos con algoritmos de clave

pública é computacionalmente moi intensivo –ou máis coloquialmente: extremadamente lento. Moitos cifrados de bloques e fluxo poden cifrar de cen a mil veces máis rápido ca os algoritmos de clave pública. Así, de xeito irónico, a criptografía de clave pública raramente se usa para o cifrado real de datos.

Outro problema, ao que se debe facer fronte ao empregar criptografía asimétrica, é que os algoritmos de clave pública requiren claves moi longas, o que resulta en tempos de execución lentos. A continuación discutirase a lonxitude e seguridade de ditas claves.

5.2. Lonxitude e niveis de seguridade das claves

No capítulo da ciptografía simétrica, limitámonos a dous cifrados de bloques, DES e AES, se ben existen moitos outros algoritmos simétricos. Ao longo dos anos propuxéronse varios centos de algoritmos e, aínda que se descubriu que moitos non eran seguros, existen moitos criptograficamente fortes.

A situación é radicalmente diferente para os algoritmos asimétricos. Só hai tres grandes familias de algoritmos de clave pública que son de relevancia práctica. Pódense clasificar en función do problema computacional que os substenta [17].

1. **Esquemas de factorización enteira**

   Varios esquemas de clave pública baséanse no feito de que é difícil factorizar números enteiros grandes. O representante máis destacado desta familia de algoritmos é RSA.

2. **Esquemas de logaritmo discreto**

   Hai varios algoritmos que se basean no coñecido como problema do logaritmo discreto en corpos finitos. Os exemplos máis destacados inclúen o intercambio de claves Diffie-Hellman, o cifrado ElGamal ou o algoritmo de sinatura dixital (DSA).

3. **Esquemas de curva elíptica (EC)**

   Unha xeneralización do algoritmo de logaritmo discreto son os esquemas de clave pública de curva elíptica. Os exemplos máis populares inclúen o intercambio de claves Diffie-Hellman de Curva elíptica (ECDH) e o algoritmo de sinatura dixital de curva elíptica (ECDSA).

As tres familias de algoritmos de clave pública establecidas están baseadas en funcións numéricas. Unha característica distintiva delas é que requiren aritmética con operandos e

		Criptosistemas	Nivel de seguridade (bit) 80	128	192	256	Lonxitude (bit)
Criptografía simétrica		AES, 3DES	80	128	192	256	
Criptografía asimétrica	Factorización enteira	RSA	1024	3072	7680	15360	
	Logaritmo discreto	DH, DSA, Elgamal	1024	3072	7680	15360	
	Curvas elípticas	ECDH, ECDSA	160	256	384	512	

Táboa 5.1: Lonxitudes en bits dos algoritmos para os diferentes niveis de seguridade.

claves moi longos. Non sorprende que, canto máis longos sexan os operandos e as claves, máis seguros serán os algoritmos. Para comparar diferentes algoritmos, utilízase o chamado nivel de seguridade. Dise que un algoritmo ten un ***nivel de seguridade*** n ***bit*** se o mellor ataque coñecido require 2^n pasos. É unha definición bastante natural porque os algoritmos simétricos cun nivel de seguridade n bit teñen unha clave de n bits. A relación entre potencia criptográfica e nivel de seguridade non é tan sinxela no caso asimétrico. A Táboa 5.1 mostra a lonxitude de bits recomendada para algoritmos de clave pública para os catro niveis de seguridade 80, 128, 192 e 256 bit. Vemos na táboa que os esquemas semellantes ao RSA e aos esquemas de logaritmo discreto requiren operandos e claves moi longos. A lonxitude da clave dos esquemas de curvas elípticas é significativamente menor, pero mesmo así o dobre cós cifrados simétricos coa mesma forza criptográfica.

Por outra banda, os algoritmos simétricos son pobres á hora de proporcionar funcionalidades de non repudio e establecemento de claves. Para usar o mellor de ambos os mundos, os protocolos máis prácticos son **protocolos híbridos** que incorporan algoritmos simétricos e de clave pública. Cabe destacar os seguintes [19]:

- **SSL (Secure Socket Layer).** Protocolo de cifrado ideado por Netscape para a transmisión de datos en Internet. É empregado por todas as conexións seguras https na Web.

- **SSH (Secure Shell).** Protocolo da rede co que entrar e executar programas en

computadores que non se atopen preto, é dicir, permite unha conexión segura e auténtica entre computadores sobre unha canle insegura. Emprega o procedemento RSA para lograr a conexión, tras o cal, fará uso de rápidos procedementos simétricos para o cifrado da comunicación, como é o AES.

- **PGP (Pretty Good Privacy).** Comezou empregando o RSA para o intercambio de claves, ata finales do 2002, cando pasou a usar o algoritmo ElGamal. Este protocolo permite cifrar mensaxes electrónicas.

Pola discusión ata agora vimos que unha gran vantaxe dos esquemas asimétricos é que podemos distribuír libremente claves públicas. Non obstante, na práctica, as cousas son un pouco máis complicadas porque aínda temos que asegurar a autenticidade de ditas claves públicas, é dicir, ter a seguridade de que unha determinada clave pública pertence a unha determinada persoa. Na práctica, este problema adoita resolverse co que se denomina certificados dixitais, pode dicirse, grosso modo, que os certificados vinculan unha clave pública a unha determinada identidade; ou tamén coa axuda de sinaturas dixitais, como se explicará a continuación. Este é un problema importante en moitas aplicacións de seguridade, por exemplo, cando se realizan transaccións de comercio electrónico en Internet.

5.3. Sinatura dixital

As sinaturas dixitais son indispensables en infinitude de mensaxes electrónicas, como pode ser unha transacción bancaria ou un contrato de compra. Non son máis que un conxunto de datos que permiten, ao receptor dunha información, probar a orixe e a integridade do recibido ademais de protexer fronte a posibles falsificacións. Pode definirse dando o esquema de sinatura que sempre se segue para firmar, independentemente do algoritmo que se empregue [19].

Definición 5.1 (Esquema de sinatura)**.** Un **esquema de sinatura** é unha terna ($\mathcal{M}$, $\mathcal{S}$, $\mathcal{K}$) de conxuntos finitos $\mathcal{M}$, as mensaxes en claro; $\mathcal{S}$, as sinaturas e $\mathcal{K}$, as claves, de tal xeito que, para cada $k \in \mathcal{K}$ existen un par de funcións:
A función de sinatura, sig_k, coa que asinar a mensaxe, e a función de verificación, ver_k, que será pública, para comprobar que tal sinatura é auténtica:

$$sig_k \colon \mathcal{M} \longrightarrow \mathcal{S}$$
$$ver_k \colon \mathcal{M} \times \mathcal{S} \longrightarrow \{\text{verdadeiro, falso}\}.$$

Dados $m \in \mathcal{M}$ e $s \in \mathcal{S}$, esta última defínese como segue:

$$ver_k(m,s) = \begin{cases} \text{verdadeiro}, & \text{se } sig_k(m) = s, \\ \\ \text{falso}, & \text{noutro caso.} \end{cases}$$

Ademáis, o único coñecedor da sinatura s que verifique $ver_k(m,s) =$ verdadeiro hai de ser o asinante da mensaxe.

Observación 5.2. É importante ter en conta que:

- A sinatura dixital non pode adxuntarse ao final dunha mensaxe, senón que se firmará antes. Existen ataques que permiten manipular con éxito mensaxes primeiro cifradas e logo firmadas.
- Do mesmo xeito que se pode comprobar a autenticidade da sinatura manual, comparándoa por exemplo coa do Documento Nacional de Identidade, o feito de que os algoritmos de verificación sexan públicos farán posible, autentificar a dixital.

5.3.1. Sinatura dixital con RSA

Como xa se dixo, o algoritmo asimétrico RSA –visto na Sección 4.3– permite a sinatura dixital e, a continuación, deterémonos a estudar o procedemento co que efectivamente se obtén:

O RSA traballa co produto de dous primos, $n = p \cdot q$ e obtén como claves pública e privada, (n,e) e (n,d), respectivamente, verificando ademáis que $d \cdot e \equiv 1 \pmod{(p-1)\cdot(q-1)}$.

Tomando entón e como a clave pública e d a privada dun emisor que pretende asinar unha mensaxe $m \in \mathcal{M}$, definido neste caso $\mathcal{M} = \mathbb{Z}_n$, non terá máis que calcular

$$sig_k(m) = m^d \mod n \in \mathcal{S} = \mathbb{Z}_n,$$

e publicar como función de verificación

$$ver_k(m,s) = \begin{cases} \text{verdadeiro}, & \text{se } m \equiv s^e \pmod{n}, \\ \\ \text{falso}, & \text{noutro caso.} \end{cases}$$

Ao ser e a clave pública do emisor, calquera poderá verificar a firma. No caso de obter, $m \not\equiv s^e \pmod{n}$ entón,

$$s \equiv s^{d\cdot e} \not\equiv m^d \pmod{n},$$

e esta desigualdade na equivalencia tense gracias a que a potenciación con $d \in \mathbb{Z}_n$ é inxectiva. O que quere dicir que a sinatura non corresponde á mensaxe m e, polo tanto, é falsa.

Neste procedemento terá que garantirse a autenticidade da clave pública, por exemplo, a través dun certificado dunha organización confiable, do contrario un terceiro podería suplantar dita sinatura.

5.3.2. Sinatura dixital con ElGamal

O algoritmo ElGamal, descrito polo criptógrafo exipcio Taher ElGamal no 1984, foi diseñado para crear sinaturas dixitais, aínda que logo se empregou tamén no cifrado de mensaxes. Este é un método de encriptación asimétrico que se basea no problema do logaritmo discreto e en Diffie-Hellman.

En canto á obtención das claves pública e privada deste algoritmo, pode simplificarse dicindo que, o usuario comeza elixindo un número primo n e outros dous aleatorios, p e x menores que n. Calcula

$$y = p^x \mod n,$$

conseguindo así as claves buscadas: a clave pública será (p, y, n) e a privada x.

Xa estamos en condicións de estudar o esquema de sinatura dunha mensaxe m. Bastará con escoller unha clave aleatoria k, sendo k coprimo con $n-1$, e por suposto hai de ser secreta e dun só uso. Así, a firma virá dada polo par (a, b) –denotemos $s \equiv (a, b)$– onde

$$\begin{aligned} a &= p^k \mod n, \\ b &= (m - x \cdot a)k^{-1} \mod (n-1). \end{aligned}$$

Neste caso suministrarase a seguinte función de verificación

$$ver_k(m, s) = \begin{cases} \text{verdadeiro}, & \text{se } p^m \equiv y^a \cdot a^b \pmod n, \\ \\ \text{falso}, & \text{noutro caso.} \end{cases}$$

Unha vez estudados os esquemas de sinatura empregando os algoritmos RSA e ElGamal, pode observarse que no primeiro caso, a sinatura terá a mesma lonxitude que a mensaxe en claro, e no segundo, o dobre. Isto, ademáis de ocasionar un importante gasto de tempo, tamén ocupa demasiada memoria, polo que se buscará reducir tales lonxitudes. Ademais, estos algoritmos presentan outra importante limitación: existen ataques de suplantación aos que non son capaces de facer fronte. Unha posible solución a estos problemas será o DSA, algoritmo que se estudará a continuación.

5.3.3. DSA

O algoritmo de sinatura dixital coñecido, polas súas siglas en inglés, como DSA (Digital Signature Algorithm) é unha variante eficiente do ElGamal proposta polo NIST no 1991. É o actual algoritmo de sinatura dixital estándar do Goberno Federal de Estados Unidos, América [9].

As claves pública e privada neste algoritmo obtéñense cos seguintes pasos:

1. Comeza coa seleción de q, un número primo verificando $2^{159} < q < 2^{160}$.
2. Buscaranse t, p e q tales que $0 \leq t \leq 8$, $2^{511+64\cdot t} < p < 2^{512+64\cdot t}$ e q sexa divisor de $(p-1)$, respectivamente.
3. Tómase un elemento do grupo cíclico de orde p, $g \in \mathbb{Z}_p^*$, e calcúlase $\alpha = g^{(p-1)/q}$ mód p.
4. Se $\alpha = 1$ repítese o paso 3.
5. E finalmente, escóllese aleatoriamente un enteiro a tal que $1 \leq a \leq q-1$ e calcúlase $y = \alpha^a \mod p$.

A clave pública será (p, q, α, y) e a privada a.

Para a obtención e verificación da sinatura, o DSA fai uso dunhas funcións dun só sentido que son capaces de calcular, a partir dunha cadea de bits de lonxitude arbitraria, outra, aparentemente aleatoria e de lonxitude fixa, as coñecidas como funcións resumen ou hash [2].

Sexa entón h a saída dunhas destas funcións aplicada a mensaxe m, e k un número ao azar, positivo e menor que o valor público q; a sinatura de dita mensaxe será o par (u, v) onde

$$u = (\alpha^k \mod p) \mod q \qquad \text{e} \qquad v = k^{-1}(h + a \cdot u) \mod q.$$

Para verificar a autenticidade da firma, bastará con efectuar as seguintes operacións, tendo en conta que a clave pública (p, q, α, y) é coñecida. Comezarase verificando que $0 < u < q$ e $0 < v < q$, noutro caso xa se rechazaría a sinatura. E finalmente, esta aceptarase se, e só se, unha vez calculado $w = v^{-1} \mod q$ e obtido o valor de h –a partir de m–, se verifica

$$u = (\alpha^{w\cdot h \mod q} \cdot y^{w\cdot r \mod q} \mod p) \mod q.$$

Bibliografía

[1] Abarca, J. F. (2018). *Fundamentos matemáticos del algoritmo RSA* (Tese de maestría). Universidad Autónoma de Guerrero, Chilpancingo de los Bravo, Guerrero, México.

[2] Centro Criptológico Nacional. (2015). *CCN-STIC-401 Glosario y Abreviaturas*. Consultado por última vez o 3 de xuño de 2021 en `https://www.ccn-cert.cni.es/guias/glosario-de-terminos-ccn-stic-401.html`

[3] Chung-Wei Phan, R., (2002), Mini advanced encryption standard (mini-aes): A testbed for cryptanalysis students, *Cryptologia*, **26** (4), 283-306.

[4] Garfinkel S. L. (1996). Public key cryptography. *Computer*, **29** (6), 101–104.

[5] Gómez Pardo, J. L. (2013). *Introduction to Cryptography with Maple*. Berlín. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG.

[6] Granados, G. (2006). Introducción a la criptografía. *Revista Digital Universitaria*, **7** (7).

[7] Kroll, S. (2016). Evolución de los sistemas criptográficos dende la Edad Media a la Moderna. *Memoria y Civilización*, **7**. 181–199.

[8] Lucena, M. J. (2021). *Criptografía y Seguridad en Computadores* (Versión 5-1.1.1). [Libro electrónico], Jaén, España.

[9] Plaza, F. J. (2021). *Manual de Criptografía: Fundamentos matemáticos de la Criptografía para un estudiante de Grado*. Salamanca, España. Ediciones Universidad de Salamanca.

[10] Prieto, M. J. (2020). *Historia de la criptografía: Cifras, códigos y secretos dende la antigua Grecia a la Guerra Fría*. Madrid, España. La Esfera de los Libros.

[11] Ramió, J. [UPM]. (2015). Píldora formativa 27: ¿Qué es mejor, la criptografía simétrica o la asimétrica? [Arquivo de vídeo]. Recuperado o 9 de maio de 2021 de `https://www.youtube.com/watch?v=0qfOVm-dtcQ`

[12] [Recurso electrónico da Universidade Politécnica de Madrid con información sobre cifrados clásicos e modernos]. Recuperado o 8 de abril de 2021 de `http://www.dma.fi.upm.es/recursos/aplicaciones/matematica_discreta/web/aritmetica_modular/bibliografia.html`

[13] Reyad, O. (2018). *Cryptography and Data Security: An Introduction.* (Pendente de publicación).

[14] Schaefer, E. (1996). A Simplified Data Encryption Standard Algorithm. *Cryptologia*, **20** (1). 77–84.

[15] Sgarro A.. (1989). *Códigos secretos.* Madrid, España. Pirámide.

[16] Sharma, M., Garg, R. B. (2016). DES: The oldest symmetric block key encryption algorithm. *2016 International Conference System Modeling & Advancement in Research Trends (SMART).* 53–58.

[17] Talens-Oliag, S. (1999). Seguridad en JAVA. Valencia, España: Universidad de Valencia. Recuperado o 7 de xuño de 2021 de `https://www.uv.es/~sto/cursos/seguridad.java/html/sjava.html#toc2`

[18] Universidad Politécnica de Madrid [UPM]. *Crypt4you - Aula Virtual de Criptografía y Seguridad de la información Crypt4you.* Recuperado o 14 de maio de 2021 de `http://www.criptored.upm.es/crypt4you/portada.html`

[19] Willems, W., Gutierrez, I. (2010). *Una introducción a la criptografía de clave pública.* Barranquilla. (2ª ed.) Ediciones Uninorte.

Apéndices

Apéndice A

Implementación de S-DES en SageMath

A.1. S-DES (Simplyfied) Data Encryption Standard

O S-DES é unha versión reducida do algoritmo DES que toma bloques de texto claro de 8 bits e claves de 10 bits para, en dúas roldas, devolver texto cifrado de 8 bits, conservando todos os ingredientes do propio DES –estudado na Sección 3.2. A continuación, explicaranse os pasos para a implementación en `SageMath` dunha versión adaptada de [14].

A.1.1. Xeración das claves de rolda

Para a xeración das claves de rolda, precisarase dunha permutación, P10, que permuta os 10 bits da clave; dúas funcións, LS1 e LS2, que desprazan circularmente 1 e 2 posicións, respectivamente, os bits dun bloque de 5, e unha selección permutada PC8 que permuta 10 bits e selecciona 8.

```
def P10(c):
    """
    permuta os 10 bits de c segundo a lista p10
    """
    p10 = [2,4,1,6,3,9,0,8,7,6]
    s = []
    for i in range(10):
        bi = c[p10[i]]
        s.append(bi)
    return s
```

```
def LS1(c):
    """
    despraza 1 posición cara a esquerda os bits dunha cadea de 5
    """
    return [c[(i+1)%5] for i in range(5)]
```

```
def LS2(c):
    """
    despraza 2 posición cara a esquerda os bits dunha cadea de 5
    """
    return [c[(i+2)%5] for i in range(5)]
```

```
def P8(c):
    """
    permuta os 8 bits de c e selecciona 8 bits
    """
    p8 = [5,2,6,3,7,4,9,8]
    s = []
    for i in range(8):
        bi = c[p8[i]]
        s.append(bi)
    return s
```

```
def K(k):
    """
    xeración das claves de rolda
    """
    pk = P10(k)       # permutación inicial da clave
    E = pk[:5]        # metade esquerda
    D = pk[5:]        # metade dereita
    E = LS1(E)        # desprazamento dos bits á esquerda 1
    D = LS1(D)        # desprazamento dos bits á esquerda 1
    K1 = P8(E+D)      # xuntamos as metades e permutamos
    E = LS2(E)        # desprazamento dos bits á esquerda 2
    D = LS2(D)        # desprazamento dos bits á esquerda 2
    K2 = P8(E+D)      # xuntamos as metades e permutamos
    return [K1,K2]
```

A.1.2. Proceso de cifrado

Antes de comezar coa primeira rolda, os 8 bits de texto claro pasan pola permutación inicial, IP; e dentro de cada rolda, considérese que estamos a traballar con dúas cadeas de 4 bits (a parte esquerda e dereita da cadea de entrada), en cada iteración do algoritmo tense unha función Feistel, que colle os catro bits da dereita e a clave de rolda para producir uns novos 4 bits para dita parte dereita. Finalmente, a función SW intercambiará as metades esquerda e dereita.

En canto á función de Feistel, unha función $F : (\mathbb{Z}_2)^4 \times (\mathbb{Z}_2)^8 \to (\mathbb{Z}_2)^4$, dados m e k, alarga m a 8 bits coa estensión permutada, EP, que pasa de a 8 bits que se sumarán con XOR con k antes de pasar pola S-caixa. En realidade, son dúas S-caixas S1 e S2, que actúan sobre as metades esquerda e dereita da cadea de 8 bits, respectivamente, para devolver, cada unha delas, 2 bits que concatenaremos e permutaremos coa permutación P4 para obtermos os 4 bits da saída.

```
def IP(c):
    """
    permutación inicial dunha cadea de 8 bits
    """
    ip = [1,5,2,0,3,7,4,6]
    return [c[ip[i]] for i in range(8)]
```

```
def EP(c):
    """
    duplica e permuta os bits dunha cadea de 4 para facer unha de 8
    """
    ep = [3,0,1,2,1,2,3,0]
    s = []
    for i in range(8):
        bi = c[ep[i]]
        s.append(bi)
    return s
```

```
def xor(a,b):
    """
    o operador XOR consiste na suma de cadeas mod 2
    """
    return[(a[i]+b[i])%2 for i in range(len(a))]
```

A.1.3. S-caixa

Para entender o funcionamento das S-caixas detallarase, por exemplo, o de S1. Vexamos como se define $S1([b_0, b_1, b_2, b_3])$, partindo da matriz

$$M1 = \begin{pmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{pmatrix}$$

e tomando como i o número binario dado de contatenar b_0b_3, polo tanto, $i = 2b_0 + b_3$ e, analogamente, tómase $j = 2b_1 + b_2$: defínese $S1([b_0, b_1, b_2, b_3]) = (A_{ij})_2$, en binario, é dicir, as dúas cifras (completando con 0 á esquerda, se é preciso) que representan o número A_{ij}.

```
def S1(c):
    """
    S-caixa que pasa de 4 a 2 bits
    """
    s1 = matrix([[1,0,3,2],[3,2,1,0],[0,2,1,3],[3,1,3,2]])
    i = 2*c[0]+c[3]
    j = 2*c[1]+c[2]
    t = (s1[i,j]).digits(2,padto=2)
    t. reverse()
    return t
```

```
def S2(c):
    """
    S-caixa que pasa de 4 a 2 bits
    """
    s2 = matrix([[0,1,2,3],[2,0,1,3],[3,0,1,0],[2,1,0,3]])
    i = 2*c[0]+c[3]
    j = 2*c[1]+c[2]
    t = (s2[i,j]).digits(2,padto=2)
    t. reverse()
    return t
```

```
def S(x):
    """
    S-caixa de 8 a 4 bits
    """
    return S1(x[:4])+S2(x[4:])
```

```
def P4(c):
    """
    permuta cadeas de 4 bits
    """
    p4 = [1,3,2,0]
    s = []
    for i in range(4):
        bi = c[p4[i]]
        s.append(bi)
    return s
```

```
def F(d,k):
    """
    Función Feistel, con d 4 bits e k 8 bits
    """
    y = xor(EP(d),k)
    return P4(S(y))
```

```
def SW(c):
    """
    intercambia as metades esquerda e dereita dunha cadea de 8 bits
    """
    return c[4:]+c[:4]
```

Xa temos os ingredientes para definir a actuación por cada unha das dúas roldas

```
def R(i,x,k):
    """
    rolda i-ésima para x de 8 bits e k de 10 bits (1=1,2)
    """
    ki = K(k)[i-1]
    e = x[:4]
    d = x[4:]
    d1 = F(d,ki)
    e = xor(e,d1)
    return SW(e+d)
```

```
def IP_1(c):
    """
    inversa da permutación anterior
    """
    ip = [1,5,2,0,3,7,4,6]
    return [c[ip.index(i)] for i in range(8)]
```

A.1.4. Cifrado con S-DES

S-DES cifra bloques de 8 bits usando claves de 10 bits:

$$sDES_{Enc} : (\mathbb{Z}_2)^8 \times (\mathbb{Z}_2)^{10} \to (\mathbb{Z}_2)^8.$$

Dados m e k, aplícaselle a permutación inicial, IP, a m para despois aplicarlle a consecutivamente as roldas 1 e 2, e finalmente a inversa da permutación inicial despois de intercambiar as metades esquerda e dereita:

$$sDES_{Enc}(m,k) = \Big(IP^{-1} \circ SW \circ R_2 \circ R_1 \circ IP\Big)(m),$$

onde $R_i(m,k) \coloneqq R(i,m,k)$, para $i = 1, 2$.

```
def sDES_Enc(m,k):
    x = IP(m)
    for i in [1,2]:
        x = R(i,x,k)
    return(IP_1(SW(x)))
```

```
    m=[0,1,1,1,0,0,1,0]
    k=[1,0,1,0,0,0,0,0,1,0]
```

```
    sDES_Enc(m,k)
```

```
[0, 1, 1, 1, 0, 1, 1, 1]
```

A.1.5. Descifrado

Tendo en conta que, salvo o intercambio final das dúas metades, cada rolda é autoinversa, no sentido de que $(SW \circ R_i) \circ (SW \circ R_i) = \mathrm{id}$ ou, equivalentemente, $R_i \circ WS \circ R_i = SW$, o desencriptado é esencialmente o mesmo proceso ca o encriptado, só que as roldas son en orde inversa, primeiro K_2 e despois K_1.

$$sDES_{Dec}(c,k) = \Big(IP^{-1} \circ SW \circ R_1 \circ R_2 \circ IP\Big)(c),$$

onde $F_{K_i}(m) \coloneqq F(m, K_i)$, e K_1 e K_2 son as claves de rolda que se obteñen de k.

```
def sDES_Dec(m,k):
    x = IP(m)
    for i in [2,1]:
        x = R(i,x,k)
    return(IP_1(SW(x)))

sDES_Dec(sDES_Enc(m,k),k) == m
```

```
True
```

A.2. S-DES en SageMath

SageMath conta cunha implementación de S-DES, e podemos comprobar que, formalmente, coincide coa acabada de realizar neste apéndice, aínda que as súas saídas son como listas de elementos do monoide libre de cadeas binarias, mentres que aquí sae como lista de enteiros.

```
from sage.crypto.block_cipher.sdes import SimplifiedDES
sdes = SimplifiedDES()
sdes.encrypt(m,k)
sDES_Enc(m,k)
```

```
[0, 1, 1, 1, 0, 1, 1, 1]
[0, 1, 1, 1, 0, 1, 1, 1]
```

Apéndice B

Implementación de M-AES en SageMath

B.1. M-AES (Mini) Advanced Encryption Standard

Neste apéndice presentamos unha posible implementación en `SageMath` dunha versión reducida do algoritmo AES (**Advanced Encryption Standard**), o M-AES, baseado en [3] e no que se modificará a definición da S-box para respectar a súa estrutura alxébrica.

A versión de AES que se deu na sección 3.3 cifra bloques de 128 bits (16 bytes) usando claves de tamén 128 bits en 10 roldas, cada unha delas, agás a última, con catro capas: **substitución de bytes**, **desprazamentos nas filas**, **mestura por columnas** e **suma coa claves de rolda** detalladas na Figura 3.4. M-AES cifra bloques de 16 bits (4 cuartetos) usando claves de 16 bits (4 cuartetos) en 2 roldas nas que se implementarán as seguintes funcións:

- SubCuart (Substitucións de cuartetos)
- ShiftRows (desprazamentos nas filas)
- MixColumns (mestura por columnas)
- ExpCla (expansión da claves)

Para estas transformacións, multiplicación de matrices incluída, é necesario contar cunha suma e un produto, para o que consideraremos a estructura de corpo de Galois de orde 2^4, $\mathbb{F}_{16}$.

Unha maneira de facelo é a través das identificacións

$$\mathbb{F}_{16} \longrightarrow (\mathbb{F}_2)^4 \longrightarrow \mathbb{F}_2[x]/_{\langle x^4+x+1\rangle}$$

onde a primeira é simplemente unha enumeración dos 16 elementos de $\mathbb{F}_{16}$ mediante as cifras en base 2 (completadas ata 4) dos números 0 a 15. A segunda asócialle a cada polinomio a súa clase módulo o ideal maximal xenerado por x^4+x+1 (polinomio irreducible), cun representante canónico, un polinomio de grao menor ou igual que 3. Este último corpo é o que dá estrutura ás outras dúas representacións.

En `SageMath` podemos construír este anel de polinomios e facer operacións módulo o ideal, ou, utilizar directamente a implementación de GF(16) e as opcións para cambiar dunha a outra representación, que iremos usando a medida que sexan útiles.

B.2. O algoritmo

Inicia coa xeración das claves. A continuación, tomando como k_0 a clave inicial, súmaa ao texto claro, para comezar coa primeira rolda. Nesta primeira rolda, lévanse a cabo as substitución de cuartetos, os desprazamentos nas filas e a substitución por columnas e finalmente, súmase o resultado con k_1 para passar á segunda rolda. Nesta segunda rolda, farase de novo substitución de cuartetos e desprazamentos nas filas e, sen mestura por columnas, súmase con k_2 para obter o texto cifrado.

- Comezamos coa **xeración das claves** k_0, k_1 e k_2.
- A continuación –o que pode chamarse **rolda 0**–: o algoritmo comeza sumándolle k_0 ao texto claro e pasa á rolda 1.
- **Rolda 1**: aplícálle a cada catro bits unha permutación **SubCuart**. Despois, vendo os 16 bits como unha matriz 2×2 con entradas de 4 bits, aplícanse dúas transformacións, **ShiftRows** e **MixColumns**, rolda 1 remata tras sumar a claves de dita rolda, k_1.
- **Rolda 2**: aplícase de novo **SubCuart** a cada cuarteto e despois **ShiftRows**, para rematar a rolda 2 coa suma da clave da rolda 2 (esta vez sen MixColumns)

B.2.1. SubCuart

A parte máis importante do algoritmo, en termos de seguridade, é a S-Box que cambia os cuartetos en que se divide o texto plano. Como comentamos, na implementación que

existe en **SageMath** seguindo [3], é unha permutación de $\mathbb{F}_{16}$ que usa a táboa

$$\mathrm{S0} = [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7]$$

aquí representada con números do 0 ao 15. $\mathrm{S0}(0) = 14,\ \mathrm{S0}(1) = 4,\ \mathrm{S0}(2) = 13,\ \ldots$ que é fácil utilizar para definir a correspondente versión para cuartetos (listas de 4 bits).

```
S0 = [14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7]
"""
pasando a cadeas as cifras dos números de S0
"""
taboa0=
[''.join([str(i) for i in a.digits(2,padto=4)]) for a in S0]
taboa0
```

```
['0111', '0010', '1011', '1000', '0100', '1111', '1101', '0001', '1100',
'0101', '0110', '0011', '1010', '1001', '0000', '1110']
```

```
def NS0(c):
    """
    NS (Nibble Substitution, na teminoloxía do autor na referencia
    que estamos a empregar): actúa sobre c unha lista de catro bits
    """
    pos = sum([int(c[i])*2^(i) for i in range(4)])
    # convertimos en número a cadea
    return [int(s) for s in taboa0[pos]]
    # devolvemos a cadea que ocupa esa posición
```

```
NS0([1,1,0,0])   # o elemento que ocupa a posición 3 da táboa
```

```
[1, 0, 0, 0]
```

Como dicíamos no apartado 3.3, e como se pode ver, por exemplo, en [5], cando se fala de AES, esta permutación ten unha estrutura alxébrica interesante: é unha aplicación afín (mutiplicar por unha matriz e sumar un vector) pero aplicada ao inverso en $\mathbb{F}_{16}$ da lista de partida (tomando [0,0,0,0], se se trata da lista [0,0,0,0]). É dicir, unha función SC tal que

$$\mathrm{SC}(n) = \mathrm{inv}(n) * M + V$$
$$\text{onde } V = \mathrm{NS0}([0,0,0,0])$$

Primeiro definiremos a función **inv** e despois, para determinar a matriz M, terase en conta que $e_i * M = \text{SC}(\text{inv}(e_i)) + V$ é a fila i-ésima de M.

```
F16.<x> = GF(16)                    # corpo de Galois de 2^4 elementos
def inv(n):                         # lista con catro bits
    if n == [0,0,0,0]: return [0,0,0,0]
    else:
        num = sum([n[i]*2^(i) for i in range(4)])
        # enteiro que en binario ten por cifras a n
        p = F16.fetch_int(num)  # representación de num en F16
        q = p.inverse_of_unit() # inverso de i (!= 0)
        d = ZZ(q.integer_representation())
        # número enteiro que corresponde a q en F_16
        ou = d.digits(2,padto=4)
        # cifras en binario (en orde inversa) completadas ata 4
    return ou
```

```
inv([1,0,1,1])
```

```
[0, 0, 1, 0]
```

A función SC debería de coincidir coa permutación NS0 dada pola táboa, non obstante, acaba de probarse que non é así. Intentouse buscar, polo tanto, a matriz M e o vector V que fixeran que dita aplicación coincidise con NS0, pero chegouse a que a matriz resultante non sería inversible e, polo tanto, non definiría unha permutación.

En [3], o autor di que esa táboa se obtén da primeira fila da matriz que se usa para a S-Box S0 na descrición de DES. Vexamos se tomando a primeira fila da S-Box S1 e repetindo o proceso nos serve

S1 = [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10]

```
S1 = [15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10]
T1 = [''.join([str(i) for i in a.digits(2,padto=4)]) for a in S1]
def NS1(n):
    pos = sum([int(n[i])*2^(i) for i in range(4)])
    return [int(s) for s in T1[pos]]
```

```
N1 = NS1([0,0,0,0])
P1 = []
```

```
for i in range(4):
    """
    lista de 4 bits do inverso de e_i
    """
    inv_ei = inv((2^i).digits(2,padto=4))
    """
    gárdase en P0 a lista de catro bits coa imaxe por S01 do
    inverso de e_i
    """
    P1.append(NS1(inv_ei))
P1 = matrix(GF(2),P1)
# matriz que ten por filas estes valores pos S01
Q1 = matrix([N1,N1,N1,N1])
# matriz con filas iguais a N0 para sumarlle a P0
M1 = P1+Q1
# matriz buscada para a transformación afín
V1 = matrix(GF(2),N1)
# vector da transformación (como matriz fila)
M1
M1.determinant()
```

```
[0 1 1 1]
[0 0 0 1]
[1 1 1 1]
[1 0 1 0]
1
```

B.2.2. A nova substitución para cuartetos

Multiplicar por M1 e sumar V1 si define unha permutación, vexámolo:

```
def SC(n):
    """
    substitución para cuartetos
    """
    v = matrix(inv(n))
    return (v*M1 + V1).list() # para que a saída sexa unha lista
```

```
SC([1,0,0,0])
```

```
[1, 0, 0, 0]
```

B.2.3. Comparación das permutacións NS1 e SC

```
TSB1 = []
cab = []
for i in [0..7]:
    c = i.digits(2,padto=4)
    cab.append(''.join([str(s) for s in c]))
    TSB1.append(''.join([str(s) for s in SC(c)]))
cab
print('------------------------------------------------------')

taboa1[:8]
TSB1
print('\n')
TSB2 = []
cab = []
for i in [8..15]:
    c = i.digits(2,padto=4)
    cab.append(''.join([str(s) for s in c]))
    TSB2.append(''.join([str(s) for s in SC(c)]))
cab
print('------------------------------------------------------')

taboa1[8:]
TSB1
```

```
['0000', '1000', '0100', '1100', '0010', '1010', '0110', '1110']
------------------------------------------------------------
['1111', '1000', '0001', '0111', '0110', '1101', '1100', '0010']
['1111', '1000', '0010', '1011', '1101', '0011', '0110', '0001']

['0001', '1001', '0101', '1101', '0011', '1011', '0111', '1111']
------------------------------------------------------------
['1001', '1110', '0100', '1011', '0011', '0000', '1010', '0101']
['1111', '1000', '0010', '1011', '1101', '0011', '0110', '0001']
```

Pódese observar que non coincide en todas partes, pero si o fan nos inversos dos vectores da base canónica, é dicir, $[0,0,0,0]$, $[1,0,0,1]$, $[1,0,1,1]$ e $[1,1,1,1]$.

B.3. Funcións auxiliares

Definiremos unha serie de funcións que nos permiten cambiar de modo de representacion dos datos e que podemos usar para simplificar o código das definicións.

- **cuart_to_pol**: pasar de cuartetos a polinomios
- **pol_to_cuart**: pasar de polinomios a cuartetos
- **oct_to_char**: pasar de 8 bits a caracter
- **char_to_oct**: pasar de caracter a 8 bits

```
def cuart_to_pol(c):
    return F16(c)
```

```
def pol_to_cuart(p):
    a = p.integer_representation()
    n = ZZ(a)
    return n.digits(2,padto=4)
```

```
def oct_to_char(o):
    n = ZZ(o,2)
    return chr(n)
```

```
def char_to_oct(s):
    a = ZZ(ord(s))
    return a.digits(2,padto=8)
```

B.3.1. A permutación en notación polinomial

Dado que usaremos multiplicación de matrices en $\mathbb{F}_{16}$ a través da identificación con polinomios, o primeiro que faremos a continuación será dar a versión da permutación anterior en polinomios, para a substitución de cuartetos.

```
def SubCuart(p):
    n = pol_to_cuart(p)
    c = SC(n)
    return cuart_to_pol(c)
```

```
p = F16([1,1,0,0])
SubCuart(p)        # comparar con SC([1,1,0,0])
SC([1,1,0,0])
```

```
x^3 + x^2 + 1
[1, 0, 1, 1]
```

B.4. Encriptado

B.4.1. Xeración das claves de rolda

Na xeración das claves de rolda vimos no apartado 3.3 que interviña unha S-box –explicada nas Figuras 3.6 e 3.7– e actuando, alí sobre bytes, mentres que aquí o fará sobre cuartetos a mesma SC que acabamos de definir. A clave estendida constará de 12 cuartetos, os catro primeiros definen k_0, os catro seguintes, k_1, e os restantes, k_2. Neste caso, os cuartetos serán polinomios, pensados coma elementos do corpo $\mathbb{F}_{16}$.

```
def XenCla(k):
    """
    k é unha lista de 16 bits
    """
    K=[cuart_to_pol(k[4*i:4*(i+1)]) for i in range(4)] # -> F16
    """
    constantes que se suman en cada rolda
    """
    cron1=F16.fetch_int(1)
    cron2=F16.fetch_int(2)
    cron=[cron1,cron2]
    W=K[:]+[0]*8        # engadimos 8 elementos que iremos cambiando
    for i in [4..11]:
        if 1%2==0:
            r = 1//2 - 1
            W[i]=W[i-4]+SubCuart(W[i-1])+cron[r]
        else:
            W[i-4]+W[i-1]
    return W            # lista de 12 polinomios
```

```
XenCla([1,1,0,0,0,1,1,1])
```

```
[x + 1, x^3 + x^2 + x, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

B.4.2. ShifRows e MixColumns

M-AES cifra bloques de 16 bits, considerados formados por catro cadeas de 4 bits, P0 P1 P2 P3, que se pensan como unha matriz da forma

$$\begin{pmatrix} P0 & P1 \\ P2 & P3 \end{pmatrix}.$$

Observación B.1. Obsérvese que se usa a matriz trasposta da que se usa en xeral, cambiando a orde das multiplicacións e transformacións, co único obxectivo de simplificar o paso de lista de 16 bits a matriz de 4 cuartetos, e viceversa. Para a multiplicación destas matrices, úsase o produto en F16, polo que a idea foi utilizar a representación como polinomios destes cuartetos para facer os cálculos.

Estas funcións, ShifRows e MixColumns realizan os seguintes desprazamentos, respectivamente –nótese que os desprazamentos por filas son agora por columnas na matriz trasposta–:

$$\begin{pmatrix} P0 & P1 \\ P2 & P3 \end{pmatrix} \to \begin{pmatrix} P0 & P3 \\ P2 & P1 \end{pmatrix},$$

$$\begin{pmatrix} P0 & P1 \\ P2 & P3 \end{pmatrix} \to \begin{pmatrix} P0 & P2 \\ P1 & P3 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix}.$$

Considerando 2 e 3 como elementos de F16

```
def ShiftRows(A):
    """
    A unha lista de 4 elementos de F16
    """
    return [A[0],A[3],A[2],A[1]]
```

```
def MixColumns(A):
    """
    A unha lista de 4 elementos de F16
    """
    p2 = F16.fetch_int(2)
    p3 = F16.fetch_int(3)
    B = matrix(F16,2,2,[p3,p2,p2,p3])
    return ((matrix(2,2,A))*B).list()
    # para volver a matriz a formato lista
```

B.4.3. Encriptar un bloque de 16 bits

```
def maes_Enc(m,k):          # m e k son listas de 16 bits
    W = XenCla(k)           # xeramos as claves das roldas
    """
    de cada cuarteto facemos un polinomio
    e sumamos cada un deles cos da claves da rolda 0
    """
    A = [F16(m[4*i:4*(i+1)]) for i in range(4)]
    B = [A[i]+W[i] for i in srange(4)]
    """
    comeza a rolda 1, aplicámoslle a cada un deles a permutacion
    """
    C = [SubCuart(p) for p in B]
    D = ShiftRows(C)
    A = MixColumns(D)
    B = [A[i]+W[4+i] for i in range(4)] # remata a rolda 1
    C = [SubCuart(p) for p in B]
    A = ShiftRows(C)
    B = [A[i]+W[8+i] for i in range(4)] # remata a rolda 2
                                       # (sen MixColumns)
    R = []
    for p in B:
        c = pol_to_cuart(p) # convertimos en cuarteto cada polinomio
        R += c
    return R
```

```
m = [0,1,0,0,1,1,0,1,0,1,1,1,0,0,1,0]
k = [0,1,1,1,0,0,1,0,1,1,1,0,0,0,1,1]
maes_Enc(m, k)
```

```
[1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0]
```

B.4.4. Encriptar un texto

Unha posibilidade é empregar a seguinte función.

```
def Enc_mAES(s,k):
    """
    a idea é de cada dous caracteres facer 16 bits
    (8 de cada un deles)
    """
```

```
    if len(s)%2==1:
        s+=' '
        TP = ''
    j = 0
    while j+2<=len(s):
        m = (ZZ(ord(s[j]))).digits(2,padto=8)+
            (ZZ(ord(s[j+1]))).digits(2,padto=8)
        c = maes_Enc(m,k)
        TP += chr(ZZ(c[:8],2))+chr(ZZ(c[8:],2))
        # caracter que lle coresponde a cada octeto
        j += 2
    return TP
```

```
texto =
'''Tampouco sei si o sol brila, ou se o que brila es ti;
mais sei que se non te vexo sempre é noite para min'''
claves = [0,1,1,1,0,0,1,0,1,1,1,0,0,0,1,1]
Enc_mAES(texto, claves)
```

```
'a6NgozcTImPlFwpbqjXuhohT9G93J3vWpgX3RJIPbUkVh07xThj4yz591zB+Yba8QVXdB5ele0vO
1pN6GOkMIdAj8xWhAEEpwVxYEYfQz5AhrqoDDSCSbcllzdGrW08fOp7zMd+l3eEmtqk7qoG33fjW/
gaFR4RSO6HbMrTeZwE='
```

B.5. Desencriptado

Comézase coa xeración das claves de rolda, coma no encriptado, pero agora van ser aplicadas en orde inversa. Como MixColumns e ShiftRows son autoinversibles, só se require da inversa de SubCuart, definida a continuación.

```
def SC_1(n):                    # n un cuarteto
    """
    convertimos n en matriz e restámoslle V1,
    multiplicamos polo inverso de M1
    e recuperamos a lista co nentradas enteiras
    """
    u = matrix(n) + V1
    v = u*M1.inverse()
    w = [ZZ(a) for a in v.list()]
    return inv(w)                # o inverso
```

```
def SubCuart_1(p):                    # p en notación polinomial
    a = p.integer_representation()    # recuperar notacion de número
    b = ZZ(a)                         # convertilo nun número enteiro
    n = b.digits(2,padto=4)           # a lista de 4 bits
    c = SC_1(n)                       # a lista asociada
    return F16(c)                     # representación como polinomio
```

```
SubCuart_1(F16([1,1,0,0]))
```

```
x^3
```

```
# comprobemos nun exemplo que son inversas
p = F16([1,1,0,1])
p
SubCuart_1(SubCuart(p))
SubCuart(SubCuart_1(p))
```

```
x^3 + x + 1
x^3 + x + 1
x^3 + x + 1
```

B.5.1. Desencriptar un bloque de 16 bits

```
def maes_Dec(m,k):            # m e k son listas de 16 bits
    W = XenCla(k)             # xeramos as claves das roldas
    """
    A: de cada cuarteto facemos un polinomio
    B: sumamos cada un deles cos da claves da rolda 2
    D: rolda 1: aplicámoslle a cada polinomio a permutacion
    """
    A = [F16(m[4*i:4*(i+1)]) for i in range(4)]
    B = [A[i]+W[8+i] for i in srange(4)]
    C = ShiftRows(B)
    D = [SubCuart_1(p) for p in C]
    A = MixColumns(D)
    B = [A[i]+W[4+i] for i in range(4)] # fin rolda 1
    C = [SubCuart_1(p) for p in B]
    A = ShiftRows(C)
    B = [A[i]+W[i] for i in range(4)]   # fin rolda 2 (sen MixCol)
```

```
    R = []
    for n in B:
        c = pol_to_cuart(n) # convertimos en cuarteto cada polinomio
        R += c
    return R
```

```
maes_Dec([1]*16,[0,1,1,0]*4)
```

```
[1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1]
```

```
maes_Enc(maes_Dec([1]*16,[0,1,1,0]*4),[0,1,1,0]*4)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

B.5.2. Desencriptar un texto

```
def Dec_mAES(s,k):
    if len(s)%2==1:
        s+=' '
    TP = ''
    j = 0
    while j+2<=len(s):
        m = (ZZ(ord(s[j]))).digits(2,padto=8)+
            (ZZ(ord(s[j+1]))).digits(2,padto=8)
        c = maes_Dec(m,k)
        TP += chr(ZZ(c[:8],2))+chr(ZZ(c[8:],2))
        j += 2
    return TP
```

```
plano = '''Tampouco sei si o sol brila, ou se o que brila es ti;
mais sei que se non te vexo sempre é noite para min'''
claves = [0,1,1,1,0,0,1,0,1,1,1,0,0,0,1,1]
cripto = Enc_mAES(plano,claves)
Dec_mAES(cripto,claves)
```

```
'Tampouco sei si o sol brila, ou se o que brila es ti;
mais sei que se non te vexo sempre é noite para min'
```

Apéndice C

Implementación dos algoritmos do RSA en SageMath

C.1. Algoritmo de Euclides

O algoritmo de Euclides para o cálculo do máximo común divisor de dous números a e b, baséase no seguinte resultado, que usaremos para dar un algoritmo recursivo,

$$\text{mcd}(a, b) = \text{mcd}(b, a \mod b) :$$

```
def mcd(a,b):
    if b == 0: return a
    else: return mcd(a,a%b)
```

C.2. Algoritmo de Euclides estendido

O algoritmo de Euclides estendido permite determinar os valores de s e t do Teorema de Bézout –que conta co seguinte enunciado–: dados a a b, enteiros, existen s e t, tamén enteiros, tales que

$$\text{mcd}(a, b) = sa + tb.$$

De novo usaremos un algoritmo recursivo para determinalo, neste caso basearase no feito de que se

$$s_1 b + t_1(a \mod b) = \text{mcd}(b, a \mod b) = \text{mcd}(a, b) = sa + tb,$$

entón podemos despexar s e t obtendo $s = t_1$ e $t = s_1 - t_1 q$, onde q é o cociente enteiro de a entre b.

```
def mcdx(a, b):
    if b == 0 :
        return a,1,0
    else:
        m,s1,t1 = mcdx(b, a%b)
        s = t1
        t = s1 - t1*(a//b)
        return m,s,t
```

```
mcdx(15,6)
```

```
(3, 1, -2)
```

```
xgcd(15,6)                # a función implementada en {\tt SageMath}
```

```
(3, 1, -2)
```

C.3. Inverso modular

Unha consecuencia inmediata do Teorema de Bézout –enunciado recentemente– é que se x e m son dous números enteiros sen divisores en común, entón existe y tal que $xy \equiv 1$ mód m, é dicir,

$$\mathbb{Z}_m^* = \{x \in \mathbb{Z}_m / \text{mcd}(x, m) = 1\}$$

onde $\mathbb{Z}_m^*$ e o grupo das unidades do anel $\mathbb{Z}_m$.

Podemos aproveitar o algoritmo estendido de Euclides para determinar o inverso de x módulo m, sempre que exista.

```
def inv_mod(x,m):
    mc, s, t = mcdx(x,m)
    if mz != 1: print ('non existe inverso')
    else: return s
```

C.4. Exponenciación modular

O algoritmo de RSA utiliza a exponenciación de números moi grandes en aritmética modular, polo que non é factible calcular $x^e \mod m = (x \mod m) \times (x \mod m) \times (x \mod m) \times \cdots$ e moito menos facer $(x^e) \mod n$.

Implementaremos en SageMath o algoritmo de exponenciación rápida, ou exponenciación binaria, un algoritmo que reduce drasticamente o número de operacións necesarias

para levar a cabo os cálculos. A idea é que, se $e = e_0 + e_1 2^1 + e_2 2^2 + \cdots + e_k 2^k$ terase que

$$\begin{aligned} b^{e_0+e_1 2^1+e_2 2^2+\cdots+e_k 2^k} &= b^{e_0} \times b^{e_1 2^1} \times b^{e_2 2^2} \times \cdots \times b^{e_k 2^k} \\ &= \left(b^{2^0}\right)^{e_0} \times \left(b^{2^1}\right)^{e_1} \times \left(b^{2^2}\right)^{e_2} \times \cdots \times \left(b^{2^k}\right)^{e_k}. \end{aligned}$$

Basta entón con observar que:

1. Se $e_i = 0$ o factor $\left(b^{2^i}\right)^{e_i}$ toma o valor 1, polo que non é necesario multiplicalo.
2. As bases son, cada unha, o cadrado da anterior, polo que a función pode ser así:

```
def exp_mod(b,e,n):
    E = e.digits(2)       # a lista de cifras de e en base 2
    t = len(E)            # a cantidade de cifras
    base = mod(b,n)       # dende aquí, opera en Zn
    num = 1               # para irmos multiplicando
    for i in range(t):
        if E[i] == 1:     # se a correspondente cifra é 1
            num *= base   # multiplicamos pola base
        base *= base      # elevando a base ao cadrado
    return num
```

C.5. Tests de primalidade

O **Teorema Pequeno de Fermat** (para p primo e a coprimo con p, $a^{p-1} \equiv 1 \pmod{p}$) proporciona unha condición necesaria para que un número sexa primo. Un número n composto para o que existe a coprimo con el, e tal que $a^{n-1} \equiv 1 \pmod{n}$, chámase **$a$-pseudo primo**. A existencias de *números de Carmichael* (números que pasan o test de Fermat para todas as bases posibles e son compostos) fai que este test non sexa fiable. Non obstante, hai unha mellora posible para este test baseada no seguinte resultado:

Se p é un número primo impar, $p - 1 = 2^s t$, con t impar, e a coprimo con p. Entón, ou $a^t \equiv 1 \pmod{p}$, ou $a^{2^i t} \equiv -1 \pmod{p}$, para algún i, con $0 \leq i \leq s - 1$. Un número composto que pasa este test para algún a chámase **a-pseudo primo forte**. Todo a-pseudo primo forte é a-pseudo primo, pero hai moitos menos pseudo primos fortes ca pseudo primos e, en particular, non existen pseudo primos fortes con respecto a todas as bases.

Baseado neste resultado temos o seguinte Test de Primo Altamente Probable (TPAP): se para algún a non pasa o test, non é primo, se o pasa, ou é primo ou é a-pseudo primo forte.

```
def TPAP(n,a):
    """
    n >3 e 1< a < n-1
    """
    s = 0
    t = n-1
    while t % 2 == 0:
        t /= 2
        s += 1
    u = mod(a^t, n)
    if u == 1: return True
    i = 1
    while (u != 1) and (u != n-1) and (i < s):
        u = u^2 % n
        i += 1
    if u == n-1: return True
    else: return False
```

```
TPAP(77,2)
TPAP(23,5)
```

```
False
True
```

Comprobemos que hai poucos 2-pseudo primos fortes menores ca 5000, só 4.

```
[n for n in range(3,5000,2) if TPAP(n,2) and not is_prime(n)]
```

```
[2047, 3277, 4033, 4681]
```

Como comentamos antes, non existen números que sexan b-pseudo primos fortes para todas as bases e, de feito, o **Teorema de Monier-Rabin** afirma que para un número composto n, o número de bases b para os que n pode ser b-pseudo primo forte é como moito $\phi(n)/4$, onde $\phi(n)$ é o número de unidades de $\mathbb{Z}_n$.

Baseado nisto, e usando o TPAP, temos o que se coñece como **test de Miller-Rabin**, que consiste en someter un número n a unha cantidade $k > 0$ de bases collidas aleatoriamente entre as posibles.

```
def MillerRabin(n,k):
    """
    n > 4 e k > 0
    """
    for i in range(k):
        b = randint(2,n-2)
        if not TPAP(n,b):
            return False
    return True
```

```
MillerRabin(2047,2)
```

```
true
```

Aínda que é non determinista, poderá estudarse con $k = 4$, por exemplo, que números menores de 9000 pasan o test.

```
[n for n in range(5,9000,4) if MillerRabin(n,2) and not is_prime(n\
    )]
```

```
[]
```

Apéndice D

Implementación de RSA en SageMath

D.1. Xeración de claves

A implementación que faremos a continuación é puramente ilustrativa, sen mención explícita a criterios de seguridade, coma o tamaño dos primos, o formato da entrada ou saída dos datos e outros detalles que poden verse nas sucesivas versións dos *Public-Key Cryptography Standars* (**PKCS #1**).

Comezamos cun algoritmo de xeración de claves. A partir do parámetro k buscamos primos, p e q, ao azar, con k bits e calcúlase $n = p \cdot q$. Tómase $e = 2^{16} + 1$ e verifícase que sexa invertible módulo $\Phi(n) = (p-1) \cdot (q-1)$. A elección de e é arbitraria, pero $2^{16} + 1$ é primo, o que aumenta as posibilidades de existencia de d, ademáis ten poucos 1 na súa expansión binaria, o que simplifica a exponenciación modular como vimos no apartado C.4.

```
def xen(k):
    p = randint(2^(k-1),2^k)
    while not is_prime(p):
        p = randint(2^(k-1),2^k)
        q = randint(2^(k-1),2^k)
    while not is_prime(q):
        q = randint(2^(k-1),2^k)
    n = p*q
    e = 2^16 + 1
    fi = (p-1)*(q-1)
    if gcd(e,fi) == 1:
        d = inverse_mod(e,fi)
        return [[n,e],[n,d]]
    else: return xen(k)
```

D.2. Cifrado e descifrado con RSA básico

Tomando como entradas de texto plano e criptotextos unha cadea de bits, verase que o número que corresponde a ese texto plano é menor ca n, pois os cálculos son módulo n.

```
def RSA(b,pk):
    """
    para cifrar unha cadea de bits correspondente
    a un número inferior ó módulo da claves pública
    """
    n, e = pk
    x = ZZ(b,2)
    if x >= n: print('número moi grande')
    else:
        y = power_mod(x,e,n)
    return y.digits(2)
```

Para o descifrado non se requirirá de control de tamaño, xa que só se empregará con cadeas que proceden de RSA.

```
def ASR(c,sk):
    n,d = sk
    y = ZZ(c,2)
    x = power_mod(y,d,n)
    return x.digits(2)
```

```
pk1, sk1 = xen(256)
pk1
sk1
```

```
[73462966250187629275257593265273439236229576771119919853935525297492119094822161071707116270477115651709511664737327206268160461942469576260118448884497
7,65537]
[73462966250187629275257593265273439236229576771119919853935525297492119094822161071707116270477115651709511664737327206268160461942469576260118448884497
7,
23844610190792853654321669094311452461103003733883551177400424941831716980308348046129404170786655375985216098497322505832786787398368429475646352224991
69]
```

```
m1 = 1000.bits(2)
c1 = RSA(m1,pk1)
ASR(c1,sk1) == m1
```

```
True
```

D.3. Padded RSA

Presentaremos unha implementación escolar para visualizar como se podería converter RSA nun algoritmo non determinista. Non faremos, polo tanto, ningún estudo do tipo de recheo nin análise das súas propiedades criptográficas.

A idea consistirá en engadirlle unha cantidade fixa de bits ao texto plano antes de preceder ao seu cifrado con RSA. Unha opción é engadirlle exactamente k bits, e será coa que traballemos. No contexto desta implementación, usamos $k = 256$, pero deixamos dentro do algoritmo a variable k, que podería tamén ser introducida como parámetro.

Observación D.1. Cabe ter en conta que:

1. Debido a que a cadea correspondente ó texto plano se completa con k bits pola esquerda, o tamaño dos números a cifrar vese reducido para aplicar RSA. Non incluímos test, porque ó aplicar RSA salta a mensaxe de erro se o tamaño excede o permitido.

2. randint devolve un elemento na clase int de Python, que non admite o método .digits(). Por iso o convertemos a un enteiro de `SageMath`, vía ZZ().

```
def pRSA(b,pk):
    k = 256
    r = ZZ(randint(0,2^k-1)).digits(2,padto=k)
    x = r + b
    return RSA(x,pk)
```

O descifrado lévase a cabo directamente co descifrado básico, bastará con eliminar os k bits menos significativos do número resultante.

```
def pASR(c,sk):
    k = 256
    return ASR(c,sk)[k:]
```

```
c2 = pRSA(m1,pk1)
pASR(c2,sk1) == m1
```

```
True
```

D.4. Cifrado de textos

Os algoritmos anteriores están pensados para traballar con cadeas de bits. Se queremos aplicarllos a textos directamente, podemos primeiro fabricar un número a partir dos caracteres e, se é maior ca n, separalo en bloques para cifrar cada bloque por separado. Para isto contamos con ord que lle asocia a cada carácter un número do 0 a 255. Con esta lista de números, pensándoa como unha lista de cifras en base 256, xa podemos fabricar o noso número. O seguinte paso será determinar as súas cifras en base n, que cifraremos para inverter o proceso.

```
def RSAt(m,pk):
    n,e = pk
    x256 = [ord(i) for i in m]
    x = ZZ(x256,256)
    xn = x.digits(n)
    x2 = [a.digits(2) for a in xn]
    y2 = [RSA(b,pk) for b in x2]
    yn = [ZZ(s,2) for s in y2]
    y = ZZ(yn,n)
    y256 = y.digits(256)
    c = ''.join([chr(m) for m in y256])
    return c
```

```
texto = '''En vez de finxir, ou estrelarme unha copa de celos,
deulle por rir.'''
print(RSAt(texto,pk1))
```

```
    ßÅ ·@ ¡¯  èí ýI´  ÙØ¬ó:·Ü_æ6ÔXD  '  hÈ]úÕN²ç(ÝcÞå  4s²J ÚL.nÁ
   ¥    ÚWVX_¸K?   µ*Þ °¹dqíÜeÇ ã½¦  c-Åð Ê˜ÿUU´  ÿ!Rà m}KO =1-=â
```

```
def ASRt(c,sk):
    n,d = sk
    y256 = [ord(i) for i in c]
    y = ZZ(y256,256)
    yn = y.digits(n)
    y2 = [a.digits(2) for a in yn]
    x2 = [ASR(b,sk) for b in y2]
    xn = [ZZ(s,2) for s in x2]
    x = ZZ(xn,n)
    x256 = x.digits(256)
    c = ''.join([chr(m) for m in x256])
    return c
```

```
ASRt(RSAt(texto,pk1),sk1)
```

```
'En vez de finxir, ou estrelarme unha copa de celos, deulle por rir.'
```

No caso de padded RSA, sería similar. Para permitir o recheo con k bits, en lugar de separar o número tomando as súas cifras en base n, tomarémolas en base 2^{k-1}, para ter a seguridade de que co recheo non pasamos de n.

```
def pRSAt(m,pk):
    k = 256
    n = pk[0]
    x256 = [ord(i) for i in m]
    x = ZZ(x256,256)
    xk = x.digits(2^(k-1))
    x2 = [a.digits(2) for a in xk]
    y2 = [pRSA(b,pk) for b in x2]
    yn = [ZZ(s,2) for s in y2]
    y = ZZ(yn,n)
    y256 = y.digits(256)
    c = ''.join([chr(m) for m in y256])
    return c
```

```
print(pRSAt(texto,pk1))
```

```
def pASRt(c,sk):
    k = 256
    n = sk[0]
    y256 = [ord(i) for i in c]
    y = ZZ(y256,256)
    yn = y.digits(n)
    y2 = [b.digits(2) for b in yn]
    x2 = [pASR(s, sk) for s in y2]
    xk = [ZZ(t,2) for t in x2]
    x = ZZ(xk,2^(k-1))
    x256 = x.digits(256)
    m = ''.join([chr(u) for u in x256])
    return m
```

```
pASRt(pRSAt(texto,pk1),sk1)
```

```
'En vez de finxir, ou estrelarme unha copa de celos, deulle por rir.'
```