Universidad Nacional de La Matanza

Criptografía y Criptoanálisis. Desde los cifrados clásicos hasta la actualidad.

Licenciatura en Matemática Aplicada

Trabajo Final de Grado

Autor: Alba, Maximiliano Emanuel

Profesores:

Lic. Jorge Barreto

Dr. Galardo Osvaldo

Año 2018

INDICE

Resumen

En este trabajo se realizará un recorrido por la criptografía, desde algunos cifrados clásicos que fueron significativos en su momento, y actualmente tienen sentido histórico, hasta algunos de los modernos que se utilizan en la actualidad. También, se analizará en general, las debilidades que tienen los cifrados, si quisiéramos intentar vulnerarlos. De esta forma, podremos comprobar de que depende la seguridad de cada uno y en particular, de que depende la seguridad de nuestras comunicaciones.

Introducción

¿Qué entendemos por criptografía?

La Real Academia Española, la define de la siguiente manera:

"*Del gr. κρυπτός kryptós 'oculto' y -grafía.*

Arte de escribir con clave secreta o de un modo enigmático."

"*Criptografía: Emplean este modo de escribir especialmente los estadistas, los príncipes, los diplomáticos, etc., con el fin de asegurar el secreto de su correspondencia, para el caso que pueda caer en manos enemigas o extrañas. […]*

La Criptografía también ha sido llamado este arte: Criptología, de kruptus, oculto, y logos, discurso; Poligrafía, de polus, muchos y grafos, escribir"

Sin embargo, actualmente, la criptografía ya no es utilizada por un cierto grupo de personas, sino que se aplica en todas nuestras comunicaciones para garantizar su seguridad.

El origen de ella si bien no se puede precisar, se sospecha que es anterior al 2500 a.C, se ha encontrado una tablilla babilónica en donde se ha eliminado la primera consonante, o aparecen caracteres en variaciones poco habituales. Esta tablilla se descubrió que tenía oculta la elaboración de la cerámica vidriada, en ese entonces los alfareros y comerciantes tenían secretos de su oficio y no querían que se divulguen.

Codificación: método de escritura en clave que consiste en sustituir unas palabras por otras.

Cifrado: se transforma el mensaje original en otro ilegible, mediante un cierto método. Estas formas de cifrar un mensaje dan origen a la variedad de criptosistemas que se conocen.

Descifrado: reconstrucción del texto original a partir del cifrado.

Por lo tanto, la finalidad de la criptografía es mantener la privacidad de la información que contiene un mensaje, haciéndolo ver ilegible a simple vista y también asegurar la reconstrucción del mismo por parte del destinatario.

¿Por qué puede ser necesario mantener la privacidad en una comunicación? Acá entran en juego muchos factores, en la antigüedad comenzó para compartir

secretos entre gobernadores en una guerra para tener una ventaja estratégica en la misma. En la actualidad su utilidad es fundamental, desde el uso de tarjetas de crédito, transacciones bancarias, hasta el envío de correos electrónicos, y compras por internet. Casi todo servicio que se pueda acceder por internet cuenta con un sistema que utiliza a la criptografía para garantizar la seguridad.

Como bien dice una frase atribuida a Francis Bacon "La información es poder", y depende quien la posea y las intenciones que tenga, podría tener una seria influencia en el desarrollo de cualquier acontecimiento. Por este motivo, siempre existirán personas que por diversas razones necesitan descifrar esos mensajes, y me refiero a mensajes que no necesariamente están destinados para estas. Por ejemplo, el interceptar mensajes en una guerra y saber su misterioso contenido puede definirla o al menos otorgar una gran ventaja táctica, hecho verídico en que el descifrado de la maquina Enigma gracias a Alan Turing y su equipo, logró que la Segunda Guerra Mundial haya durado dos años menos de lo que se había estimado.

Ahora nos surge inmediatamente la siguiente pregunta: ¿Cómo descifrar un mensaje si no sabemos su clave, si la forma en que está cifrado fue para que otra persona que no sea el remitente no pueda leerlo? En la historia muchas personas se hicieron la misma pregunta y esto dio lugar al criptoanálisis.

Veamos dos definiciones para entender lo que es el criptoanálisis:

"*Criptoanálisis: (Del gr. Kryptós, oculto y análisis) m. Arte de descifrar criptogramas.*"

"*Criptograma: m. Texto cifrado para transmitir informaciones o instrucciones, de forma que no sean compresibles para quien no disponga de la clave. También cifra.*"

El criptoanálisis, buscará debilidades en los cifrados o sistemas criptográficos, y así intentará descifrarlos u obtener información que pueda ser útil a posteriori como las claves, por ejemplo.

Objetivos del trabajo

A lo largo del trabajo nos focalizaremos en los cifrados criptográficos y veremos como el criptoanálisis (a veces) puede descifrar su contenido exitosamente. Evidentemente si un cifrado es susceptible al criptoanálisis ese cifrado no es seguro, y no cumple su propósito por lo cual no tendría sentido usarlo. Sin embargo, el criptoanálisis aparece posteriormente luego de mucho trabajo sobre los textos cifrados.

El trabajo está dividido en tres partes: en la primera se analizará la criptografía clásica con una idea de como efectuar su criptoanálisis, en la segunda se establecerá detenidamente las bases teoricas (propiedades de aritmética modular, teoremas fundamentales, residuos cuadráticos y reciprocidad cuadrática) y fundamentación de algunos algoritmos para la tercera parte en donde los utilizaremos; en la tercera parte veremos algunos cifrados modernos, y su criptoanálisis en donde se expondrán algoritmos para analizar la primalidad de un número.

Como objetivo del trabajo, además de hacer un recorrido por la criptografía y el criptoanálisis, se analizarán algoritmos viendo la matemática que los sustenta y se crearán programas: los de la primera parte tienen como idea mostrar el funcionamiento de algunos cifrados clásicos; en la segunda parte la creación de un programa de utilidad que nos permitirá calcular en un módulo dado: sumas, productos, potencias, factoriales, como también mcd, la combinación lineal del mcd, y el cálculo del símbolo de Jacobi; en la tercera parte, principalmente, tres test probabilísticos de primalidad, con un automatización de los mismos para generar una lista de números pseudoprimos.

¿Cifrar o encriptar?

Existe un debate y confusión al respecto, hay quienes defienden que la palabra correcta es cifrar y descifrar diciendo que encriptar y desencriptar son anglicismos de los términos ingleses encrypt y decrypt. Sin embargo, aunque hace unos años atrás encriptar no estaba aceptado en el diccionario de la Real Academia española, hoy en día su uso está aceptado.

“Encriptar: Del ingl. to encrypt; cf. gr. ἐγκρύπτειν enkrýptein 'ocultar'.

1. tr. cifrar (‖ transcribir con una clave).”

(Extraído de http://dle.rae.es/?id=FABu3oz)

En el trabajo se usará la palabra cifrado para referirnos a este proceso.

El proceso de comunicación (cuando la información es cifrada)

¿Qué es lo que ocurre desde que se envía un mensaje cifrado a un destinatario?

El emisor quiere enviar un mensaje al receptor de forma privada, por lo cual el emisor cifra al mensaje mediante un cifrado el cual consiste en un algoritmo criptográfico que puede o no ser públicamente conocido y el uso de una clave donde ésta será la seguridad de esa privacidad. El mensaje ya cifrado pasará a manos del receptor quien sabrá el algoritmo que se usó para cifrarlo y usará una clave (que puede ser la misma o no, que al momento del cifrado) para volver a tener el mensaje original.

Tenemos entonces, mensaje original -> mensaje ilegible (para que nadie excepto el o los receptores puedan saber lo que dice) -> mensaje original.

Sin embargo, a la hora de ponerse en práctica presenta ciertas consideraciones. De tener que enviar un mensaje de forma urgente, el cifrado llevaría mucho tiempo realizándolo a mano, por lo cual hoy en día se realiza a través de una computadora o dispositivos como teléfonos celulares, tabletas, PDAS, que puedan compilar o interpretar lenguajes de programación. En el S.O. Android, un compilador muy bueno de Lenguaje C/C++ es C4Droid el cual se usó en repetidas ocasiones para la elaboración de los programas adjuntos presentes en este trabajo.

La segunda dificultad es el conocimiento del cifrado (siendo por un humano o máquina) y la clave. Una forma de enviar un mensaje cifrado sin usar una clave es que tanto el emisor como el receptor sepan el algoritmo y que éste no sea público. Sin embargo, la seguridad por oscurantismo (así se le dice) no es tan segura como parece, en el momento puede ser útil sin embargo vamos a ver un ejemplo:

Supongamos que dos amigos quieren enviarse un mensaje de forma privada y usan un algoritmo secreto ideado por ellos, logran su cometido. Pero a la semana ellos se pelean y uno de ellos envía un mensaje cifrado a otra persona. Pero ahora si el otro amigo puede interceptar el mensaje sabrá su contenido, y esto en un mundo complejo de comunicaciones no es para nada viable.

Además, el avance actual en Inteligencia Artificial y sus aplicaciones, harían que cualquier algoritmo sin clave sea altamente vulnerable.

La segunda dificultad que mencionábamos es el conocimiento de la clave. Si dos personas usan un cifrado que es seguro pero la clave que descifra es la misma que cifra, la pregunta es ¿Cómo se debe acordar la clave? Pues si la clave es interceptada, todo el mensaje se verá comprometido. Actualmente se habla de clave pública y clave privada, pero antes esto no era así y la clave que cifraba era la misma para descifrar.

La última dificultad que presentan las comunicaciones, es la seguridad en sí de los métodos criptográficos usados, y su respectivo análisis.

Un sistema criptográfico es inseguro cuando, sea cual sea la clave el mensaje original puede ser reconstruido sin necesitarla.

Por lo tanto, en un sistema criptográfico seguro, la clave es un elemento fundamental de su seguridad; a esto se lo conoce como principio de Kerckhoffs.

Hay personas que estudian cómo vulnerar cifrados, se los llama criptoanalistas. Puede ser tanto como para mejorar la seguridad de los cifrados como para poder descubrir la información y usarla a su beneficio, o beneficio de entidades y organizaciones que los financian.

En los cifrados modernos un factor importante es el tiempo, refiriéndonos a la demora en que puede descifrarse un mensaje sin conocer la clave. Pues **toda información es importante en un intervalo de tiempo razonable,** fuera de éste la información ya no tiene demasiada importancia. Por ejemplo, develar los mensajes enemigos en una guerra es importante dentro de la guerra. Cuando la guerra finalizó quizás quede como una curiosidad o hecho histórico, pero ya no tiene la importancia que tenía en ese tiempo.

Comenzaremos el trabajo estudiando a algunos cifrados clásicos, los primeros que se originaron, y detrás de esto también analizaremos el trabajo del criptoanálisis para vulnerarlos, dando ideas de que errores no deben cometerse.

Cifrados Clásicos

Nos referiremos como cifrados clásicos, a los cifrados que fueron utilizados en la antigüedad que ya no son utilizados debido a lo inseguros que son actualmente, en su momento muchos se creyeron irrompibles, pero con el progreso del criptoanálisis, y formas de procesamiento de la información, se estima que tardarían pocas horas en vulnerarse con nuestra tecnología.

En la introducción hemos hablado un poco de la seguridad por oscurantismo. En la antigüedad vemos que surgen los principios de la idea de poner a la información en secreto y comienza con la **esteganografía**: ocultar la existencia misma del mensaje.

El griego Heródoto en sus investigaciones durante el siglo V a.C. Nos relata lo siguiente:

"*Histieo, el tirano de Mileto, ordenó a un hombre que se rapara la cabeza. A continuación, escribió en el cráneo del sujeto el mensaje que deseaba transmitir y esperó a que volviera a crecerle el cabello, momento en el cual despachó al hombre hacia su destino, el campamento de Aristágoras. Una vez a salvo, el mensajero advirtió la treta y, rapada de nuevo la cabeza, mostró el ansiado mensaje a su destinatario*" (Gómez, 2010, p.21)

La esteganografía es una de las ideas que desencadenan la aparición de la criptografía, porque si bien esta última no oculta la existencia del mensaje, oculta la información del mismo. Su objetivo es el mismo, mantener la confidencialidad de la información.

Los primeros cifrados eran rudimentarios, y algunos dieron lugar a la **criptografía por transposición**; por ejemplo, una cinta arrollada a un bastón de cierto diámetro y longitud, con un mensaje escrito sobre ella, luego se escribían caracteres basura en el medio de la cinta y sin tener un bastón de esas dimensiones especificas era imposible descifrarlo. Estos caracteres basura estaban para que, al desenrollarlo, lo que quede sea algo ilegible. Este método de la **escitala** lo utilizaron los espartanos y atenienses.

Sin embargo, al mismo tiempo que surgían estos **cifrados por transposición** aparecieron los **cifrados por sustitución**, los cuales consisten en sustituir letras por otras.

Entre los primeros que se tienen registro de sustitución están el **Cifrado de Polibio** y el **Cifrado de César**.

Cifrado de César

Éste cifrado fue utilizado por Julio César (Roma, 100 a.C – 44 a.C) General y estadista romano. Consiste en reemplazar a cada letra de un mensaje por la que le sigue 3 posiciones adelante. Por ejemplo:

Si quisiéramos cifrar la palabra DURAZNO. La letra que sigue a la D en tres posiciones es la G, por lo que reemplazaríamos a la D por G, U por X, R por U, A por D, Z por C, N por Q y O por R.

¿Por qué a la Z por C? A las letras como la X, Y, y Z cuando se termina el abecedario volvemos a arrancarlo.

Finalmente, nuestra palabra cifrada queda como GXUDCQR.

También podemos cifrar construyendo la siguiente tabla:

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

La primera línea corresponde con nuestro alfabeto, y la segunda al alfabeto cifrado, es decir por cual lo debo reemplazar. En este cifrado sustituimos teniendo cuenta un único alfabeto, por lo cual a este tipo de cifrados se los conoce como **monoalfabéticos**. Veremos más adelante que hay otros cifrados que usan más de un alfabeto para cifrar.

¿Qué tiene que ver la matemática con este cifrado? Si bien aquí no parece actuar, en realidad está, aunque algo escondida.

Vamos a asignarle un número a cada letra de la siguiente forma: A=0, B=1, C=2 … Z=26.

Esta asignación no es más que una función biyectiva del conjunto de las letras al subconjunto de los Naturales (con el cero) comprendidos del 0 al 26 inclusive. Por lo cual a cada letra le corresponderá un único número y viceversa. En el trabajo nos referiremos a esta relación biunívoca como $\boldsymbol{a}$ y aclararemos si tiene 27 o 26 elementos (dependerá si consideramos a la letra ñ o no).

Al cifrado de César se lo puede modelizar matemáticamente mediante la siguiente función.

$$C: A \rightarrow B \ / \ C(x) = x + 3 \ (mod\ 27)$$

Desde ahora vamos a ver a la aritmética modular como algo bastante común en la criptografía. Usamos el módulo 27 pues el abecedario consta de 27 letras, si excluimos la letra ñ, por ejemplo, trabajaríamos con módulo 26.

Si queremos cifrar la letra “Z”, aplicamos la biyección $\boldsymbol{a}$ obteniendo el número 26, luego calculamos:

$$C(26) \equiv 26 + 3\ (mod\ 27) \Rightarrow C(26) \equiv 29\ (mod\ 27) \Rightarrow C(25) \equiv 2\ (mod\ 27)$$

Luego buscamos $\boldsymbol{a}^{-1}$ de 1 y obtenemos la letra C.

¿Cómo desciframos? César reemplazaba a cada letra codificada por la que está 3 lugares antes. Aunque para descifrar también podemos usar una función:

$$C^{-1}(x) = x - 3\ (mod\ 27)$$

Y usaremos también la misma biyección de antes para asignar las letras a números y números a letras.

¿Por qué esa función?

Sea $a \in A,\ b = C(a) \in B$:

Como $C^{-1}(b) = a$:

$$C(a) \equiv a + 3\ (27)\ \Rightarrow C(a) - 3 \equiv a\ (27) \Rightarrow b - 3 \equiv C^{-1}(b)\ (27)$$

La idea de este cifrado de César da lugar a una generalización.

Cifrado César Generalizado

Este cifrado está descrito por la siguiente función:

$$C(x) = x + k\ (mod\ 27)$$

Siendo k la clave del cifrado, en el cifrado de César k=3.

Por ejemplo: Siendo la clave 10, si queremos cifrar la letra A, primero aplicamos la biyección tenemos el 0, y luego 0+10 (mód 27)=10 que le corresponde la letra K.

Para descifrar se generaliza lo que habíamos visto en César.

$$C^{-1}(x) = x - k\ (mod\ 27)$$

Cifrado Afín

Generalizando más al cifrado de César Generalizado se llega al Cifrado Afín. El cual está dado por la siguiente función:

$$C_{(a,b)}(x) = (ax + b)\ (mod\ n)$$

Se pide que mcd (a,n)=1, pues sino se podría cifrar de forma diferente una misma letra ya veremos por qué. La clave en este cifrado es el par ordenado (a,b).

¿Cómo lo desciframos? Vamos a tener que operar un poco. Según Gómez, 2010:

$$C_{(a,b)}(x) = ax + b = y\ (mod\ n)$$

$$ax = y - b\ (mod\ n)$$

$$a^{-1}ax = a^{-1}\ (y - b)\ (mod\ n)$$

$$x = a^{-1}\ (y - b)\ (mod\ n) \quad con\ a^{-1} \mid a^{-1}a = 1$$

Es por esto que se pide que mcd(a,n)=1, si es distinto de 1 no puede asegurarse que existe el inverso de a módulo n, por lo cual no podría descifrarse correctamente.

Si a=1, se tiene el Cifrado de César Generalizado, si además b=3 se tiene el de César.

Cifrado de Polibio

Inventado hacia 150 a. C. por el historiador Polibio, es anterior a César pero lo dejamos para este momento para comprender una cuestión: en el Cifrado de César se reemplazaba una letra por otra, aquí se reemplazará a una letra por un dígrafo. Haber trabajado con César primero nos ayudará a comprender mejor el método de sustitución ya que iniciar con una sustitución de un grafo a un dígrafo puede resultar algo confuso al empezar con estos métodos.

Este cifrado es por ***sustitución monoalfabético monográmico*** *(monográmico porque se reemplaza letra a letra, en particular aquí se reemplaza a cada letra por un dígrafo.)*

El cifrado consiste en realizar una tabla de 5x5, aunque también se puede utilizar una de 6x6 con el alfabeto a usar. Si usamos una de 5x5 las letras I y J las pondremos en una misma casilla y excluiremos la ñ; si usamos una de 6x6 usaremos un alfabeto de 26 letras excluyendo la ñ e incluiremos los números del 0 al 9.

Usaremos una de tabla de 5x5, las letras se pueden colocar de cualquier forma que el receptor al igual que el emisor deben saber. Por lo tanto, la tabla sería la clave de este cifrado. Como encabezado de columnas y filas se usarán las letras de la A hasta la E o también pueden usarse los números del 1 al 5. Supongamos que elegimos crear la tabla de la siguiente forma:

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

Es decir, se propone colocar primero las letras de posición par y luego las impares (esto es irrelevante, pero es algo que le podemos comunicar fácilmente a nuestro receptor).

Supongamos que queremos cifrar la palabra: CEREZA

Observando la fila y la columna de cada letra de la palabra, sustituiremos la C por AB (pues tiene fila A y columna B), luego la E por AC, y así continuaremos… R por BD, E por AC nuevamente, Z por CC y A por AA.

Por lo que nuestra palabra CEREZA quedará cifrada como ABACBDACCCAA.

Para descifrar: construimos la tabla (nuestra clave) y la intersección de las letras de fila y columna en ese orden, nos dirá la letra descifrada. En nuestro ejemplo: AB=C, AC=E ,BD=R, AC=E, CC=Z, AA=A.

Por lo tanto, la palabra original es: CEREZA

Cifrado de Playfair

Creado por el barón Lyon Playfair y Sir Charles Wheatstone, a mediados del siglo XIX.

Este cifrado es por ***sustitución monoalfabético poligrámico*** *(poligrámico porque se reemplaza un conjunto de letras por otro, en particular aquí se reemplazan los dígrafos por otros dígrafos.)*

Al igual que en Polibio también tenemos que construir una tabla de 5x5 o 6x6. Depende el autor, en la tabla de 5x5 se excluye la ñ y se juntan en la misma casilla la I y J, como también la I y K, entre otras combinaciones. En este trabajo tomaremos en una misma casilla la I y K.

La tabla se construirá con una palabra o frase clave, se escribirá en la tabla la palabra/frase quitando las letras que se repitieron y a continuación se ponen las letras restantes del abecedario en orden.

Veamos un ejemplo, si la clave es LOBACHEVSKY le corresponderá la siguiente tabla:

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

Cifraremos a continuación la frase:

NO HAY RAMA DE LA MATEMATICA POR ABSTRACTA QUE SEA QUE NO PUEDA APLICARSE ALGUN DIA A LOS FENOMENOS DEL MUNDO REAL

Para cifrarlo vamos a tener en cuenta lo siguiente:

Según Orga (1986):

"*El proceso es como sigue:*

1. *Se divide el texto en grupos de dos caracteres; si los caracteres en algún grupo son iguales, se separan y al primero de ellos se le añade un carácter poco frecuente […]*
2. *Se van sustituyendo cada par de caracteres, de forma que se pueden dar tres casos:*

2.1. *Que los dos caracteres estén en la misma fila de la matriz de sustitución, en cuyo caso se sustituyen por los dos caracteres inmediatamente siguientes en la fila.*

2.2. *Si están en la misma columna, se sustituyen por los siguientes en la columna de la matriz.*

2.3. *Si no están en la misma fila ni en la misma columna, se sustituyen por los que forman un rectángulo con ellos en la matriz; es decir, los caracteres* $C_{ij}C_{kl}$ *se sustituyen por los caracteres* $C_{il}C_{kl}$ *de la matriz de sustitución."*

Original:

NO HA YR AM AD EL AM AT EM AT IC AP OR AB ST RA CT AQ UE SE AQ UE NO PU ED AA PL IC AR SE AL GU ND IA AL OS FE NO ME NO SD EL MU ND OR EA L

Modificado (apto para aplicar Playfair):

NO HA YR AM AD EL AM AT EM AT IC AP OR AB ST RA CT AQ UE SE AQ UE NO PU ED AX AP LI CA RS EA LG UN DI AX AL OS FE NO ME NO SD EL MU ND OR EA LX

(Es necesario separar el digrafo AA con una X por regla 1, al hacer esto se juntaron otras letras AA que antes estaban separadas, luego también tuvo que agregarse una X entre ellas, y al final para tener una cantidad par de letras se tuvo que agregar otra X)

Comencemos:

“NO” están en la misma columna así que reemplazo por “UE” (por 2.2)

“HA” están en distintas filas y columnas, así que reemplazo por “SL” (por 2.3)

“YR” por “JM”, “AM” por ”LQ”, “AD” por “OG”, “EL” por “HO”, “AM” por “LQ”, “AT” por “LX” (todos estos por 2.3) y así continuamos, en algún momento llegaremos a “SE” que están en la misma fila y lo remplazamos por “IV” (por 2.1), y así terminaremos el mensaje.

Por lo tanto, queda cifrado como:

UESLJMLQOGHOLQLXHNLXJIBQCNCAHXQCLZSXODIVSXODUENWDNSABQCHLCQISOAYOUJESACOAEDVUENHUEEGHONTUNCNSOAT

Para descifrarlo: se crea la tabla como hicimos anteriormente y se realizan los pasos al revés, si antes moví hacia la derecha ahora muevo hacia la izquierda, si moví hacia abajo ahora hacia arriba, etc; reconstruyendo así el mensaje original.

Cifrados Polialfabéticos y el cifrado de Vigenère

Dada la victoria del criptoanálisis frente a los cifrados por sustitución monoalfabéticos, fue entonces cuando surgieron algunos artilugios en los cifrados como el **cifrado de sustitución homofónica** (del cual hablaremos más adelante) y los cifrados polialfebéticos.

El primer cifrado polialfabético fue el llamado cifrado de Alberti en 1467, creó artilugios llamados ‘discos de Alberti’.

Según Gómez (2010): “*Estos cifradores portátiles consisten de un armazón fijo en el que está grabado un alfabeto convencional, y unido a él una pieza circular concéntrica y móvil con otro alfabeto grabado*”

Sin embargo, el más conocido y potente fue el **Cifrado de Vigenère**.

El **Cifrado de Vigenère** es un cifrado polialfabético, consiste en un alfabeto de n letras y n alfabetos cifrados. Es decir, tenemos una matriz de 27x27 o 26x26

(si no consideramos la ñ). En donde la primera fila es el alfabeto sin desplazar y se le asigna la letra ‘A’ a ese alfabeto; en la segunda fila tenemos al alfabeto desplazado una letra a la izquierda y se le asigna la B, la tercera lo mismo respecto de la segunda y se le asigna la C y así con los 24 o 23 alfabetos restantes. De esta forma se construye por ejemplo la siguiente tabla de 26x26:

		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
2	B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
3	C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
4	D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
5	E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
6	F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
7	G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
8	H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
9	I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
10	J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
11	K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
12	L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
13	M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
14	N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
15	O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
16	P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
17	Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
18	R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
19	S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
20	T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
21	U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
22	V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
23	W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
24	X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
25	Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
26	Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

(Gómez, 2010, p.44)

Una vez que se tiene la tabla, y el mensaje en texto llano, se piensa en una clave. Veamos un ejemplo de cómo cifrar:

Mensaje: BUENAS TARDES

Clave: VERDE

Ahora se repite la clave hasta tener la cantidad de letras del mensaje original, y se busca el par de letras en la tabla anterior, *en los encabezados,* para saber por cual letra reemplazar *su intersección* (como la tabla es simétrica es lo mismo buscar por fila-columna que por columna-fila):

Mensaje Original:	B	U	E	N	A	S	T	A	R	D	E	S
Clave:	V	E	R	D	E	V	E	R	D	E	V	E
Mensaje Cifrado:	W	Y	V	Q	E	N	X	R	U	H	Z	W

Para descifrarlo observando la tabla buscamos en la fila de la letra de la palabra clave correspondiente a la letra cifrada y vemos a que columna le pertenece. La columna será la letra descifrada.

Veamos cómo cifrarlo con ayuda de la matemática:

Es válido pensar que a la letra A le corresponde el cifrado monoalfabético por César con clave 0, y para la letra B la clave 1, y así sucesivamente. Esto nos va a ayudar a entender la función que cifra por Vigenère.

Por el otro lado como procedimos anteriormente necesitamos ambas letras la de la clave y la del texto original, entonces hay que encontrar el número de la posición de la letra a cifrar del texto original y también debemos saber la letra de la clave que le corresponde, donde ésta nos dirá que cifrado de César utilizar.

Para saber la letra de la clave vamos a tener que saber la longitud de la palabra clave en el ejemplo la longitud de VERDE es 5, y ahora vamos a la posición de la letra a cifrar a dividirla por nuestra longitud de clave y nos quedaremos con su resto. Para cifrar la letra R del mensaje de BUENASTARDES, vemos que su posición empezando a contar desde 0 es 8, luego hacemos el resto de dividir 8 por 5 y será la posición de la letra de la clave en este caso 3. Buscando la posición 3 en VERDE, vemos que es la letra D. Aplicando la biyección $\boldsymbol{a}$ que usamos en César, pero teniendo en cuenta que la cantidad de elementos es 26 y no 27 como usamos anteriormente, a la D le correspondería el 3 (en este caso coincide con el 3 anterior porque la posición 3 en la palabra VERDE coincide con el número en la biyección $\boldsymbol{a}$), entonces nuestra clave César es el 3. Ahora cifremos nuestra R que le corresponde el 17, y evaluamos 17+3 en módulo 26, que nos da 20, aplicando la inversa de la biyección a 20 llegamos a que la letra cifrada es la U. Por lo que la letra R quedaría cifrada como U.

Ahora vamos a escribir esto de una forma más general:

Sea f la función con Dominio $\mathbb{N}_0$ e Imagen el conjunto de las letras del alfabeto, f nos devolverá la letra correspondiente a la posición sobre la palabra clave.

Sea L la longitud de la palabra clave; π la letra a cifrar y p la posición correspondiente a la dicha letra en el texto.

Sea $\mathbf{a}$ la función que habíamos definido en César.

Para cifrar haremos lo siguiente:

$V(\pi, p, L) = a(\pi) + a\big(f(r_L(p))\big)\ (mod\ 26)$; y luego $a^{-1}(V)$ la letra correspondiente al cifrado. Con $r_L(p)$ el resto de dividir a p por L.

Para descifrar, siendo π' la letra a descifrar y p su lugar correspondiente en el texto:

$V^{-1}(\pi', p, L) = a(\pi') - a\Big(f\big(r_L(p)\big)\Big)\ (mod\ 26)$ y luego $a^{-1}(V^{-1})$ la letra correspondiente al descifrado.

Al ejemplo anterior podríamos cifrarlo de esta forma:

$$V(R, 17, 5) \equiv a(R) + a(f\big(r_5(8)\big)\ (mod\ 26)$$

$$\Rightarrow V(R, 17, 5) \equiv 17 + a\big(f(3)\big)\ (mod\ 26)$$

$$\Rightarrow V(R, 17, 5) \equiv 17 + a(D)\ (mod\ 26) \Rightarrow V(R, 17, 5) \equiv 17 + 3\ (mod\ 26)$$

$$\Rightarrow V(R, 17, 5) \equiv 20\ (mod\ 26)\ \Rightarrow\ a^{-1}(20) = U$$

Cifrado ADFGX/ADFGVX

Según Gómez, 2010: En junio de 1918 las tropas alemanas se aprestaban a atacar la capital francesa, los aliados interceptaron las comunicaciones. Las comunicaciones estaban encriptadas por ADFGVX, considerada irrompible, combina algoritmos de sustitución y transposición. Sin embargo, un criptoanalista talentoso George Painvin rompió el cifrado el 1 de junio de 1918, esto permitió a los franceses impedir el ataque, deteniendo así el avance alemán a finales de la I Guerra Mundial.

Este cifrado es por ***sustitución monoalfabético monográmico y transposición****.*

El cifrado comienza similar a Polibio, realizamos una tabla de 5x5, aunque también se puede utilizar una de 6x6 con el alfabeto a usar. Si usamos una de 5x5 las letras I y J las pondremos en una misma casilla y excluiremos la ñ; si usamos una de 6x6 usaremos un alfabeto de 26 letras excluyendo la ñ e incluiremos los números del 0 al 9. Como encabezado de columnas y filas se usarán las letras ADFGX en caso de ser de 5x5 o ADFGVX en caso de ser de 6x6.

¿Por qué esas letras? Porque son fácilmente diferenciables al enviarlas por un telégrafo. A su vez el telégrafo se usa con código morse que es en esencia un *cifrado monoalfabético poligrámico*, pero sin clave.

A	. _	G	_ _ .
D	_ . .	V	. . . _
F	. . _ .	X	_ . . _

Como estamos usando muchas tablas de 5x5 vamos a hacer una de 6x6, las letras se pueden colocar de cualquier forma que el receptor al igual que el emisor deben saber, y será una de las claves del cifrado. Supongamos que elegimos crear la tabla de la siguiente forma:

	A	D	F	G	V	X
A	A	C	E	G	I	K
D	M	0	2	4	O	Q
F	S	6	8	1	U	W
G	Y	3	5	7	B	D
V	F	9	H	J	L	N
X	P	R	T	V	X	Z

También supongamos que queremos cifrar el texto: POMELO2018

Procedemos igual que Polibio:

P	O	M	E	L	O	2	0	1	8
XA	DV	DA	AF	VV	DV	DF	DD	FG	FF

Ahora elegimos una palabra clave, por ejemplo: PLANTAS. Construiremos una tabla que tiene como encabezado cada letra de la palabra clave y rellenaremos la tabla con lo que ciframos en el paso anterior.

P	L	A	N	T	S
X	A	D	V	D	A
A	F	V	V	D	V
D	F	D	D	F	G
F	F				

Ahora *transponemos*: ordenaremos a las columnas por el encabezado en orden alfabético.

A	L	N	P	S	T
D	A	V	X	A	D
V	F	V	A	V	D
D	F	D	D	G	F
	F		F		

Finalmente escribiremos lo que aparece en las columnas de arriba hacia abajo sin contar la clave: DVDAFFFVVDXADFAVGDDF

Por lo tanto, dada la combinación de la tabla del principio, y la palabra clave PLANTAS, el mensaje POMELO2018 queda cifrado como DVDAFFFVVDXADFAVGDDF.

Para descifrarlo: Buscaremos primero rearmar la última tabla. La longitud del mensaje es 20 y la de la clave 6 (clave que no repite letras), entonces buscamos techo(20,6)=4 (la función techo es similar a la parte entera pero a diferencia de esta última, techo redondea hacia el entero de arriba) por lo cual la tabla tendrá 4 filas. Hasta ahora sabemos que la tabla es de 6x4, 6 por la longitud de clave y 4 por la función techo. Ahora necesitamos saber que casillas quedan en blanco, para eso hallamos el resto de dividir la longitud del mensaje por la longitud de la palabra clave 20 (mod 6)=2. El 2 me indica que las primeras dos letras de la palabra clave (PL) terminan en la última fila y las otras terminan en la anterior.

Rearmaremos la tabla, así que escribimos las letras de la clave ordenadas alfabéticamente y las llenamos de arriba hacia abajo y consideramos lo que dijimos anteriormente sobre las celdas en blanco.

A	L	N	P	S	T
D	A	V	X	A	D
V	F	V	A	V	D
D	F	D	D	G	F
	F		F		

Ordenaremos las columnas por la letra clave:

P	L	A	N	T	S
X	A	D	V	D	A
A	F	V	V	D	V
D	F	D	D	F	G
F	F				

Reescribimos de izquierda a derecha el contenido de la tabla y las agrupamos de a dos a las letras, obteniendo:

XA	DV	DA	AF	VV	DV	DF	DD	FG	FF

Finalmente aplicando 'el descifrado de Polibio' a la tabla clave del principio volvemos a obtener el mensaje original: POMELO2018.

Programas desarrollados

Programa: Cifrado de César Generalizado

***Importante**: no cifrar textos o palabras con acentos o la letra ñ, porque no serán contemplados debido a limitaciones del lenguaje, y de cómo fue programado. Luego de cada interacción con el programa como ingresar un número o un texto se debe presionar Enter para que el programa continúe.*

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
int ascii_num (char c) {
        int num=c;
```

```
	//asignamos a mayusculas y minusculas, los numeros correspondientes al
abecedario siendo a=0, b=1, c=2, ..., z=25 sin contar la ñ pues no la reconoce
	if ((num>=65) && (num<=122)) return num%32-1; /* mod 32 pues hay
un espacio de 32 caracteres entre A y a y entre cualquier par de letras mayus y
minus */
	return -1;
}

char num_ascii (int n) {
	n+=65; /* asigno solo las mayus */
	return n;
}

int main () {
	int opc;
	printf("Cifra y Descifra por Cesar Generalizado\n\n1. Cifrar\n2.
Descifrar\n");
	scanf("%d",&opc);
	system("CLS");
	if (opc==2) cesar_descif(); else cesar_cif();
	system("CLS");
	main();
}

int cesar_cif () {
int clave, a;
char texto[1002];
printf("Escriba texto a cifrar:\n");
fflush(stdin);
fgets(texto, 1003, stdin);
fflush(stdin);
printf("\nIngrese la Clave (1/25): ");
scanf("%d",&clave);
printf("\nTexto Cifrado:\n");
for (a=0; a<=(strlen(texto)-2); a++) {
//toma el caracter lo traduce a numero con ascii_num, luego le sumo la clave y lo
evaluo modulo 26 segun Cesar Generalizado, y luego asigno las nuevas letras por
num_ascii, los caracteres que no correspondan los ignora
if (ascii_num(texto[a])>=0) printf("%c",
num_ascii((ascii_num(texto[a])+clave)%26));
}
printf("\n\nPresione una tecla para volver al menu...");
getch();
return 0;
}

int cesar_descif () {
int clave, a;
char texto[1002];
printf("Escriba texto a descifrar:\n");
fflush(stdin);
fgets(texto, 1003, stdin);
fflush(stdin);
```

```
printf("\nIngrese la Clave (1/25): ");
scanf("%d",&clave);
printf("\nTexto Cifrado:\n");
for (a=0; a<=(strlen(texto)-2); a++) {
//toma el caracter lo traduce a numero con ascii_num, luego le resto la clave y le
sumo 26 pues en modulo 26 no afecta y el numero obtenido sera positivo por lo
cual puedo sacarle el resto de la division modulo 26 segun Cesar Generalizado, y
luego asigno las letras correspondientes por num_ascii descifrando asi el mensaje
printf("%c", num_ascii((ascii_num(texto[a])+26-clave)%26));
}
printf("\n\nPresione una tecla para volver al menu...");
getch();
return 0;
}
```

Programa: de Cifrado de Playfair

***Importante**: no cifrar textos o palabras con acentos o la letra ñ, porque no serán contemplados debido a limitaciones del lenguaje, y de cómo fue programado. Luego de cada interacción con el programa como ingresar un número o un texto se debe presionar Enter para que el programa continúe. En el descifrado de Playfair usar solamente minúsculas y sin espacios (preferentemente copiar la salida del cifrado).*

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

int cadmit(char a) {
	char
admitidos[52]="abcdefghijlmnopqrstuvwxyzABCDEFGHIJLMNOPQRSTUVWXYZkK";
	int i, e=0;
	for (i=0;i<52;i++) {
		if (a==admitidos[i]) e=1;
	}
	return e;
	//devuelve 1 si el caractar es valido y 0 si no lo es
}

char convenvalido(char a) {
	char aconver[27]="ABCDEFGHIJkKLMNOPQRSTUVWXYZ";
	char econver[27]="abcdefghijiilmnopqrstuvwxyz";
	int i;
	for (i=0;i<27;i++) {
		if (a==aconver[i]) return econver[i];
	}
	return a;
	//convierte al caracter en minuscula valido
}

int buscaenclave (char* clave, char a) {
```

```
        int i;
        for (i=0; i<25; i++) {
                if (clave[i]==a) return i;
        }
}

void cif_playfair () {
        printf("Cifrado de Playfair - Cifrado\n\nIngrese palabra o frase clave: ");
        char frase[100];
        fflush(stdin);
   fgets(frase, 101, stdin);
   fflush(stdin);
   //printf("N: %d",strlen(frase));
   printf("\nGenerando Tabla de la Clave...");
   char frasecompl[125];
   int i, j=0;
   for (i=0;i<strlen(frase)-1;i++) {
        if (cadmit(frase[i])==1) {   frasecompl[j]=convenvalido(frase[i]);
        j++;
        }
        }
   char abc[25]="abcdefghijlmnopqrstuvwxyz";
   for (i=0;i<25;i++) {
        frasecompl[j+i]=abc[i];
   }
   /*printf("frasecompl: ");
   for (i=0;i<strlen(frasecompl);i++) {
        printf("%c",frasecompl[i]);
   }*/

        char clave[25];
        clave[0]=frasecompl[0];
        //printf("c0: %c",clave[0]);
        int k=0, e=0;
        j=1;
        for (i=1; i<strlen(frasecompl); i++) {
                for (k=0;k<j;k++) {
                        if (frasecompl[i]==clave[k]) e=1;
                }
                if (e==0) {clave[j]=frasecompl[i];j++;}
                e=0;
        }

        /*
        printf("clave: ");
   for (i=0;i<25;i++) {
        printf("%c",clave[i]);
   } */

   //creando tabla de 5x5
   char tabla[5][5];
   for (i=0;i<25;i++) {
        tabla[i/5][i%5]=clave[i];
```

```
    }
    printf(" [Listo]\nTabla Clave:");
    for (i=0;i<5;i++) {
        printf("\n");
        for (j=0;j<5;j++) {
                printf("%c   ",tabla[i][j]);
        }
    }

    printf("\n\nIngrese texto a cifrar: ");
    char texto[1500];
        fflush(stdin);
    fgets(texto, 1501, stdin);
    fflush(stdin);
    printf("\nFormateando el texto para cifrar...");
        char textomod[1500];
        j=0;
        for (i=0;i<strlen(texto)-1;i++) {
                if (cadmit(texto[i])==1) {
                if (j%2==0) {textomod[j]=convenvalido(texto[i]);j++;} else {
                        if (convenvalido(texto[i])!=textomod[j-1])
{textomod[j]=convenvalido(texto[i]);j++;} else {
                                if (textomod[j-1]=='x') textomod[j]='z'; else
textomod[j]='x';
                        j++;
                         textomod[j]=convenvalido(texto[i]);
                        j++;
                        }
                }
                }
        }
        if ((j%2)==1) {
                textomod[j]='x';
                j++;
        }

        printf(" [Listo]\nTexto Formateado: ");
        for (i=0;i<j;i++) {
                printf("%c",textomod[i]);
        }

        printf("\n\nCifrando...\nTexto Cifrado: ");
        int c1, c2, caso, c1f, c1c, c2f, c2c;
        for (i=0; i<j;i+=2) {
                c1=buscaenclave(clave,textomod[i]);
                        c2=buscaenclave(clave,textomod[i+1]);
        //printf("c1: %d - c2: %d",c1,c2);

        caso=3;
        //misma fila
        if ((c1/5)==(c2/5)) caso=1;
        //misma columna
        if ((c1%5)==(c2%5)) caso=2;
```

```
	//los supuse en distinta fila y columna, de ser todo negativo seria caso 3

	switch (caso) {
		case 1: {
			c1f=c1/5;
			c1c=c1%5;
			c1c++;
			if (c1c==5) c1c=0;
			c1=c1f*5+c1c;
			printf("%c",clave[c1]);
			c2f=c2/5;
			c2c=c2%5;
			c2c++;
			if (c2c==5) c2c=0;
			c2=c2f*5+c2c;
			printf("%c",clave[c2]);
		}
		break;
		case 2: {
			c1f=c1/5;
			c1c=c1%5;
			c1f++;
			if (c1f==5) c1f=0;
			c1=c1f*5+c1c;
			printf("%c",clave[c1]);
			c2f=c2/5;
			c2c=c2%5;
			c2f++;
			if (c2f==5) c2f=0;
			c2=c2f*5+c2c;
			printf("%c",clave[c2]);
		}
		break;
		case 3: {
			c1f=c1/5;
			c1c=c1%5;
			c2f=c2/5;
			c2c=c2%5;
			c1=c1f*5+c2c;
			c2=c2f*5+c1c;
			printf("%c%c",clave[c1],clave[c2]);
		}
		break;
	}
	}
	printf("\n\n<<Presione una tecla para volver al menu...>>");
}

void des_playfair () {
	printf("Cifrado de Playfair - Descifrado\n\nIngrese palabra o frase clave:
");
	char frase[100];
	fflush(stdin);
```

```
fgets(frase, 101, stdin);
fflush(stdin);
printf("\nGenerando Tabla de la Clave...");
char frasecompl[125];
int i, j=0;
for (i=0;i<strlen(frase)-1;i++) {
     if (cadmit(frase[i])==1) {   frasecompl[j]=convenvalido(frase[i]);
     j++;
     }
     }
char abc[25]="abcdefghijlmnopqrstuvwxyz";
for (i=0;i<25;i++) {
     frasecompl[j+i]=abc[i];
}

     char clave[25];
     clave[0]=frasecompl[0];
     int k=0, e=0;
     j=1;
     for (i=1; i<strlen(frasecompl); i++) {
             for (k=0;k<j;k++) {
                     if (frasecompl[i]==clave[k]) e=1;
             }
             if (e==0) {clave[j]=frasecompl[i];j++;}
             e=0;
     }
//creando tabla de 5x5
char tabla[5][5];
for (i=0;i<25;i++) {
     tabla[i/5][i%5]=clave[i];
}
printf(" [Listo]\nTabla Clave:");
for (i=0;i<5;i++) {
     printf("\n");
     for (j=0;j<5;j++) {
             printf("%c   ",tabla[i][j]);
     }
}

printf("\n\nIngrese texto a descifrar: ");
char texto[1500];
     fflush(stdin);
fgets(texto, 1501, stdin);
fflush(stdin);
     printf("\n\nDescifrando...\nTexto Descifrado: ");
     int c1, c2, caso, c1f, c1c, c2f, c2c;
     for (i=0; i<strlen(texto)-1;i+=2) {
             c1=buscaenclave(clave,texto[i]);
                     c2=buscaenclave(clave,texto[i+1]);

     caso=3;
     //misma fila
     if ((c1/5)==(c2/5)) caso=1;
```

```
	//misma columna
	if ((c1%5)==(c2%5)) caso=2;
	//los supuse en distinta fila y columna, de ser todo negativo seria caso 3

	switch (caso) {
		case 1: {
			c1f=c1/5;
			c1c=c1%5;
			c1c--;
			if (c1c==-1) c1c=4;
			c1=c1f*5+c1c;
			printf("%c",clave[c1]);
			c2f=c2/5;
			c2c=c2%5;
			c2c--;
			if (c2c==-1) c2c=4;
			c2=c2f*5+c2c;
			printf("%c",clave[c2]);
		}
		break;
		case 2: {
			c1f=c1/5;
			c1c=c1%5;
			c1f--;
			if (c1f==-1) c1f=4;
			c1=c1f*5+c1c;
			printf("%c",clave[c1]);
			c2f=c2/5;
			c2c=c2%5;
			c2f--;
			if (c2f==-1) c2f=4;
			c2=c2f*5+c2c;
			printf("%c",clave[c2]);
		}
		break;
		case 3: {
			c1f=c1/5;
			c1c=c1%5;
			c2f=c2/5;
			c2c=c2%5;
			c1=c1f*5+c2c;
			c2=c2f*5+c1c;
			printf("%c%c",clave[c1],clave[c2]);
		}
		break;
	}
	}
	printf("\n\n<<Presione una tecla para volver al menu...>>");
}

int main() {
	printf("Cifra y Descifra por Playfair\n\nSeleccione (1/2):\n1. Cifrar\n2.
Descifrar\n>>");
```

```
        int a;
        scanf("%d",&a);
        system("CLS");
        if (a==2) des_playfair(); else cif_playfair();
        getch();
        system("CLS");
        main();
        return 0;
}
```

Programa: Cifrado de Vigenère

***Importante**: no cifrar textos o palabras con acentos o la letra ñ, porque no serán contemplados debido a limitaciones del lenguaje, y de cómo fue programado. Luego de cada interacción con el programa como ingresar un número o un texto se debe presionar Enter para que el programa continúe.*

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
int ascii_num (char c) {
        int num=c;
        //asignamos a mayusculas y minusculas, los numeros correspondientes al
abecedario siendo a=0, b=1, c=2, ..., z=25 sin contar la ñ pues no la reconoce
        if ((num>=65) && (num<=122)) return num%32-1; /* mod 32 pues hay
un espacio de 32 caracteres entre A y a y entre cualquier par de letras mayus y
minus */
        return -1;
}

char num_ascii (int n) {
        n+=65; /* asigno solo las mayus */
        return n;
}

int main () {
        int opc;
        printf("Cifra y Descifra por Vigenere\n\n1. Cifrar\n2. Descifrar\n");
        scanf("%d",&opc);
        system("CLS");
        if (opc==2) vigenere_descif(); else vigenere_cif();
        system("CLS");
        main();
}

int vigenere_cif () {
int a, pos=-1, ccesar;
char texto[1002];
char clave[1002];
char clave_admit[1002];
printf("Escriba texto a cifrar:\n");
```

```
fflush(stdin);
fgets(texto, 1003, stdin);
fflush(stdin);
printf("\nIngrese la palabra o frase clave: ");
fflush(stdin);
fgets(clave, 1003, stdin);
fflush(stdin);
//recreando la clave a una admitida
for (a=0; a<=(strlen(clave)-2); a++) {
if (ascii_num(clave[a])>=0) {pos++;clave_admit[pos]=clave[a];}
}
clave_admit[pos+1]='\0';
printf("[Clave admitida: %s]",clave_admit);
pos=-1;
printf("\n\nTexto Cifrado:\n");
for (a=0; a<=(strlen(texto)-2); a++) {
//toma el caracter lo traduce a numero con ascii_num, luego le sumo la clave que
vendra dada por Vigenere y lo evaluo modulo 26 segun Cesar Generalizado, y
luego asigno las nuevas letras por num_ascii, los caracteres que no correspondan
los ignora
    if (ascii_num(texto[a])>=0) {
        pos++;
        //calculando ccesar por vigenere
        ccesar=ascii_num(clave_admit[pos%strlen(clave_admit)]);
        printf("%c", num_ascii((ascii_num(texto[a])+ccesar)%26));
    }
}

printf("\n\nPresione una tecla para volver al menu...");
getch();
return 0;
}

int vigenere_descif () {
int a, pos=-1, ccesar;
char texto[1002];
char clave[1002];
char clave_admit[1002];
printf("Escriba texto a descifrar:\n");
fflush(stdin);
fgets(texto, 1003, stdin);
fflush(stdin);
printf("\nIngrese la palabra o frase clave: ");
fflush(stdin);
fgets(clave, 1003, stdin);
fflush(stdin);
//recreando la clave a una admitida
for (a=0; a<=(strlen(clave)-2); a++) {
if (ascii_num(clave[a])>=0) {pos++;clave_admit[pos]=clave[a];}
}
clave_admit[pos+1]='\0';
printf("[Clave admitida: %s]",clave_admit);
pos=-1;
```

```
printf("\n\nTexto Descifrado:\n");
for (a=0; a<=(strlen(texto)-2); a++) {
//toma el caracter lo traduce a numero con ascii_num, luego le resto la clave que
vendra dada por Vigenere y lo evaluo modulo 26 segun Cesar Generalizado, y
luego asigno las nuevas letras por num_ascii, los caracteres que no correspondan
los ignora
    if (ascii_num(texto[a])>=0) {
        pos++;
        //calculando ccesar por vigenere
        ccesar=ascii_num(clave_admit[pos%strlen(clave_admit)]);
        printf("%c", num_ascii((ascii_num(texto[a])-ccesar+26)%26));
    }
}
printf("\n\nPresione una tecla para volver al menu...");
getch();
return 0;
}
```

Criptoanálisis (Parte I)

Criptoanálisis para cifrados por sustitución monoalfabéticos:

El Análisis de Frecuencias

Los cifrados como el de César, César Generalizado y Afín son de sustitución monoalfabéticos, y son también casos particulares de algo más general.

Dado que, si a cada letra le corresponde otra única, y viceversa. Si aplicamos la biyección del conjunto de letras a números, podemos llegar a la conclusión que los cifrados mencionados pueden considerarse también de otra manera.

Veamos un ejemplo: Para un Cifrado Cesar Generalizado de clave 7 la tabla que se usará para cifrar se corresponde a la siguiente matriz:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & \dots & 26 \\ 7 & 8 & 10 & 11 & \dots & 6 \end{pmatrix}$$

Pero en otro cifrado de sustitución monoalfabético, le puede corresponder por ejemplo esta otra:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & \dots & 26 \\ 8 & 5 & 16 & 9 & \dots & 2 \end{pmatrix}$$

En general, si tenemos en cuenta los cifrados de este estilo, todos los alfabetos cifrados son elementos del grupo de permutaciones S_{27}, donde la cantidad de elementos sería de 27!=*10888869450418352160768000000 alfabetos cifrados posibles*. Aunque algunos no serían tan prácticos por su trivialidad.

'A mano' sería impracticable hallar la única combinación que permite descifrar un texto que este cifrado de esta forma. Por lo cual, a los criptoanalistas se les ocurrió un método con el cual pueden descifrar cualquier texto que haya sido cifrado a través de la sustitución con un cifrado monoalfabético, este método es el **Análisis de Frecuencias**.

Según Gómez, 2010: "*Una manera de resolver un mensaje cifrado, si sabemos en qué lengua está escrito, es encontrar un texto llano escrito en la misma lengua, suficientemente largo, y luego contar cuantas veces aparece cada letra. A la letra que aparece con más frecuencia la llamamos "primera", a la siguiente en frecuencia la llamaremos "segunda" y así hasta que hayamos cubierto todas las letras que aparecen en nuestro texto. Luego observamos el texto cifrado que queremos resolver y clasificamos sus símbolos de la misma manera. Encontramos el símbolo que aparece con mayor frecuencia y lo sustituimos por la "primera" y así sucesivamente, hasta que hayamos cubierto todos los símbolos*

del criptograma que queremos resolver." al-Kindi 1987, Estambul, tratado "Sobre el desciframiento de mensajes criptográficos" *(Fue hallado en 1987 pero fue en el siglo IX cuando vivió al-Kindi)*

Además de lo que nos dejó al-Kindi, también se suelen analizar los llamados dígrafos y trígrafos, donde se analiza también el porcentaje de la aparición de dos letras juntas y de tres letras juntas. Uniendo lo que nos dice al-Kindi + dígrafos + trígrafos, podemos descifrar de esta forma cualquier texto.

En el idioma español el porcentaje de aparición de las letras es el siguiente.

A	11.96%	H	0.89%	Ñ	0.29%	U	4.80%
B	0.92%	I	4.15%	O	8.69%	V	0.39%
C	2.92%	J	0.30%	P	2.77%	W	0.01%
D	6.87%	K	0.01%	Q	1.53%	X	0.06%
E	16.78%	L	8.37%	R	4.94%	Y	1.54%
F	0.52%	M	2.12%	S	7.88%	Z	0.15%
G	0.73%	N	7.01%	T	3.31%		

(Gómez, 2010, p.39)

Luego teniendo un texto cifrado de gran tamaño, buscamos los porcentajes de aparición de las letras y los asociamos con los de esta tabla, y así podemos encontrar la combinación que se utilizó para ser cifrado. Sin embargo, hay veces que no funciona muy bien hacer solo esto pues quizás el texto no es lo suficientemente largo, por lo cual hay que incorporar al análisis los dígrafos y trígrafos, de esta forma el análisis estará completo, y se podrá obtener el texto original. Además, podremos saber la clave que se usó (de haber utilizado algún cifrado conocido) y utilizarla para descifrar más textos provenientes del mismo autor que los cifró.

Polibio y Playfair

El Cifrado de Polibio es por *sustitución monoalfabético monográmico* al igual que César, así que si tomamos a los dígrafos del texto cifrado como 'una letra' el análisis de frecuencias es perfectamente aplicable a Polibio, por lo cual Polibio no es seguro.

El Cifrado de Playfair es por *sustitución monoalfabético poligrámico* si bien es 'más seguro' que los anteriores, solo basta con tomar los dígrafos del texto cifrado y hacer un análisis de frecuencias con los dígrafos del idioma del texto original. Por lo cual, también es reconstruible y ya no es seguro.

El criptoanálisis nos da una importante lección:

Los cifrados por sustitución monoalfabéticos sean monográficos o poligráficos: son inseguros.

Una respuesta por parte de los criptógrafos

Para contrarrestar a esta potente herramienta diseñada por los criptoanalistas surgió el **cifrado de sustitución homofónica**, por ejemplo, si la E tenía una probabilidad de aparición de un 10%, este cifrado nos decía que a la E hay que reemplazarla por 10 caracteres alternativos, este método fue el preferido hasta comienzos del siglo XVIII; luego se optó por los cifrados polialfabéticos, aunque éstos ya existían, pero eran más difíciles de llevar a cabo.

Sin embargo, las características y ciertos patrones que tienen las letras podrían dar pautas de las sustituciones realizadas, un análisis más profundo claramente pero tampoco resultó ser infalible este cifrado.

Criptoanálisis para cifrados por sustitución polialfabéticos: Vigenère

Charles Babbage (1791- 1871) científico británico del siglo XIX, se dedicó a romper el cifrado polialfabético. Babbage (1854) se dio cuenta que si enumeraba las letras que se repetían era posible que éstas pertenezcan al mismo alfabeto cifrado, y que la aparición de éstas eran un múltiplo de la cantidad de letras de la palabra clave. Por ejemplo, si una letra se repite en 8 caracteres, sus divisores: 8, 4, 2, 1 pueden ser la longitud de la palabra clave.

"*El procedimiento es el siguiente: se listan todos los caracteres repetidos y el espacio que queda entre ellos, y se buscan divisores enteros de los caracteres de los espacios. Los divisores comunes son los números candidatos a representar la longitud de la clave.*"

(Gómez, 2010, p.50)

Empezamos con la longitud de clave más probable, y separamos a todas esas letras que posiblemente pertenezcan al mismo alfabeto, siendo una cadena C1 C2 C3 C4 C5 C6 C7 … si el candidato de longitud es 5: C1 C6 C11 … pertenecerían a un mismo alfabeto y C2 C7 C12… a otro. Luego a cada alfabeto se le aplica el **análisis de frecuencias**, y podría descifrarse el texto en cuestión sin

problemas, en caso que no se llegue se procede a buscar la otra longitud de letras más probable y se vuelve a realizar el procedimiento hasta descifrar el texto. Sin embargo, Babbage mantuvo su método en secreto, luego en 1863 Friedrich Krasiski propuso públicamente un método similar.

Pongamos en práctica esa idea:

Texto original: Hay cuentas que cuentan y cuentas que no cuentan, ¿Cuántas cuentas cuentas que cuentan y cuántas que no cuentan?

Clave: RESTO

Texto cifrado por Vigenère (usando el programa desarrollado):

YEQVIVRLTGHYWVIVRLTBPGMXBKEKJIVRGVIVRLTBTYSGHRWUNSEXSLQLIFMO
JUMXQLIFMOECUNOEXSLELIFHQLIFMOE

Se ha sombreado algunas cadenas de caracteres que coinciden. Analizamos ahora las distancias que hay entre ellas:

"VIVRLT": 10, 20, 30.

"EXSL": 25.

"QLIFMO": 10, 20, 30.

La longitud de clave debe ser divisor de todas estas distancias, así que propongo calcular el mcd entre todos los números.

mcd(10,20,30)=10 y mcd (10,25)=5

Luego el mcd entre todas las longitudes encontradas es 5.

Por lo que es probable que 5 sea la longitud de la clave utilizada. Luego la cadena YVHVP… pertenecerá a un alfabeto, mientras que la cadena ERYRG… a otro, QLWLM… a otro, VTVTX… a otro y IGIBB…a otro alfabeto. Realizando el análisis de frecuencias a cada cadena perteneciente a un alfabeto, podremos recuperar el mensaje original. El único problema es que perderemos el análisis de los dígrafos y trígrafos, por lo que puede complejizarse si el texto cifrado en cuestión no es de gran tamaño.

Bases Teóricas

En esta parte se dará una fundamentación matemática, a propiedades y teoremas que nos referiremos en el resto del trabajo. Las demostraciones que se verán a continuación, están basadas mayormente en el libro "Aritmética", aunque con ciertas modificaciones y aclaraciones, la esencia y los artilugios utilizados son los del libro.

Congruencias

Definición: Se dice que a es congruente con b módulo m si y solo si m divide a "a – b".

$$Sea\, m \in \mathbb{N}, a, b \in \mathbb{Z} : \; a \equiv b \; (m) \Leftrightarrow \; m|a-b$$

Además $m|a-b \Leftrightarrow \; \exists\, k \; \in \; \mathbb{Z} \mid \; a-b = mk$

Propiedad 1.1: $a \equiv r_m a \;\; (m)$ siendo $r_m(a)$ el resto de dividir a a por m.

Demostración

$$a = km + r_m a \,, con\; 0 \leq r_m a < m$$

$$\Leftrightarrow \; a - r_m(a) = km$$

$$m|m \Rightarrow m|km \Rightarrow \; m|a - r_m a \;\; (m) \Rightarrow \; a \equiv r_m a \;\; (m)$$

Propiedad 1.2: a es múltiplo de m $\Leftrightarrow \; a \; \equiv 0 \;\; (m)$

Demostración:

Probemos la ida:

$$a\; es\; múltiplo\; de\; m \; \Rightarrow \; m|a \; \Rightarrow \; m|a-0 \; \Rightarrow \; a \equiv 0 \;\; (m)$$

Probemos la vuelta:

$$a \equiv 0 \;\; (m) \Rightarrow \; m|a-0 \; \Rightarrow \; m|a \; \Rightarrow \; a\; es\; múltiplo\; de\; m.$$

Propiedad 1.3:

La $\equiv$ es una relación de equivalencia

- Reflexividad: $a \equiv a(m)$

Demostración:

$m \mid 0 \Rightarrow m \mid a - a \Rightarrow a \equiv a\ (m)$

- Simetría : $a \equiv b\ (m) \Rightarrow b \equiv a\ (m)$

Demostración:

a $\equiv b\ (m) \Rightarrow m \mid a - b \Rightarrow m \mid (a-b)(-1) \Rightarrow m \mid b - a \Rightarrow b \equiv a\ (m)$

$\therefore a \equiv b\ (m) \Rightarrow b \equiv a\ (m)$

- Transitividad: $a \equiv b\ (m) \wedge b \equiv c\ (m) \Rightarrow a \equiv c\ (m)$

Demostración:

$$a \equiv b\ (m) \Rightarrow m \mid a - b$$

$$b \equiv c\ (m) \Rightarrow m \mid b - c$$

$$\Rightarrow m \mid (a-b) + (b-c) \Rightarrow m \mid a - c \Rightarrow a \equiv c\ (m)$$

$$\therefore a \equiv b\ (m) \wedge b \equiv c\ (m) \Rightarrow a \equiv c\ (m)$$

Por lo tanto, como la relación de congruencia es reflexiva, simétrica y transitiva entonces es una relación de equivalencia, por serlo genera una partición en clases de equivalencia.

Definición: Clase de congruencia

$\bar{a} = \{x \in \mathbb{Z} \,/\, x \equiv a\ (m)\}$

Igualdad entre clases: $\bar{a} = \bar{b} \Leftrightarrow a \equiv b\ (m)$

Propiedad 1.4: Sea a, b $\in \mathbb{Z}$, $n, m \in \mathbb{N} : a \equiv b\ (m) \wedge n \mid m \Rightarrow a \equiv b\ (n)$

Demostración:

$$a \equiv b\ (m) \Rightarrow m \mid a - b$$

$$n \mid m \wedge m \mid a - b \Rightarrow n \mid a - b \Rightarrow a \equiv b\ (n)$$

Propiedad 1.5: $a \equiv b\ (m) \wedge c \equiv d\ (m) \Rightarrow a \pm c \equiv b \pm d\ (m)$

Demostración:

$a \equiv b(m) \Rightarrow m \mid a - b$, además, $c \equiv d(m) \Rightarrow m \mid c - d$

$$\Rightarrow m \mid (a-b) + (c-d) \wedge m \mid (a-b) - (c-d)$$

$$\Rightarrow a + c \equiv b + d\ (m) \wedge a - c \equiv b - d\ (m)$$

$$\Rightarrow a \pm c \equiv b \pm d\ (m)$$

<u>Propiedad 1.6</u>: $a \equiv b\ (m) \wedge c \equiv d\ (m) \Rightarrow ac \equiv bd\ (m)$

<u>Demostración</u>:

$a \equiv b\ (m) \Rightarrow m \mid a - b \Rightarrow m \mid (a - b)\, c \Rightarrow m \mid ac - bc$

$c \equiv d\ (m) \Rightarrow m \mid c - d \Rightarrow m \mid (c - d)\, b \Rightarrow m \mid bc - bd$

$$\Rightarrow m \mid (ac - bc) + (bc - bd) \Rightarrow m \mid ac - bd$$

$$\Rightarrow ac \equiv bd\ (m)$$

<u>Propiedad 1.7</u>: $a \equiv b\ (m) \wedge n \in \mathbb{N} \Rightarrow a^n \equiv b^n\ (m)$

<u>Demostración</u>:

$a \equiv b\ (m)\ por\ hipótesis \Rightarrow a^1 \equiv b^1\ (m)$ verifica para n=1

Queremos ver que $a^h \equiv b^h\ (m) \Rightarrow a^{h+1} \equiv b^{h+1}\ (m)$

$a^h \equiv b^h\ (m)$ (por hipótesis inductiva) $\wedge\ a \equiv b\ (m)$ (por hipótesis)

$$\Rightarrow a^h . a \equiv b^h . b\ (m) \Rightarrow a^{h+1} \equiv b^{h+1}\ (m)$$

Habiéndolo verificado para n =1, de suponerlo verdadero para n=h y habiéndolo probado para n = h+1 , queda demostrado que:

$$\forall\, n \in \mathbb{N}: a \equiv b\ (m) \Rightarrow a^n \equiv b^n\ (m)$$

<u>Propiedad 1.8</u>: $a \equiv b\ (m) \wedge d \mid a \wedge d \mid b \wedge d \mid m \Rightarrow \frac{a}{d} \equiv \frac{b}{d}\left(\frac{m}{d}\right)$ (con $d \in \mathbb{N}$)

<u>Demostración</u>:

$$a \equiv b\ (m) \Rightarrow m \mid a - b \Rightarrow \exists\, k \in \mathbb{Z} \,/\, \frac{a-b}{d} = \frac{km}{d}$$

$$\Rightarrow \frac{a}{d} - \frac{b}{d} = k \,.\, \frac{m}{d}$$

$d \mid a \wedge d \mid b \Rightarrow \frac{a}{d}, \frac{b}{d} \in \mathbb{Z}$; además $d \mid m \Rightarrow \frac{m}{d} \in \mathbb{N}$

$$\Rightarrow \frac{m}{d} \mid \frac{a}{d} - \frac{b}{d} \Rightarrow \frac{a}{d} \equiv \frac{b}{d}\left(\frac{m}{d}\right)$$

<u>Definición</u>:

$\mathbb{Z}_m$: Conjunto de las clases de congruencia módulo m

$\mathbb{Z}_m^*$: Conjunto de las clases de congruencia módulo m, sin la clase nula.

Se define: $\bar{a} + \bar{b} = \overline{a+b}$ y $\bar{a}.\bar{b} = \overline{ab}$, por propiedades 1.5 y 1.6 están bien definidas.

En la suma: En $\mathbb{Z}_m$ la clase $\bar{0}$ es elemento neutro y cada elemento tiene inverso aditivo, siendo $\overline{-a}$ = $\overline{m-a}$ la clase opuesta de $\bar{a}$.

<u>Propiedad 1.9</u>: $\bar{a}$ *es inversible módulo* $m \Leftrightarrow (a:m) = 1$

<u>Demostración</u>

Probemos la ida:

$$\bar{a} \text{ es inversible módulo } m \Leftrightarrow \exists\, \bar{b} \;/\; \bar{a}\,.\,\bar{b} = \bar{1} \Leftrightarrow \overline{ab} = \bar{1}$$

$$\Leftrightarrow ab \equiv 1\,(m) \Leftrightarrow m \mid ab - 1$$

$$\Leftrightarrow \exists\, k \in \mathbb{Z} \mid ab - 1 = mk$$

$$ab - mk = 1$$

$$ab + m(-k) = 1$$

$$\Rightarrow (a : m) \mid 1 \Rightarrow (am) = 1 \text{ (I)}$$

Probemos la vuelta de la propiedad

$(a:m) = 1 \Rightarrow \exists\, b,t \in \mathbb{Z} \,/\, ab + mt = 1$ (por Lema de Bézout)

$$ab - 1 = -mt$$

$$ab - 1 = m\,(-t)$$

$$\Rightarrow m \mid ab - 1 \Rightarrow ab \equiv 1\,(m)$$

$$\Rightarrow \overline{ab} = \bar{1}$$

$$\bar{a}\,.\,\bar{b} = \bar{1}$$

$$\Rightarrow \bar{b} \text{ es inverso multiplicativo de } a \text{ en } \mathbb{Z}_m$$

$$\Rightarrow \bar{a} \text{ es inversible módulo } m$$

$$\therefore \bar{a} \text{ es inversible módulo } m \Leftrightarrow (a:m) = 1$$

(I) Pues $a \mid (a:n) \Rightarrow a = (a : n)q$

$$m \mid (a:m) \Rightarrow m = (a:m)q'$$

$$\Rightarrow 1 = ab + m\,(-k) = (a:m)qb + (a:m)\,q'\,(-k)$$

$$\Rightarrow 1 = (a:m)\big(qb + q'(-k)\big) \Rightarrow (a:m) \mid 1$$

Por lo tanto, en $\mathbb{Z}_m$ tendrán inversos los elementos coprimos con m.

En particular, si un módulo es un p primo todas sus clases no nulas serán inversibles.

<u>1.10. Teorema de Wilson</u>: Un número natural n es primo $\Leftrightarrow$ $n \mid (n-1)\,! + 1$

<u>Demostración</u>:

Probemos la ida,

n=p, con p primo

Pensemos en la clase producto de todas las de $\mathbb{Z}_p$ no nulas : $\overline{(p-1)!}$

Como p es primo en $\mathbb{Z}_p$ cada clase no nula tiene inversa. Las clases cuya inversa no son si mismas su producto será 1 y no afectará al resultado de $\overline{(p-1)!}$.

Analicemos entonces aquellas clases cuya inversa es si misma:

$$a.a \equiv 1\,(p) \Rightarrow\ a^2 \equiv 1\,(p) \Rightarrow\ a^2 - 1 \equiv 0\,(p) \Rightarrow p \mid a^2 - 1$$

$$\Rightarrow p \mid (a-1)(a+1) \Rightarrow p \mid a-1\ \vee p \mid a+1$$

$$\Rightarrow\ a \equiv 1\,(p)\ \vee\ a \equiv\ -1\,(p)$$

$$\Rightarrow\ \bar{a} =\ \bar{1}\ \vee\ \bar{a} =\ \overline{-1} =\ \overline{p-1}$$

Luego, $\overline{(p-1)!} =\ \bar{1}\,.\bar{1}\ \overline{p-1} \Rightarrow\ \ \overline{(p-1)!} =\ \overline{p-1} =\ \overline{-1}$

$$\Rightarrow\ \overline{(p-1)!} +\ \bar{1} =\ \overline{p-1} +\ \bar{1}$$

$$\Rightarrow\ \overline{(p-1)!+1} =\ \bar{0}$$

$$\Rightarrow (p-1)! + 1 \equiv 0\,(p)$$

$$\Rightarrow p \mid (p-1)! + 1 \Rightarrow n \mid (n-1\,)!\ + 1\ pues\ n = p$$

Probemos la vuelta,

Supongamos n compuesto $\wedge\, n \mid (n-1\,)! + 1$

n compuesto $\Rightarrow\ \exists\ a\ primo\ /\ a \mid n, como\ n \mid (n-1)! + 1\ \Rightarrow a\ \mid\ (n\ -\ 1)!\ +\ 1$

$a \mid (n-1)!\ (pues\ a\ es\ factor\ de\ n) \Rightarrow\ a \mid (n-1)!\ + 1 - (n-1)!\ \Rightarrow\ a \mid 1$ Absurdo! pues a es primo, luego n es primo.

<u>Propiedad 1.11</u>: $ax \equiv ay\ (m) \wedge (a : m) = 1 \Rightarrow x \equiv y\ (m)$

Demostración:

Como $(a : m) = 1 \Rightarrow \exists\, b \in \mathbb{Z} \;/\; ab \equiv 1\,(m)$

$$\Rightarrow b\,(ax) \equiv b\,(ay)\;\;(m)$$

$$(ba)x \equiv (ba)\,y\;\;(m)$$

$$x \equiv y\,(m)$$

Luego, $(a{:}m) = 1 \wedge x \not\equiv y(m) \Rightarrow\; ax \not\equiv ay\,(m)$

(por la propiedad lógica $\big((p \wedge q) \Rightarrow r\big) \Leftrightarrow \big((q \wedge \sim r) \Rightarrow \sim q\big)$)

Como $\{0,1,2,\dots,n-1\}$ es un sistema completo de restos módulo m,

$si\;(a{:}m) = 1 \Rightarrow \{0, a\,, 2a, \dots, (m-1)a\}$ es un sistema completo de restos módulo m.

Por lo tanto, la multiplicación por $\bar{a}$ determina una permutación de las clases de congruencia módulo m.

<u>Definición:</u> $\{x_1, x_{2,} \dots, x_s\}$ es un sistema reducido de restos módulo m (con m $\in \mathbb{N}$) $\Leftrightarrow\; \forall\, c \in \mathbb{Z} \,/\, (c : m) = 1, \exists\, i \in \mathbb{N} \,/\, c \equiv x_i\;(m) \wedge 1 \leq i \leq s$

Sabiendo que:

$$(a{:}m) = 1 \wedge (b{:}m) = 1 \Rightarrow\; (ab : m) = 1$$

En $\mathbb{Z}_m^*$ todo elemento x que pertenezca verifica $(x : m) = 1$

Dados x, y $\in \mathbb{Z}_m^*$: x $\not\equiv y\,(m)$, tomando $(a : m) = 1$ y pensando en el corolario de la propiedad 1.11. Podemos decir que $\{a\,x_{1,} a\,x_{2,} \dots, a\,x_{s,}\}$ tiene las mismas clases que $\mathbb{Z}_m^*$ por lo que multiplicar por $\bar{a}$ permuta los elementos de $\mathbb{Z}_m^*$.

<u>Proposición 1.12</u>: $La\ ecuación\ ax \equiv b\;\;(m)\;\;es\ resoluble \Leftrightarrow (a{:}m)|b$

Demostración:

Probemos la ida,

$ax \equiv b(m) \Rightarrow m \mid ax - b \Rightarrow\; \exists\, k \in \mathbb{Z} \;/\; ax - b = mk$

$$\Rightarrow ax + m\,(-k) = b \underset{*}{\Rightarrow} (a{:}m) \mid b$$

Probemos la vuelta,

$(a{:}m) \mid b \underset{*}{\Rightarrow} \exists\, x, y \in \mathbb{Z} \;/\; ax + my = b$

$$\Rightarrow ax - b = m\,(-y) \Rightarrow m \mid ax - b \Rightarrow ax \equiv b\ (m)$$

Las implicaciones marcadas con (*) se justificarán con la siguiente proposición.

(*) <u>Proposición 1.13</u>: $\exists\, x, y \in \mathbb{Z} \;/\; ax + by = c \Leftrightarrow (a{:}b) \mid c$

Probemos la ida,

$(a{:}b) \mid a \Rightarrow \exists\, q \in \mathbb{Z} \,/\, a = (a{:}b)\,.\,q$

$(a{:}b) \mid b \Rightarrow \exists\, q' \in \mathbb{Z} \,/\, b = (a{:}b)\,.\,q'$

$$ax + by = c \Rightarrow (a{:}b).qx + (a{:}b).q'y = c$$

$$\Rightarrow (a{:}b)(qx + q'y) = c \Rightarrow (a{:}b) \mid c$$

Probemos la vuelta,

$(a{:}b) \mid c \;\wedge\; (a{:}b) = d \Rightarrow d \mid c$

$(a{:}b) = d \Rightarrow \exists\, s, t \in \mathbb{Z} \;/\; a\,s + b\,t = d$ (por Lema de Bézout)

$\Rightarrow as + bt \mid c \Rightarrow \exists\, k \in \mathbb{Z} \;/\; c = (as + bt)k$

$$\Rightarrow c = a\,(ks) + b\,(kt)$$

$$\Rightarrow \exists\, x_0 \in \mathbb{Z} \;/\; x_0 = ks$$

$$\Rightarrow \exists\, y_0 \in \mathbb{Z} \;/\; y_0 = kt$$

$$\Rightarrow \exists\, x_0, y_o \in \mathbb{Z} \;/\; c = ax_0 + b\,y_0$$

Analicemos la unicidad de la solución:

Supongamos que $\overline{x_0}$ es solución y $\bar{x}$ también. Tomo $a' = \frac{a}{d} \wedge m' = \frac{m}{d}$ (con $d = (a{:}m)$)

$$ax \equiv b\ (m) \wedge ax_0 \equiv b\ (m) \Rightarrow ax \equiv ax_0\ (m)$$

$\Rightarrow \frac{a}{d}\,x \equiv \frac{a}{d}\,x_0\,(\frac{m}{d})$ (Pues $a \equiv b(m) \wedge d \mid a \wedge d \mid b \wedge d \mid m \Rightarrow \frac{a}{d} \equiv \frac{b}{d}\ \left(\frac{m}{d}\right)$)

Luego si $a'x \equiv a'x_0\,(m') \Rightarrow x \equiv x_0\,(m')$ (pues $(a' : m') = 1$, entonces existe el inverso de a´ módulo m'.)

$\therefore$ *La solución es única en módulo m´.*

$\Rightarrow m' \mid x - x_0 \Rightarrow \exists\, k \in \mathbb{Z} \,/\, x - x_0 = m'k \Rightarrow x = x_0 + km' \Rightarrow x = x_0 + k\,\frac{m}{d}$

Veamos cuales soluciones hay en módulo m:

$$x_0 + k\,\frac{m}{d} \equiv x_0 + k'\,\frac{m}{d}\ (m)$$

$$k\,\frac{m}{d} \equiv k'\,\frac{m}{d}\ (m) \Rightarrow m \mid k\,\frac{m}{d} - k'\,\frac{m}{d}$$

$$\Rightarrow \exists\, q \in \mathbb{Z}\ /\ k\,\tfrac{m}{d} - k'\,\tfrac{m}{d} = mq$$

$$m\left(\frac{k}{d} - \frac{k'}{d}\right) = mq \Rightarrow m\left(\frac{k-k'}{d}\right) = mq$$

$$\Rightarrow \frac{k-k'}{d} = q\ \ (con\ q \in \mathbb{Z}) \Rightarrow d \mid k-k' \Rightarrow\ k \equiv k'(d)$$

Luego hay d soluciones distintas pues $k \in \mathbb{Z}_d$ *en módulo m, y serán*

$\left\{\overline{x_0}, \overline{x_{0+\frac{m}{d}}}, \overline{x_{0+2\frac{m}{d}}}, \dots, \overline{x_{0+(d-1)\frac{m}{d}}}\right\}$ (pues $0 \leq k < d$)

Corolario: la ecuación $ax \equiv b(m),\ si\ (a{:}m) = 1$, admite solución única en módulo m.

Teoremas fundamentales

2.1. Teorema chino del resto

Sean $m_1, m_2, \dots, m_r \in \mathbb{N}\ /\ (m_i : m_j) = 1\ \ \forall\, i \neq j\ (coprimos\ 2\ a\ dos)$

Sean $a_1, a_2, \dots, a_r \in \mathbb{Z}$ *y sea el sistema de r ecuaciones de congruencia*

$$x \equiv a_i\ (m_i)\ \ \forall\, i, con\ i = 1, 2, \dots, r$$

$\Rightarrow$ *El sistema tiene solución. Si x e y son soluciones de las ecuaciones anteriores se tiene que*

$$x \equiv y\ (m)\ siendo\ m_1, m_2, \dots, m_r$$

Demostración:

Sea $\prod_{t=1}^{j} m_t = m_1.m_2\ \dots\ m_r,\ como\ (m_i : m_J) = 1\ \forall\, i \neq j;\ \ m\ será\ su\ mcm.$

Sea $q_i = \frac{m}{m_i}\ \forall i \in \{1,2,\dots,r\}$

Como $(q_i : m_i) = 1 \Rightarrow \exists\, v_i\ inverso\ de\ q_i\ en\ \mathbb{Z}_i$: $\ q_i\, v_i \equiv 1\ (m_i), \forall\, i \in \{1, 2, \dots, r\}$

Defino el número entero x:

$$x = \sum_{i=1}^{r} q_i v_i\, a_i = q_1 v_1\, a_1 + q_2 v_2\, a_2 + \cdots + q_r v_r\, a_r$$

Queremos ver que x es solución

Siendo $i \neq j$, con $j \in \{1, 2, \dots, r\}$

$q_j = \frac{m}{m_j} = \frac{m_i . m_j . \prod_{t=1, t \neq i,j}^{r} m_t}{m_j}$

$$\Rightarrow q_j = m_i \prod_{t=1, t \neq i,j}^{r} m_t \Rightarrow m_i \mid q_j \; \forall \, i \neq j \; si \; i,j \in \{1, 2, \dots, r\}$$

Sea $i \in \{1, 2, \dots r\}$

$x \equiv x \, (m_i) \Rightarrow x \equiv \sum_{t=1}^{r} q_t \, v_t \, a_t \;\; (m_i)$

$$x \equiv q_1 v_1 \, a_i + \cdots + q_i v_i \, a_i + q_r v_r \, a_r \;\; (m_i)$$

Como $m_i \mid q_j \Rightarrow q_j \, v_j \, a_j \equiv 0 \, (m_i)$

$\Rightarrow x \equiv q_i v_i \, a_i \;\; (m_i) \;)$

Como $q_i v_i \equiv 1 \, (m_i) \Rightarrow x \equiv a_i \;\; (m_i) \;\; \forall_i \;\; con \; i \in \{1, 2, \dots, r\}$

$$\Rightarrow x \; es \; solución \; del \; sistema$$

Supongamos que y es otra solución $\Rightarrow y \equiv a_i \, (m_i) \; \forall i$

$$\Rightarrow m_i \mid x - y \;\; \forall i \in \{1, 2, \dots r\}$$

$$\Rightarrow mcm(m_1, m_2, .., m_r) \mid x - y \Rightarrow m \mid x - y$$

$\therefore x \equiv y \, (m) \;\; (con \; m = \prod_{t=1}^{r} m_t \;)$

2.2. Pequeño teorema de Fermat:

Si p primo, $a \in \mathbb{N}$ y p no divide a $a \Rightarrow a^{p-1} \equiv 1 \, (p)$

Demostración:

Sabemos que la multiplicación por $\bar{a}$ con $\bar{a} \neq \bar{0}$ genera una permutación de $\mathbb{Z}_p^*$ pues $(p{:}a) = 1$

Luego, dado $t \in \mathbb{N} \; / \; t < p, \; \exists \, x_t \in \mathbb{Z}, \; 1 \leq x_t \leq p - 1 \; / \; ta \equiv x_t \, (p)$

De forma que $x_1, x_2, \dots, x_{p-1}$ es un reordenamiento de 1, 2, …, p-1

$\Rightarrow 1a \equiv x_1 \;\; (p), \; 2a \equiv x_2 \, (p), \; \dots, \; (p-1)! \, a = x_{p-1} \;\; (p)$

Multiplicando miembro a miembro las congruencias $ta \equiv x_t \, (p)$, se tiene:

$$(p-1)! \; a^{p-1} \equiv \prod_{t=1}^{p-1} x_t \equiv (p-1)! \;\; (p)$$

$$\Rightarrow (p-1)! \; a^{p-1} \equiv (p-1)! \;\; (p)$$

Como $\big((p-1)!:(p)\big) = 1 \Rightarrow \exists\, p' \in \mathbb{Z} \;\; / \; p'^{(p-1)}! \equiv 1\,(p)$

$\Rightarrow p'\,(p-1)!\; a^{p-1} \equiv p'.(p-1)! \;\; (p)$ (componiendo a izquierda por p')

$$\therefore a^{p-1} \equiv \; 1\,(p)$$

Analizando la relación entre p y a:

$$Si\; p \mid a \;\Rightarrow\; a \equiv 0\,(p) \wedge \; a^{p-1} \equiv 0\,(p) \Rightarrow a^p \equiv a \;\; (p)$$

$$Si\;(\,a : p) = 1 \;\Rightarrow\; a^{p-1} \equiv 1\,(p) \Rightarrow a^p \equiv a\,(p)$$

Lo que nos lleva al corolario: $Sea\; p\; primo \;\wedge a \in \mathbb{N} \Rightarrow \; a^p \equiv a\,(p)$

<u>Definición</u>:

Orden de a módulo p : $ord_p\,(a)$

$ord_p\,(a)$= min $\{k \in \mathbb{N} \mid a^k \equiv 1 \;\; (p)\}$

<u>2.3. Propiedades:</u>

I) $ord_p\,(a) \mid p-1$

II) $a^n \equiv 1\,(p) \Leftrightarrow \; ord_p\,(a) \mid n$

III) $a^r \equiv \; a^s\,(p) \;\Leftrightarrow\; r \equiv s\,(ord_p\,(a))$

<u>Demostración I</u>

Tomo $m = ord_p\,(a)$

$$a^{p-1} \;\equiv 1\,(p) \;\wedge p-1 = mk + r \;\; con\; k \in \mathbb{N}, 0 \;\leq r \;\leq m$$

$$\Rightarrow \; a^{p-1} \;\; \equiv (a^m)^k\, a^r\,(p)$$

$1 \equiv 1\, a^r \;\; (p)$ pues $a^{p-1} \equiv 1\,(p) \;\wedge\; a^m \equiv 1\,(p)$ (por pequeño teorema de Fermat y por definición de orden), luego $a^r \equiv 1\,(p)$

Supongamos que $0 < r < m$ absurdo! pues $m = \min\,\{k \in \mathbb{N} \mid a^k \equiv 1\,(p)\}$, sino r sería menor que el mínimo, por lo que $r = 0$

$\Rightarrow \; p-1 = mk \;\Rightarrow\; m \mid p-1 \;\Rightarrow\; ord_p\,(a) \mid p-1$

<u>Demostración II</u>

Queremos ver que $a^n \equiv 1\,(p) \Leftrightarrow \; ord_p\,(a) \mid n$

Tomo $m = ord_p(a)$

Probemos la vuelta,

$ord_p(a) \mid n \Leftrightarrow m \mid n \Leftrightarrow \exists\, k \in \mathbb{Z} \mid n = mk$

$$a^n \equiv a^{mk}\ (p)$$

$$a^n \equiv (a^m)^k\ (p)$$

$$a^n \equiv 1\ (p) \quad (pues\ a^m \equiv 1\ (p)\ por\ definición\ de\ orden)$$

Probemos la ida,

Supongamos que $a^n \equiv 1\ (p) \wedge m\ no\ divide\ a\ n$

$$m\ no\ divide\ a\ n \Rightarrow n = mk + r\ \ con\ k \in \mathbb{Z} \wedge 0 < r < m$$

$$a^n \equiv a^{mk+r}\ (p)$$

$$a^n \equiv (a^m)^k a^r\ (p)$$

$$a^n \equiv 1\ a^r\ (p)\ \ (por\ a^m \equiv 1\ (p)\ por\ definición\ de\ orden)$$

Como $0 < r < m$, entonces r es menor que el mínimo m . Absurdo!

$$\Rightarrow m \mid n \Rightarrow ord_p(a) \mid n$$

$$\therefore a^n \equiv 1\ (p) \Leftrightarrow ord_p(a) \mid n$$

<u>Demostración III</u>

Queremos ver que: $a^r \equiv a^s\ (p) \Leftrightarrow r \equiv s\ (ord_p(a))$

Probemos la vuelta,

Tomo $m = ord_p(a)$

$r \equiv s\ (m) \Rightarrow m \mid r - s \Rightarrow \exists\, k \in \mathbb{Z} \mid\ r - s = m\,k \Rightarrow r = mk + s$

$a^r \equiv a^r\ (p) \Rightarrow a^r \equiv a^{mk+s}\ (p) \Rightarrow a^r \equiv (a^m)^k . a^s\ (p) \Rightarrow a^r \equiv 1\ a^s\ (p)\ \ (pues\ a^m \equiv 1\ (p))$

Probemos la ida,

(A) $Si\ r = s \Rightarrow r \equiv s\ (m)$

(B) $Si\ r < s \Rightarrow s = r + t\ \ (con\ t \in \mathbb{N})$

$$a^r \equiv a^s\ (p) \Rightarrow a^r \equiv a^r . a^t\ (p) \Rightarrow a^t \equiv 1\ (p) \Rightarrow m \mid t$$

$$s - r = r + t - r = t \wedge m \mid t \Rightarrow m \mid s - r \Rightarrow s \equiv r\ (m)$$

(C) Análogamente para $r > s$ se demuestra que $r \equiv s\ (m)$

De (A), (B) y (C): $r \equiv s\ (m)$

Como consecuencia si $a^i \equiv a^j \ (p) \wedge \ 0 \leq i \leq j \leq m-1 \Rightarrow i \equiv j \ (m)$

Dado que $0 \leq i \leq j \leq m-1 \Rightarrow i = j$

Además, $a^n \equiv a^r \ (p)$ con r el resto de dividir a n por m

$pues\ a^n \equiv a^{mk+r} \equiv (a^m)^k \ a^r \equiv a^r \ (p)$

$\Rightarrow a^n \equiv a^r \ (p) \wedge 0 \leq r < m-1$

Por lo que cuando $m = p-1$, las potencias son un sistema reducido de restos módulo p, sino si m divide a p – 1 y m no es p-1 generan ciclos que no recorren a todos los elementos de $\mathbb{Z}_p^*$.

<u>Definición</u>:

Si m es un número natural, la cantidad de números naturales menores o iguales que m y coprimos con m, se llamará ***indicador de Euler*** de m, lo simbolizamos $\varphi(m)$.

$$\varphi(m) = \#\{n \in \mathbb{N} |\ n \leq m \wedge (n{:}m) = 1\}$$

<u>2.4. TEOREMA DE FERMAT – EULER:</u> Sea $m \in \mathbb{N} \wedge (a : m) = 1 \Rightarrow a^{\varphi(m)} \equiv 1 \ (m)$

<u>Demostración</u>

Sea $I = \{x_1, x_2 \ldots, x_{\varphi(m)}\} = \{x_i / \ (x_i : m) = 1\}$, al ser los elementos restos coprimos con m, son inversibles módulos m.

Si a es inversible módulo m , sabemos que la multiplicación por $\overline{a}$ permuta las clases inversibles módulo m. Luego por cada índice i, $\exists\, j_i \mid \ ax_i \equiv x_{j(i)} \ (m)$

Por lo tanto, $\left\{x_{j(1)}, x_{j(2)}, \ldots, \ x_{j(\varphi(n))}\right\}$ es una permutación de I.

$$a\, x_1 \equiv x_{j\,(1)} \ (m)$$

$$a\, x_2 \equiv x_{j\,(2)} \ (m)$$

$$\ldots$$

$$a\, x_{\varphi\,(m)} \equiv x_{j\,(\,\varphi\,(m))} \ (m)$$

Multiplicando miembro a miembro, se tiene:

$a^{\varphi(m)} \prod_{i=1}^{\varphi\,(m)} x_i \equiv \prod_{i=1}^{\varphi\,(m)} j_i \equiv \prod_{i=1}^{\varphi\,(m)} x_i \ (m)$

$$\Rightarrow a^{\varphi(m)} \prod_{i=1}^{\varphi(m)} x_i \equiv \prod_{i=1}^{\varphi(m)} x_i \quad (m)$$

Como $(x_i : m) = 1 \Rightarrow (\prod_{i=1}^{\varphi(m)} x_i$: m) = 1 , por lo existe el inverso de ese elemento módulo m.

$$\Rightarrow \exists\, x' / \prod_{i=1}^{\varphi(m)} x_i \,.\, x' \equiv 1 \ (m)$$

$$a^{\varphi(m)} \prod_{i=1}^{\varphi(m)} x_i \,.\, x' \equiv \prod_{i=1}^{\varphi(m)} x_i \,.\, x' \ (m) \ (componiendo\ a\ derecha\ por\ x')$$

$$a^{\varphi(m)}.1 \equiv 1\,(m) \Rightarrow a^{\varphi(m)} \equiv 1\,(m)$$

<u>Propiedad 2.5</u>: $\varphi(p) = p - 1$ (con p primo)

<u>Demostración</u>

Si p primo: $(1 : p) = 1$, $(2 : p) = 1$, ..., $(p - 1 : p) = 1$

$\Rightarrow (n : p) = 1 \ \ (con\ 1 \leq n \leq p - 1)$

$\Rightarrow \varphi\,(p) = p - 1$

La función φ es posible calcularla especializada en cualquier natural, sin embargo, no entraremos en detalles sobre esto, pues solo nos interesa en el trabajo esta última propiedad.

Residuos cuadráticos, reciprocidad cuadrática y símbolos

<u>Definición</u>: Si p primo y $g \in \mathbb{Z} \mid \ p\ no\ divide\ a\ g$,

g es una **raíz primitiva** módulo p $\Leftrightarrow ord_p(g) = p - 1 \Leftrightarrow g^{ord_{p(g)}} \equiv 1 \ (p)$

Podemos inferir que las potencias de g al ocurrir esto generan a $\mathbb{Z}_p^*$, luego se dice que la clase g es un generador de $\mathbb{Z}_p^*$.

3.1. Teorema de Lagrange:

Sea p primo y $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ un polinomio con coeficientes enteros, donde p no divide a a_n, entonces la ecuación $f(x) = 0$, tiene a lo sumo n soluciones distintas en $\mathbb{Z}_p$.

Demostración:

Si $n = 1$, la ecuación $a_1 x + a_0 \equiv 0\ (p)$ tiene 1 solución pues $(a_1 : a_0) = 1$ (por hipótesis)

Sea n>1, si f no tiene soluciones, no hay nada que demostrar. Luego si x_0 solución de

$f(x) \equiv 0\ (p)$, dividiendo a f por $x - x_0$, tenemos que:

$f(x) = (x - x_0)\, h(x) + r$, con h polinomio de coeficientes enteros y r entero.

Si $x = x_0 \Rightarrow f(x_0) = (x_0 - x_0)h(x_0) + r \Rightarrow f(x_0) = r$

Luego, como $f(x) \equiv 0\ (p) \Rightarrow f(x_0) \equiv 0\ (p) \Rightarrow r \equiv 0\ (p)$.

Tomando a otra solución z distinta de x_0 en módulo p.

Tenemos que: $f(z) = (z - x_0)h(z) + r$

Luego , $f(z) \equiv (z - x_0)h(z)\ (p)$, pues $(r \equiv 0\ (p))$

Además, $f(z) \equiv 0\ (p) \Rightarrow 0 \equiv (z - x_0)h(z)\ (p) \Rightarrow p|\ (z - x_0)h(z)$

Como $z \not\equiv x_0\ (p) \Rightarrow p\ no\ divide\ a\ (z - x_0)$

$$p|\ (z - x_0)h(z) \wedge p\ no\ divide\ a\ (z - x_0) \Rightarrow p|h(z) \Rightarrow h(z) \equiv 0\ (p)$$

Por lo tanto, de ser z solución de f (distinta de x_0), llegamos a que z es solución de h, por lo tanto las soluciones de f distintas de x_0 son soluciones de h, siendo h un polinomio de grado n-1. De repetirse este proceso n-1 veces, se tiene que f tiene a lo sumo n soluciones distintas en $\mathbb{Z}_p$. Debido a que si en algún momento no hubiese más soluciones o no hubiera soluciones distintas, tendría una cantidad de soluciones distintas menor a n, por lo tanto, se cumple lo que queremos demostrar.

Proposición 3.2: Sea p primo y d tal que $d|p - 1$. Si $\mathbb{Z}_p^*$ admite algún elemento de orden d, entonces admite $\varphi(d)$ elementos de orden d.

Demostración:

Supongamos que $ord_p a = d$, y los elementos $a^0, a^1, \ldots a^{d-1}$, representan elementos distintos en $\mathbb{Z}_p^*$.

Sea un exponente i de los anteriores: $\left(a^i\right)^d \equiv \left(a^d\right)^i \equiv 1\ (p)$

Entonces, $\left(a^i\right)^d - 1 \equiv 0 \ (p) \Rightarrow a^i \ es\ solución\ de\ x^d - 1 = 0\ en\ \mathbb{Z}_p^*$

Por el teorema de Lagrange, la ecuación admite a lo sumo d soluciones distintos, como los elementos a^i son distintos entre sí por generar a $\mathbb{Z}_p^*$. Luego $a^0, a^1, \dots a^{d-1}$ son soluciones de $x^d - 1 = 0\ en\ \mathbb{Z}_p^*$

Por lo tanto, todo elemento b de orden d es de la forma a^j con $0 \leq j < d$.

¿Para qué valores de j entonces, ocurre que $ord_p a^j = d$?

Queremos ver si $(j{:}\,d) = 1 \Leftrightarrow ord_p\left(a^j\right) = d$

Probemos la vuelta,

Sea t tal que $(j{:}\,d) = t$

$\left(a^j\right)^{\frac{d}{t}} \equiv \left(a^d\right)^{\frac{j}{t}} \equiv 1 \ (p)$, pues $ord_p a = d$.

$$\Rightarrow \left(a^j\right)^{\frac{d}{t}} \equiv 1 \ (p)$$

$\Rightarrow d|\frac{d}{t}$ (pues d el orden), luego t=1, $\Rightarrow (j{:}\,d) = 1$

Probemos la ida,

Queremos ver si, $d|k, \forall k \in \mathbb{N}|\ \left(a^j\right)^k \equiv 1 \ (p)$

$$a^{jk} \equiv \left(a^j\right)^k \equiv 1 \ (p)$$

Como $ord_p a = d \Rightarrow d|jk$, y como $(d{:}\,j) = 1, \Rightarrow d|k \ \forall k \in \mathbb{N}$, entonces $ord_p\left(a^j\right) = d$

$$\therefore (j{:}\,d) = 1 \Leftrightarrow ord_p\left(a^j\right) = d$$

$\Rightarrow$ la cantidad de elementos de orden d en $\mathbb{Z}_p^*$ coincide con el número de exponentes j coprimos con d, tal que $0 \leq j < d$, por definición $\varphi(d)$.

Como corolario de esta proposición tenemos que si $a^d \equiv 1 \ (p) \Rightarrow$ los demás elementos de orden d en $\mathbb{Z}_p^*$ serán los a^j tales que $(j{:}\,d) = 1$.

<u>Definición</u>: Dado un divisor d de p-1, designamos a $\psi(d)$ el número de elementos de orden d en $\mathbb{Z}_p^*$. Luego por lo anterior sabemos que $\psi(d) = 0 \vee \psi(d) = \varphi(d)$. Por lo tanto,

$p - 1 = \sum_d \psi(d)$, donde $\sum_d$ indica que la suma se toma sobre todos los divisores positivos de $p - 1$

<u>Proposición 3.3</u>: Sea $n \in \mathbb{N} \Rightarrow n = \sum_d \varphi(n)$.

Demostración:

Sea $A_d = \{a \in [1;n] |\ (a:n) = d\}$

$$\Rightarrow [1;n] = \bigcup_d A_d \Rightarrow n = \sum_d \#A_d$$

Sea $a \in A_d \Rightarrow (a:n) = d \Rightarrow \left(\frac{a}{d}:\frac{n}{d}\right) = 1$

Como $a \leq n \Rightarrow \frac{a}{d} \leq \frac{n}{d}$, como $d|a \Rightarrow d \leq a \Rightarrow 1 \leq \frac{a}{d}$

$$\Rightarrow 1 \leq \frac{a}{d} \leq \frac{n}{d} \wedge \left(\frac{a}{d}:\frac{n}{d}\right) = 1$$

Si $b \in \left[1;\frac{n}{d}\right] \wedge \left(b:\frac{n}{d}\right) = 1$

$$\Rightarrow d\left(b:\frac{n}{d}\right) = d \Rightarrow \left(db:\frac{dn}{d}\right) = d \Rightarrow (db:n) = d$$

$$1 \leq b \leq \frac{n}{d} \Rightarrow 1 \leq d \leq bd \leq n$$

$$\therefore (db:n) = d \wedge 1 \leq bd \leq n \Rightarrow bd \in A_d$$

Si $b = \frac{a}{d} \wedge 1 \leq b \leq \frac{n}{d} \wedge \left(b:\frac{n}{d}\right) = 1 \Rightarrow a \in A_d$

Por lo tanto, la aplicación $x \to \frac{x}{d}$ establece una relación biunívoca entre A_d y los elementos del intervalo $\left[1;\frac{n}{d}\right]$ coprimos con $\frac{n}{d}$.

$$\Rightarrow \#A_d = \varphi\left(\frac{n}{d}\right) = \varphi(n)$$

$$\Rightarrow n = \sum_d \#A_d = \sum_d \varphi(n) \Rightarrow n = \sum_d \varphi(n)$$

<u>3.4. Teorema de existencia de raíces primitivas</u>: si p primo, existen raíces primitivas módulo p.

<u>Demostración</u>:

$$p - 1 = \sum_d \psi(d) \leq \sum_d \varphi(d) = p - 1$$

$$\Rightarrow \sum_d \psi(d) = \sum_d \varphi(d)$$

Luego para que sean iguales, todos sus términos deben serlo, y que si hubiera alguno en que $\psi(d) \leq \varphi(d)$, las sumatorias serían distintas.

Por lo tanto, para cada divisor d de p-1, existen en $\mathbb{Z}_p^*$, $\varphi(d)$ elementos de orden d. Tomando uno en particular tenemos que existen raíces primitivas en módulo p.

Residuos Cuadráticos

Definición: Dado un entero a, decimos que a es un residuo cuadrático módulo p (RC mod p) $\Leftrightarrow \exists x \in \mathbb{Z} |\ x^2 \equiv a\ (p)$ (con p primo)

Observaciones: La definición de RC depende de la clase de a módulo p, por lo que podríamos pensar a los valores de a y x en el intervalo [0; p-1] y como 0 es RC mod p, cualquiera sea el primo p. Podríamos analizar en las siguientes demostraciones considerando a $I_{p-1} = [1; p-1]$.

Definición: Sean $a, x_0 \in I_{p-1} |\ x_0^2 \equiv a\ (p)$, se dice que x_0 es una raíz cuadrada de a módulo p.

Proposición 3.5: La mitad de los elementos de I_{p-1} son RC mod p. Es decir, hay $\frac{p-1}{2}$ residuos cuadráticos módulo p. (con p primo impar)

Demostración

Sea g raíz primitiva módulo p, $\Rightarrow \forall a \in I_{p-1}: a = g^i\ \ con\ 0 \leq i \leq p-2$ (pues g genera a I_{p-1})

Quiero ver que: $a\ es\ RC\ mod\ p \Leftrightarrow i\ es\ par$

Probemos la ida:

Si $i\ es\ par \Rightarrow i = 2j\ \ con\ j \in \mathbb{N}_o$

$$a \equiv g^i \equiv g^{2j} \equiv \left(g^j\right)^2\ (p) \Rightarrow\ \ a \equiv \left(g^j\right)^2\ (p) \Rightarrow a\ es\ RC\ mod\ p$$

Probemos la vuelta:

$$a\ es\ RC\ mod\ p \Rightarrow\ \exists x \in I_{p-1} |\ a \equiv x^2\ (p)$$

Como x es un elemento de I_{p-1}, luego $\exists k |\ \ x \equiv g^k\ \ (p)$

Sabemos que: $g^i \equiv a \equiv x^2 \equiv \left(g^k\right)^2 \equiv g^{2k}\ \ (p) \Rightarrow g^i \equiv g^{2k}\ (p)$

Como $ord_p g = p-1$ (por ser g una raíz primitiva módulo p), luego $i \equiv 2k\ \ (p-1)$

$i - 2k \equiv 0\ \ (p-1) \Rightarrow p-1 | i-2k$

Como p es impar $\Rightarrow 2|p-1 \wedge p-1|i-2k \Rightarrow\ \ 2|i-2k \Rightarrow i \equiv 2k\ \ (2) \Rightarrow i \equiv 0\ \ (2) \Rightarrow i\ es\ par$

$$\therefore a\ es\ RC\ mod\ p \Leftrightarrow i\ es\ par$$

Luego como la mitad de los elementos de I_{p-1} son pares $\Rightarrow$ hay $\frac{p-1}{2}\ RC\ mod\ p\ en\ I_{p-1}$.

Con esta proposición previa, estamos en condiciones de demostrar lo siguiente:

<u>3.6. Criterio de Euler</u>: Sea $a \in \mathbb{Z} |\ (a:p) = 1,\ a\ es\ RC\ mod\ p \Leftrightarrow a^{\frac{p-1}{2}} \equiv 1\ (p)$, (con p primo impar).

<u>Demostración</u>:

Probemos la ida,

Sea a RC mod p y b cualquiera de sus raíces cuadradas módulo p.

$\Rightarrow a^{\frac{p-1}{2}} \equiv (b^2)^{\frac{p-1}{2}} \equiv b^{p-1} \equiv 1\ (p)$ (pues $(b:p) = 1$, ya que $(a:p) = 1$, y por pequeño teorema de Fermat)

$$\Rightarrow a^{\frac{p-1}{2}} \equiv 1\ (p)$$

Probemos la vuelta,

El teorema de Lagrange nos dice que $x^{\frac{p-1}{2}} \equiv 1\ (p)$ admite a lo sumo $\frac{p-1}{2}$ soluciones módulo p, puesto que hay $\frac{p-1}{2}$ residuos cuadráticos módulo p en I_{p-1}, luego cada RC mod p satisface la ecuación

$\Rightarrow x = a\ \ RC\ mod\ p$.

<u>Corolario</u>: Sea $a \in \mathbb{Z} |\ (a:p) = 1,\ \ a\ no\ es\ RC\ mod\ p \Leftrightarrow a^{\frac{p-1}{2}} \equiv -1\ (p)$, (con p primo impar)

<u>Demostración</u>:

Probemos la ida,

Por el pequeño teorema de Fermat, si $a \in \mathbb{N} \wedge (a:p) = 1 \Rightarrow a^{p-1} \equiv 1\ (p)$

$$\Rightarrow\ \ a^{\frac{p-1}{2}} \equiv \pm 1\ (p)$$

Si $a \in \mathbb{Z}^- \wedge (a:p) = 1 \Rightarrow -a \in \mathbb{N} \wedge (-a:p) = 1 \Rightarrow (-a)^{p-1} \equiv 1\ (p) \Rightarrow (-1)^{p-1}a^{p-1} \equiv 1\ (p)$

Como p es impar, p-1 es par $\Rightarrow (-1)^{p-1} = 1 \Rightarrow a^{p-1} \equiv 1\ (p) \Rightarrow\ \ a^{\frac{p-1}{2}} \equiv \pm 1\ (p)$

Si a=0, no podría ser coprimo con p.

Luego, si $a \in \mathbb{Z} \wedge (a:p) = 1$, tenemos que $a^{\frac{p-1}{2}} \equiv \pm 1\ (p)$

Si $a^{\frac{p-1}{2}} \equiv 1\ (p)$, entonces a sería RC mod p, lo cual es absurdo, por lo tanto

$$a^{\frac{p-1}{2}} \equiv -1\ (p)$$

Probemos la vuelta,

$a^{\frac{p-1}{2}} \equiv -1\ \ (p) \Rightarrow a^{\frac{p-1}{2}} \not\equiv 1\ \ (p) \Rightarrow a\ no\ es\ RC\ mod\ p$ (pues p impar y por contrarrecíproco del criterio de Euler)

<u>Proposición 3.7</u>: $-1\ es\ RC\ mod\ p \Leftrightarrow p \equiv 1\ (4)$ (con p primo impar)

<u>Demostración</u>:

Probemos la ida,

$$-1\ es\ RC\ mod\ p \Rightarrow (-1)^{\frac{p-1}{2}} \equiv 1\ (p) \Rightarrow \frac{p-1}{2} \equiv 0\ \ (2) \Rightarrow \frac{p-1}{2}.2 \equiv 0.2\ (2.2) \Rightarrow p-1 \equiv 0\ \ (4)$$

$$\Rightarrow p \equiv 1\ \ (4)$$

Probemos la vuelta,

$p \equiv 1\ \ (4) \Rightarrow\ p-1 \equiv 0\ \ (4)$, como p es impar entonces p-1 es par, luego $2|p-1$

$$p-1 \equiv 0\ \ (4) \wedge\ 2|p-1\ \wedge 2\ |0 \wedge\ 2|4 \Rightarrow \frac{p-1}{2} \equiv 0\ \ (2) \Rightarrow (-1)^{\frac{p-1}{2}} \equiv 1\ \ (p)$$

$$\Rightarrow\ -1\ es\ RC\ mod\ p$$

<u>Proposición 3.8</u>: $2\ es\ RC\ mod\ p \Leftrightarrow p \equiv \pm 1\ \ (8)$

<u>Demostración</u>:

Algunas notaciones que vamos a utilizar: I_{p-1}^{+} (primera mitad de I_{p-1}): $[1;\frac{p-1}{2}]$; I_{p-1}^{-} (segunda mitad de I_{p-1}): $[\frac{p+1}{2};p-1]$. Sean los elementos de $I_{p-1}^{+} = \{x_1, x_2, \dots, x_{\frac{p-1}{2}}\}$.

Sea $y_i = 2x_i \Rightarrow$ tendremos elementos $y_1, y_2, \dots, y_{\frac{p-1}{2}}$, algunos serán de I_{p-1}^{+} y otros de I_{p-1}^{-}.

Todo elemento y de I_{p-1}^{-} es inverso aditivo módulo p de algún elemento de I_{p-1}^{+}, debido a que $p-y \in I_{p-1}^{+}$, y vale la relación $y_i \equiv -(p-y_i)\ \ (p)$

$$\Rightarrow\ \ 2x_i \equiv \pm z_i\ \ (p) \qquad ,con\ z_i \in I_{p-1}^{+}$$

Ocurrirá $y_i \equiv 2x_i \equiv -z_i\ \ (p)$ cuando y_i sea mayor que $\frac{p-1}{2}$

Queremos ver que: $z_i = z_j \Leftrightarrow i = j$

Probemos la ida,

Si $z_i = z_j$, se tiene por su definición que: $2x_i \equiv 2x_j\ (p) \vee -2x_i \equiv -2x_j\ (p) o\ 2x_i = -2x_j\ (p)$

Como p impar, $(2{:}p) = 1$, luego : $x_i \equiv x_j\ (p) \vee x_i \equiv -x_j\ (p)$

Si $x_i \equiv x_j\ (p)$, como son positivos y menores que p, se tiene que $i = j$

Si $x_i \equiv -x_j\ (p) \Rightarrow x_i + x_j \equiv 0\ \ (p) \Rightarrow p | x_i + x_j \Rightarrow p \leq x_i + x_j$, sin embargo,

como $x_i < \frac{p}{2} \wedge x_j < \frac{p}{2} \Rightarrow\ x_i + x_j < p$, absurdo!, luego $x_i \not\equiv -x_j\ (p)$

$$\therefore i = j$$

Probemos la vuelta, $z_i = z_i \wedge i = j \Rightarrow z_i = z_j$

$$\therefore z_i = z_j \Leftrightarrow i = j$$

Si multiplico miembro a miembros las $\frac{p-1}{2}$ relaciones $2x_i \equiv \pm z_i\ \ (p)$, se tiene que:

$2^{\frac{p-1}{2}} \prod_i x_i \equiv (-1)^m \prod_i z_i\ \ (p)$ (con m la cantidad de elementos de índice i tales que $2x_i \in I_{p-1}^-$)

Además, como los elementos de I_{p-1}^+ son los x_i y también los z_i, se tiene que $(-1)^m \prod_i z_i = (-1)^m \prod_i x_i$

Luego, $2^{\frac{p-1}{2}} \prod_i x_i \equiv (-1)^m \prod_i x_i\ \ (p)$

Como cada $x_i \in I_{p-1}$: $(x_i{:}p) = 1 \Rightarrow (\prod_i x_i : p) = 1 \Rightarrow\ \ 2^{\frac{p-1}{2}} \equiv (-1)^m\ \ (p)$

Luego por criterio de Euler, $2\ es\ RC\ mod\ p \Leftrightarrow m\ es\ par$

Como m es la cantidad de elementos de índice i tales que $2x_i \in I_{p-1}^-$, analizamos los elementos de la forma $2x_i \in I_{p-1}^-$,

$$2x_i \in I_{p-1}^- \Leftrightarrow \frac{p-1}{2} < 2x_i \leq p-1 \Leftrightarrow \frac{p-1}{4} < x \leq \frac{p-1}{2}$$

Puede calcularse la cantidad de enteros comprendidos en el intervalo $\left(\frac{p-1}{4}; \frac{p-1}{2}\right]$, calculando $\left[\frac{p-1}{2}\right] - \left[\frac{p-1}{4}\right]$, siendo [ ] la parte entera.

Los valores de la parte entera dependen de la congruencia en módulo 4. Luego, en módulo 4, los primos pueden ser de la forma $p = 4k_1 + 1$ y $p = 4k_2 + 3$.

Si $p = 4k_1 + 1 \Rightarrow \left[\frac{p-1}{4}\right] = [k_1] = k_1\ \wedge\ \left[\frac{p-1}{2}\right] = [2k_1] = 2k_1 \Rightarrow \left[\frac{p-1}{2}\right] - \left[\frac{p-1}{4}\right] = 2k_1 - k_1 = k_1$

Tenemos que, $2\ es\ RC\ mod\ p \Leftrightarrow k_1\ par \Leftrightarrow p = 4(2k_3) + 1 = 8k_3 + 1 \Leftrightarrow p \equiv 1\ \ (8)$

Si $p = 4k_2 + 3 \Rightarrow \left[\frac{p-1}{4}\right] = \left[k_2 + \frac{1}{2}\right] = k_2 \wedge \left[\frac{p-1}{2}\right] = [2k_2 + 1] = 2k_2 + 1$

$$\Rightarrow \left[\frac{p-1}{2}\right] - \left[\frac{p-1}{4}\right] = 2k_2 + 1 - k_2 = k_2 + 1$$

Luego, $2\ es\ RC\ mod\ p \Leftrightarrow k_2\ impar \Leftrightarrow p = 4(2k_4 + 1) + 3 = 8k_4 + 7 \Leftrightarrow p \equiv -1\ \ (8)$

$\therefore 2\ es\ RC\ mod\ p \Leftrightarrow p \equiv \pm 1\ \ (8)$ (con p impar)

<u>El símbolo de Legendre</u>

<u>Definición</u>: Si $a \in \mathbb{Z} \wedge (a{:}p) = 1$, se define al símbolo de Legendre de a con respecto a p (con p primo impar):

$$\left(\frac{a}{p}\right): \begin{cases} 1 & si\ a\ es\ un\ RC\ mod\ p \\ -1 & si\ a\ no\ es\ un\ RC\ mod\ p \end{cases}$$

La definición tiene sentido debido al criterio de Euler y su corolario.

<u>3.9. Propiedades</u>:

I) $\left(\frac{a^2}{p}\right) = 1$

II) $a \equiv b\ \ (p) \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$

III) $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$ (es multiplicativo)

IV) $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$

V) $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$

<u>Demostración I:</u>

$$a^2 \equiv a^2\ (p) \Rightarrow a^2\ es\ RC\ mod\ p \Rightarrow \left(\frac{a^2}{p}\right) = 1$$

<u>Demostración II:</u>

Si $\left(\frac{a}{p}\right) = 1 \Rightarrow \exists x |\ x^2 \equiv a\ (p) \wedge a \equiv b\ (p) \Rightarrow x^2 \equiv b\ (p) \Rightarrow \left(\frac{b}{p}\right) = 1$

Análogamente, si $\left(\frac{b}{p}\right) = 1 \Rightarrow \left(\frac{a}{p}\right) = 1$

$$\therefore \left(\frac{a}{p}\right) = 1 \Leftrightarrow \left(\frac{b}{p}\right) = 1$$

Luego $\left(\frac{b}{p}\right) \neq 1 \Leftrightarrow \left(\frac{a}{p}\right) \neq 1$, luego $\left(\frac{b}{p}\right) = -1 \Leftrightarrow \left(\frac{a}{p}\right) = -1$

$$\therefore a \equiv b\ (p) \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$$

<u>Demostración III:</u>

$$\left(\frac{ab}{p}\right) \equiv (ab)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} b^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \quad (p)$$

<u>Demostración IV:</u>

$\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}}\ (p)$, por criterio de Euler, luego $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$

<u>Demostración V:</u>

$$\left(\frac{2}{p}\right) = 1 \Leftrightarrow 2\ es\ RC\ mod\ p \Leftrightarrow p \equiv \pm 1\ (8) \Leftrightarrow \frac{p^2-1}{8}\ es\ par$$

$$\Rightarrow \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$$

<u>Corolario</u>: $\left(\frac{ab}{p}\right) = 1 \Leftrightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$, con a y b coprimos con p.

Luego, dado $\left(\frac{a_1 a_2 \dots a_r}{p}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{p}\right)\dots\left(\frac{a_r}{p}\right)$, y en particular: $\left(\frac{a^m}{p}\right) = \left(\frac{a}{p}\right)^m \quad \forall a \in \mathbb{Z} \wedge \forall m \in \mathbb{N}$

<u>3.10. Lema de Gauss</u>:

Sea a tal que $(a{:}p) = 1$ y $m = \#\{x \in S |\ ax \in T\}$,

$$\Rightarrow a\ es\ RC\ mod\ p \Leftrightarrow m\ es\ par \Leftrightarrow \left(\frac{a}{p}\right) = (-1)^m$$

<u>Demostración</u>:

Sea $S \subset \mathbb{Z}_p^*,\ S = \{x_1, x_2, \dots, x_{\frac{p-1}{2}}\}$

Y sean los elementos $y_1, y_2, \dots, y_{\frac{p-1}{2}}$, tales que $y_i = ax_i$,

$\Rightarrow y \equiv -(p-y)\ \ (p)$ (ya que cualquier resto par es el inverso aditivo de uno impar, por ser p impar)

Luego, $\forall x_i \in \mathbb{Z}_p^*{:}\ ax_i \equiv \pm z_i\ \ (p)$, con $z_i \in S$

Queremos ver que los elementos z_i van a recorrer a los restos impares, por lo que nos será útil probar que $z_i = z_j \Leftrightarrow i = j$, para que luego si son distintos entre sí, su cardinal coincida con #S.

Probemos la ida,

$$z_i = z_j \Rightarrow ax_i \equiv ax_j\ (p) \vee ax_i \equiv -ax_j\ (p)$$

Como $(a{:}p) = 1$, existe el inverso de a módulo p $\Rightarrow x_i \equiv x_j\ \ (p)\ \vee x_i \equiv -x_j\ \ (p)$

Si $x_i \equiv x_j\ \ (p)$, como ambos son positivos y menores que p $\Rightarrow i = j$

Si $x_i \equiv -x_j\ \ (p) \Rightarrow\ \ x_i + x_j \equiv 0\ \ (p) \Rightarrow\ \ p|\, x_i + x_j$, lo cual es absurdo pues $x_i + x_j$ sería un múltiplo par positivo de p menor que 2p.

Para la vuelta, $i = j \wedge z_i = z_i \Rightarrow z_i = z_j$

$\therefore z_i = z_j \Leftrightarrow i = j$, luego $i \neq j \Leftrightarrow z_i \neq z_j$

Por lo tanto, los elementos z_i recorren a los restos impares.

Si multiplicamos miembro a miembro las expresiones $ax_i \equiv \pm z_i\ \ (p)$, se tiene que:

$a^{\frac{p-1}{2}} \prod_i x_i \equiv (-1)^m \prod_i z_i\ \ (p)\ \wedge\ \ \prod_i z_i = \prod_i x_i$ (con m el número de índices tal que $ax_i \in T$)

$$\Rightarrow a^{\frac{p-1}{2}} \prod_i x_i \equiv (-1)^m \prod_i x_i\ \ (p)$$

Como $\forall x_i{:}\ (x_i{:}p) = 1 \Rightarrow (\prod_i x_i : p) = 1 \Rightarrow$ existe su inverso módulo p

$$\Rightarrow a^{\frac{p-1}{2}} \equiv (-1)^m\ \ (p)$$

Aplicando el criterio de Euler se tiene que, $a\ es\ RC\ mod\ p \Leftrightarrow a^{\frac{p-1}{2}} \equiv 1\ \ (p) \Leftrightarrow m\ es\ par$

$$\therefore \left(\frac{a}{p}\right) = (-1)^m$$

<u>3.11. Ley de Reciprocidad Cuadrática</u>:

$\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}} \left(\frac{p}{q}\right)$, con p y q primos impares.

Lo cual es equivalente a: $\begin{cases} \left(\frac{q}{p}\right) = -\left(\frac{p}{q}\right) & si\ p \equiv q \equiv 3\ \ (4) \\ \left(\frac{q}{p}\right) = \left(\frac{p}{q}\right) & en\ otro\ caso \end{cases}$

<u>Demostración:</u>

Notaciones que vamos a utilizar:

S_p: $conjunto\ de\ elementos\ impares\ de\ I_{p-1}$

T_p: $conjunto\ de\ elementos\ pares\ de\ I_{p-1}$

S_q: $conjunto\ de\ elementos\ impares\ de\ I_{q-1}$

T_q: *conjunto de elementos pares de* I_{p-1}

$Q_p = \{i \in S_p : qi \in T_p \ (mod\ p)\}, \quad m = \#Q_p$

$P_q = \{j \in S_q : pj \in T_q \ \ (mod\ q)\}, \ n = \#P_q$

$G = \{qi - pj : \ (i,j) \in S_p x S_q\}$

Nos será útil demostrar primero las siguientes proposiciones:

I) Todo elemento de G es par y $0 \notin G$.

II) $\#G = \frac{p-1}{2}.\frac{q-1}{2}$

III) La función $f(x) = p - q - x$ determina una biyección de G en sí mismo.

<u>Demostración I</u>:

$qi \in T_p \wedge pj \in T_q \Rightarrow qi - pj \ es\ par \Rightarrow todo\ elemento\ de\ G\ es\ par$

Como $(q:p) = 1 \wedge (i:p) = 1 \Rightarrow (qi:p) = 1 \Rightarrow p\ no\ divide\ a\ q_i$ (con $0 < i < p$)

$$\Rightarrow qi \neq pj \quad \forall (i,j) \in S_p x S_q \Rightarrow \quad 0 \notin G$$

<u>Demostración II</u>:

Queremos ver que la correspondencia $(i,j) \to qi - pj$ es inyectiva.

$$qi_o - pj_0 = qi_1 - pj_1 \Rightarrow qi_0 \equiv qi_i \ \ (p) \wedge \ -pj_o \equiv -pj_1 \ \ (q)$$

$\Rightarrow i_o \equiv i_1 \ \ (p) \wedge \ \ j_0 \equiv j_1 \ \ (q)$ (por ser $(p:q) = 1$, existen los inversos de q y p, en módulos p y q).

$\Rightarrow (i_0, j_0) = (i_1, j_1)$ (por ser i_0 e i_1 positivos menores que p, y j_0 e j_1 positivos menores que q)

$\Rightarrow$ la correspondencia $(i,j) \to qi - pj$ es inyectiva.

$\therefore \#S_p x S_q = \#G \Rightarrow \#G = \frac{p-1}{2}.\frac{q-1}{2}$

<u>Demostración III</u>:

- f considerada de $\mathbb{Z} \to \mathbb{Z}$ es biyectiva.

- $f\big(f(x)\big) = f(p - q - x) = p - q - (p - q - x) = x \Rightarrow f\big(f(x)\big) = x$

- Queremos ver que: $x \in G \Rightarrow f(x) \in G$

$$x \in G \Rightarrow \exists\, (i,j) \in S_p x S_q |\ \ x = qi - pj$$

$$\Rightarrow f(x) = f(qi - pj) \Rightarrow f(x) = p - q - (qi - pj) = -q - qi + p + pj$$

$$\Rightarrow f(x) = q(-1 - i) - p(-1 - j) + pq - pq$$

$$\Rightarrow f(x) = q(p - 1 - i) - p(q - 1 - j)$$

$$i \in S_p \Rightarrow i\ impar \wedge 1 \leq i \leq p - 1 \Rightarrow\ 1 - p \leq -i \leq -1$$

$$\Rightarrow p - 1 + 1 - p \leq p - 1 - i \leq p - 1 - 1 \Rightarrow 0 \leq p - 1 - i \leq p - 2$$

Supongamos que $p - 1 - i = 0 \Rightarrow i = p - 1 \Rightarrow i\ es\ par$ Absurdo! (pues i es impar)

$$\Rightarrow p - 1 - i \neq 0 \Rightarrow 0 < p - 1 - i \leq p - 2$$

Como p impar e i impar, entonces p-1-i es impar.

$$\Rightarrow p - 1 - i \in S_p$$

Análogamente se puede demostrar que si $j \in S_q \Rightarrow q - 1 - j\ \in S_q$

$$\Rightarrow (p - 1 - i; q - 1 - j) \in S_p x S_q \Rightarrow\ \ f(x) \in G$$

∴ f determina una biyección de G en sí mismo.

Contaremos a los elementos de G considerando las siguientes particiones:

$$G_1 = \{x \in G: x < -q\}$$

$$G_2 = \{x \in G: x > p\}$$

$$G_3 = \{x \in G: -q < x < p\}$$

Las cuales son particiones ya que p y –q no pertenecen a G por ser impares, y por I sabemos que son pares los elementos de G.

Luego, $\#G = \#G_1 + \#G_2 + \#G_3$

<u>Proposición IV</u>: $\#G_1 = \#G_2 \wedge\ \ \#G_3 = m + n$

<u>Demostración IV</u>:

Si $x \in G_1 \Leftrightarrow x < -q \Leftrightarrow\ -q - x > 0 \Leftrightarrow p - q - x > p \Leftrightarrow f(x) > p$

$$f(x) > p \wedge f(x) \in G \Leftrightarrow x \in G_2$$

$$\therefore \#G_1 = \#G_2$$

Para demostrar $\#G_3 = m + n$, vamos a ver primero que los x positivos de G_3 pertenecen a T_p

$x > 0 \wedge x \in G_3 \Rightarrow\ -q < x < p \wedge x > 0 \wedge x \in G \Rightarrow 0 < x < p \wedge x = qi - pj$, con $(i,j) \in S_p x S_q$

$\Rightarrow 1 \leq x \leq p - 1\ \wedge x\ es\ par$ (pues $x \in G$) $\Rightarrow x \in T_p$

$x = qi - pj \Rightarrow qi = x + pj \Rightarrow\ qi \equiv x\ \ (p) \wedge i \in S_p \Rightarrow i \in Q_p$ (por definición de Q_p)

$\therefore x > 0 \wedge x \in G_3 |\ \ x = qi - pj\ \Rightarrow i \in Q_p$ (A)

Sea $r \in Q_p \wedge z \in T_p |\ \ qr \equiv z\ (p)$

$r \in Q_p \Rightarrow r \in S_p \wedge qr \equiv z\ \ (p)$ (con $z \in T_p$)

$$\Rightarrow \exists s |\ qr = ps + z$$

$\Rightarrow z = qr - ps$

Como z par y p, q, r impares, entonces s es impar.

$$s = \frac{qr - z}{p} \wedge z > 0\ \left(pues\ z \in T_p\right) \Rightarrow \frac{qr - z}{p} < \frac{qr}{p} \wedge r < p\ \left(pues\ r \in S_p\right) \Rightarrow \frac{qr}{p} < \frac{qp}{p} \Rightarrow \frac{qr}{p} < q$$

$$\Rightarrow s < q$$

Supongamos que $s \leq 1 \Rightarrow\ \ qr = ps + z \wedge ps + z < z - p\ \ \wedge z - p < 0$ (pues $z < p$ ya que $z \in T_p$)

$\Rightarrow\ \ qr < 0$ Absurdo!

$\Rightarrow 0 < s < q \wedge s\ impar \Rightarrow s \in S_q$, como $r \in S_p$ (por hipótesis) $\Rightarrow qr - ps \in G_3$

$\Rightarrow z \in G_3 \wedge z > 0$ (pues $z \in T_p$) (B)

Por (A) y (B), hay una biyección entre los elementos positivos de G_3 y los de Q_p, como $m = \#Q_p$, la cantidad de elementos positivos de G_3 es m.

Se procede análogamente probando que hay una biyección entre los elementos negativos de G_3 y los de P_q, y como $n = \#P_q$, la cantidad de elementos negativos de G_3 es n.

$\therefore G_3 = m + n$, pues $0 \notin G$

Como $\#G_1 = \#G_2 \wedge \#G_3 = m + n \wedge \#G = \#G_1 + \#G_2 + \#G_3$

$$\Rightarrow \#G = 2\#G_1 + \#G_3 \wedge \#G = \frac{p-1}{2}.\frac{q-1}{2} \Rightarrow\ \ 2\#G_1 + m + n = \frac{p-1}{2}.\frac{q-1}{2}$$

$$\Rightarrow m + n \equiv \frac{p-1}{2}.\frac{q-1}{2}\ \ (2)$$

Por lo que $\frac{p-1}{2}$ y $\frac{q-1}{2}$ tienen la misma paridad.

$$\Rightarrow (-1)^{m+n} = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}$$

$$(-1)^m(-1)^n = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}$$

$\Rightarrow \left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}$ (por Lema de Gauss)

$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right)\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\left(\frac{p}{q}\right)$ (componiendo a derecha con $\left(\frac{p}{q}\right)$)

$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right)^2 = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\left(\frac{p}{q}\right) \Rightarrow \left(\frac{q}{p}\right)\left(\frac{p^2}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\left(\frac{p}{q}\right)$

$$\therefore \left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\left(\frac{p}{q}\right)$$

<u>Símbolo de Jacobi</u>

<u>Definición</u>: Puede extenderse el símbolo de Lengedre $\left(\frac{a}{n}\right)$ (definido para n primo impar) a los n impares.

Sea a un número entero, y n un natural impar mayor que 1, suponiendo que

$n = \prod_i p_i$ es su descomposición en números primos, se define al símbolo de Jacobi como:

$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right)$, donde $\left(\frac{a}{p_i}\right)$ refiere al símbolo de Legendre.

Además, definimos que $\left(\frac{a}{1}\right) = 1$.

Luego, el símbolo queda definido de esta forma para a entero y n siendo un natural impar.

Puede notárselo con la misma simbología, debido a que para n es primo impar el símbolo es el de Legendre.

<u>Aclaración</u>: A diferencia del símbolo de Legendre, el símbolo de Jacobi no determina el carácter cuadrático pues la productoria puede arrojar 1, sin que cada uno de los factores sea 1.

<u>3.12. Propiedades</u>:

I) $\left(\frac{a}{n}\right) = 0 \Leftrightarrow (a:n) \neq 1$

II) $(a:n) = 1 \Rightarrow \left(\frac{a^2}{n}\right) = 1$

III) 1) $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$, 2) $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$ (multiplicidad)

IV) $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$, también $\left(\frac{-1}{n}\right) = 1 \Leftrightarrow n \equiv 1\ (4)$

V) $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$, también $\left(\frac{2}{n}\right) = 1 \Leftrightarrow n \equiv \pm 1\ (8)$

VI) $\left(\frac{m}{n}\right) = (-1)^{\frac{n-1}{2}\cdot\frac{m-1}{2}}\left(\frac{n}{m}\right)$ (reciprocidad cuadrática)

<u>Demostración I:</u>

$(a{:}n) \neq 1 \Leftrightarrow \exists p_k \in \mathbb{N} |\ \ p_k | a \wedge p_k | n$ (con p_k primo)

Por definición de símbolo de Jacobi: $\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\dots\left(\frac{a}{p_k}\right)\dots\left(\frac{a}{p_r}\right)$, siendo $n = \prod_{i=1}^{r} p_i$

Luego como $p_k | a \Rightarrow \left(\frac{a}{p_k}\right)$ no está definido, por lo que podemos decir que si $p_k | a \Rightarrow \left(\frac{a}{p_k}\right) = 0$, luego el símbolo de Jacobi $\left(\frac{a}{n}\right) = 0 \Leftrightarrow (a{:}n) \neq 1$.

Respecto a las propiedades II y III.1: son consecuencias de las propiedades del símbolo de Legendre. La propiedad III.2 es por definición del símbolo de Jacobi.

<u>Demostración IV</u>:

Si n es primo, entonces $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$

Si n es compuesto (producto de dos primos) $\Rightarrow n = pr$ con p y r primos impares.

$$\left(\frac{-1}{n}\right) = \left(\frac{-1}{p}\right)\left(\frac{-1}{r}\right) = (-1)^{\frac{p-1}{2}}(-1)^{\frac{r-1}{2}} = (-1)^{\frac{p+r-2}{2}}$$

Analizando la paridad de $\frac{n-1}{2}\ y\ \frac{p+r-2}{2}$, tenemos que: $\frac{n-1}{2} - \frac{p+r-2}{2} = \frac{pr-1-p-r+2}{2} = \frac{(p-1)(r-1)}{2}$

Como $p-1\ y\ r-1$ son pares, entonces $\frac{(p-1)(r-1)}{2}$ es par, luego tienen la misma paridad. Por lo que , $(-1)^{\frac{p+r-2}{2}} = (-1)^{\frac{n-1}{2}} \wedge \left(\frac{-1}{n}\right) = (-1)^{\frac{p+r-2}{2}} \Rightarrow \left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$

Si n es compuesto (producto de un primo por un compuesto) $\Rightarrow n = ps$ con p primo impar y s compuesto menor que n.

$$\left(\frac{-1}{n}\right) = \left(\frac{-1}{p}\right)\left(\frac{-1}{s}\right)$$

Si s es producto de dos primos, tenemos por lo anterior que $\left(\frac{-1}{s}\right) = (-1)^{\frac{s-1}{2}}$. Por lo que,

$$\left(\frac{-1}{n}\right) = \left(\frac{-1}{p}\right)\left(\frac{-1}{s}\right) = (-1)^{\frac{p-1}{2}}(-1)^{\frac{s-1}{2}}$$

Razonando análogamente con el análisis de la paridad entre $\frac{n-1}{2}$ y $\frac{p-1}{2} + \frac{s-1}{2}$, se llega a que $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$

Luego si s no es el producto de dos primos, puede ser el producto de un primo por un compuesto que sea el producto de dos primos, en dicho caso, $\left(\frac{-1}{s}\right) = (-1)^{\frac{s-1}{2}}$ y seguirá valiendo que $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$. Si s es el producto de un primo por un compuesto que no es el producto de dos primos, analizando a $\left(\frac{-1}{s}\right)$, eventualmente descomponiendo a s en algún momento llegaremos a un primo por un compuesto producto de dos primos, e inductivamente $\left(\frac{-1}{s}\right) = (-1)^{\frac{n-1}{2}}$, por lo que $\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}}$.

La propiedad V, se puede demostrar de forma similar a IV, analizando la paridad de los exponentes. Mientras que la demostración de la propiedad VI puede realizarse considerando a la definición del símbolo de Jacobi, luego aplicando la ley de reciprocidad cuadrática a cada factor de la productoria, y realizando un análisis similar sobre la paridad de los exponentes.

Fundamentos de algoritmos de aritmética modular

En esta parte del trabajo se justificarán los principales algoritmos con los que se desarrollarán posteriormente programas.

Representante de clase

Dado un valor a entero, si deseamos calcular el resto de dividirlo por m:

En caso que $a \geq 0$, no tenemos inconvenientes pues en el lenguaje C existe el operador %. En caso que $0 \leq a < m$, el resto será a, sino se procederá a calcularlo con el operador %.

Si $a < 0$, siendo $a = -a'$ $(a' > 0)$ tenemos dos posibilidades: que m divida a *a'*, o que no lo divida.

Si $m|a' \Rightarrow m|a \Rightarrow r_m(a) = 0$

Sino $a' = mk + r_m a'$ $(con\ 0 < r_m a' < m)$

$$\Rightarrow -a' = m(-k) + m - m - r_m a'\ (con - m < -r_m a' < 0)$$

$$\Rightarrow -a' = m(-k-1) + m - r_m a'\ (con\ 0 < m - r_m a' < m)$$

$$\Rightarrow r_m(-a') = m - r_m a' \Rightarrow r_m a = m - r_m(-a)$$

Siguiendo esa idea se creó la siguiente función:

```
long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}
```

Adición

Si queremos calcular a+b módulo m. Vamos a pensar lo siguiente

$$a \equiv r_{,m} a\ (m)\ \wedge b \equiv r_m b\ (m) \Rightarrow a + b \equiv r_m a + r_m b$$

Además, $a + b \equiv r_m(a + b)\ (m) \Rightarrow a + b \equiv r_m(r_m a + r_m b)\ (m)$

De esta forma, estaremos reduciendo siempre a números más manejables. Por lo anterior, tenemos una función para calcular los representantes, luego sumado a esto último mencionado se puede crear la siguiente función:

```
long long int suma_modular (long long int a, long long int b, long long int modulo) {

 return representante_modular(representante_modular(a,
modulo)+representante_modular(b, modulo), modulo);

}
```

Producto

Si queremos calcular a.b módulo n, similarmente a lo visto en 'Adición' tenemos que:

$$ab \equiv r_m a.r_m b \ \ (m) \wedge \ \ ab \equiv r_m(ab) \Rightarrow \ \ ab \equiv r_m(r_m a.r_m b) \ \ (m)$$

Lo que se puede ver en la siguiente función:

```
long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}
```

MCD

Pensando en el Algoritmo de Euclides, se puede realizar la siguiente función:

```
long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}
```

Inverso

Si deseamos calcular el inverso de un número en un módulo q, en un principio se puede pensar en que se prueben los números del 1 hasta q-1 y ver cual cumple la condición, sin embargo, la cantidad de operaciones que realiza el algoritmo crece rápidamente a medida que q es mayor, si bien es una manera sencilla de pensarlo, en algún momento se hace inviable debido a que el tiempo en hallarlo resulta ser absurdo.

Necesitamos entonces de un algoritmo con una mejor convergencia para calcular el inverso modular, ello es encontrar un $p'|\ \ p.p' \equiv 1\ (mod\ q)$, por definición de congruencia tenemos que:

$$q|p.p'-1 \Leftrightarrow \exists t\epsilon\mathbb{Z}\ |\ p.p'-1 = q.t\ \ \Leftrightarrow\ p.p' + q.(-t) = 1$$

Es decir que al inverso lo podemos hallar (si existe) al resolver dicha ecuación diofántica y existirá si $(p{:}q)|1$, por lo tanto $(p{:}q) = 1$.

Lo que necesitamos es expresar a 1 como combinación lineal de p y q, pues al hacerlo tendríamos una solución particular de la ecuación diofántica y una manera de hacerlo es pensar que 1 es el mcd de p y q , entonces si desarrollamos el Algoritmo de Euclides para el cálculo del mcd, tendríamos la manera de encontrar la combinación lineal buscada. Así que el primer algoritmo a desarrollar será para expresar a 1 como combinación lineal de p y q, utilizando el Algoritmo de Euclides.

Como en el algoritmo de Euclides en algún momento se llegará a un resto nulo, y como p y q no son divisibles, puedo definir la siguiente sucesión finita:

$$a(p,q) = \begin{cases} a_1 = p \\ a_2 = q \\ a_n = r_{a_n-1}(a_{n-2}) \end{cases}$$ (si $n \leq m$, siendo m tal que a_m=0)

siendo $r_{a_n-1}(a_{n-2})$ el resto de dividir a a_{n-2} por a_{n-1}.

Sea $\bar{a} = (a_1, a_2)$ y $\bar{k} \in \mathbb{Z}^2$, tenemos que $a_1 = (a_1, a_2).(1{,}0) = \bar{a}.\overline{k_1}$ y que $a_2 = (a_1, a_2).(0{,}1) = \bar{a}.\overline{k_2}$. Quiero ver que: $a_{i-2} = \bar{a}.\overline{k_{\iota-2}} \wedge a_{i-1} = \bar{a}.\overline{k_{\iota-1}} \Rightarrow a_i = \bar{a}.\overline{k_\iota}$, de esta manera sabríamos que cada iteración del algoritmo es una combinación lineal de $a_1\ y\ a_2$.

Dividiendo a_{i-2} por a_{i-1}, tenemos que: $a_{i-2} = a_{i-1}.c_{i-2} + r_{a_i-1}(a_{i-2})$

$a_{i-2} = a_{i-1}.c_{i-2} + a_i$ (por definición de a_n)

$a_i = a_{i-2} - a_{i-1}.c_{i-2}$

$a_i = \bar{a}.\overline{k_{\iota-2}} - \bar{a}.\overline{k_{\iota-1}}.c_{i-2}$ (por hipótesis)

$a_i = \bar{a}.(\overline{k_{\iota-2}} - \overline{k_{\iota-1}}.c_{i-2})$

$a_i = \bar{a}.\overline{k_\iota}$, siendo $\overline{k_\iota} = \overline{k_{\iota-2}} - \overline{k_{\iota-1}}.c_{i-2}$

Por lo tanto, se verifica que cada número de la serie puede ser expresado como el producto escalar de $\bar{a}$ y de $\bar{k}$, siendo $\overline{k_1} = (1,0), \overline{k_2} = (0,1)\ y\ \overline{k_n} = \overline{k_{n-2}} - \overline{k_{n-1}}.c_{n-2}$.

El último término de la serie T_u será el MCD entre p y q, por lo tanto, será 1. Finalmente,

$$1 = \bar{a}.\overline{k_u} = (a_1, a_2)(k_{u,1}, k_{u,2}) = (p, q)(k_{u,1}, k_{u,2}) = p.k_{u,1} + q.k_{u,2} \Rightarrow p.k_{u,1} + q.k_{u,2} = 1,$$

y ésta es la combinación lineal que estábamos buscando.

Dada esta fundamentación teórica se procede a detallar el algoritmo:

Se partirá de una matriz que en cada iteración irá modificándose, en la iteración n será de la forma $MatGen(n) = \left(\frac{\overline{k_{n+1}}}{\overline{k_{n+2}}}\right) = \begin{pmatrix} k_{n+1;1} & k_{n+1;2} \\ k_{n+2;1} & k_{n+2;2} \end{pmatrix}$, y su siguiente

$MatGen(n+1) =$
$\begin{pmatrix} MatGen(n)_{2;1} & MatGen(n)_{2;2} \\ MatGen(n)_{2;1}.(-c_n) + MatGen(n)_{1;1} & MatGen(n)_{2;2}.(-c_n) + MatGen(n)_{1;2} \end{pmatrix}$, siendo c_n el cociente de dividir a a_{n-1} por a_n, obteniéndose los cocientes por el algoritmo de Euclides.

Por ejemplo al inicio del algoritmo o iteración 0, será $MatGen(0) = \left(\frac{\overline{k_1}}{\overline{k_2}}\right) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, luego en la iteración 1: $MatGen(1) = \left(\frac{\overline{k_2}}{\overline{k_3}}\right) = \begin{pmatrix} 0 & 1 \\ 0 * (-c_1) + 1 & 1 * (-c_1) + 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -c_1 \end{pmatrix}$.

Al finalizar el algoritmo la segunda fila de la matriz serán los coeficientes de la combinación lineal buscada, es decir que la última iteración será $MatGen(u-2) = \begin{pmatrix} k_{u-1;1} & k_{u-1;1} \\ k_{u;1} & k_{u;2} \end{pmatrix}$.

En lenguaje C se puede programar de la siguiente manera, siendo $a_1\ y\ a_2$ naturales:

```
#include <stdio.h>
#include <conio.h>

int main() {
    long long int a,b,r,c;
    //matriz generada
    long long int matgen[2][2]={{1,0},{0,1}};
    long long int aux0,aux1;
    printf("Halla la combinacion lineal del MCD de A y B:\n\nIngrese A(espacio)B: ");
    scanf("%lld %lld",&a,&b);
    printf("\n[%lld,%lld].",a,b);
    while (b!=0) {
        c=a/b;
        r=a%b;
        a=b;
        b=r;
        //iteracion
        aux0=matgen[1][0]*(-c)+matgen[0][0];
```

```
        aux1=matgen[1][1]*(-c)+matgen[0][1];
        matgen[0][0]=matgen[1][0];
        matgen[0][1]=matgen[1][1];
        matgen[1][0]=aux0;
        matgen[1][1]=aux1;
    }

    printf("[%lld,%lld]=%lld",matgen[0][0],matgen[0][1],a);
    getch();
}
```

Luego, para hallar el inverso se puede modificar parte del algoritmo, omitiendo la columna correspondiente al coeficiente de q (es decir la segunda columna), y solo quedándonos con la de p, pues en el algoritmo anterior las columnas son independientes, no se relacionan entre ellas, solo las filas. Por lo cual se puede programar la siguiente función:

```
long long int inverso_modular (long long int numero, long long int modulo) {
    long long int a=numero,b=modulo,r,c;
    //matriz generada modificada
    long long int matgen[2]={1,0};
    long long int aux;
    while (b!=0) {
        c=a/b;
        r=a-b*c;
        a=b;
        b=r;
        //iteracion
        aux=matgen[1]*(-c)+matgen[0];
        matgen[0]=matgen[1];
        matgen[1]=aux;
    }
    if (a!=1) return 0; else return representante_modular(matgen[0],modulo);
}
```

Aquí la función inverso_modular tomará como parámetros a p y a q, y devolverá el inverso si existe y es única su clase, sino devolverá 0. Luego actuará la función representante_modular devolviendo el valor del inverso entre 0 y q-1.

Potenciación

La calcularemos mediante el método de exponenciación binaria, este nos dice que si queremos calcular a^b, hagamos lo siguiente:

-Si b=1, el resultado es a.

-Si b es par, se procede a calcular $\left(a^{\frac{b}{2}}\right)^2$

-Si b es impar, se procede a calcular $a.a^{b-1}$

Luego como b cada en cada iteración es menor se llega a b=1, con lo que concluye el cálculo.

Luego si queremos calcular a^b módulo m, podríamos hacer lo siguiente:

$$a^b\ (mod\ m) = \begin{cases} r_m a & si\ b = 1 \\ r_m(r_m a^{\frac{b}{2}}.r_m a^{\frac{b}{2}}) & si\ b\ es\ par \\ r_m(r_m a.r_m a^{b-1}) & si\ b\ impar \end{cases}$$

Con lo cual podríamos escribir a la siguiente función:

```
long long int exponenciacion_binaria_modular(long long int base, long long int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base, exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return producto_modular(base,exponenciacion_binaria_modular(base, exponente-1,modulo),modulo);

}
```

Factorial

Si queremos calcular el factorial de a en módulo m, pensando que $a! \equiv r_m a!\ (m) \wedge a.(a-1)! \equiv r_m(r_m a.r_m(a-1)!)\ (m)$, podemos escribir la siguiente función:

```
long long int factorial_modular (long long int numero, long long int modulo) {

    if ((numero==1)||(numero==0)) return 1;

    return producto_modular(numero, factorial_modular(numero-1,modulo),modulo);

}
```

Símbolo de Jacobi

Algo que nos resultará muy útil es poder calcular el símbolo de Jacobi $\left(\frac{m}{n}\right)$, ya que hemos visto que es una generalización del símbolo de Legendre, el cual nos permitía saber si m es residuo cuadrático módulo n (si n es primo impar), la cual tiene ciertas ventajas.

El algoritmo que se desarrolló para este trabajo es el siguiente:

Sea $m \in \mathbb{Z}\ y\ n \in \mathbb{N}\ (n\ impar)$,

1. Procedemos a calcular $\left(\frac{m}{n}\right)$
2. Si m=1 o n=1, el algoritmo termina arrojando 1. (Pues $\left(\frac{1}{n}\right) = 1$ y $\left(\frac{m}{1}\right) = 1$)
3. Si no se cumple que $0 \leq m \leq n$, entonces $\left(\frac{m}{n}\right) = \left(\frac{r_n m}{n}\right)$ (aplicando la propiedad $a \equiv b\ (n) \Rightarrow \left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$, pues $m \equiv r_n m\ (n)$). Luego $0 \leq r_n m < m$, y se vuelve al paso 1.

4. Dado que se cumple que $0 \leq m \leq n$, se calcula (m:n). Si $(m{:}n) \neq 1$. Termina el algoritmo arrojando 0.

5. Si $m \neq n - 1$, salto al paso 7.

6. Como m=n-1, calculo n en módulo 4.

6.1 Si $n \equiv 1\ (4)$, el algoritmo termina arrojando 1.

6.2 SI $n \not\equiv 1\ (4)$, el algoritmo termina arrojando -1.

(Pues $\left(\frac{-1}{n}\right) = 1 \Leftrightarrow n \equiv 1\ \ (4)$)

7. Si $m \neq 2$, salto al paso 9.

8. Como m=2, calculo n en módulo 8.

8.1 Si $n \equiv 1\ (8) \vee n \equiv 7(8)$, el algoritmo termina arrojando 1.

8.2 Si $n \not\equiv \pm 1\ (8)$, el algoritmo termina arrojando -1.

(pues $\left(\frac{2}{n}\right) = 1 \Leftrightarrow n \equiv \pm 1\ (8)$)

9. Si m es par, $\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right).\left(\frac{\frac{m}{2}}{n}\right)$ y se vuelve al paso 1. (Por propiedad $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$)

10. Si n es par, $\left(\frac{m}{n}\right) = \left(\frac{m}{2}\right)\left(\frac{m}{\frac{n}{2}}\right)$ y se vuelve al paso 1. (Por propiedad $\left(\frac{m}{ab}\right) = \left(\frac{m}{a}\right)\left(\frac{m}{b}\right)$)

11. Llegado hasta acá m, n son impares mayores o iguales a 3. Así que estamos en condiciones de aplicar la ley de reciprocidad cuadrática.

11.1 Si $\frac{m-1}{2}$ es par $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$ y se vuelve al paso 1.

11.2 Si $\frac{n-1}{2}$ es par $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)$ y se vuelve al paso 1.

11.3 Si $\frac{m-1}{2}$ y $\frac{n-1}{2}$ son impares, entonces $\left(\frac{m}{n}\right) = -\left(\frac{n}{m}\right)$ y se vuelve al paso 1.

(pues $\left(\frac{m}{n}\right) = (-1)^{\frac{n-1}{2}.\frac{m-1}{2}}\left(\frac{n}{m}\right)$)

Resumiendo: Si m y n no son coprimos arroja 0, si alguno es 1 arroja 1. Sino, por el paso 3 se garantiza que m<n, luego si m es 1, 2 o n-1 el algoritmo termina y si no lo es, por ley de reciprocidad cuadrática, n pasa a ser m, haciendo que cada vez m sea menor hasta llegar a 1, 2 o n-1.

Podemos escribir la siguiente función, que nos vaya mostrando como se va calculando:

```
int jacobi(long long int m, long long int n) {

	if ((m==1)||(n==1)) {printf("\n(%lld/%lld)=1",m,n);return 1;}

	if ((m>n)||(m<0)) {


	  printf("\n(%lld/%lld)=(%lld/%lld)",m,n,representante_modular(m,n),n);

		return jacobi(representante_modular(m,n),n);}

	if ((mcd(m,n))!=1) {printf("\nNo son coprimos el mcd es %lld",mcd(m,n));return 0;}

	if (m==(n-1)) {if (representante_modular(n,4)==1) {printf("\n(%lld/%lld)=1",m,n);return 1;} else {printf("\n(%lld/%lld)=-1",m,n);return -1;}}

	if (m==2) {if ((representante_modular(n,8)==1)||(representante_modular(n,8)==7)) {printf("\n(%lld/%lld)=1",m,n);return 1;} else {printf("\n(%lld/%lld)=-1",m,n);return -1;}}

	if (representante_modular(m,2)==0) {printf("\n(%lld/%lld)=(2/%lld)(%lld/%lld)",m,n,n,m/2,n);return jacobi(2,n)*jacobi(m/2,n);}

	if (representante_modular(n,2)==0) {printf("\n(%lld/%lld)=(%lld/2)(%lld/%lld)",m,n,m,m,n/2);return jacobi(m,2)*jacobi(m,n/2);}
```

```
        if (representante_modular((m-1)/2,2)==0) {printf("\n(%lld/%lld)=1*(%lld/%lld)",m,n,n,m);return jacobi(n,m);} else {

        if (representante_modular((n-1)/2,2)==0) {printf("\n(%lld/%lld)=1*(%lld/%lld)",m,n,n,m);return jacobi(n,m);} else {printf("\n(%lld/%lld)=(-1)*(%lld/%lld)",m,n,n,m);return (-1)*jacobi(n,m);}

    }

}
```

Programa: Aritmética Modular

Para concluir con esta parte, se desarrolló un programa con todas las funciones mencionadas, para observar su funcionamiento y ser una herramienta de utilidad para más adelante.

```
#include <stdio.h>

#include <conio.h>

#include <windows.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;} else return numero%modulo;} else return numero%modulo;

}


long long int suma_modular (long long int a, long long int b, long long int modulo) {

 return representante_modular(representante_modular(a, modulo)+representante_modular(b, modulo), modulo);

}


long long int producto_modular (long long int a, long long int b, long long int modulo) {

 return representante_modular(representante_modular(a, modulo)*representante_modular(b, modulo), modulo);

}
```

```
long long int mcd (long long int a, long long int b) {
        long long int r=1, aux;
        if (a>b) {aux=a; a=b; b=aux;}
        if (b==0) return a;
        while (r!=0) {
                r=a%b;
                a=b;
                b=r;
        }
        return a;
}

void combinacionlinealmcd() {
   long long int a,b,r,c;
   //matriz generada
   long long int matgen[2][2]={{1,0},{0,1}};
   long long int aux0,aux1;
   printf("Halla la combinacion lineal del MCD de A y B:\n\nIngrese A(espacio)B: ");
   scanf("\n%lld %lld",&a,&b);
   printf("\n[%lld,%lld].",a,b);
   while (b!=0) {
      c=a/b;
      r=a%b;
      a=b;
      b=r;
      //iteracion
      aux0=matgen[1][0]*(-c)+matgen[0][0];
      aux1=matgen[1][1]*(-c)+matgen[0][1];
      matgen[0][0]=matgen[1][0];
```

```
        matgen[0][1]=matgen[1][1];
        matgen[1][0]=aux0;
        matgen[1][1]=aux1;
    }

    printf("[%lld,%lld]=%lld",matgen[0][0],matgen[0][1],a);
}



long long int inverso_modular (long long int numero, long long int modulo) {
    long long int a=numero,b=modulo,r,c;
    //matriz generada modificada
    long long int matgen[2]={1,0};
    long long int aux;
    while (b!=0) {
        c=a/b;
        r=a-b*c;
        a=b;
        b=r;
        //iteracion
        aux=matgen[1]*(-c)+matgen[0];
        matgen[0]=matgen[1];
        matgen[1]=aux;
    }
    if (a!=1) return 0; else return representante_modular(matgen[0],modulo);
}


long long int exponenciacion_binaria_modular (long long int base, long long int
exponente, long long int modulo) {
    long long int aux;
    if (exponente==1) return representante_modular(base, modulo);
```

```
   if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int factorial_modular (long long int numero, long long int modulo) {

   if ((numero==1)||(numero==0)) return 1;

   return producto_modular(numero, factorial_modular(numero-1,modulo),modulo);

}


int jacobi(long long int m, long long int n) {

      if ((m==1)||(n==1)) {printf("\n(%lld/%lld)=1",m,n);return 1;}

      if ((m>n)||(m<0)) {


        printf("\n(%lld/%lld)=(%lld/%lld)",m,n,representante_modular(m,n),n);

                return jacobi(representante_modular(m,n),n);}

      if ((mcd(m,n))!=1) {printf("\nNo son coprimos el mcd es
%lld",mcd(m,n));return 0;}

      if (m==(n-1)) {if (representante_modular(n,4)==1)
{printf("\n(%lld/%lld)=1",m,n);return 1;} else {printf("\n(%lld/%lld)=-
1",m,n);return -1;}}

      if (m==2) {if
((representante_modular(n,8)==1)||(representante_modular(n,8)==7))
{printf("\n(%lld/%lld)=1",m,n);return 1;} else {printf("\n(%lld/%lld)=-
1",m,n);return -1;}}

      if (representante_modular(m,2)==0)
{printf("\n(%lld/%lld)=(2/%lld)(%lld/%lld)",m,n,n,m/2,n);return
jacobi(2,n)*jacobi(m/2,n);}

      if (representante_modular(n,2)==0)
{printf("\n(%lld/%lld)=(%lld/2)(%lld/%lld)",m,n,m,m,n/2);return
jacobi(m,2)*jacobi(m,n/2);}

      if (representante_modular((m-1)/2,2)==0)
{printf("\n(%lld/%lld)=1*(%lld/%lld)",m,n,n,m);return jacobi(n,m);} else {

      if (representante_modular((n-1)/2,2)==0)
{printf("\n(%lld/%lld)=1*(%lld/%lld)",m,n,n,m);return jacobi(n,m);} else
{printf("\n(%lld/%lld)=(-1)*(%lld/%lld)",m,n,n,m);return (-1)*jacobi(n,m);}
```

```
    }
}
int main() {
        int op;
        long long int a, b, m;
        printf("Aritmetica Modular\n\nCalcular...\n1. Suma \n2. Producto\n3. Inverso\n4. Potencia\n5. Factorial\n6. Simbolo de Jacobi\n7. MCD\n8. Combinacion lineal MCD\n\nOpcion: ");
        scanf("%d",&op);
        system("CLS");
        switch(op) {
                case 1: {
                        printf("Suma\n\nIngrese Sumando 1: ");
                        scanf("%lld",&a);
                        printf("Ingrese Sumando 2: ");
                        scanf("%lld",&b);
                        printf("Ingrese Modulo: ");
                        scanf("%lld",&m);
                        printf("\n%lld + %lld = %lld (mod %lld)",a,b,suma_modular(a,b,m),m);
                }
                break;
                case 2: {
                        printf("Producto\n\nIngrese Factor 1: ");
                        scanf("%lld",&a);
                        printf("Ingrese Factor 2: ");
                        scanf("%lld",&b);
                        printf("Ingrese Modulo: ");
                        scanf("%lld",&m);
                        printf("\n%lld* %lld = %lld (mod %lld)",a,b,producto_modular(a,b,m),m);
                }
```

```
        break;
        case 3: {
                printf("Inverso\n\nInverso de: ");
                scanf("%lld",&a);
                printf("Modulo: ");
                scanf("%lld",&m);
                b=inverso_modular(a,m);
                if (b==0) printf("\nEl inverso multiplicativo de %lld mod %lld ~ NO
EXISTE",a,m); else printf("\nInverso de %lld = %lld (mod %lld)",a,b,m);
        }
        break;
        case 4: {
                printf("Potencia\n\nIngrese Base: ");
      scanf("%lld",&a);
                printf("Ingrese Exponente: ");
                scanf("%lld",&b);
                printf("Ingrese Modulo: ");
                scanf("%lld",&m);
                printf("\n%lld ^ %lld = %lld (mod
%lld)",a,b,exponenciacion_binaria_modular(a,b,m),m);
        }
        break;
        case 5: {
      printf("Factorial\n\nIngrese numero (de a lo sumo 4 cifras): ");
                scanf("%lld",&a);
      printf("Ingrese Modulo: ");
                scanf("%lld",&m);
                printf("\n%lld!=%lld (mod %lld)",a,factorial_modular(a,m),m);
        }
        break;
   case 6: {
```

```
        printf("Simbolo de Jacobi (m/n)\n\nIngrese m: ");
        scanf("%lld",&a);
        printf("Ingrese n: ");
        scanf("%lld",&b);
        if (((b%2)==0)||(b<0)) printf("\n\nEl numero n debe ser un natural impar");
else printf("\n\nResultado Final: (%lld/%lld)=%d", a, b, jacobi(a,b));    }
    break;
    case 7: {
        printf("Maximo comun divisor\n\nIngrese a: ");
        scanf("%lld",&a);
        printf("Ingrese b: ");
        scanf("%lld",&b);
        printf("\nmcd(%lld,%lld)=%lld",a,b,mcd(a,b));
    }
    break;
    case 8: combinacionlinealmcd();
    break;
          }
          printf("\n\n<<Presione una tecla para volver al menu...>>");
          getch();
          system("CLS");
          main();
}
```

Cifrados Modernos

Dado los avances del criptoanálisis, se buscaron más formas de cifrar mensajes con artilugios matemáticos más sofisticados.

La clave de todo cifrado exitoso siempre es la matemática. En los cifrados antiguos algunos parecen no tenerla (al menos visible), lo cierto es que sin un razonamiento matemático es imposible cifrar o descifrar.

Entre los cifrados que se analizarán, veremos a continuación uno que opera con matrices.

Cifrado de Lester S. Hill

Según Gomez, 2010: Fue creado en 1929 por el matemático estadounidense Lester S. Hill ideado, patentado y puesto a la venta sin éxito por él, combina aritmética modular y álgebra lineal.

Este cifrado tiene una importante debilidad: en caso de que el receptor disponga de un fragmento pequeño del texto sin cifrar, es factible de descifrar el mensaje en su totalidad.

La clave de este cifrado es una matriz entera de NxN, generalmente N es 2, 3 o 4, cuyo determinante debe ser ± 1. Usaremos a modo de ejemplo una matriz entera de 3x3 con determinante 1.

Hallaremos una particular, para poder aplicar el cifrado:

como $2.3 - 5.1 = 1 \Rightarrow \begin{vmatrix} 2 & 1 \\ 5 & 3 \end{vmatrix} = 1 \Rightarrow \begin{vmatrix} 0 & 0 & 1 \\ 2 & 1 & 4 \\ 5 & 3 & 7 \end{vmatrix} = 1$

como $5.2 - 3.3 = 1 \Rightarrow \begin{vmatrix} 5 & 3 \\ 3 & 2 \end{vmatrix} = 1 \Rightarrow \begin{vmatrix} 5 & 2 & 3 \\ 0 & 1 & 0 \\ 3 & 4 & 2 \end{vmatrix} = 1$

$\begin{pmatrix} 0 & 0 & 1 \\ 2 & 1 & 4 \\ 5 & 3 & 7 \end{pmatrix} . \begin{pmatrix} 5 & 2 & 3 \\ 0 & 1 & 0 \\ 3 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 3 & 4 & 2 \\ 22 & 23 & 14 \\ 46 & 39 & 29 \end{pmatrix}$, y su determinante será 1 por ser producto de matrices de determinante 1.

Tenemos a nuestra matriz entera: $A = \begin{pmatrix} 3 & 4 & 2 \\ 22 & 21 & 14 \\ 46 & 41 & 29 \end{pmatrix}$ (será la clave de nuestro cifrado)

Como tomamos una matriz de 3x3 vamos a agrupar de a 3 a las letras del mensaje a cifrar.

Supongamos que queremos cifrar: FRUTILLA

FRU TIL LAX (completamos los espacios con X para tener bloques de 3 letras)

Aplicamos la biyección ***a,*** a cada letra del mensaje (vamos a considerar a la ñ en la biyección):

5 18 21 20 8 11 11 0 24

Y consideraremos estos arreglos como vectores columna de 3x1:

$$\begin{pmatrix}5\\18\\21\end{pmatrix},\begin{pmatrix}20\\8\\11\end{pmatrix},\begin{pmatrix}11\\0\\24\end{pmatrix}$$

Ahora se multiplica la matriz clave por cada matriz del mensaje, y nos dará por cada una, una matriz cifrada.

$$\begin{pmatrix}3 & 4 & 2\\22 & 21 & 14\\46 & 41 & 29\end{pmatrix}.\begin{pmatrix}5\\18\\21\end{pmatrix}=\begin{pmatrix}129\\782\\1577\end{pmatrix}$$

$$\begin{pmatrix}3 & 4 & 2\\22 & 21 & 14\\46 & 41 & 29\end{pmatrix}.\begin{pmatrix}20\\8\\11\end{pmatrix}=\begin{pmatrix}114\\762\\1567\end{pmatrix}$$

$$\begin{pmatrix}3 & 4 & 2\\22 & 21 & 14\\46 & 41 & 29\end{pmatrix}.\begin{pmatrix}11\\0\\24\end{pmatrix}=\begin{pmatrix}81\\578\\1202\end{pmatrix}$$

Luego, calculamos a los elementos de las matrices en módulo 27 porque hemos considerado la ñ, tomando el resto de la división por 27.

$$\begin{pmatrix}129\\782\\1577\end{pmatrix}\rightarrow\begin{pmatrix}21\\26\\11\end{pmatrix},\begin{pmatrix}114\\762\\1567\end{pmatrix}\rightarrow\begin{pmatrix}6\\6\\1\end{pmatrix},\begin{pmatrix}81\\578\\1202\end{pmatrix}\rightarrow\begin{pmatrix}0\\11\\14\end{pmatrix}$$

Por lo tanto, podemos reescribirlo de la siguiente forma:

21 26 11 6 6 1 0 11 14, luego se aplica la biyección $\mathbf{a^{-1}}$.

El mensaje quedará cifrado finalmente como: UZLGGBALÑ

Para descifrarlo, calculamos la matriz inversa de A.

¿Cómo sabemos si siempre existe?

Sabemos que $A^{-1}=\frac{1}{|A|}(A^*)^T$, siendo que se pide ||A||=1, al ser el determinante distinto de cero la matriz A es inversible y además:

$A^{-1}=(A^*)^T\ v\ A^{-1}=-(A^*)^T$, pero no sabemos si será entera.

A^* es la matriz adjunta, el determinante de una matriz entera será entero por Ley Interna en $\mathbb{Z}$, por lo cual como cada elemento de la matriz es su adjunto, A^* será entera.

Luego la transpuesta de una entera también será entera pues contiene a los mismos elementos.

Por lo tanto, la inversa existirá y será entera.

Sabiendo esto estamos en condiciones de continuar con el descifrado. Como en nuestra matriz tomada |A|=1, entonces $A^{-1} = (A^*)^T$

Como $A = \begin{pmatrix} 3 & 4 & 2 \\ 22 & 21 & 14 \\ 46 & 41 & 29 \end{pmatrix}$, tenemos que

$$A^* = \begin{pmatrix} \begin{vmatrix} 21 & 14 \\ 41 & 29 \end{vmatrix} & -\begin{vmatrix} 22 & 14 \\ 46 & 29 \end{vmatrix} & \begin{vmatrix} 22 & 21 \\ 46 & 41 \end{vmatrix} \\ -\begin{vmatrix} 4 & 2 \\ 41 & 29 \end{vmatrix} & \begin{vmatrix} 3 & 2 \\ 46 & 29 \end{vmatrix} & -\begin{vmatrix} 3 & 4 \\ 46 & 41 \end{vmatrix} \\ \begin{vmatrix} 4 & 2 \\ 21 & 14 \end{vmatrix} & -\begin{vmatrix} 3 & 2 \\ 22 & 14 \end{vmatrix} & \begin{vmatrix} 3 & 4 \\ 22 & 21 \end{vmatrix} \end{pmatrix} = \begin{pmatrix} 35 & 6 & -64 \\ -34 & -5 & 61 \\ 14 & 2 & -25 \end{pmatrix}$$

$$(A^*)^T = \begin{pmatrix} 35 & -34 & 14 \\ 6 & -5 & 2 \\ -64 & 61 & -25 \end{pmatrix}$$

$$\therefore\ A^{-1} = \begin{pmatrix} 35 & -34 & 14 \\ 6 & -5 & 2 \\ -64 & 61 & -25 \end{pmatrix}$$

Ahora operaremos las matrices cifradas con la inversa de nuestra matriz original.

Las cuales son: $\begin{pmatrix} 21 \\ 26 \\ 11 \end{pmatrix}, \begin{pmatrix} 6 \\ 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 11 \\ 14 \end{pmatrix}$

$$\begin{pmatrix} 35 & -34 & 14 \\ 6 & -5 & 2 \\ -64 & 61 & -25 \end{pmatrix}\begin{pmatrix} 21 \\ 26 \\ 11 \end{pmatrix} = \begin{pmatrix} 5 \\ 18 \\ -33 \end{pmatrix}$$

$$\begin{pmatrix} 35 & -34 & 14 \\ 6 & -5 & 2 \\ -64 & 61 & -25 \end{pmatrix}\begin{pmatrix} 6 \\ 6 \\ 1 \end{pmatrix} = \begin{pmatrix} 20 \\ 8 \\ -43 \end{pmatrix}$$

$$\begin{pmatrix} 35 & -34 & 14 \\ 6 & -5 & 2 \\ -64 & 61 & -25 \end{pmatrix}\begin{pmatrix} 0 \\ 11 \\ 14 \end{pmatrix} = \begin{pmatrix} -178 \\ -27 \\ 321 \end{pmatrix}$$

Luego, calculo a los elementos de las matrices en módulo 27

$$\begin{pmatrix} 5 \\ 18 \\ -33 \end{pmatrix} \to \begin{pmatrix} 5 \\ 18 \\ 21 \end{pmatrix}, \begin{pmatrix} 20 \\ 8 \\ -43 \end{pmatrix} \to \begin{pmatrix} 20 \\ 8 \\ 11 \end{pmatrix}, \begin{pmatrix} -178 \\ -27 \\ 321 \end{pmatrix} \to \begin{pmatrix} 11 \\ 0 \\ 24 \end{pmatrix}$$

Así obtenemos: 5 18 21 20 8 11 11 0 24, que equivale a: FRUTILLAX, nuestro mensaje original.

Teorema Criptográfico de Kaisa Nyberg

Según Penazzi, 2010: El teorema de Kaisa Nyberg, profesora de criptografía de la Universidad Tecnológica de Helsinki, nos dice lo siguiente, en forma resumida:

"*Si se toma la substitución que manda el 0 al 0, y para un A no nulo se lo manda a 1/A módulo p, con p primo, entonces esa substitución es "altamente no lineal"*"

Consideraremos a p=29 y se considera la siguiente tabla de inversos módulo p, donde en la columna izquierda tendremos a los números que le querremos calcular su inverso.

Una sustitución no lineal entonces es la siguiente, aplicando el teorema:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	15	10	22	6	5	25	11	13	3	8	17	9	27

15	16	17	18	19	20	21	22	23	24	25	26	27	28
2	20	12	21	26	16	18	4	24	23	7	19	14	28

Nos explica Penazzi, que esta sustitución tiene un problema, y es que corresponde al 0 con el 0, al 1 con el 1 y al 28 con el 28. Por lo tanto, propone sumar 2 en módulo 29. También nos dice, que la suma de una sustitución lineal a una no lineal, es una no lineal y que al hacer esto evita que haya puntos fijos.

Sin embargo, al ser una sustitución monoalfabética monográfica, es susceptible a un *análisis de frecuencias*. Y también será vulnerable a un *ataque por fuerza bruta* (luego veremos en que consiste).

Entonces, ¿Qué utilidad tiene semejante teorema si es vulnerable? La idea no es usarlo solo, sino acompañado de más cosas. En el próximo cifrado veremos esto mismo.

Cifrado AES

AES es la abreviación de Advanced Encryption Standard: Estándar de Encriptamiento Avanzado.

Según Penazzi (2010): "*A mediados de los 90, se observó que DES ya no era un algoritmo seguro. Luego de una competencia internacional para elegir al sucesor, que ganaron dos profesores de Bélgica, resulto elegido AES*"

Hablaremos resumidamente de su estructura sin profundizar mucho.

Según Penazzi, 2010: Para incrementar la seguridad en los cifrados modernos se habla de añadir rondas.

La idea en AES es: a un texto sin cifrar, aplicarle Vigenère + sustitución no lineal (y aquí es donde entra el teorema de Nyberg) + transformación lineal para evitar puntos fijos (como sugirió Penazzi de sumar 2 en módulo 29) + Hill, y a todo este proceso lo llamaremos una ronda. La idea es que Hill evite que no se pueda realizar el análisis de frecuencias y la sustitución prevendrá el ataque a través de ecuaciones lineales. En AES se usan al menos 10 de estas rondas.

En lugar de usar bloques de letras se usarán bloques de al menos 128 bits, entre otros detalles técnicos.

AES es uno de los cifrados criptográficos más utilizados actualmente, se encuentra presente, por ejemplo, en la mayoría de nuestras comunicaciones.

Criptografía Asimétrica

Hasta ahora los cifrados que hemos analizados son todos de criptografía simétrica, es decir, que la clave que se usa para cifrar y descifrar es la misma. Excepto el cifrado AES, el resto son vulnerables por técnicas de criptoanálisis que fuimos mencionando. Este es un problema y el hecho de tener que intercambiar la clave con alguien, suponiendo que no podamos hacerlo presencialmente o tengamos alguna ya acordada, también es un problema.

Sin embargo, existe una solución: la criptografía asimétrica.

El intercambio de claves

Esta idea se introdujo en 1976, por dos criptógrafos Whitfield Diffie y Martin Hellman, lograr crear una forma en que puedan acordar una clave sin tener que enviarla.

Imaginemos que Patricia e Inés, desean acordar una clave k que podrían utilizarla luego para lo que necesiten.

Eligen un número primo p y una raíz primitiva g módulo p, esta información será considerada **pública**, no hay ningún problema que alguien más lo sepa. Luego

Patricia elige un número mayor que $1 \leq a \leq p-1$ que es una clave **privada** (por lo tanto, nadie puede saberla, sólo ella), Inés por su parte hace lo mismo y toma una clave b (también privada) con la misma condición.

Veamos ahora el proceso que ocurre:

Patricia: Calcula $u \equiv g^a\ (p)$

Inés: Calcula $v \equiv g^b\ (p)$

Se envían entre ellas u y v, y luego

Patricia: Calcula $k_p \equiv v^a\ (p)$

Inés: Calcula $k_i \equiv u^b\ (p)$

Sin embargo, $k_p \equiv v^a \equiv \left(g^b\right)^a \equiv (g^a)^b \equiv u^b \equiv k_i\ (p)$ por lo que termina siendo que sus claves k son la misma, luego ya han acordado una clave k común, sin tener que enviarla.

Supongamos que alguien esta escuchando su intercambio de claves, podría conocer g, p, y también como máximo a u y v.

Si tiene g y tiene u, esta persona intentaría conocer a. O si tiene g y v a b. Luego sería cuestión de hacer u^a en módulo p y tener la clave k.

Pero hay un inconveniente, el hecho de querer conocer $a|\ g^a = u\ (p)$, implica el cálculo del logaritmo discreto $\log_g u$ y aún no se conoce un algoritmo eficiente para lograr esto. Por lo cual la seguridad de este sistema está en la complejidad de la resolución de dicho problema.

RSA

El criptosistema RSA, creado en 1978, es uno de los sistemas más famosos de clave pública, su nombre lo debe a que es una sigla formada por los apellidos de quienes los crearon Rivest, Shamir y Adelman.

En esta ocasión Patricia e Inés intentan enviarse un mensaje cifrado por RSA.

Patricia: Elije dos números primos p y q (de unas 200 cifras o más cada uno), los mantiene en secreto, calcula n=p.q, luego calcula $\varphi(n) = (p-1)(q-1)$ y toma a un número e coprimo con $\varphi(n)$ entre 1 y $\varphi(n)$, y además calcula d, que será el inverso de e módulo $\varphi(n)$ (el cual existirá pues $(e{:}\varphi(n)) = 1$). Luego le envía a Inés los números n y e que son públicos, el resto son privados.

Inés: Convierte a su mensaje en un número natural m menor que n. Posteriormente calcula $M = m^e\ (n)$, y le envía M a Patricia.

Patricia: Calcula M^d módulo n, y obtiene así el mensaje descifrado.

¿Por qué funciona?

$$M^d \equiv (m^e)^d \equiv m^{ed}\ (n)$$

Como $e.d \equiv 1\ \big(\varphi(n)\big) \Rightarrow ed = k\varphi(n) + 1$

Luego, $m^{ed} \equiv m^{k\varphi(n)+1} \equiv \left(m^{\varphi(n)}\right)^k.m \equiv 1.m\ \ (n)$, (pues $m^{\varphi(n)} \equiv 1\ (n)$ por teorema de Fermat-Euler)

Por lo tanto, $M^d = m$ módulo n. (si (m:n)=1)

Sin embargo, para aplicar el teorema de Fermat-Euler destaquemos que (m:n)=1, y si Inés no conoce los primos de n, podría ocurrir que un primo divida el mensaje. Analicemos esa situación:

Si $(m{:}n) = d\ (con\ d \neq 1)\ \Rightarrow d = p\ \vee d = q\ \vee d = n$, d no puede ser n porque se pidió que m fuera menor que n, entonces d podría ser p o q, en ambos casos la demostración es análoga, así que supongamos que p|m (m=ps con s natural).

Como $M^d \equiv m^{ed}\ (n)\ \Rightarrow\ M^d \equiv m^{ed}\ (p)\ \wedge\ M^d \equiv m^{ed}\ (q)$

$$M^d \equiv m^{ed}\ \equiv (ps)^{ed} \equiv p^{ed}s^{ed} \equiv 0\ \equiv m\ (p)$$

$M^d \equiv m^{ed}\ (n) \equiv m^{k\varphi(n)+1} \equiv \left(m^{\varphi(n)}\right)^k.m \equiv m\ (q)$ (ya que (m:q)=1, pues suponemos que solo p divide a m, podemos aplicar Fermat-Euler).

Dado que $M^d \equiv m\ (p) \wedge M^d \equiv m\ (q)$, por teorema chino del resto, tenemos finalmente que $M^d \equiv m\ (n)$

Por lo cual, vemos que el cifrado RSA funciona correctamente.

Veamos un ejemplo para entender mejor el funcionamiento:

Patricia (en lugar de tomar primos de más de 200 cifras) eligió los primos p=23117 y q=7741. Calcula n=pq=178948697, luego $\varphi(n) = 23116.7740 = 178917840$, toma e entre 1 y $\varphi(n)$, coprimo con $\varphi(n)$, e= 18469, luego halla su inverso d en módulo $\varphi(n)$, d=138792349.

Inés que recibió los números n y e, quiere cifrar la palabra LIMON, antes de empezar observa que $27^5 \leq n \leq 27^6$, por lo cual deberá dividir a su mensaje en bloques de 5 letras, en este caso LIMON tiene 5 letras, así que cifrará solo ese bloque, si tuviera una cantidad que no sea un múltiplo de 5, completaría con letras que tengan bajo porcentaje de aparición en el idioma para poder dividir en bloques de a 5. Luego puede aplicar la función ***a*** en cada letra para llevarlo a números, 11 8 12 15 13, posteriormente considera que está en base 27, así que lo transforma a base 10: $11.27^4 + 8.27^3 + 12.27^2 + 15.27 + 13 = 6012481$

Luego, su mensaje m=6012481.

Calcula entonces: $m^e\ (mod\ n),\ \ 6012481^{18469} \equiv 80574990\ (mod\ 178948697)$,

Su mensaje cifrado 80574990, considera que esta en bloques de 6 letras su texto, y operando un poco llega a que en base 27 es: 5 16 16 17 2 24, lo cual podemos comprobar haciendo $5.27^5 + 16.27^4 + 16.27^3 + 17.27^2 + 2.27 + 24 = 80574990$, luego aplica la inversa de ***a***. Llegando al mensaje cifrado FPPQCX, y lo envía.

Patricia al saber que $27^5 \leq n \leq 27^6$, deduce que para descifrar deberá tomar bloques de a 6, en este caso son 6 letras, así que será un bloque. Aplica la biyección ***a***, y llega a que M=80574990, utilizando su clave privada, realiza $M^d\ (mod\ n), 80574990^{138792349} \equiv 6012481\ (mod\ 178948697)$

Luego lo convierte a base 27, llegando a 11 8 12 15 13, y aplica la inversa de la biyección llegando al mensaje descifrado LIMON.

Los cálculos fueron realizados con la ayuda de los programas "Aritmetica Modular" y "Cifrado RSA" para la generación de claves.

<u>Aclaración</u>: El siguiente programa opera sin la letra ñ, los números primos con los que puede operar son pequeños, asi que no es seguro, para ser seguro se deberían utilizar primos de más de 200 cifras, por lo tanto, lo hecho es a modo de muestra del funcionamiento del criptosistema.

Programa: Cifrado RSA

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

      if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int modulo)
{

 return                    representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);
```

```
}


long long int exponenciacion_binaria_modular (long long int base, long long int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base, exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return producto_modular(base,exponenciacion_binaria_modular(base, exponente-1,modulo),modulo);

}


long long int inverso_modular (long long int numero, long long int modulo) {

    long long int a=numero,b=modulo,r,c;

    //matriz generada modificada

    long long int matgen[2]={1,0};

    long long int aux;

    while (b!=0) {

        c=a/b;

        r=a-b*c;

        a=b;

        b=r;

        //iteracion

        aux=matgen[1]*(-c)+matgen[0];

        matgen[0]=matgen[1];

        matgen[1]=aux;

    }

    if (a!=1) return 0; else return representante_modular(matgen[0],modulo);

}


long long int mcd (long long int a, long long int b) {

          long long int r=1, aux;
```

```
        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


int ascii_num (char c) {

        int num=c;

        //asignamos a mayusculas y minusculas, los numeros correspondientes al abecedario siendo a=0, b=1, c=2, ..., z=25 sin contar la ª pues no la reconoce

        if ((num>=65) && (num<=122)) return num%32-1; /* mod 32 pues hay un espacio de 32 caracteres entre A y a y entre cualquier par de letras mayus y minus */

        return -1;

}


char num_ascii (int n) {

        n+=65; /* asigno solo las mayus */

        return n;

}


long long int potencia_ent (long base, long exp) {

        if (exp==0) return 1;

    if ((exp%2)==0) {long long int aux=potencia_ent(base, exp/2); return aux*aux;} else return base*potencia_ent(base,exp-1);

}


void GeneraClavesRSA() {
```

```
    long long int p, q, n, fi, e, d;

    printf("Genera claves para RSA (para primos de pocas cifras)\nNOTA: La suma de la cantidad de cifras de ambos no debe superar a 9\n\nIngrese un primo p: ");

    scanf("%lld",&p);

    printf("Ingrese un primo q: ");

    scanf("%lld",&q);

    printf("\nCalculando...");

    n=p*q;

    //siendo f, fi(n)

    fi=(p-1)*(q-1);

    //busco e pseudoaleatorio coprimo con fi

    e=fi;

    while (mcd(e,fi)!=1) {

        e=2+rand()%(fi-2);

    }

    //busco d inverso de e en modulo fi

    d=inverso_modular(e,fi);

    printf("Listo.\nClave Publica: N=%lld, E=%lld (Datos para el cifrado)",n,e);

    printf("\nInformacion Privada: P=%lld, Q=%lld, Fi(n)=%lld, D=%lld\n\nGuarde la clave publica y su clave privada N, D (para el descifrado).",p,q,fi,d);

}


int RSA (char cifrado) {

//cifrado=c para cifrar, cifrado=d para descifrar

if ((cifrado!='c')&&(cifrado!='d')) {printf("Ingrese una opcion valida."); return 1;

}


int a, pos=0, longbloqueent, longbloquesal, longtexto;

int base=26;

long long int aux, n, e;

int cociente;
```

```
char texto[1002];
int textonum[1002];
int textonumcif[1002];
printf("Escriba el texto:\n");
fflush(stdin);
fgets(texto, 1003, stdin);
fflush(stdin);
printf("Ingrese clave n: ");
scanf("%lld",&n);
if (cifrado=='c') printf("Ingrese clave e: "); else printf("Ingrede clave d: ");
scanf("%lld",&e);

//detecta la longitud del bloque de entrada
longbloqueent=1;
while (potencia_ent(base, longbloqueent)<n) longbloqueent++;
if (cifrado=='c') longbloqueent-=1;

//toma los caracteres aceptados y los convierte a numeros, guardadolo en textonum
for (a=0; a<=(strlen(texto)-2); a++) {
   if (ascii_num(texto[a])>=0) {
        textonum[pos]=ascii_num(texto[a]);
       pos++;
   }
}
textonum[pos]='\0';
longtexto=pos;
//completando el ultimo bloque
for (a=pos; (a%longbloqueent)!=0; a++) {
        textonum[a]=23;
}
```

```
//ha cambiado la longitud del texto
longtexto=a;
//textonum es el mensaje con la biyeccion a aplicada, es decir, esta en numeros


//divido en bloques de longbloqueent
if (cifrado=='c') printf("Texto cifrado: "); else printf("Texto descifrado: ");
for (cociente=0; cociente<(longtexto/longbloqueent); cociente++) {
	aux=0;
	//tomo al numero en base "base"
	for (a=(cociente*longbloqueent); a<((cociente+1)*longbloqueent); a++) {
		//convierto al bloque en base 10
				aux+=textonum[a]*potencia_ent(base,longbloqueent-1-
a%longbloqueent);
	}
	//aux es el numero en base 10
	//deberia aplicarse una funcion a aux
	aux=exponenciacion_binaria_modular(aux,e,n);


	//y luego, si cifro longbloquesal=longbloque+1 sino longbloquesal=longbloque-
1
	if (cifrado=='c') longbloquesal=longbloqueent+1; else
longbloquesal=longbloqueent-1;
	for (a=0;a<longbloquesal;a++) {  textonumcif[longbloquesal-1-
a]=aux%base;
		aux/=base;
	}


	//imprimo al bloque de un numero a un bloque de longbloquesal letras
	for (a=0; a<longbloquesal; a++) {
	printf("%c",num_ascii(textonumcif[a]));
	}
}
```

```
 return 0;
}


int main() {
	int opc;
	printf("Cifrado RSA\n\nElija:\n1. Generar claves\n2. Cifrar\n3. Descifrar\n>");
	fflush(stdin);
	scanf("%d",&opc);
	fflush(stdin);
	printf("\n\n");
	switch (opc) {
		case 1: GeneraClavesRSA();
		break;
		case 2: RSA('c');
		break;
		case 3: RSA('d');
		break;
	}
	printf("\n\nPulse una tecla para volver al menu...");
	getch();
	printf("\n\n\n");
	main();
}
```

Criptoanálisis (Parte II)

Ecuaciones Lineales y el ataque por fuerza bruta

Como habíamos anunciado previamente la debilidad del cifrado de Hill, está en que si se tiene una parte del mensaje descifrado y el correspondiente cifrado. Se puede revelar la clave. ¿Cómo?

Solo basta con hacer un sistema de ecuaciones lineales y sabiendo la matriz sin cifrar y la cifrada, con un conjunto de éstas se puede revelar la clave. O al menos, tener una matriz 'aproximada' porque en Hill se aplica aritmética modular y por lo cual la matriz cifrada no necesariamente es exactamente igual a su equivalencia en módulo 27. Sin embargo, mientras más información tengamos y teniendo esta matriz, es cuestión de probar con todas las posibilidades con ayuda de una computadora que haga los cálculos. A este método de agotar todas las posibilidades se lo conoce como **ataque por fuerza bruta**.

El **ataque por fuerza bruta** también es válido para la mayoría de los cifrados clásicos, y es una herramienta potente del criptoanálisis, aunque su única desventaja es el tiempo.

De hecho, estos ataques son los que se usan hoy en día con supercomputadoras para quebrar algunos cifrados en donde la clave no es la adecuada y es insegura. Es por esto que en el correo electrónico por ejemplo se aconseja usar mayúsculas, minúsculas, números y símbolos, porque de esta manera un ataque a fuerza bruta no llegara jamás a obtener la clave, por el inconveniente del tiempo.

Algunos ataques por fuerza bruta tardarían millones de años en cumplir su objetivo, pero con la aparición de procesadores cada vez más potentes van reduciendo un poco su tiempo, aunque no demasiado y por lo tanto en algunos sistemas criptográficos un ataque de este estilo es improbable que logre su objetivo en un tiempo prudente, y como antes mencionamos toda información es valiosa en un tiempo, fuera de ese tiempo ya pierde valor.

El criptoanálisis de Hill nos da otra importante lección:

Todo buen cifrado no debe ser lineal, pues el cifrado de Hill es un cifrado monoalfabético poligrámico (reemplazando grupos de grafos por otros), lo cual lo hace lineal y vulnerable. De aquí la utilidad del teorema de Kaisa Nyberg.

Tests de primalidad

El cifrado RSA, por ejemplo, basa su seguridad en el problema de encontrar un algoritmo eficiente que permita la factorización de números de gran tamaño,

hasta ahora no se ha encontrado. Es decir, no se ha encontrado un algoritmo eficiente que permita resolver esto en tiempo polinómico, de ahí la importancia del problema abierto P versus NP.

Sería útil tener de todos modos, una forma de hallar a números primos, y en eso se basan estos tests que vamos a trabajar.

Tenemos los **tests deterministas**: ellos nos dirán si un número es primo o no, la desventaja que tienen estos es la lentitud debida a la cantidad de operaciones a realizar para determinar la primalidad de un número.

Entre estos tests deterministas, se encuentra el teorema de Wilson, que nos dice que $n\ primo \Leftrightarrow\ (n-1)!\ \equiv\ -1\ (mod\ n)$, por lo tanto, podríamos calcular el factorial en módulo n de n-1 y ver si es congruente a -1 en modulo n, si lo cumple es primo. Sin embargo, no resulta algo muy eficiente a la hora de determinar si un número es primo o no.

También tenemos a la Criba de Eratóstenes para hallar a todos los números primos menores que a un número n, la cual resulta más eficiente que la aplicación del teorema de Wilson. Antes de explicar el funcionamiento y ver como podría aplicarse, vamos a demostrar el siguiente teorema:

Teorema: $n\ es\ primo \Leftrightarrow\ \nexists a \in [2;\sqrt{n}]|\ \ a|n$

Probaremos primero la vuelta, por el absurdo:

Supongamos que $\nexists a \in [2;\sqrt{n}]|\ \ a|n\ \wedge\ \ n\ compuesto$

$$n\ compuesto \Rightarrow\ \exists b,c \in [2;n-1]|\ bc = n$$

$$\Rightarrow b \notin [2;\sqrt{n}]\ por\ hipótesis \Rightarrow b > \sqrt{n}$$

Análogamente, $c > \sqrt{n}$

Luego $bc > \sqrt{n}\sqrt{n} \Rightarrow n > n$ Absurdo!

Por lo tanto, $\nexists a \in [2;\sqrt{n}]|\ \ a|n \Rightarrow n\ es\ primo$

Probemos la ida:

$$n\ primo \Rightarrow\ \nexists a \in [2;n-1]|\ \ a|n$$

Como $[2;\sqrt{n}] \subset [2;n-1] \Rightarrow \nexists a \in [2;\sqrt{n}]|\ \ a|n$

Por lo tanto, basta con probar con los números menores a la raíz cuadrada de un numero n que queramos ver si es primo para saber si lo es o no.

Sin embargo, para ahorrarnos este proceso la Criba de Eratóstenes nos dice que listemos primero a los números menores que n, y vayamos descartando en nuestro análisis a los que son múltiplos de los primos encontrados antes de $\sqrt{n}$, si ninguno de los primos hallados divide a n, entonces por el teorema anterior garantizamos que n es primo, sino es claramente compuesto.

La programación de este algoritmo, requeriría de hacer un vector de n elementos en donde sus componentes sean 0 y 1, e ir recorriéndolo de la manera recién especificada. Sin embargo, por la dificultad que tiene el manejo de la memoria en el lenguaje C, he optado por realizar un programa que solamente pruebe con los números menores a $\sqrt{n}$ sean primos o no, lo cual hará más lento el proceso.

Se podrían mencionar y hablar de varios algoritmos determiníticos, pero la mayoría se aplican a casos particulares, o tienen ciertas consideraciones que los hacen poco útiles, sobre todo por el tiempo que demandan en determinar la primalidad. Por lo tanto, la idea va a ser analizar a continuación, otro tipo de tests que son de gran utilidad.

Tests probabilísticos de primalidad

Estos tipos de tests a diferencias de los anteriores, no determinan la primalidad de un número, sino que nos dice cuán probable es que lo sea. Es decir, los tests anteriores eran de condiciones necesarias y suficientes.

En estos tenemos que, si p es primo entonces cumple con algo. Ahora si bien es necesario que sea p primo para que cumpla la condición, no es suficiente. Por lo tanto, no podemos afirmar que los números que pasen el test sean primos, y depende el tipo de test, es más o menos fiable los resultados que podamos obtener.

Test de Fermat

Tenemos a un número n natural que queremos analizar su primalidad. Tomamos aleatoriamente a un natural a comprendido entre 1 y n-1, luego calculamos $x = a^{n-1}\ (mod\ n)$. Por el pequeño teorema de Fermat tenemos que si n es primo entonces $x \equiv 1\ (mod\ n)$, por lo tanto si $x \not\equiv 1\ (mod\ n)$, el número n será compuesto, pero si $x \equiv 1\ (mod\ n)$ no podemos afirmar que n sea primo, se dice en este caso que n pasa el test respecto a la base a, por lo que n es un probable primo. Luego tomamos a otra base a y repetimos el test, nuevamente si $x \not\equiv 1\ (mod\ n)$ será compuesto y si no, habrá pasado otra ronda del test.

No tomaremos a a=1, ni a=n-1, en el primer caso pasaría el test sin importar n, y en el segundo no lo pasaría nunca pues sería congruente a -1. Para reforzar la seguridad en el test, podría calcularse mcd(a:n), si da un valor que no es 1, significa que hay un número que divide a n, luego no es primo.

Hagamos un ejemplo:

Si n=105, tomamos a un a en el intervalo (1;104), por ejemplo a=82

Luego calculamos: $82^{105-1} \equiv 46\ (mod\ 105)$, luego 105 no es primo pues no ha pasado el test para la base 82.

¿Qué tan fiable es este test, cómo se analiza la probabilidad? Para responder a esto, se encuentra el siguiente Lema.

"Supongamos que n no pasa el test con respecto a una cierta base b. Entonces n es un probable primo respecto a lo sumo el 50% de las bases coprimas con n" (Becker, Petrocola, Sanchez, 2001, p. 191)

Demostración:

Sea (b:n)=1, la inyección $a \to ab$, determina una permutación de las bases coprimas con n.

Sabemos que b no pasa el test para una base $\Rightarrow b^{n-1} \not\equiv 1\ (mod\ n)$

Si una base a pasara el test tenemos que: $(ab)^{n-1} \equiv a^{n-1}b^{n-1} \equiv b^{n-1} \not\equiv 1\ (mod\ n)$, por lo tanto, ab no pasaría el test. Entonces para cada base a que pasa, habrá una que no pasa el test. Luego cómo $a \to ab$ es una inyección, tenemos que la cantidad de las bases que pasan el test es menor o igual a las que no lo pasan. De ahí podemos afirmar el lema.

Por lo tanto, la probabilidad de que n sea compuesto si una base pasa el test es de a lo sumo $\frac{1}{2}$. Así mismo si fueran dos bases sería $\frac{1}{2}.\frac{1}{2} = \frac{1}{4}$, luego para una cantidad k de bases que pasen el test la probabilidad sería de $\frac{1}{2^k}$, por lo tanto, la probabilidad de que sea primo es de al menos $1 - \frac{1}{2^k}$. Es decir que a medida que el número n vaya pasando mas rondas del test, es más probable que sea primo y podríamos determinar de esa manera.

Esto, nos hace surgir otra pregunta, ¿qué ocurriría si a siempre pasa el test? ¿el número sería primo?

Pues no, existen números que se pasan el test de Fermat para todas las bases, y se los conoce como pseudoprimos o números de Carmichael. Entre ellos están

el 561, 1105, 1729, …, por mencionar algunos. Sin embargo, hay una forma de saber si un número es de Carmichael.

Teorema:

$Un\ número\ impar\ n\ es\ pseudoprimo\ \Leftrightarrow$

$n\ es\ libre\ de\ cuadrados \wedge\ \forall i: p_i - 1 | n - 1\ , siendo\ p_i\ un\ factor\ primo\ de\ n$

Probaremos la vuelta del teorema:

Si $p_i - 1 | n - 1\ \forall i \Rightarrow\ \exists k_i |\ \ n - 1 = (p_i - 1)k_i$

Luego $a^{n-1} \equiv a^{(p_i-1)k_i}\ \equiv (a^{p_i-1}\ )^{k_i} \equiv 1\ (mod\ p_i)$ por pequeño teorema de Fermat, si a y p_i son coprimos.

Si no fueran coprimos, entonces $p_i | a$, luego $a \equiv 0\ (mod\ p_i) \Rightarrow a^n \equiv 0 \equiv a\ (mod\ p_i)$

Dado que en ambos casos se llega a que $a^n \equiv a\ (mod\ p_i)\ \forall i$

Como n es libre de cuadrados, $(p_i : p_j) = 1\ \forall i \neq j$, entonces por teorema chino del resto se tiene que $a^n \equiv a\ (mod\ n) \Rightarrow n\ impar\ es\ pseudoprimo$.

Probaremos la ida:

Como $a^n \equiv a\ (mod\ n), si\ a = p_i\ \ , siendo\ p_i\ un\ factor\ de\ n$

$$\Rightarrow p_i^n \equiv p_i\ (mod\ n) \Rightarrow n | p_i^n - p_i$$

Supongamos que $p_i^2 | n \Rightarrow p_i^2 | p_i^n - p_i \Rightarrow p_i^2 | p_i(p_i^{n-1} - 1) \Rightarrow p_i | p_i^{n-1} - 1$

Como $p_i | p_i^{n-1} \Rightarrow p_i | 1$ Absurdo!, por lo tanto n, es libre de cuadrados.

Sea g raíz primitiva módulo p_i: $g^n \equiv g\ (mod\ n)$ por hipótesis,

como $p_i | n \Rightarrow g^n \equiv g\ (mod\ p_i)$

Siendo que $g \not\equiv 0\ (mod\ p_i)$ (pues g es raíz primitiva)

$\Rightarrow (g : p_i) = 1 \Rightarrow g^{n-1} \equiv 1\ (mod\ p_i)$

Como $ord_{p_i} g | n - 1 \Rightarrow p_i - 1 | n - 1\ \forall i$ (pues g es raíz primitiva en módulo p_i).

Por lo tanto, el test de Fermat nos arrojará también a estos números de Carmichael. Sin embargo, si reforzamos al test exigiendo que $(a : n) = 1$, tal vez algunos de ellos no aparezcan.

Se han desarrollado para este test dos programas, uno que determina con el test si un número es un probable primo, y otro que realiza un listado de todos

los números que cumplen el test bajo una cierta cantidad de rondas, claramente el listado puede variar ligeramente de una ejecución a otra pues las bases tomadas son pseudoaleatorias.

Test de Solovay-Strassen (o de Jacobi)

Tenemos a un número n natural impar que queremos analizar su primalidad. Tomamos aleatoriamente a un natural a comprendido entre 1 y n-1, luego calculamos (a:n), si el mcd da distinto de 1, entonces ese número que arroja será un divisor de n, por lo que n será compuesto. Si son coprimos, seguimos a calcular el símbolo de Jacobi $\left(\frac{a}{n}\right)$ y $a^{\frac{n-1}{2}}$ módulo n. El criterio de Euler nos dice que si n es un primo impar $\Rightarrow \left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}} \ (mod\ n)$, por lo tanto, si esa congruencia no se cumple el número será compuesto, de lo contrario el número n habrá superado el test y se dice que n es un pseudoprimo de Euler respecto a la base a, y procedemos a tomar otra base a para aplicar otra ronda del test.

Hagamos un ejemplo:

Si n=7453, tomamos aleatoriamente una base a, por ejemplo, 2045.

Podemos verificar que (7453:2045)=1, luego calculamos el símbolo de Jacobi:

$\left(\frac{2045}{7453}\right) = (-1)^{\frac{2045-1}{2}\cdot\frac{7453-1}{2}}\left(\frac{7453}{2045}\right)$ (por Ley de reciprocidad cuadrática)

$\left(\frac{2045}{7453}\right) = 1.\left(\frac{7453}{2045}\right) = \left(\frac{1318}{2045}\right)$ (pues $7453 \equiv 1318\ (mod\ 2045)$)

$\left(\frac{2045}{7453}\right) = \left(\frac{1318}{2045}\right) = \left(\frac{2}{2045}\right)\left(\frac{659}{2045}\right)$ (pues $\left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right)\left(\frac{b}{m}\right)$)

$\left(\frac{2045}{7453}\right) = (-1).(-1)^{\left(\frac{2045-1}{2}\right)\left(\frac{659-1}{2}\right)}\left(\frac{2045}{659}\right)$ (pues $2045 \equiv 5\ (mod\ 8)$ y por Ley de reciprocidad cuadrática)

$\left(\frac{2045}{7453}\right) = -\left(\frac{2045}{659}\right) = -\left(\frac{68}{659}\right)$ (pues $2045 \equiv 68\ (mod\ 659)$)

$\left(\frac{2045}{7453}\right) = -\left(\frac{68}{659}\right) = -\left(\frac{2^2}{659}\right)\left(\frac{17}{659}\right)$ (pues $\left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right)\left(\frac{b}{m}\right)$)

$\left(\frac{2045}{7453}\right) = -1(-1)^{\left(\frac{659-1}{2}\right)\left(\frac{17-1}{2}\right)}\left(\frac{659}{17}\right)$ (pues $\left(\frac{a^2}{n}\right) = 1, con\ (a:n) = 1$ y por Ley de reciprocidad cuadrática)

$\left(\frac{2045}{7453}\right) = -\left(\frac{13}{17}\right)$ (pues $659 \equiv 13\ (mod\ 17)$)

$\left(\frac{2045}{7453}\right) = -(-1)^{\left(\frac{17-1}{2}\right)\left(\frac{13-1}{2}\right)}\left(\frac{17}{13}\right)$ (por Ley de reciprocidad cuadrática)

$\left(\frac{2045}{7453}\right) = -\left(\frac{4}{13}\right) = -\left(\frac{2^2}{13}\right) = -1$ (pues $17 \equiv 4\ (mod\ 13)\ y\ \left(\frac{a^2}{n}\right) = 1, con\ (a{:}n) = 1$)

Luego $\left(\frac{2045}{7453}\right) = -1$

Calculando $2045^{\frac{7453-1}{2}} \equiv 2045^{3726} \equiv 6424\ (mod\ 7453)$

Como $6424 \not\equiv -1\ (mod\ 7453)$, entonces 7453 es compuesto.

Analicemos la relación que tiene con el test de Fermat:

"Si n supera el test de Jacobi respecto a una cierta base a entonces también supera al test de Fermat en dicha base" (Becker, Petrocola, Sanchez, 2001, p. 193)

Demostración:

Si n supera el test de Jacobi respecto a una cierta base a $\Rightarrow \left(\frac{a}{n}\right) \equiv a^{\frac{n-1}{2}}\ (mod\ n)$, puesto que el símbolo de Jacobi puede arrojar 1 o -1 (ya que a y n son coprimos, no puede arrojar 0)

Luego $a^{\frac{n-1}{2}} \equiv \pm 1\ (mod\ n)\ \Rightarrow\ \left(a^{\frac{n-1}{2}}\right)^2 \equiv 1\ (mod\ n) \Rightarrow a^{n-1} \equiv 1\ (mod\ n)$

Entonces supera al test de Fermat en dicha base.

Luego podemos deducir que la cantidad de los números que pasan al test de Jacobi es menor o igual a la cantidad de los números que pasan del test de Fermat.

Respecto a la probabilidad de que un número que lo pase sea primo, tenemos al siguiente lema, el cual solo enunciaremos:

"Supongamos que n es compuesto. Entonces no pasa el test de Jacobi con respecto a una base b. En tal caso, pasará el test respecto a lo sumo el 50% de las bases" (Becker, Petrocola, Sanchez, 2001, p. 194).

Por lo cual, la probabilidad de que n sea primo es la misma que en el test de Fermat, al menos $1 - \frac{1}{2^k}$, si pasó k veces el test.

Se han desarrollado para este test dos programas, uno que determina con el test si un número es un probable primo, y otro que realiza un listado de todos los números que cumplen el test bajo una cierta cantidad de rondas.

Test de Rabin-Miller

También conocido como TFP (test fuerte de primalidad), es el test de uso más difundido debido a que posee ventajas respecto a los otros dos que analizamos. Tenemos a un número n natural impar que queremos analizar su primalidad. Tomamos aleatoriamente a un natural a comprendido entre 1 y n-1, luego calculamos (a:n), si el mcd no es 1, n será compuesto, sino continuamos con el test. Basándonos en la siguiente propiedad:

$$a^{n-1} \equiv 1\ (n) \Leftrightarrow a^{\frac{n-1}{2}} \equiv \pm 1\ (mod\ n)$$

En lugar de calcular a a^{n-1}, calculamos $a^{\frac{n-1}{2}}$ en módulo n, si $a^{\frac{n-1}{2}} \not\equiv \pm 1\ (mod\ n) \Rightarrow a^{n-1} \not\equiv 1\ (mod\ n) \Rightarrow a\ es\ compuesto$

De lo contrario, es decir si $a^{\frac{n-1}{2}} \equiv \pm 1\ (mod\ n)$, tenemos dos posibilidades:

$$a^{\frac{n-1}{2}} \equiv 1\ (\ mod\ n) \vee a^{\frac{n-1}{2}} \equiv -1\ (mod\ n)$$

Si $a^{\frac{n-1}{2}} \equiv -1\ (mod\ n)$, no podemos volver a aplicar la propiedad inicial, así que diremos que n paso el test respecto a la base a.

Si $a^{\frac{n-1}{2}} \equiv 1\ (mod\ n)$, debemos analizar si $\frac{n-1}{2}\ es\ par\ o\ impar$

Si $\frac{n-1}{2}$ es impar entonces n es compuesto, de lo contrario estamos en la misma situación inicial para aplicar la propiedad, y esta vez deberíamos analizar primero si $a^{\frac{n-1}{4}} \equiv \pm 1\ (mod\ n)$, y se continua esta idea hasta que no pueda efectuarse más.

Luego, pensando los pasos al revés se tiene el siguiente algoritmo (siendo una base b):

"1) Se factoriza a n-1 en la forma $n - 1 \equiv 2^s d$, donde $s \geq\ 1$ y d es impar (esto corresponde a dividir a n-1 por 2 todas las veces que sea posible).

2) Se calcula $x_0 = b^d$

3) Si $x_0 \equiv \pm 1\ (n)$ el algoritmo finaliza, con cierta presunción de que n es primo. Se dice entonces que n pasa el test respecto a la base b ó que n es un *pseudoprimo fuerte* respecto a la base b.

4) Si $x_0 \not\equiv \pm 1\ (n)$, se van calculando $x_1 = x_0^2, x_2 = x_1^2, \dots, x_k = x_{k-1}^2, \dots. (k \leq s-1)$, analizando tras cada paso las siguientes alternativas (obsérvese que $x_k = b^{2^k d}$, siendo $x_{s-1} = b^{\frac{n-1}{2}}$):

4.i) $x_k \equiv -1\ (n)$. El algoritmo termina ahí mismo, con la misma conclusión que en 3).

4.ii) $x_k \equiv 1\ (n)$. Entonces n es compuesto con certeza y el test finaliza.

4.iii) $x_k \not\equiv \pm -1\ (n)$. En tal caso se procede a calcular x_{k+1} y se repite el análisis anterior.

5) Si se llega al valor k=s-1 y no se verifica que $x_{s-1} \equiv -1$, entonces el test finaliza con la seguridad de que n es compuesto." (Becker, Petrocola, Sanchez, 2001, p. 197)

Hagamos un ejemplo: si n=1413 y b= 349

Comprobamos que 1413 y 349 son coprimos, luego se calculan s y d, en este caso serán 2 y 353 respectivamente (pues $1412 = 2^2.353$)

Luego, $x_o \equiv 349^{353}\ (mod\ 1413) \Rightarrow x_o \equiv 283\ (mod\ 1413)$, como no es congruente a ± 1 en módulo n, procedemos a calcular x_1.

$$x_1 \equiv x_o^2 \equiv 283^2 \equiv 961\ (mod\ 1413)$$

Como $1 \leq 2-1$, y $x_1 \not\equiv -1\ (mod\ 1413) \Rightarrow 1413\ es\ compuesto$

Respecto a la probabilidad de que un número que lo pase sea primo, tenemos al siguiente lema, el cual solo enunciaremos:

"Si n pasa el test de Rabin-Miller respecto a una cierta base, entonces también pasa el test de Jacobi con respecto a dicha base. Además, si n es compuesto, sólo pasará el test de Rabin-Miller respecto a lo sumo el 25% de las bases." (Becker, Petrocola, Sanchez, 2001, p. 198)

Por lo cual, la probabilidad de que n sea primo es de al menos $1-\frac{1}{4^k}$, si pasó k veces el test. Esto significa que necesitamos iterar la mitad de veces para tener los mismos resultados que en los test anteriores.

Se han desarrollado para este test dos programas, uno que determina con el test si un número es un probable primo, y otro que realiza un listado de todos los números que cumplen el test bajo una cierta cantidad de rondas.

Programas desarrollados

Se han desarrollado como parte del trabajo los siguientes programas escritos en lenguaje C.

Programa: Test determinista basado en el teorema de Wilson

```
#include <stdio.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}


long long int factorial_modular (long long int numero, long long int modulo) {

    if ((numero==1)||(numero==0)) return 1;

    return producto_modular(numero, factorial_modular(numero-1,modulo),modulo);

}
```

```
int main () {
        long long int num;
        printf("Test de Wilson\n\nIngrese numero a testear (de hasta 4 cifras): ");
        scanf("%lld",&num);
   if (factorial_modular(num-1,num)==(num-1)) printf("\n%lld es PRIMO",num);
else printf("\n%lld es COMPUESTO",num);
        printf("\n--------------------\n\n");
        main();
        return 1;
}
```

Programa: Test determinista (basado en la propiedad mencionada)

```
#include <stdio.h>
#include <math.h>

int main() {
   long long int num, i;
   int primo;
   printf("Ingrese numero a saber si es primo: ");
   scanf("%lld",&num);
   //lo supongo primo
   primo=1;
   for (i=2;((i<=sqrt(num))&&(primo==1));i++) {
      if ((num%i)==0) {printf("El numero %lld es compuesto, es divisible por
%lld",num,i);primo=0;}
   }
   if (primo==1) printf("El numero %lld es primo",num);
   printf("\n--------------------\n\n");
        main();
}
```

Programa: Generar una lista de primos (utilizando el test anterior)

```
#include <stdio.h>

#include <math.h>

int primo (long long int n) {

    long long int i;

    for (i=2;i<=sqrt(n);i++) {

        if ((n%i)==0) return 0;

    }

    return 1;

}

int main() {

    long long int num, posicion=3;

    for (num=2; num<9999999; num++) {

        if (primo(2*num+1)==1) {printf("\nPrimo
#%lld:\t%lld",posicion,2*num+1);posicion++;}

    }

    printf("\n\nFinalizado.");

    getch();

}
```

Programa: Test de Fermat (considerando bases coprimas con con n)

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>
```

```
long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}


unsigned long long int exponenciacion_binaria_modular(long long int base, long long
int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}
```

```
        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


long long int potencia_ent (long base, long exp) {

   if (exp==0) return 1;

   if ((exp%2)==0) {long long int aux=potencia_ent(base, exp/2); return
aux*aux;} else return base*potencia_ent(base,exp-1);

}


int main () {

        long long int num, base,x=1,iteracion;

        srand(time(NULL));

        printf("Test de Fermat\n\nIngrese numero a testear: ");

        scanf("%lld",&num);


        for (iteracion=1;((iteracion<11)&&(x==1));iteracion++) {

       //tomo un numero pseudoaleatorio base

                base=2+rand()%(num-3);

                //tomo una precaucion adicional ver si (base:num)=1
```

```
                if (mcd(base,num)==1) {

                x=exponenciacion_binaria_modular(base, num-1,num);

        if (x!=1) printf("\n%lld COMPUESTO. Ya que la base %lld no paso el test, pues x=%lld",num,base,x); else printf("\n%lld paso el test para la base %lld.\n[Iteracion: %lld - Probabilidad: %.0f %% (aprox.)]\n",num,base,iteracion,(100-100/(potencia_ent(2,iteracion))+.0));

                } else {printf("\n%lld COMPUESTO. Ya que el mcd(%lld,%lld)=%lld", num, base, num, mcd(base,num));x=0;}

        }

        printf("\n--------------------\n\n");

        main();

}
```

Programa: Generar una lista de pseudoprimos (utilizando el test de Fermat)

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;} else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int modulo) {

 return representante_modular(representante_modular(a, modulo)*representante_modular(b, modulo), modulo);
```

```
}


unsigned long long int exponenciacion_binaria_modular(long long int base, long long
int exponente, long long int modulo) {

   long long int aux;

   if (exponente==1) return representante_modular(base, modulo);

   if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


int Fermat(long long int n) {

        long long int base,x=1,iteracion;

        for (iteracion=1;((iteracion<11)&&(x==1));iteracion++) {

       //tomo un numero pseudoaleatorio base

                base=2+rand()%(n-3);

                //tomo una precaucion adicional ver si (base:num)=1

                if (mcd(base,n)==1) {

                x=exponenciacion_binaria_modular(base, n-1,n);
```

```
        if (x!=1) return 0;
                    } else return 0;
          }
          return 1;
}


int main() {
    long long int num, posicion=3;
    srand(time(NULL));
    for (num=2; num<9999999; num++) {
        if (Fermat(2*num+1)==1) {printf("\nPosible Primo
#%lld:\t%lld",posicion,2*num+1);posicion++;}
    }
    getch();
}
```

Programa: Test de Solovay-Strassen

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


long long int representante_modular (long long int numero, long long int modulo) {
if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {
        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;
}


long long int suma_modular (long long int a, long long int b, long long int modulo) {
 return representante_modular(representante_modular(a,
modulo)+representante_modular(b, modulo), modulo);
}
```

```
long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}


unsigned long long int exponenciacion_binaria_modular(long long int base, long long
int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


int jacobi (long long int m, long long int n) {

        if ((m==1)||(n==1)) return 1;

        if ((m>n)||(m<0)) return jacobi(representante_modular(m,n),n);

        if ((mcd(m,n))!=1) return 0;
```

```
        if (m==(n-1)) {if (representante_modular(n,4)==1) return 1; else return -
1;}

        if (m==2) {if
((representante_modular(n,8)==1)||(representante_modular(n,8)==7)) return 1;
else return -1;}

   if (representante_modular(m,2)==0) return jacobi(2,n)*jacobi(m/2,n);

   if (representante_modular(n,2)==0) return jacobi(m,2)*jacobi(m,n/2);

   if (representante_modular((m-1)/2,2)==0) return jacobi(n,m); else {

      if (representante_modular((n-1)/2,2)==0) return jacobi(n,m); else return (-
1)*jacobi(n,m);

   }

}


long long int potencia_ent (long base, long exp) {

   if (exp==0) return 1;

   if ((exp%2)==0) {long long int aux=potencia_ent(base, exp/2); return
aux*aux;} else return base*potencia_ent(base,exp-1);

}


int main () {

        long long int num, base,x=1,iteracion, y=1;

        srand(time(NULL));

        printf("Test de Solovay-Strassen\n\nIngrese numero a testear: ");

        scanf("%lld",&num);

   if (((num%2)==0)||(num<0)) printf("\n\nEl numero n debe ser un natural
impar"); else {

        for (iteracion=1;((iteracion<11)&&(suma_modular(x,(-
1)*y,num)==0));iteracion++) {

      //tomo un numero pseudoaleatorio base

                base=2+rand()%(num-3);


                if (mcd(base,num)==1) {

                x=exponenciacion_binaria_modular(base, (num-1)/2,num);
```

```
                y=jacobi(base,num);

                //calculo si x=y (mod num)

                if (suma_modular(x,(-1)*y,num)!=0) printf("\n%lld COMPUESTO. Ya
que la base %lld no paso el test, pues (%lld)^((%lld-1)/2)=%lld (mod %lld) y
(%lld/%lld)=%lld",num,base,base,num,x,num,base,num,y); else printf("\n%lld
paso el test para la base %lld.\n[Iteracion: %lld - Probabilidad: %.0f %%
(aprox.)]\n",num,base,iteracion,(100-100/(potencia_ent(2,iteracion))+.0));



                } else {printf("\n%lld COMPUESTO. Ya que el mcd(%lld,%lld)=%lld",
num, base, num, mcd(base,num));x=0;}



        }}

        printf("\n--------------------\n\n");

        main();

}
```

Programa: Generar una lista de pseudoprimos (utilizando el test de Solovay-Strassen)

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int suma_modular (long long int a, long long int b, long long int modulo) {

 return representante_modular(representante_modular(a,
modulo)+representante_modular(b, modulo), modulo);

}
```

```
long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}


unsigned long long int exponenciacion_binaria_modular(long long int base, long long
int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


int jacobi (long long int m, long long int n) {

        if ((m==1)||(n==1)) return 1;

        if ((m>n)||(m<0)) return jacobi(representante_modular(m,n),n);

        if ((mcd(m,n))!=1) return 0;
```

```
        if (m==(n-1)) {if (representante_modular(n,4)==1) return 1; else return -
1;}

        if (m==2) {if
((representante_modular(n,8)==1)||(representante_modular(n,8)==7)) return 1;
else return -1;}

   if (representante_modular(m,2)==0) return jacobi(2,n)*jacobi(m/2,n);

   if (representante_modular(n,2)==0) return jacobi(m,2)*jacobi(m,n/2);

   if (representante_modular((m-1)/2,2)==0) return jacobi(n,m); else {

      if (representante_modular((n-1)/2,2)==0) return jacobi(n,m); else return (-
1)*jacobi(n,m);

   }

}


int SolovayStrassen (long long int n) {

        long long int base,x=1,iteracion, y=1;


        for (iteracion=1;iteracion<11;iteracion++) {

      //tomo un numero pseudoaleatorio base

                base=2+rand()%(n-3);


                if (mcd(base,n)==1) {

                x=exponenciacion_binaria_modular(base,(n-1)/2,n);

                y=jacobi(base,n);

                //calculo si x=y (mod num)

                if (suma_modular(x,(-1)*y,n)!=0) return 0;


                } else return 0;

        }

        return 1;

}
```

```
int main() {

    long long int num, posicion=3;

    srand(time(NULL));

    for (num=2; num<9999999; num++) {

        if (SolovayStrassen(2*num+1)==1) {printf("\nPosible Primo
#%lld:\t%lld",posicion,2*num+1);posicion++;}

    }

    getch();

}
```

Programa: Test de Rabin-Miller

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

        if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}


long long int exponenciacion_binaria_modular(long long int base, long long int
exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base,
exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return
```

```
producto_modular(base,exponenciacion_binaria_modular(base, exponente-
1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


long long int potencia_ent (long base, long exp) {

   if (exp==0) return 1;

   if ((exp%2)==0) {long long int aux=potencia_ent(base, exp/2); return
aux*aux;} else return base*potencia_ent(base,exp-1);

}


int main () {

        long long int num, base, iteracion, s, d, xo,xk, k;

        int pasa, compuesto;

        srand(time(NULL));

        printf("Test de Rabin-Miller\n\nIngrese un numero IMPAR >=5 a testear: ");

        scanf("%lld",&num);

        if (num%2==0) printf ("%\n%lld es compuesto, pues es divisible por
2",num); else {
```

```
        //determino d y s
        d=num-1;s=0;
        while ((d%2)==0) {
      s++;
      d/=2;
        }
   printf("\nEncuentro s=%lld d=%lld",s,d);


   compuesto=0;
        for (iteracion=1;((iteracion<6)&&(compuesto==0));iteracion++) {
      //tomo un numero pseudoaleatorio base
                base=2+rand()%(num-3);
                if (mcd(base,num)!=1) {compuesto=1;printf("\n%lld es compuesto,
pues (%lld:%lld)=%lld",num,base,num,mcd(base,num));} else {
      xo=exponenciacion_binaria_modular(base,d,num);
      if ((xo==1)||(xo==num-1)) printf("\n%lld es un pseudoprimo fuerte respecto
a la base %lld, pues xo=%lld (mod %lld)\n[Iteracion: %lld - Probabilidad: %.0f %%
(aprox.)]\n",num,base,xo,num,iteracion,(100-100/(potencia_ent(4,iteracion))+.0));
else {
          xk=xo;
          printf("\n[xo=%lld (mod %lld)  (con base=%lld)]",xo,num,base);
          pasa=0;
          for (k=1;((k<=(s-1))&&(pasa==0)&&(compuesto==0));k++) {
             xk=exponenciacion_binaria_modular(xk,2,num);
             if (xk==(num-1)) {pasa=1;printf("\n%lld es un pseudoprimo fuerte
respecto a la base %lld, pues xk=-1 (mod %lld) (con k=%lld)\n[Iteracion: %lld -
Probabilidad: %.0f %% (aprox.)]\n",num,base,num,k,iteracion,(100-
100/(potencia_ent(4,iteracion))+.0));} else {
                if (xk==1) {compuesto=1;printf("\n%lld es compuesto, pues con la
base %lld xk=%lld (mod %lld) (con k=%lld)",num,base,xk,num,k);} else {
                   printf("\n[xk=%lld (mod %lld) (con k=%lld)  (con
base=%lld)]",xk,num,k,base);
                }
```

```
            }

          }

          if ((pasa==0)&&(compuesto==0)) {compuesto=1;printf("\n%lld es
compuesto, pues con la base %lld xk=%lld (mod %lld) (con k=s-
1=%lld)",num,base,xk,num,k-1);}

      }

        }

        }

        }

        printf("\n--------------------\n\n");

        main();

}
```

Programa: Generar una lista de pseudoprimos (utilizando el test de Rabin-Miller)

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>


long long int representante_modular (long long int numero, long long int modulo) {

if ((numero>=0)&&(numero<modulo)) return numero; else if (numero<0) {

      if ((numero%modulo)!=0) {numero*=-1;return modulo-numero%modulo;}
else return numero%modulo;} else return numero%modulo;

}


long long int producto_modular (long long int a, long long int b, long long int
modulo) {

 return representante_modular(representante_modular(a,
modulo)*representante_modular(b, modulo), modulo);

}
```

```
unsigned long long int exponenciacion_binaria_modular(long long int base, long long int exponente, long long int modulo) {

    long long int aux;

    if (exponente==1) return representante_modular(base, modulo);

    if ((exponente%2)==0) {aux=exponenciacion_binaria_modular(base, exponente/2, modulo);return producto_modular(aux,aux,modulo);} else return producto_modular(base,exponenciacion_binaria_modular(base, exponente-1,modulo),modulo);

}


long long int mcd (long long int a, long long int b) {

        long long int r=1, aux;

        if (a>b) {aux=a; a=b; b=aux;}

        if (b==0) return a;

        while (r!=0) {

                r=a%b;

                a=b;

                b=r;

        }

        return a;

}


int RabinMiller(long long int n) {


        long long int base, iteracion, s, d, xo,xk, k;

        int pasa;

        if (n%2==0) return 0; else {

        //determino d y s

        d=n-1;s=0;

        while ((d%2)==0) {

      s++;

      d/=2;
```

```
        }
        for (iteracion=1;(iteracion<6);iteracion++) {
      //tomo un numero pseudoaleatorio base
                base=2+rand()%(n-3);
                if (mcd(base,n)!=1) return 0; else {
      xo=exponenciacion_binaria_modular(base,d,n);
      if ((xo!=1)&&(xo!=n-1)) {
         xk=xo; pasa=0;
         for (k=1;((k<=(s-1))&&(pasa==0));k++) {
            xk=exponenciacion_binaria_modular(xk,2,n);
            if (xk==(n-1)) pasa=1; else {
               if (xk==1) return 0;
            }
         }
         if (pasa==0) return 0;
      }
        }
        }
        return 1;
        }
}


int main() {
   long long int num, posicion=3;
   srand(time(NULL));
   for (num=2; num<9999999; num++) {
      if (RabinMiller(2*num+1)==1) {printf("\nPosible Primo
#%lld:\t%lld",posicion,2*num+1);posicion++;}
   }
   getch();
}
```

Conclusiones

En los cifrados clásicos notamos que éstos solían ser ingenuos en cuanto a la seguridad ofrecida, si bien parecían ser seguros en su momento, el criptoanálisis demostró sus falencias y nos muestra como vulnerarlos. Cuando un sistema criptográfico es vulnerable, no es seguro y por lo tanto ya no es viable de aplicarse. A partir de esto, de prueba y error la criptografía fue avanzando y cada vez utilizando matemática más sofisticada, y algoritmos más complejos para garantizar la seguridad del cifrado.

Si bien podemos diferenciar criptografía simétrica y asimétrica, no es del todo cierto afirmar que la criptografía simétrica es insegura, si bien lo es la clásica, el cifrado AES que utiliza rondas de cifrados simétricos es seguro. Es importante destacar que nos referimos igualmente al cifrado AES con los estándares usados actualmente, en el trabajo no entramos en su análisis al detalle, sino que se mencionó en que se basaba su funcionamiento.

Los cifrados modernos como el AES y el RSA, se estiman que tardarían mucho tiempo en ser descifrados, por lo tanto, actualmente son seguros. Es interesante ver las implicaciones que tiene la matemática en esto, por ejemplo, el cifrado RSA basa su seguridad en el desconocimiento actual de la resolución del problema matemático de la factorización de números de gran cantidad de cifras, y la falta de un algoritmo eficiente que lo logre, hace que sea imposible de descifrar en un tiempo razonable, haciendo las comunicaciones y transferencias de datos seguras.

Observando el funcionamiento de los programas que generan una lista de números pseudoprimos, se puede observar el porqué del favoritismo al test de Rabin-Miller, se observa que avanza (con los mismos recursos informáticos) mucho más rápido que los test de Fermat y Solovay-Strassen, además que su probabilidad es mayor por cada iteración. De 10 ejecuciones de los 4 programas en simultáneo, bajo la misma probabilidad de $1-\frac{1}{2^{10}}$, se observó que solo en una oportunidad el test de Rabin-Miller arrojó un número más que los primos existentes menores a 2.10^7, mientras el test de Fermat arrojaba por ejecución de 20 a 30 pseudoprimos más, y el de Solovay-Strassen se acercaba más a los resultados de Rabin-Miller, y era más lento que ambos. Aún así los tres eran más rápidos al test determinista propuesto (que tal vez no fue tan rápido como debiera, por no aplicar la criba de Eratóstenes).

Finalmente, este trabajo nos sirve para apreciar tanto a la criptografía como a su criptoanálsis, y a la matemática aplicada a su funcionamiento, además del valor histórico que tienen los cifrados por si mismos.

Bibliografía

Libros:

M. Becker, N. Pietrocola, C. Sanchez (2001), Aritmética, Buenos Aires, Argentina, RED OLIMPICA.

Joan Gómez (2010), Matemáticos, espías y piratas informáticos (Codificación y Criptografía), España, RBA Coleccionables S.A.

Rossetti (Ed.) (2010), Aventuras Matemáticas, Argentina, CEI

Grossman (1988), Aplicaciones de álgebra lineal, México, Grupo Editorial Iberoamérica.

Craig P. Bauer (2013), Secret History: The Story of Cryptology, Florida, EEUU, CRC Press.

Montaner y Simón, Sociedad Internacional (1912), Diccionario Enciclopédico Hispano-Americano de Literatura, Ciencias, Artes, Etc., Edición Profusamente Ilustrada (TOMO VI), España.

Enciclopedia Clarín (Vol. 5), VISOR Enciclopedias Audiovisuales S.A., 1999 - Arte Gráfico Editorial Argentino (AGEA), Argentina.

Enciclopedia Clarín (Vol. 7), VISOR Enciclopedias Audiovisuales S.A., 1999 - Arte Gráfico Editorial Argentino (AGEA), Argentina.

Vicente Martinez Orga (1986), Enciclopedia práctica de la informática aplicada, Criptografía. La ocultación de mensajes y el ordenador, Ediciones Siglo Cultural, España.

Webs:

Diccionario de la Real Academia Española. Recuperado el 01/12/2018. De: http://dle.rae.es/?id=BHcfHjo

Diccionario de la Real Academia Española. Recuperado el 01/12/2018. De: http://dle.rae.es/?id=FABu3oz