UNIVERSIDAD DE ALMERÍA

Simulación de algoritmos criptográficos de clave pública para grupos mediante el uso de Mathematica

Simulation of public key cryptographic algorithms for groups using Mathematica

---

Trabajo de Fin de Master en Matemáticas

Autor: **David Trujillo Gradit**

Tutor: **Juan Antonio López Ramos**

Área de conocimiento: **Criptografía**

Mes y año: **Julio de 2020**

Índice general

Simulación de algoritmos criptográficos de clave pública para grupos mediante el uso de Mathematica

Resumen

En primer lugar, tratamos el conocido como criptosistema RSA [9], primer criptosistema de los llamados de clave pública y que tiene como base los conocidos grupos RSA, que son grupos de la forma $Z_{\phi(n)}$, para n un número entero positivo y $\phi(n)$ el número de unidades de Z_n.

A continuación abordamos el criptosistema de ElGamal [1], construido originalmente sobre el grupo multiplicativo Z_p^* para p un número primo. Seguidamente, en los capítulos 4, 5 y 6 llevamos a cabo implementaciones del mismo criptosistema de ElGamal, pero sobre otros grupos, tales como un cuerpo finito cualquiera, que es una extensión natural del caso de Z_p^*, el caso no conmutativo de las matrices circulantes y, para finalizar, el grupo de puntos de una curva elíptica, grupo este ampliamente usado en la actualidad debido a sus reducidas necesidades de ancho de banda, es decir, de información enviada a través de la red o el medio inalámbrico.

En todos y cada uno de los casos tratados en esta memoria se ha hecho una implementación de los métodos matemáticos necesarios para un uso real de los criptosistemas, usando para ello el software Mathematica [10].

Palabras clave:

Criptografía, comunicación, claves pública y privada, RSA, ElGamal, grupo finito, matriz circulante, curva elíptica

Simulation of public key cryptographic algorithms for groups using Mathematica

Abstract

First, we treat the RSA cryptosystem [9], the first cryptosystem of public key calls and which is based on the known RSA groups, which are groups of the form $Z_{\phi(n)}$, for n a positive integer and $\phi(n)$ the number of units of Z_n.

Then, we study the ElGamal cryptosystem [1], originally built on the multiplicative group Z_p^* for p a prime number. Next, in Chapters 4, 5 and 6 we carry out implementations of the same ElGamal cryptosystem, but on other groups, such as any finite body, which is a natural extension of the case of Z_p^*, the non-commutative case of the circulating matrices and, finally, the group of points on an elliptic curve, this group is widely used today due to its low bandwidth needs, that is, information sent through the network or wireless medium .

In each and every one of the cases treated in this report, an implementation of the mathematical methods necessary for a real use of cryptosystems has been made, using Mathematica software for this [10].

key words:

Cryptography, communication, public and private keys, RSA, ElGamal, finite group, circulating matrix, elliptic curve

Introducción

Necesidad del ser humano por mantener secreta ciertos tipos de información.

Desde épocas antiguas, siempre ha existido cierto tipo de información que el ser humano ha querido mantener en secreto ya sea para uso propio o para envió hacia otra persona de forma confidencial. En la actualidad todo ser humano tiene este derecho a la intimidad y a la privacidad, aunque es uno de los que más ha sufrido por el avance de la tecnología. El hecho de que nuestros datos se hayan convertido en un bien preciado demanda una modernización en los sistemas que los aseguran ante posibles atacantes. Además este no es un hecho que nos afecta de forma individual, sino que también es necesario en entidades públicas y privadas para el envío y recepción de información.

Seguridad en las comunicaciones.

La Seguridad de las comunicaciones, se encarga de prevenir que alguna entidad no autorizada que intercepte la comunicación pueda acceder de forma inteligible a información. Esta tarea de protección se lleva a cabo campos de estudios como la criptografía, la emisión segura, la transmisión segura, la seguridad del flujo del tráfico y la seguridad física del equipo que se encarga de las comunicaciones. En nuestro trabajo trataremos la seguridad de la comunicación a través de la criptografía.

En la actualidad su uso lo podemos ver reflejado en numerosos ejemplo de la vida cotidiana, como banca electrónica, comercio electrónico, servicios de audio, etc:

- Para la seguridad financiera, una de las más robustas, la mayoría de los bancos aplican protocolos y algoritmos de cifrado. Un ejemplo, RSA (Rivest Shamir Adleman) en los tockens, como factor de doble autenticación ante diferentes tipos de servicios bancarios.

  "En nuestro caso, en el sistema financiero debemos garantizar que tanto las transacciones de nuestro cliente, así como las comunicaciones en él, se mantengan protegidas con mecanismos de encriptación, para reducir los riesgos y proteger la información", Jonathan Fernández.

- En el comercio electrónico es muy común el uso de firmas electrónicas: Quien envía un mensaje, cifra su contenido con su clave privada y quien lo recibe, lo descifra con su clave pública, determinando así la autenticidad del origen del mensaje y garantizando que el envío de la firma electrónica es de quien dice serlo.

- WhatsApp: el cifrado extremo a extremo. El protocolo que implementa esta característica, Signal Protocol, permite que los mensajes se transmitan cifrados entre

emisor y receptor, y no solo los mensajes entre dos usuarios, sino los grupos de chat, el envío de mensajes multimedia y las conversaciones de voz. Esto es debido al uso de diferentes sistemas criptográficos utilizadas por esta aplicación, entre las que destacamos la curva elíptica que estudiaremos en nuestro trabajo.

Criptografía

La Criptografía, como ciencia, tiene una gran importancia en este ámbito. Sus principales funciones son la confidencialidad de la información, es decir, que la información no sea accesible por parte de terceras partes no autorizadas; la autenticación de la información, o lo que es lo mismo, la posibilidad de verificar el origen de la misma; integridad de la información, lo que significa que es posible verificar que la información no ha sido alterada desde el proceso de creación de esta y, finalmente, no repudio, es decir, que la fuente de información no puede negar el origen de la misma. En esta memoria nos vamos a centrar en la primera de sus funciones, es decir, la confidencialidad y más precisamente en el uso de diversas estructuras de grupos algebraicos que permiten la definición de distintos algoritmos que proporcionan esta deseada funcionalidad.

Capítulo 1

Tratamiento digital de la comunicación

1. Criptografía y criptosistemas.

La palabra criptografía proviene en un sentido etimológico del griego Kriptos = ocultar, Graphos = escritura, lo que significaría ocultar la escritura.

En un sentido más amplio, la Criptografía es la ciencia encargada de diseñar funciones o dispositivos, capaces de transformar mensajes legibles a mensajes cifrados de tal manera que esta transformación (cifrar) y su transformación inversa (descifrar) sólo pueden ser factibles con el conocimiento de una o más llaves.

La criptografía se puede clasificar históricamente en dos: La criptografía clásica y la criptografía moderna.

La criptografía clásica es aquella que se utilizó desde los inicios más básicos y sencillos hasta la mitad del siglo XX. También puede entenderse como la criptografía no computarizada. Los métodos utilizados eran variados, algunos muy simples y otros muy complicados de criptoanalizar para su época.

Se puede decir que la criptografía moderna se inició después de tres hechos: el primero fue la publicación de la "Teoría de la Información" por Shannon; el segundo, la aparición del estándar del sistema de cifrado DES (Data Encryption Standard) en 1974 y finalmente con la aparición del estudio realizado por Whitfield Diffie y Martin Hellman sobre la aplicación de funciones matemáticas de un solo sentido a un modelo de cifrado, denominado cifrado de clave pública en 1976.

2. Criptosistema Cesar.

Uno de los primeros sistemas criptográficos surgió en el siglo I a.c., un método de cifrado conocido con el nombre de cifrado de César en honor al emperador Julio César y en el que se realiza una transformación al mensaje que se desea enviar a un receptor, de tipo monoalfabética. El cifrado del César aplica un desplazamiento constante de tres caracteres al mensaje inicial, de forma que el alfabeto de cifrado es el mismo que el alfabeto del texto del mensaje, pero desplazado 3 espacios hacia la derecha módulo n, con n el número de letras del mismo.

Ejemplo 1. *Un ejemplo de este tipo de crifrado sería el siguiente.*

- *Mensaje original: MENSAJE DE PRUEBA*
- *Mensaje cifrado: OHPVDM GH SUXHED*

Al describir el cifrado de César se utilizó un concepto muy usado en las matemáticas y más en criptografía: el módulo.

De esta forma, podemos describir el sistema de cifrado y descifrado de la siguiente forma:

Para cifrar:

$C_i = (3 + M_i)$ mod 27 con $i = 0, 1, ., n$; $n =$ número de letras del mensaje donde C_i es la letra cifrada y M_i es la letra a cifrar el alfabeto comienza con $A = 0$, $B = 1$, ..., $Z = 26$.

Para descifrar:

$M_i = (C_i - 3)$ mod 27 $= (C_i + 24)$ mod 27 con $i = 0, 1, ., n$; $n =$ número de letras del mensaje donde C_i es la letra cifrada y M_i es la letra a cifrar el alfabeto comienza con $A = 0$, $B = 1$, ..., $Z = 26$.

Observación 1. *Como podemos ver en el ejemplo, el uso de las matemáticas facilita la búsqueda y el análisis de los métodos que se usan en Criptografía, permitiendo, al mismo tiempo estudiar y asignar niveles de seguridad de los métodos usados en la actualidad, haciendo de la Criptografía una ciencia moderna y que evoluciona a la par que las Matemáticas.*

3. Criptografía por bloques y sus tipos: simétrica y de clave pública.

En el caso del criptosistema de César puede observarse que la sustitución de caracteres uno a uno tiene un análisis muy sencillo desde el punto de vista de un adversario sin más que usar métodos estadísticos muy básicos. Si en un mensaje cifrado mediante dicho criptosistema aparece un carácter con mucha frecuencia, eso nos lleva a pensar que dicho carácter es uno que, en el idioma en el que está escrito el mensaje original, es un carácter muy común. Por ejemplo, si en un texto cifrado aparece con mayor frecuencia la letra J, ello podría indicar que dicha letra estaría sustituyendo a una A o una E, con lo que fácilmente podríamos inducir fácilmente la clave del mismo. Un razonamiento similar puede usarse incluso con grupos de dos letras. Este hecho nos lleva a considerar una forma de cifrado diferente, como es la de sustituir bloques completos de texto original por otros, con lo que nace la Criptografía por bloques. En este ámbito podemos distinguir dos tipos esenciales: la Criptografía Simétrica y la Asimétrica o de Clave Pública.

- Criptografía Simétrica:

  La criptografía simétrica es aquella que utiliza un sistema de cifrado para cifrar y descifrar un mensaje utilizando únicamente una clave secreta. Se puede observar en la figura 1.1 como el eje de simetría divide al sistema en dos completamente iguales,

esto ilustra el hecho del porqué se le da el nombre de criptografía simétrica.

Este tipo de criptografía sólo utiliza una llave para cifrar y descifrar, esto es: si yo cifro un mensaje m con una clave secreta k entonces el mensaje cifrado resultante m' únicamente lo voy a poder descifrar con la misma clave secreta k.

El problema de este sistema criptográfico es que además de la necesidad de generar claves privadas distintas para cada uno de los receptores, es el modo de compartir dichas claves privadas al receptor sin comprometer la confidencialidad de las mismas. Estos problemas se resuelven de cierta manera con criptografía asimétrica.

- Criptografía Asimétrica

Si se observa la figura 1.2, que ilustra la idea de criptografía de clave pública, se puede ver claramente que no existe simetría en ella, ya que de un lado de la figura se cifra o descifra con una clave pública y en el otro lado con una clave privada. Algunos ejemplos de este tipo de criptografía son RSA y ElGamal.

Veamos la solución al problema del intercambio de claves que presenta la criptografía simétrica: Lo que se hace es que se toma la clave pública de la persona a la que se le va a enviar el mensaje y se cifra con un sistema asimétrico la clave secreta, esto implica que sólo la persona poseedora de la clave privada pueda descifrar lo que se está enviando y con ello tener la clave secreta, tal y como se muestra en la figura 1.2. Dicho procedimiento lo explicaremos mas detalladamente en capítulos posteriores.

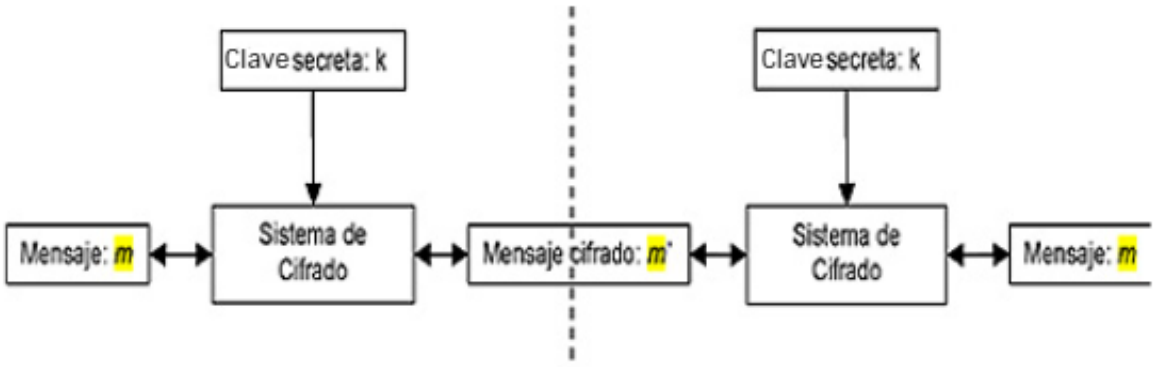


Figura 1.1: Criptografía simétrica

Figura 1.2: Criptografía asimétrica

4. Codificación a binario.

El objetivo de este trabajo fin de máster es el desarrollo de diferentes sistemas de encriptación como son el RSA y ElGamal. En cada capítulo haremos una descripción detallada de cada uno de ellos, además de crear un programa desde cero para su implantación en Wolfram Mathematica. Realizaremos ejemplos de su uso en la vida real y para ello necesitaremos codificar el mensaje que se desea enviar al receptor a binario. El objetivo de este capítulo es el desarrollo de un pequeño programa que convierta cualquier mensaje compuesto por letras y números en una cadena compuesta por unos y ceros.

El contenido del mensaje se compone de una secuencia de caracteres. Los caracteres representan letras del alfabeto, puntuación, etc. Pero el contenido se almacena en un ordenador como una secuencia de bytes, que son valores numéricos. La forma en que la secuencia de bytes se convierte en caracteres depende de la clave que se haya utilizado para codificar el texto. En este contexto, esa clave se denomina codificación de caracteres. Dicho método es el que utilizamos para convertir un carácter de un lenguaje natural como puede ser el alfabeto en un símbolo de otro sistema de representación. En nuestro caso convertiremos cada carácter en un número aplicando una serie de normas de codificación.

Los principales tipos de codificación de carácteres son Base64 y ASCII:

- Base64: es un sistema de numeración posicional que usa 64 como base para representar los carácteres A-Z, a-z y 0-9. Esto ha propiciado su uso para codificación de correos electrónicos, PGP y otras aplicaciones. También es denominado Radix-64
- ASCII (Código Estándar Estadounidense para el Intercambio de Información): es un código basado en el alfabeto latino. En la actualidad define códigos para 32 caracteres no imprimibles, de los cuales la mayoría son caracteres de control que tienen efecto sobre cómo se procesa el texto, más otros 95 caracteres imprimibles que les siguen en la numeración. Casi todos los sistemas informáticos actuales utilizan el código ASCII o una extensión compatible para representar textos y para el control de dispositivos que manejan texto como el teclado.

El tipo de codificación ASCII es en el que nos centraremos y usaremos en este trabajo. A medida que la tecnología avanzó se desarrollaron variantes de este código para incluir caracteres del alfabeto latino y se le denominó ASCII extendido. Por lo tanto, la tabla de código que utilizaremos será la siguiente:

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

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

ASCII extendido

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Una vez que nuestro mensaje, que estaba compuesto por una cadena de caracteres, lo hemos codificado en una secuencia de números como en el siguiente ejemplo:

- Mensaje: "Hola, hemos codificado en ASCII"
- Codificado: $72, 111, 108, 97, 44, 32, 104, 101, 109, 111, 115, 32, 99, 111, 100, 105, 102, 105, 99, 97, 100, 111, 32, 101, 110, 32, 65, 83, 67, 73, 73$

El siguiente paso que trataremos en este capítulo será convertir cada uno de los números de la secuencia en código binario y así obtener una cadena de ceros y unos. Dicha cadena será a que codificaremos utilizando diferentes tipos de criptosistemas como el RSA o ElGamal.

5. Desarrollo del programa en Wolfram Mathematica.

Debemos tener en cuenta que para pasar el mensaje que queremos enviar a binario necesitaremos convertir el texto en una cadena de caracteres, asociar cada cada carácter a un número y posteriormente pasar ese número a binario.

El objetivo de la siguiente función es convertir cualquier texto en un número compuesto por ceros y unos, para ello cada carácter del texto es convertido a binario y una vez obtenidos todos se unen para formar un único número.

Mensaje[m_]:=Module[{l}, l = {};

l = Map[FromDigits, Partition[Flatten[Map[Abinario8, Map[IntegerDigits[#, 2]&,

ToCharacterCode[m]]]], UpTo[120]]];

Return[l]]

Esta función auxiliar ha sido creada para solventar el problema de que al pasar cada carácter a binario, algunos tenían longitud 7 y otros 8. Con ella, unificamos todas las longitudes a 8 añadiendo los ceros necesarios para que a la hora de realizar el procedimiento inverso poder dividir el número en cadenas iguales de longitud 8.

Abinario8[l_]:=Module[{n}, $n = l$;

While[Length[n] < 8, n = Prepend[n, 0]];

Return[n]]

El objetivo de la siguiente función es que dado un número compuesto por ceros y unos, lo divida en cadenas de 8, cada cadena binaria la convierta en número y posteriormente a su carácter asociado.

Descifrar[c_]:=Module[{m}, m = {};

m = Map[FromCharacterCode, Map[FromDigits[#, 2]&,

Partition[Flatten[Normalizar[Flatten[IntegerDigits[c]]]], 8]]];Return[m]]

Esta función nos asegura que cada numero compuestos de ceros y unos tenga longitud 120, añadiéndole a la izquierda tantos ceros como sea necesario.

Normalizar[n_]:= Module[{c},c=n;

While[Length[c]<120, c = Prepend[c,0]];

Return[c]]

Como el texto puede ser tan largo como se desee, será dividido en bloques de 120, esta función descifra cada bloque de 120 y obtiene el mensaje de texto inicial.

Código[c_]:=Map[Descifrar, c]

6. Ejemplos.

El siguiente mensaje lo utilizaremos en capítulos siguientes ya que no es muy largo y así los cálculos posteriores no ocupan demasiadas páginas.

Mensaje[“Ejemplo prueba.”]

{1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000010010111 0}

Código[{1000101011010100110010101101101011100000110110001101111001000000111
0000011100100111010101100101011000100110000100101110}]

{{E, j, e, m, p, l, o, , p, r, u, e, b, a, .}}

Aun así como hemos indicado, este pequeño programa codifica cualquier tipo de texto con caracteres como puedes ser:

Mensaje[“La contraseña es: 2492. Introducir en el orden correcto.”]

{1001100011000010010000001100011011011110110111001110100011100100110000101110
0110110010111110000101100001001000000110010 1, 1110011001110100010000000110010001101000011100100110010001011100010000001
0100001110010011001000101110001000000100100101101110011101000111001001101111
01100100, 1110101011000110110100101110010001000000110010101101110001000000110
0101011000010000001101111011100100110010001100101, 1101110001000000110001101
1011110111001001110010011001010110001101110100011011110010110}

Código
[{1001100011000010010000001100011011011110110111001110100011100100110000101110
0110110010111110000101100001001000000110010 1,
1110011001110100010000000110010001101000011100100110010001011100010000001 0
0100101101110011101000111001001101111011001 00,
1110101011000110110100101110010001000000110010101101110001000000110010101 1
0110000100000011011110111001001100100011001 01,
1101110001000000110001101101111011100100111001001100101010110001101110100011
0111100101110}]

{{L, a, , c, o, n, t, r, a, s, e, ñ, a, , e}, {s, :, , 2, 4, 9, 2, ., , I, n, t, r, o, d}, {u, c, i, r, , e, n, , e, l, , o, r, d, e},
{, , , , n, , c, o, r, r, e, c, t, o, .}}

Capítulo 2

El Criptosistema R.S.A.

1. Introducción

Este sistema de clave pública fue diseñado en 1977 por los profesores del MIT (Massachusetts Institute of Technology) Ronald R. Rivest, Adi Shamir y Leonard M. Adleman, de ahí las siglas con las que es conocido. Desde entonces, este algoritmo de cifrado se ha convertido en el prototipo de los de clave pública.

El RSA es un criptosistema de clave pública que sirve tanto para encriptar mensajes como para autenticación de documentos o transacciones (firmas digitales). Este criptosistema basa su seguridad en que no existe una manera rápida y sencilla de factorizar cantidades que son producto de dos números primos grandes.

2. Algorítmo

- Se seleccionan dos números primos largos p y q de forma aleatoria y se calcula el producto de $n = p * q$. A este n se le denomina módulo.
- Definimos la Función φ como

$$\varphi(n) = (p-1) * (q-1) \tag{2.1}$$

- Calculamos un número e que esté dentro del rango $1 \leq e \leq (p-1)(q-1)$ tal que cumpla:

$$m.c.d(e, \varphi(n)) = 1 \tag{2.2}$$

- Se elige un número d, menor que el valor de n usando el algoritmo extendido de Euclides con u y v tales que $e * u + \varphi(n) * v = 1$

$$d = u \bmod \varphi(n) \text{ es el inverso de } e \text{ módulo } \varphi(n) \tag{2.3}$$

- A d y e se les conoce como los exponentes privado y público respectivamente.
- La clave pública la formarán la pareja (n, e).
- La clave privada viene dada por (n, d).
- Con la clave pública procederíamos a encriptar el mensaje deseado y con la clave privada se desencriptaría.

- El factor p y q debe ser guardado en secreto o ser destruido para evitar que cualquiera intente descifrar el mensaje, ya que podría obtener la clave privada.
- Cifrado con RSA:

  Definición 1. *Dado el mensaje m tal que $0 < m \leq n-1$, que queremos enviar a un receptor, llamamos c al código o mensaje cifrado. Dicho mensaje codificado se calcula de la siguiente forma:*

$$C(m) = m^e \ mod\ n. \tag{2.4}$$

- Descifrado RSA:

  Definición 2. *Dado el mensaje cifrado c obtendremos el mensaje inicial m mediante el descifrado con RSA de la siguiente forma:*

$$D(c) = c^d \ mod\ n. \tag{2.5}$$

Una vez visto el algoritmo a seguir en el desarrollo de este sistema de criptografía, veamos que funciona correctamente. Para ello demostraremos el siguiente teorema que la correcta definición del algoritmo en su descifrado:

Teorema 1. *En las condiciones definidas anteriormente, se cumple que:*

$$D(C(m)) = m^{de} \equiv m \ mod\ n. \tag{2.6}$$

Demostración. Recordemos que:

- $e * d \equiv 1 \bmod \phi(n)$
- $m \in \mathbb{Z}_n^*$
- $m^{\phi(n)} \equiv 1 \bmod n$

Entonces se tiene que:

$$D(C(m)) = m^{ed} = m^{1+k\phi(n)} = m(m^{\phi(n)})^k \bmod \ \equiv m \bmod n. \tag{2.7}$$

□

3. Seguridad

La seguridad del criptosistema depende del tamaño de n. Cuanto mayor sea el tamaño del módulo mejor será la seguridad del criptosistema. Pero el tamaño de n influye negativamente en la velocidad de las operaciones del RSA, dado que los cálculos se realizan módulo un número muy grande y los algoritmos que soportan dichas operaciones tienen una complejidad que depende directamente de la longitud del número entero que determina dicho módulo.

Presumiblemente es difícil obtener la clave privada, d, a partir de la clave pública (n, e). Si pudiéramos factorizar n en p y q, podríamos obtenerla. La seguridad de RSA se basa en la idea de que esta factorización es sumamente complicada de realizar.

Debido a que sólo el receptor conoce la clave secreta d, solamente él puede desencriptarlo. Es responsabilidad suya el guardar dicha clave secreta.

4. Diagrama de funcionamiento y pequeño ejemplo

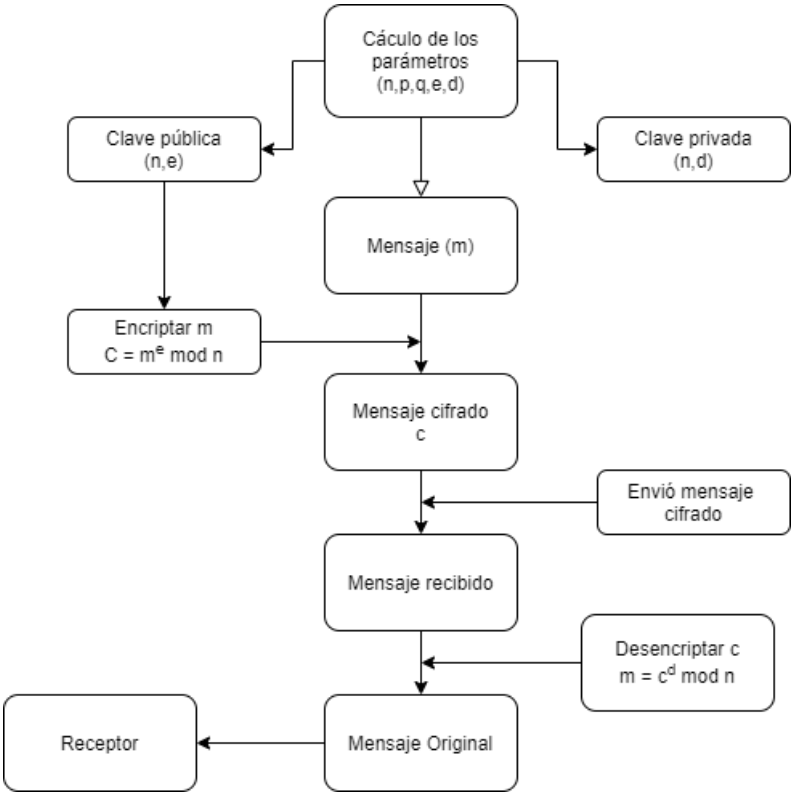
De este modo, un ejemplo de su funcionamiento para un caso simple que nos permite calcularlo sin necesidad de un programa, sería el siguiente:

Ejemplo 2. *Pequeño ejemplo*

1. $p = 11$, $q = 23$ *(elección de dos números primos)*
2. $n = p * q = 253$
3. $\phi(n) = (3-1) * (5-1) = 220$
4. *Sea* $e = 3$ *entonces* $d = 147$ *ya que* $e * d = 3 * 147 = 441$ *mod* $8 = 1$
5. *Clave privada* $(253, 147)$
6. *Clave pública* $(253, 3)$
7. *Tomamos un mensaje fácil, por ejemplo* $m = 24$
8. *El cifrado* $C(m) = C(24) = m^e$ *mod* n; $c = 24^3$ *mod* $253 = 162$
9. *Para descifrar* $D(c) = D(162) = c^d$ *mod* n; $m = 162^{147}$ *mod* $253 = 24$.

Veamos un diagrama de cómo sería el funcionamiento del algoritmo paso a paso siguiendo las etapas de su desarrollo.

Observación 2. *Diagrama RSA*



5. Implementación en Mathematica

Sistema criptográfico de clave pública (RSA.)

Fun1[a_, b_]:=Module[{p, q, n, e, d}, p = 0; q = 0; n = 0; e = 0; d = 0;

p = RandomInteger[{a, b}]; While[PrimeQ[p]==False, p = p + 1];

q = RandomInteger[{a, b}]; While[PrimeQ[q]==False, q = q + 1];

n = p * q;

e = RandomInteger[{2, ((p − 1) * (q − 1))}]; While[PrimeQ[e]==False, e = e + 1];

d = PowerMod[e, −1, ((p − 1) * (q − 1))];

Return[{n, e, d}]]

Esta función genera dos números aleatorios p y q contenidos en el intervalo (a, b), multiplica ambos para obtener n y calcula e y d como hemos explicado en el apartado anterior donde se desarrolla el algoritmo. Así pues, sus elementos de entrada son a y b enteros y sus elementos de salida n, e y d.
Exigiendo que e sea primo nos aseguramos del hecho de que e y $(p-1)*(q-1) = \phi(n)$ tengan máximo común divisor 1, tal y como se establece en la construcción del criptosistema.
Si construimos los primos de la forma $p = 2 * r1 + 1$, $q = 2 * r2 + 1$, con $r1$ y $r2$ primos, entonces, la generación de e puede ser hecha sin más que generar un número aleatorio entre 2 y el menor de $r1$ y $r2$, dado que la factorización de $(p-1)*(q-1)$ en este caso será $2^2 * r1 * r2$ y así, inmediatamente, e y $(p-1)*(q-1)$ no tendrían factores comunes más allá de 1.

CifradoRSA[m_, n_, e_]:=Module[{c}, c = 0;

c = PowerMod[m, e, n];

Return[c]]

Función que cifra el mensaje m según la definición 1.

Si el mensaje que se desea cifrar esta compuesto por mas de un bloque de longitud 120, se utilizaría la siguiente función:

CifradoRSAcadena[lm_, n_, e_]:= Map[CifradoRSA[#, n, e]&, lm]

DescifradoRSA[c_, d_, n_]:=Module[{m}, m = 0;

m = PowerMod[c, d, n];

Return[m]]

Función que descifra el código cifrado c según la definición 2.

Al igual modo si el mensaje cifrado está compuesto por mas de un bloque se utilizaría la siguiente función:

DescifradoRSAcadena[lc_, d_, n_]:=Map[DescifradoRSA[#, *d*, *n*]&, lc]

6. Implementación de un caso real en Mathematica

Veamos ahora su uso en un caso real, en el que los números elegidos son de mayor magnitud. Como hemos visto en el capítulo anterior, el mensaje será un número compuestos por 0 y 1 ya que ha sido convertido a código binario. Así pues utilizando el sistema de encriptación RSA quedaría de la siguiente forma :

Primero para gran seguridad del sistema generamos dos primos aleatorios en el intervalo $(2^{1024}, 2^{1025})$ y obtenemos con los cálculos ya explicados los elementos de salida n, e y d.

Fun1[2^1024, 2^1025]
Salida:

{881726306379559260136202890404820156121728974162609962685769218263658558810193704656228074281730608033089863917907659704974312452954044848261062503155227199845464671706615447080214456795781164579785946986597716270361722005380046895564351005089999507593217407457629210708355957064551097449647746298765789411816384696220823574898165470419232548078091406183694996676764913591717274115685186273658180883036818427828491340374351632488893911762370016720795295263003230574363005697845830576347625324216572490665390745609110508360502418959377385347704422317607037426253697627440218259281301484513711046461973708383809238328833,
468919245301154314947469888559094433890369310687666825018183190855676026737893262077437688634610488223314569330873123654033685925577628823622162413284503081963291508993248604677448462229914083092772879526405304731773145960094727284175312910369734753350808482340406741893027108402344711979461311394819202769546812978766161952534239231969600286800617047857682341957784757004504336192417588873941411301584271159339988552175457831304245387900556277890190577898865874461306779706580324622835718613218849212576379696958229598481582185252279318103335705017980350907959060762608552709173863266209536316036723397450265055172 59,
143125494947110879326192906456501499125569680949331978971411709858894657475171047867958575789031830036821007682543036496668972579336305571756038122161235531030805912140575449163904559710820922170656961911145903995395678560160017403549808256710876174600514833208911826900597197976296699066468348366516217976917652512043957886596157706312465186087265863101296877095121834020534216693104948731254867272682390209256450328486027317544590555553693199938563581270350660616317278679403383795314333319687107127975786122861001874931545436489373823004866285010815267930002642391141570559619426818602537176701077448944431210039 39}

En nuestro caso vamos a cifrar el mensaje "Ejemplo prueba" que una vez pasado el mensaje a binario con el programa introducido en el capítulo anterior queda de la siguiente forma :

- Mensaje["Ejemplo prueba."]
- Salida: 1000101011010100110010101101101011100000110110001101111001000000111 1000001110010011101010110010101100010011000010010111 0

Para ello introducimos el mensaje m y los datos calculados en el paso anterior n y e. Dicha función nos devolverá el código cifrado $C(m) = c$ con la clave pública (n, e).

CifradoRSA[
1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000 0001,

881726306379559260136202890404820156121728974162609962685769218263658558810193704656228074281730608033089863917907659704974312452954044848261062503155227199845464671706615447080214456795781164579785946986597716270361722005380046895564351005089999507593217407457629210708355957064551097449647746298765789411816384696220823574898165470419232548078091406183694996676764913591717274115685186273658180883036818427828491340374351632488893911762370016720795295263003230574363005697845830576347625324216572490665390745609110508360502418959377385347704422317607037426253697627440218259281301484513711046461973708383809238328883,
468919245301154314947469888559094433890369310687666825018183190855676026737893262077437688634610488223314569330873123654033685925577628823622162413284503081963291508993248604677448462229914083092772879526405304731773145960094727284175312910369734753350808482340406741893027108402344711979461311394819202769546812978766161952534239231969600286800617047857682341957784757004504336192417588873941411301584271159339988552175457831304245387900556277890190577898865874461306779706580324622835718613218849212576379696958229598481582185252279318103335705017980350907959060762608552709173863266209536316036723397450265055172593]

Salida:

235566434511543625865383365007392024880395020968048607411659789484365068634456667619427942564636617403595555651004462686613945214397639523184147674784558524852159227883045631334729271508874229186676604803302607037670721812323660379033673346379752569675568678143866306913736382913464178258810730636545339530744494480175545956086120062406556923804425640719908219991021512519778810181456195944659509940936060412559007215517487509448034301609214993574576879376824608494112649944267188048201347615877217120858644014141315501792845398259458451069863809570371383104827746424400327737902561741249734842511531787627440106367477

Este código cifrado es el que enviaremos al receptor que conocerá la clave privada (n, d) y utilizando la función DescifradoRSA y dicha clave privada obtendrá el mensaje descodificado $D(c) = m$.

DescifradoRSA[
235566434511543625865383365007392024880395020968048607411659789484365068634456667619427942564636617403595555651004462686613945214397639523184147674784558524852159227883045631334729271508874229186676604803302607037670721812323660379033673346379752569675568678143866306913736382913464178258810730636545339530744494480175545956086120062406556923804425640719908219991021512519778810181456195944659509940936060412559007215517487509448034301609214993574576879376824608494112649944267188048201347615877217120858644014141315501792845398259458451069863809570371383104827746424400327737902561741249734842511531787627440106367477,
143125494947110879326192906456501499125569680949331978971411709858894657475171047867958575789031830036821007682543036496668972579336305571756038122161235531030805912140575449163904559710820922170656961911145903993595678560160017403549808256710876174600514833208911826900597197976296699066468348366516217976917652512043957886596157706312465186087265863101296877095121834020534216693104948731254867272682390209256450328486027317544590555553693199938563581270350660616317278679403383795314333319687107127975786122861001874931545436489373823004866285010815267930002642391141570559619426818602537176701077448944431210039393,
881726306379559260136202890404820156121728974162609962685769218263658558810193704656228074281730608033089863917907659704974312452954044848261062503155227199845464671706615447080214456795781164579785946986597716270361722005380046895564351005089999507593217407457629210708355957064551097449647746298765789411816384696220823574898165470419232548078091406183694996676764913591717274115685186273658180883036818427828491340374351632488893911762370016720795295263003230574363005697845830576347625324216572490665390745609110508360502418959377385347704422317607037426253697627440218259281301484513711046461973708383809238328883]

Salida:

1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001

Una vez que el receptor obtiene el mensaje descifrado en binario solo quedaría pasarlo a carácteres con la función explicada en el capítulo anterior. De este modo obtendríamos el mensaje inicial:

- Codigo[1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000010010111 0]
- "E", "j", "e", "m", "p", "l", "o", " ", "p", "r", "u", "e", "b", "a", "."

Capítulo 3

El Criptosistema de ElGamal

1. Introducción

El criptosistema de clave pública ElGamal se basa en la función unidireccional exponencial discreta. Este criptosistema ha servido de base para la definición de un algoritmo de firma alternativo al RSA. Fue descrito por Taher Elgamal en 1984 y se usa en software GNU Privacity Guard, versiones recientes de PGP, y otros sistemas criptográficos. Este algoritmo no esta bajo ninguna patente lo que lo hace de libre uso. Su seguridad se basa en la intratabilidad computacional del problema del logaritmo discreto (DLP).

El problema del logaritmo discreto consiste en:

Definición 3. *Consideremos el grupo cíclico $\mathbb{Z}_p^*$ de orden $p-1$, un elemento primitivo $\alpha \in \mathbb{Z}_p^*$ y otro elemento $\beta \in \mathbb{Z}_p^*$. El problema del logaritmo discreto (PLD) es el problema de determinar el entero $1 \leq x \leq p-1$ tal que:*

$$\alpha^x \equiv \beta \ mod\ p. \tag{3.1}$$

El procedimiento de cifrado (y descifrado) está basado en cálculos sobre un grupo cíclico cualquiera G, lo que lleva a que la seguridad del mismo dependa de la dificultad de calcular logaritmos discretos en G.

Definición 4. *Para el problema generalizado del logaritmo discreto tomamos un grupo cíclico finito G con la operación $\circ$ y cardinal n. Consideremos un elemento primitivo $\alpha \in G$ y otro elemento $\beta \in G$. El problema del logaritmo discreto consiste en encontrar el entero x, donde $1 \leq x \leq n$, tal que:*

$$\beta = \alpha \circ \alpha \circ \cdots \circ \alpha = \alpha^x. \tag{3.2}$$

2. ElGamal y RSA: Diferencias.

Las principales diferencias entre los criptosistemas RSA y ElGamal son:

- RSA es un algoritmo determinista, dado un mensaje cualquiera, el resultado de encriptarlo con una misma clave es siempre el mismo. ElGamal es un algoritmo probabilista dado que el uso de una misma clave para encriptar un mensaje, no ofrece siempre el mismo resultado puesto que este depende de un parámetro aleatorio, tal y como puede observarse en la definición de la función de encriptado.

- El algoritmo ElGamal es más lento en el proceso de cifrado y descifrado porque genera más de una clave pública. De hecho el criptosistema de ElGamal se piensa que tiene una fortaleza menor en lo que se refiere a ataques por fuerza bruta.
- El algoritmo de RSA se basa en la dificultad computacional de factorizar grandes primos (recuperar p y q de $n = p*q$), mientras que el algoritmo de ELGamal se basa en la dificultad computacional de resolver el problema logaritmo dicreto (PLD) en grupos cíclicos (recuperar x de $\alpha^x \equiv \beta$). Debido a esto La encriptación RSA es mas rápida que ElGamal pero la desencriptación es mas lenta.
- RSA está completamente estandarizado y prácticamente todas las implementaciones son compatibles entre sí. ElGamal tiene una amplia gama de implementaciones, utilizando diferentes representaciones y grupos algebraicos, la mayoría de los cuales son incompatibles entre sí. Algunos de estos están realmente estandarizados.
- ElGamal puede ser implementado usando curvas elípticas, aumentando su eficiencia y disminuyendo la longitud de las claves. RSA no puede hacerse mas eficiente.

3. Algorítmo

- Se genera un numero primo p cualquiera de gran longitud tal que el logaritmo discreto no es soluble en un tiempo asumible en $\mathbb{Z}_p^*$, es decir, que $p-1$ tenga un factor primo lo suficientemente grande para que el problema logarítmico discreto sea difícil de resolver. Además un generador g del grupo cíclico $\mathbb{Z}_p^*$.

  Sea, además, un generados g tal que para su cálculo, usamos la siguiente definición:

  Teorema 2. *g es generador de $\mathbb{Z}_p^*$ si dada la factorización de $p-1 = q_1^{e_1} \cdots q_r^{e_r}$ se verifica que $g^{\frac{p-1}{q_i}}$ módulo p es distinto de 1, para todo $i = 1, \cdots, r$.*
- Seleccionar un número aleatorio a tal que $1 < a < p-1$ y se calcula $gap = g^a \bmod p$.
- La clave pública será (p, g, gap).
- La clave privada será a, donde $a = \log_a gap \bmod p$ con lo que el criptosistema es seguro siempre que sea difícil hallar el logaritmo discreto.
- Con la clave pública procederíamos a encriptar el mensaje deseado y con la clave privada se desencriptaría.
- Cifrado ElGamal:

  Definición 5. *Dado el mensaje m tal que $0 \leq m \leq p$ que queremos enviar a un receptor y $gap = g^a$ mod p. Si escogemos al azar k tal que $1 < k < p-1$, el mensaje codificado viene dado por la siguiente tupla:*

$$C_k(m) = (h, c) \quad donde, \tag{3.3}$$

$$\begin{aligned} h &= g^k \ mod\ p \\ c &= m * (gap)^k \ mod\ p. \end{aligned}$$

- Descifrado ElGamal:

  Definición 6. *Dada la tupla de cifrado (h, c) y conocida la clave privada a, se calcula $s = h^a \ mod\ p$ y $s^{-a} = s^{-1} \ mod\ p$. El mensaje inicial m se calcula de la siguiente forma:*

$$D(h, c) = s^{-a} * c \ mod\ p. \tag{3.4}$$

Una vez visto el algoritmo a seguir en el desarrollo de este sistema de criptografía, veamos que funciona correctamente. Para ello demostraremos el siguiente teorema que la correcta definición del algoritmo en su descifrado:

Teorema 3. *En las condiciones definidas anteriormente se cumple que:*

$$D(C_k(m)) = s^{-a} * m * (gap)^k \equiv m \ mod\ p \tag{3.5}$$

Demostración. Veamos que se cumple $s^{-a} * c \bmod p = m \bmod p$:

$$\begin{aligned} s^{-a} * c \bmod p &= (g^k)^{-a} * (gap)^k * m \bmod p \\ (g^k)^{-a} * (gap)^k * m \bmod p &= g^{-ak} * (g^a)^k * m \bmod p \\ g^{-ak} * (g^a)^k * m \bmod p &= (g^a)^{-k} * (g^a)^k * m \bmod p \\ (g^a)^{-k} * (g^a)^k * m \bmod p &= m \bmod p. \end{aligned}$$

□

Observación 3. *Hemos definido s^{-a} como el inverso de s^a módulo p. Por el pequeño teorema de Fermat se demuestra que $s^{-a} = s^{p-1-a}$ y lo podemos aplicar.*

Teorema 4. *(Pequeño teorema de Fermat.)*
Si p es un número primo, entonces, para cada número natural a, con $a > 0$, coprimo con p, $a^{p-1} \equiv 1 \ mod\ p$.

4. Seguridad

En las firmas digitales que es muy utilizado, un tercero puede falsificar firmas si encuentra la clave secreta a del firmante. Se considera que este problema son suficientemente difícil. El firmante debe tener cuidado y escoger una clave diferente de forma uniformemente aleatoria para cada firma. Así asegura que clave o aún información parcial sobre la clave no es deducible. Malas selecciones de claves pueden representar fugas de información que facilitan el que un atacante deduzca la clave secreta. En particular, si dos mensajes son enviados con el mismo valor de la clave escogida entonces es factible deducir el valor de la clave secreta. Un adversario con la habilidad de calcular logaritmos discretos podría ser capaz de romper un cifrado ElGamal. Sin embargo, en la actualidad, el algoritmo de computación de logaritmos discretos es subexponencial y por tanto, resulta complicado realizar tal tarea en números grandes en un tiempo razonable.

Este hecho de elegir distintos k en cada cifrado es por el siguiente motivo:

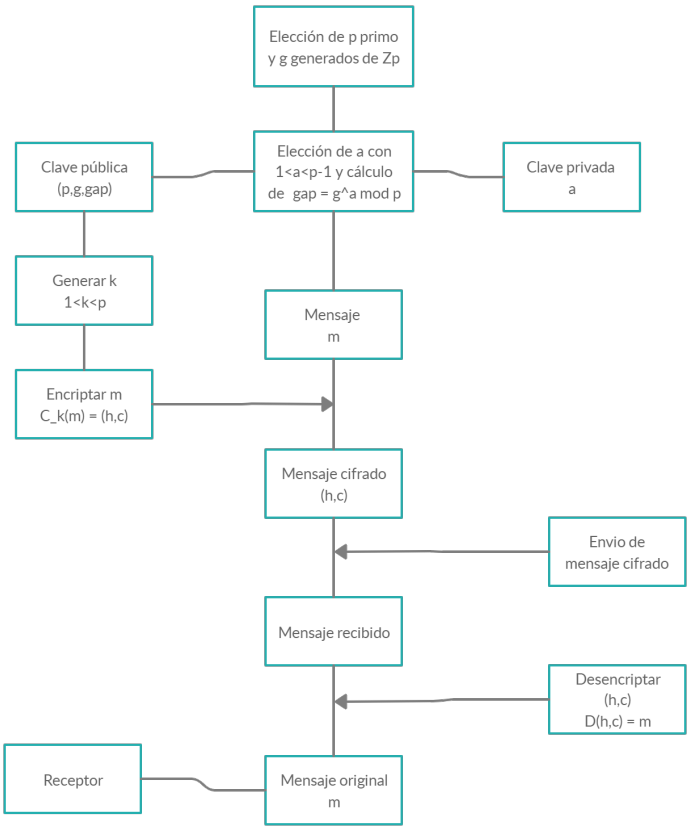
Observación 4. *Supongamos que m_1 se cifra por $(g^k, m_1 * (g^a)^k)$ y un segundo mensaje m_2 se cifra por $(g^k, m_2 * (g^a)^k)$. Si dividimos los segundos miembros tenemos que*

$$m_1 * (g^a)^k * (m_2 * (g^a)^k)^{-1} = m_1 * m_2^{-1} \ mod\ p$$

Con lo que si un atacante que conociese que se han grabado los dos encriptados y conociese el desencriptado de uno de ellos, por ejemplo, el de m_1, simplemente dividiendo los segundos miembros, obteniendo un valor x y después sabiendo que $x = m_1 * m_2^{-1} \ mod\ p$, podría conocer el valor de m_2 sin conocer el valor secreto a.

5. Diagrama de funcionamiento y pequeño ejemplo

Observación 5. *Diagrama ElGamal*



De este modo, un ejemplo de su funcionamiento para un caso simple que nos permite calcularlo sin necesidad de un programa, sería el siguiente:

Ejemplo 3. *Pequeño ejemplo.*

1. *Se elige un primo* $p = 17$ *(suponemos que* $p - 1 = 16$ *tiene un factor primo grande lo cual no es cierto). Además, sea* $g = 3$ *un generador aleatorio del grupo cíclico* $\mathbb{Z}_{17}^*$.

2. *Seleccionamos* $a = 6$ *un número aleatorio tal que* $1 < a < p - 1$ *y calculamos* $gap = g^a \ mod \ p = 3^6 \ mod \ 17 = 15$.

3. *La clave pública será (17,3,15).*

4. *La clave privada será 15.*

5. *Tomamos un mensaje fácil,* $m = 9$ *(el cual está entre* 1 *y* $p - 1$*)*

6. *Generamos* $k = 5$ *aleatorio con* $2 \leq k \leq p - 1$*, entonces el mensaje codificado será* $C_k(m) = C_5(9) = (h, c)$ *donde:*
   $h = g^k \ mod \ p = 3^5 \ mod \ 17 = 5$;
   $c = (gap)^k * m \ mod \ p = 15^5 * 9 \ mod \ 17 = 1$.
   Por lo que $C_k(m) = C_5(9) = (5, 1)$

7. *Para descifrar el código usaremos la clave privada* $a = 6$ *y el Pequeño teorema de Fermat:*
   $D(h, c) = s^{-a} * c \ mod \ p; D(h, c) = s^{p-1-a} * m \ mod \ p.$
   Por tanto, $m = 5^{10} * 1 \ mod \ 17 = 9.$

6. Implementación en Mathematica

Criptosistema de ElGammal

Fun1[a_, b_]:=Module[{p, pp, g}, p = 0; pp = 0; g = 2;

While[PrimeQ[pp] == False, p = RandomInteger[{a, b}]; While[PrimeQ[p]==False, p = p + 1];

pp = 2 * p + 1];

While[g < pp, If[PowerMod[g, p, pp]!=1&&PowerMod[g, 2, pp]!=1, Return[{pp, g}], g++]]]

En esta primera función generamos un primo p aleatorio en el intervalo (a, b), además de calcular un generador g de $\mathbb{Z}_p^*$ de la siguiente forma:

Sea p primo. Sea ahora $p' = 2p + 1$. Si p' es un primo me quedo con él. Si no, empiezo generando un nuevo p.

Despejando $p' - 1 = 2p$.

Así un generador de $\mathbb{Z}_p^*$, g, se calcula comprobando:
$g^{\frac{p'-1}{2}} \equiv'_p g^p \not\equiv_{p'} 1;$
$g^{\frac{p'-1}{p}} \equiv'_p g^2 \not\equiv_{p'} 1.$
Así pues el elemento de entrada será el intervalo (a, b) y el de salida la tupla (p, g).

Fun2[p_, g_]:=Module[{a, gap}, a = 0; gap = 0;

a = RandomInteger[{1, p − 1}];

gap = PowerMod[g, a, p];

Return[{gap, a}]]

En esta segunda función generamos de forma aleatoria la clave privada a y calculamos el número gap. Por tanto el elemento de entrada es la tupla ya calculada (p, g) y el de salida será la pareja formada por la cave pública y la clave privada (gap, a).

CifradoGAMAL[m_, p_, g_, gap_]:=Module[{k, h, c}, k = 0; h = 0; c = 0;

k = RandomInteger[{1, p − 1}];

h = PowerMod[g, k, p];

c = Mod[(m * (gap)^k), p];

Return[{h, c}]]

Función que cifra el mensaje m según la definición 4.

Si el mensaje que se desea cifrar esta compuesto por mas de un bloque de longitud 120, se utilizaría la siguiente función:

CifradoGAMALcadena[lm_, p_, g_, gap_]:=Map[CifradoGAMAL[#, p, g, gap]&, lm]

DescifradoGAMAL[hc_, p_, a_]:=Module[{h, c, s, s1, m}, h = First[hc]; c = Last[hc];

s = 0; s1 = 0; m = 0;

s = PowerMod[h, a, p];

s1 = PowerMod[s, −1, p];

m = Mod[s1 ∗ c, p];

Return[m]]

Función que descifra el código (h, c) según la definición 5.

Al igual modo si el mensaje cifrado está compuesto por mas de un bloque se utilizaría la función siguiente:

DesifradoGAMALcadena[lc_, p_, a_]:=Map[DescifradoGAMAL[#, p, a]&, lc]

7. Implementación de un caso real en Mathematica

Veamos ahora su uso en un caso real, en el que los números elegidos son de mayor magnitud. Como hemos visto en el capítulo anterior, el mensaje será un número compuestos por 0 y 1 ya que ha sido convertido a código binario. Así pues utilizando el sistema de encriptación ElGamal quedaría de la siguiente forma :

Primero para gran seguridad del sistema generamos un primo aleatorio en el intervalo $(2^{1024}, 1^{1025})$ para que el algoritmo discreto no sea soluble en $\mathbb{Z}_p^*$ y obtenemos un generados g.

Fun1[2^1024, 2^1025]
Salida:

{591556778841477265682505215295236344111480268493911454921603863901906735777812739712322319462383169065860
196202777332378112695532544041179139981553711537075393219795218018582128145009396628459566834882659990887 4
700252171819279402552143273432788637806102700071879164780057773677305682615177014925726621793583 9,
11}

En el segundo paso calculamos el valor de *gap* y generamos el número aleatorio a entre 1 y $p-1$ que será la clave privada.

Fun2[
591556778841477265682505215295236344111480268493911454921603863901906735777812739712322231946238316906586019620277733237811269553254404117913998155371153707539321979521801858212814500939662845956683488265999088747002521718192794025521432734327886378061027000718791647800577736773056826151770149257266217935839,
11]
Salida:

{5332786448758297211021893487737147768911986936334266065702289777368627836796721039933111465017602558174371626040963915280451973237808835101530355201272622014205982276128264025918261036388517712027087522708660511905137917803306867836968438774290558712218967309920508777302072368431712474099036563094793433630008,
24777727648496309557429089893634356162109554718701334890576850092512160165773991826069042906900825774467448007123038082767840906252096310981285814555594052187815394741701227150746295069380447186393178830398887543670643970168569033080799812940416989237675334460683748425730279886750188866799012831179577958967}

En nuestro caso vamos a cifrar el mensaje "Ejemplo prueba" que una vez pasado el mensaje a binario con el programa introducido en el capítulo anterior queda de la siguiente forma :

- Mensaje["Ejemplo prueba."]
- Salida: 1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001
  1000001110010011101010110010101100010011000010010111 0

Para ello introducimos el mensaje m y los datos calculados en los pasos anteriores p, g y *gap* que son la clave pública. Dicha función nos devolverá el código cifrado $C_k(m) = (h, c)$ con la clave pública (p, g, gap).

CifradoGAMAL
[1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001,
591556778841477265682505215295236344111480268493911454921603863901906735777812739712322231946238316906586019620277733237811269553254404117913998155371153707539321979521801858212814500939662845956683488265999088747002521718192794025521432734327886378061027000718791647800577736773056826151770149257266217935839,
11,
533278644875829721102189348773714776891198693633426606570228977736862783679672103993311146501760255817437162604096391528045197323780883510153035520127262201420598227612826402591826103638851771202708752270866051190513791780330686783696843877429055871221896730992050877730207236843171247409903656309479343363008]

{6623627461384421173456935081826759409567667339678417180909465910791247644245877277929913140525508972303276901793528426930422268278661980283314877130072931662490150061433052914736460078624708711585374833952086689857697154002803522298600881631043757928129625241072622569400929645114424757609743629592079491367645776607421176464111554143159818982165224911019779797853857103853091838554583013372365155837952038805805884316775961838541712327713287341846358112635599825213455604258683953056041494857533767091694533717444468522059974875201892862841719849550268221846654646518989420299801651829158892338437961812927412539346824954279590698160392007847995973454163560758671447418803380041378426420701468709871754595316330250648059558385}

En nuestro caso vamos a cifrar el mensaje "Ejemplo prueba" que una vez pasado el mensaje a binario con el programa introducido en el capítulo anterior queda de la siguiente forma :

- Mensaje["Ejemplo prueba."]
- Salida: 1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001 1000001110010011101010110010101100010011000010010111 0

Para ello introducimos el mensaje m y los datos calculados en los pasos anteriores p, g y *gap* que son la clave pública. Dicha función nos devolverá el código cifrado $C_k(m) = (h, c)$ con la clave pública (p, g, gap).

CifradoGAMAL
[1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001,
591556778841477265682505215295236344111480268493911454921603863901906735777812739712322231946238316906586019620277733237811269553254404117913998155371153707539321979521801858212814500939662845956683488265999088747002521718192794025521432734327886378061027000718791647800577736773056826151770149257266217935839,
11,
533278644875829721102189348773714776891198693633426606570228977736862783679672103993311146501760255817437162604096391528045197323780883510153035520127262201420598227612826402591826103638851771202708752270866051190513791780330686783696843877429055871221896730992050877730207236843171247409903656309479343363008]

{6623627461384421173456935081826759409567667339678417180909465910791247644245877277929913140525508972303276901793528426930422268278661980283314877130072931662490150061433052914736460078624708711585374833952086689857697154002803522298600881631043757928129625241072622569400929645114424757609743629592079491367645776607421176464111554143159818982165224911019779797853857103853091838554583013372365155837952038805805884316775961838541712327713287341846358112635599825213455604258683953056041494857533767091694533717444468522059974875201892862841719849550268221846654646518989420299801651829158892338437961812927412539346824954279590698160392007847995973454163560758671447418803380041378426420701468709871754595316330250648059558385}

Este código cifrado es el que enviaremos al receptor que conocerá la clave privada a y utilizando la función DescifradoELGAMAL y dicha clave privada obtendrá el mensaje descodificado $D(h, c) = m$.

DescifradoGAMAL[

{662362746138442117345693508182675940956766733967841718090946591079124764424587727792991314052550897230327690179352842693042226827866198028331487713007293166249015006143305291473646007862470871158537483395208668985769715400280352229860088163104375792812962524107262256940092964511442475760974362959207949136764577660742117646411155414315981898216522491101977979785385710385309183855458301337236515583795203880580588431677596183854171232771328734184635811263559982521345560425868395305604149485753376709169453371744446852205997487520189286284171984955026822184665464651898942029980165182915889233843796181292741253934682495427959069816039200784799597345416356075867144741880338004137842642070146870987175459531633025064805955838 5},

5915567788414772656825052152952363441114802684939114549216038639019067357778127397123222319462383169065860196202777332378112695532544041179139981553711537075393219795218018582128145009396628459566834882659990887470025217181927940255214327343278863780610270007187916478005777367730568261517701492572662179358396,

2477772764849630955742908989363435616210955471870133489057685009251216016577399182606904290690082577446744800712320380827678409062520963109812858145555940521878153947417012271507462950693804471863931788303988875436706439701685690330807998129404169892376753344606837484257302798867501888667990128311795779589677]

Salida:

1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001

Una vez que el receptor obtiene el mensaje descifrado en binario solo quedaría pasarlo a carácteres con la función explicada en el primer capítulo. De este modo obtendríamos el mensaje inicial:

- Codigo[1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000001001]
- "E", "j", "e", "m", "p", "l", "o", " ", "p", "r", "u", "e", "b", "a", "."

Capítulo 4

Criptosistema de ElGamal en cuerpos finitos

1. Introducción

El objetivo de este capítulo es volver a desarrollar el sistema de cifrado y descrifrado de ElGamal pero en un nuevo ámbito diferente, en extensiones finitas de cuerpos finitos. Para ello determinaremos la estructura de los cuerpos finitos, probaremos en primer lugar que todo cuerpo finito tiene p^n elementos, donde p es la característica del cuerpo y n cierto natural. Después veremos que para cada natural n y cada primo p existe un único cuerpo (salvo isomorfismos) de p^n . A este cuerpo se le llama cuerpo de Galois de orden p^n y se le representa por $GF(p^n)$ (las letras G y F son las iniciales de "Galois Field").

Definición 7. *Un anillo en el que todo elemento no nulo es invertible se denomina cuerpo. Diremos que un cuerpo es finito si tiene un número finito de elementos.*

Una vez introducida la fundamentación teórica veremos como tratar el mensaje que se quiere enviar al receptor y desarrollaremos el algoritmo de encriptación de ElGamal para grupos finitos y su implementación en Mathematica, además de un ejemplo de un caso real.

2. Estructura de un cuerpo finito

Todo cuerpo finito de característica p (para cierto primo p) tiene orden potencia de p.

Lema 1. *Sea $F \leq E$ una extensión de grado n, con $\#F = q \in \mathbb{N}$. Entonces $\#E = q^n$.*

Demostración. Sea $\alpha_1, ..., \alpha_n$ una base de E como espacio vectorial sobre F. Cada elemento $u \in F$ se escribe de forma única como $u = u_1\alpha_1 + \cdots + u_n\alpha_n$, con $u_1, ..., u_n \in F$. Como cada u_i puede ser uno de los elementos de F, es decir, hay q diferentes $u_1, ..., u_n$, el número total de elementos de E es q^n. □

Lema 2. *Si E es un cuerpo finito, entonces $\#E = p^n$, para cierto $n \in \mathbb{N}$, donde p es la característica de E.*

Demostración. Si E tiene característica p, existe un subcuerpo F de E que es isomorfo a $\mathbb{Z}_p$. Ahora aplicamos el resultado anterior a $F \leq E$, que es una extensión finita de grado, digamos n, y tenemos que $\#E = p^n$. □

Teorema 5. *Un cuerpo finito E de p^n elementos es (salvo isomorfismos) el cuerpo de descomposición del polinomio $x^{p^n} - x \in \mathbb{Z}_p[x]$.*

Demostración. Sea E un cuerpo con p^n elementos, donde p es su característica. El par $(E^*, \cdot)$, formado por los elementos no nulos de E y el producto del cuerpo, es un grupo con $p^n - 1$ elementos. Consideremos $\alpha \in E^*$. Como el orden α en $(E^*, \cdot)$ divide al orden de $(E^*, \cdot)$, que es $p^n - 1$, $\alpha^{p^n-1} = 1$. Si multiplicamos por α esta igualdad tenemos que $\alpha^{p^n} = \alpha$. Si ahora consideramos el polinomio $x^{p^n} - x \in \mathbb{Z}_p[x]$, por lo que acabamos de demostrar, resulta que α es un cero de dicho polinomio. Así, todos los elementos de E son ceros de dicho polinomio que, como tiene grado p^n, tiene a lo sumo p^n ceros en E. Esto nos dice que los elementos de E son los ceros de tal polinomio y el resultado queda probado. □

Definición 8. *Un elemento α perteneciente a un cuerpo es una raíz n-ésima de la unidad si $\alpha^n = 1$. Si $\alpha^m \neq 1$, cualquiera que sea el natural $m < n$, se dice que α es una raíz primitiva n-ésima de la unidad.*

3. Existencia de $GF(p^n)$

Lema 3. *Sea F un cuerpo finito de característica p. Entonces $f(x) = x^{p^n} - x \in F[x]$ tiene p^n ceros distintos en el cuerpo de descomposición K de $f(x)$ sobre F ($F \leq H \leq \bar{F}$).*

Demostración. Veamos que la multiplicidad de cada cero de $f(x)$ en K es 1. Sea $\alpha \neq 0$ un cero de $g(x)$. Como α es no nulo, será también un cero de $g(x) = x^{p^n-1} - 1$, luego $(x - \alpha)$ divide a $g(x)$. Sea

$$h(x) = \frac{g(x)}{(x-\alpha)} = x^{p^n-2} + \alpha x^{p^n-3} + \alpha^2 x^{p^n-4} + \cdots + \alpha^{p^n-3} x + \alpha^{p^n-2}.$$

Observemos que $h(x)$ tiene $p^n - 1$ sumandos, y que cada sumando al ser evaluado en α da $\alpha^{p^n-2} = \frac{\alpha^{p^n-1}}{\alpha} = \frac{1}{\alpha}$, luego $h(\alpha) = [(p^n-1)*1]\alpha^{-1} \neq 0$, lo que concluye la demostración. □

Teorema 6. *Para cada primo p y cada natural n existe un cuerpo que tiene p^n elementos.*

Demostración. Sea $K \leq \bar{\mathbb{Z}}_p$ el cuerpo de descomposición de $f(x) = x^{p^n} - x \in \mathbb{Z}_[x]$ sobre $\mathbb{Z}_p$, y sea F el subconjunto de K formado por todos los ceros (en K) de $f(x)$. Es inmediato probar que F es cerrado para la suma, el opuesto, el producto y el inverso; además 0 y 1 están en F, lo que demuestra que F es un cuerpo, subcuerpo de K, que contiene a $\mathbb{Z}_p$. Como K es el menor subcuerpo de $\bar{\mathbb{Z}}_p$ que contiene a $\mathbb{Z}_p$ y a las raices de $f(x)$, $K = F$. Además, por el resultado anterior, todos los ceros de $f(x)$ tienen multiplicidad 1, luego $\#F = p^n$. □

Definición 9. *Para cada primo p y cada natural n existe un cuerpo que tiene p^n elementos y es el cuerpo de descomposición de $x^{p^n} - x \in \mathbb{Z}_[x]$ (Teorema 6). Llamémosle $GF(p^n)$.*

Además, si E es otro cuerpo con p^n elementos, por el teorema 5, E es el cuerpo de descomposición de $x^{p^n} - x$ sobre $\mathbb{Z}_[x]$, luego $GF(p^n)$ y E son isomorfos, así que $GF(p^n)$ es único salvo isomorfismos. A este cuerpo de orden p^n, único salvo isomorfismos, se le llama el cuerpo de Galois de orden p^n.

Corolario 1. *Sea F un cuerpo finito. Para cada natural n existe un polinomio irreducible $f(x) \in F[x]$ de grado n.*

4. Tratamiento de la información

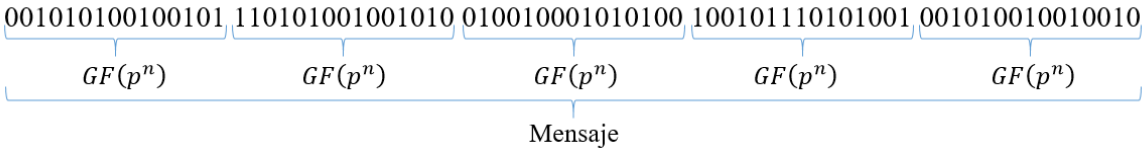
Para el tratamiento de la información usando el criptosistema de ElGamal sobre extensiones de cuerpos hay que hacer algunas modificaciones respecto al capítulo anterior.

En primer lugar vamos a fijar la notación para posteriormente programar eficientemente en Mathematica. Si $GF(p^n)$ con p primo es un cuerpo finito, entonces como hemos indicado en las secciones anteriores, tiene asociado un polinomio primitivo tal que una de sus raíces, s, genera la extensión, es decir, las potencias de s, constituyen todos los elementos no nulos de $GF(p^n)$. Dicho polinomio tiene coeficientes $\mathbb{Z}_p$ y tiene grado n.

Pues bien, $GF(p^n)$ tiene estructura de espacio vectorial generado por la base $B = \{1, s, ..., s^{n-1}\}$. De este modo, cualquier elemento de $GF(p^n)$ se expresa de la forma $x_0 + x_1 * s + ... + x_{n-1}s^{n-1}$.

Como definimos en el capítulo 1, cada mensaje de texto es codificado a binario, es decir, se convierte en una cadena de ceros y unos. Debido a esto es lógica la utilización de $p = 2$ en el grupo finito $GF(p^n)$ ya que la relación entre $\{0, 1\}$ y $\mathbb{Z}_2$ es evidente. Por ello el tratamiento del mensaje se realizará de la siguiente forma, cada elemento del mensaje codificado (01) sera un elemento de $\mathbb{Z}_2$ y n definirá la longitud en bits de cada bloque. Debido a esto habrá que definir una función que nos divida el mensajes en bloques $b_i \in GF(2^n)$ en los que cada elemento del bloque pertenece a $\mathbb{Z}_2$

Ejemplo 4. *Tratamiento de un mensaje.*
Si tomamos un primo $p = 2$ y un natural $n = 15$, tendríamos que cada bloque $b_i \in GF(p^n)$ tendría longitud $k = 15$ bits en el que cada coordenada es un elemento de $\mathbb{Z}_2$.



En el caso de que el mensaje no complete el bloque de longitud n bits, se deberá completar con tantos ceros a la izquierda como sean necesarios para obtener el bloque correspondiente. Esta observación ira reflejada en la implementación en mathematica:

Mensaje[“Ejemplo prueba.”]

{100010101101010011001010110110101110000011011000110111100100000011100000111001001110101011001010110001001100001001011110}

$n = 15$

Función que dado un mensaje te lo divide en bloques de longitud n bits.

bloques[m_, n_]:= Module[{bloque, ultimo},

bloque = Partition[Flatten[IntegerDigits[*m*]], *n*];

ultimo = Last[Partition[Flatten[IntegerDigits[*m*]], UpTo[*n*]]];

While[Length[ultimo]!=*n*, ultimo = Prepend[ultimo, 0]];

Return[Append[bloque, ultimo]]]

mensajeb = bloques[mensajeL, *n*]

$\{\{1,0,0,0,1,0,1,0,1,1,0,1,0,1,0\},\{0,1,1,0,0,1,0,1,0,1,1,0,1,1,0\},$

$\{1,0,1,1,1,0,0,0,0,0,1,1,0,1,1\},\{0,0,0,1,1,0,1,1,1,1,0,0,1,0,0\},$

$\{0,0,0,0,1,1,1,0,0,0,0,0,1,1,1\},\{0,0,1,0,0,1,1,1,0,1,0,1,0,1,1\},$

$\{0,0,1,0,1,0,1,1,0,0,0,1,0,0,1\},\{0,1,0,0,0,0,1,0,0,1,0,1,1,1,0\}\}$

Los elementos de esta lista ya son los coeficientes de los polinomios pertenecientes al grupo finito $GF(p^n)$, a los que si le aplicamos la función "ElementToPolynomial" que explicaremos en la siguiente sección quedarían de la siguiente forma:

Map[ElementToPolynomial[GF[*p*, *n*][#], *x*]&, mensajeb]

$\{1+x^4+x^6+x^8+x^9+x^{11}+x^{13}, x+x^2+x^5+x^7+x^9+x^{10}+x^{12}+x^{13},$

$1+x^2+x^3+x^4+x^{10}+x^{11}+x^{13}+x^{14}, x^3+x^4+x^6+x^7+x^8+x^9+x^{12},$

$x^4+x^5+x^6+x^{12}+x^{13}+x^{14}, x^2+x^5+x^6+x^7+x^9+x^{11}+x^{13}+x^{14},$

$x^2+x^4+x^6+x^7+x^{11}+x^{14}, x+x^6+x^9+x^{11}+x^{12}+x^{13}\}$

Una vez cifrado el mensaje y enviado al receptor siguiendo los pasos del algoritmo de ElGamal explicados en el capítulo anterior con sus respectivas claves públicas y privadas obtendremos en el descifrado polinomios de $GF(p^n)$, de los que habrá que tomar sus coeficientes para obtener el mensaje inicial.

Observación 6. *El tratamiento de todos los elementos del grupo finito $GF(p^n)$ en el programa de Mathematica será a través de sus coeficientes, es decir, por ejemplo un polinomio $p(x) = 1 + x + x^4 + x^6 \in GF(2^6)$ será representado por sus coeficientes de la siguiente forma: $\{1,1,0,0,1,0,1\}$.*

Observación 7. *Elección de $p = 2$.*
Como hemos denotado al principio de la sección hemos tomado en el grupo finito $GF(p^n)$, $p = 2$ ya que al proponer la codificación en binario es lógico que estos elementos (0 y 1) pertenezcan a $\mathbb{Z}_2$. Sin embargo el programa realizado en Mathematica siguiendo el algoritmo de criptografía de ElGamal es genérico para cualquier elección de p primo ya que en todos los resultados demostrados no se ha fijado p.

5. Implementación en Mathematica

Antes de utilizar el algoritmo de ElGamal para la encriptación del mensaje es necesario introducir y definir algunos conceptos previos.

Para poder operar con elementos de un grupo finito dado $GF(p^n)$ utilizaremos la biblioteca "FiniteFields" en la que destacamos las siguientes funciones predefinidas:

<< FiniteFields`

$p = 2$ $n = 6$ GF[p, n]

La función GF indica el cuerpo finito $GF(p^n)$ donde introducimos el primo p y el natural n.

GF[2, 6]

GF$[2, \{1, 0, 0, 0, 0, 1, 1\}]$

La siguiente función nos servirá para calcular un polinomio irreducible de cualquier cuerpo finito que le introduzcamos.

FieldIrreducible[GF[2, 6], x]

$1 + x^5 + x^6$

La función "PowerList" nos dará una lista con los coeficientes de los polinomios que forman el cuerpo introducido. Para el ejemplo que estamos tomando quedaría de la siguiente forma:

PowerList[GF[2, 6]]

$\{\{1, 0, 0, 0, 0, 0\}, \{0, 1, 0, 0, 0, 0\}, \{0, 0, 1, 0, 0, 0\}, \{0, 0, 0, 1, 0, 0\}, \{0, 0, 0, 0, 1, 0\},$

$\{0, 0, 0, 0, 0, 1\}, \{1, 0, 0, 0, 0, 1\}, \{1, 1, 0, 0, 0, 1\}, \{1, 1, 1, 0, 0, 1\}, \{1, 1, 1, 1, 0, 1\},$

$\{1, 1, 1, 1, 1, 1\}, \{1, 1, 1, 1, 1, 0\}, \{0, 1, 1, 1, 1, 1\}, \{1, 0, 1, 1, 1, 0\}, \{0, 1, 0, 1, 1, 1\},$

$\{1, 0, 1, 0, 1, 0\}, \{0, 1, 0, 1, 0, 1\}, \{1, 0, 1, 0, 1, 1\}, \{1, 1, 0, 1, 0, 0\}, \{0, 1, 1, 0, 1, 0\},$

$\{0, 0, 1, 1, 0, 1\}, \{1, 0, 0, 1, 1, 1\}, \{1, 1, 0, 0, 1, 0\}, \{0, 1, 1, 0, 0, 1\}, \{1, 0, 1, 1, 0, 1\},$

$\{1, 1, 0, 1, 1, 1\}, \{1, 1, 1, 0, 1, 0\}, \{0, 1, 1, 1, 0, 1\}, \{1, 0, 1, 1, 1, 1\}, \{1, 1, 0, 1, 1, 0\},$

$\{0, 1, 1, 0, 1, 1\}, \{1, 0, 1, 1, 0, 0\}, \{0, 1, 0, 1, 1, 0\}, \{0, 0, 1, 0, 1, 1\}, \{1, 0, 0, 1, 0, 0\},$

$\{0, 1, 0, 0, 1, 0\}, \{0, 0, 1, 0, 0, 1\}, \{1, 0, 0, 1, 0, 1\}, \{1, 1, 0, 0, 1, 1\}, \{1, 1, 1, 0, 0, 0\},$

$\{0, 1, 1, 1, 0, 0\}, \{0, 0, 1, 1, 1, 0\}, \{0, 0, 0, 1, 1, 1\}, \{1, 0, 0, 0, 1, 0\}, \{0, 1, 0, 0, 0, 1\},$

$\{1, 0, 1, 0, 0, 1\}, \{1, 1, 0, 1, 0, 1\}, \{1, 1, 1, 0, 1, 1\}, \{1, 1, 1, 1, 0, 0\}, \{0, 1, 1, 1, 1, 0\},$

$\{0, 0, 1, 1, 1, 1\}, \{1, 0, 0, 1, 1, 0\}, \{0, 1, 0, 0, 1, 1\}, \{1, 0, 1, 0, 0, 0\}, \{0, 1, 0, 1, 0, 0\},$

$\{0,0,1,0,1,0\},\{0,0,0,1,0,1\},\{1,0,0,0,1,1\},\{1,1,0,0,0,0\},\{0,1,1,0,0,0\},$

$\{0,0,1,1,0,0\},\{0,0,0,1,1,0\},\{0,0,0,0,1,1\}\}$

La función "ReduceElement" como su nombre indica en inglés, dado un cuerpo finito y los coeficientes de cualquier polinomio lo reduce a un elemento perteneciente a dicho cuerpo finito. En nuestro ejemplo, si queremos reducir el polinomio $1+5x^2+x^3+x^4$ a un elemento del cuerpo finito $GF(2^6)$ quedaría de la siguiente forma:

ReduceElement[GF[2, 6][{1, 0, 5, 1, 1}]]

$\{1,0,1,1,1,0\}_2$

Por último, otra función de gran utilidad es "ElementToPolynomial" que dado un cuerpo finito y los coeficientes de un polinomio perteneciente a dicho cuerpo te devuelve el polinomio completo.

ElementToPolynomial[GF[2, 6][{1, 0, 1, 1, 1, 0}], x]

$1+x^2+x^3+x^4$

Además de estas funciones predefinidas por la biblioteca "FiniteFields", será necesario definir algunas más para seguir el algoritmo del criptosistema de ElGamal definido en el capítulo anterior.

Primero vamos a calcular un polinomio primitivo de la extensión de cuerpos $GF(p^n)$ dada, para ello utilizaremos el teorema 5:

- Factorizamos $x^{p^n}-x \in \mathbb{Z}_p[x]$ y nos quedamos con los polinomios de grado n. Estos $p_i[x] \in \mathbb{Z}_p[x]$ serán nuestros candidatos a polinomios primitivos.

- Comprobamos para cada uno de los polinomios seleccionados y nos quedamos con el primero que cumpla el siguiente teorema:

  Teorema 7. *Sea $GF(p^n)$ un grupo finito con p primo y n un natural cualquiera. Sea $K=p^n-1$ el numero de elementos del grupo y $\{k_1,k_2,...,k_i\}$ su factorización en números primos. Si dado un polinomio $p[x] \in \mathbb{Z}_p[x]$ cumple que:*

$$\begin{aligned} x^{\frac{K}{k_1}} &\not\equiv 1 \ mod\ p[x] \\ x^{\frac{K}{k_2}} &\not\equiv 1 \ mod\ p[x] \\ &... \\ x^{\frac{K}{k_i}} &\not\equiv 1 \ mod\ p[x] \end{aligned}$$

  Entonces $p[x] \in \mathbb{Z}_p[x]$ es un polinomio primitivo de $GF(p^n)$.

- Una vez que obtenemos un polinomio que cumple este teorema ya tendremos un polinomio primitivo de $GF(p^n)$, que llamaremos en nuestro programa de Mathematica *pprim*.

De esta forma el programa quedaría de la forma siguiente:

Primero obtenemos la factorización de los $x^{p^n} - x \in \mathbb{Z}_p[x]$ y nos quedamos con los de grado igual a n.

```
FactoresGF[p_, n_]:=Module[{polaux, l, lfact}, polaux = x^(p^n) - x; lfact = {};
l = Map[First, FactorList[polaux, Modulus → p]];
For[i = 1, i < Length[l], i++, If[Exponent[l[[i]], x] == n, lfact = Append[lfact, l[[i]]]]];
Return[lfact]
]

facts = FactoresGF[p, n]
```

$$\{1 + x + x^6, 1 + x^3 + x^6, 1 + x + x^2 + x^4 + x^6, 1 + x + x^3 + x^4 + x^6, 1 + x^5 + x^6,$$
$$1 + x + x^2 + x^5 + x^6, 1 + x^2 + x^3 + x^5 + x^6, 1 + x + x^4 + x^5 + x^6\}$$

```
PolinomioPrimitivo[p_, n_, listfact_]:=Module[{fact, lmod, pprim, i}, i = 2;
fact = Map[First, FactorInteger[p^n - 1]];
lmod = While[MemberQ[Map[PolynomialMod[#, p]&,
        Map[PolynomialMod[x^((p^n - 1)/#), listfact[[i]]]&, fact]], 1], i++];
pprim = listfact[[i]];
Return[pprim]
]

pprim = PolinomioPrimitivo[p, n, facts]
```

$$1 + x + x^3 + x^4 + x^6$$

Una vez calculado el polinomio primitivo, vamos a definir la multiplicación y la potencia de elementos dentro del grupo finito $GF(p^n)$:

Para la multiplicación, realizamos el producto de dos polinomios cualesquiera y al resultado le aplicamos módulo el elemento primitivo.

```
multGF[p_, n_, pprim_, a_, b_]:=PolynomialMod[PolynomialMod[a * b, pprim], p]

a = x^6 + x^4 + x + 1
```

$$1 + x + x^4 + x^6$$

b = x^7 + x^6 + x^3 + x

$x + x^3 + x^6 + x^7$

multGF[p, n, pprim, a, b]

$1 + x + x^2 + x^3$

Para definir la potencia, utilizamos el algoritmo de la potencia rápida para reducir el cálculo de multiplicaciones de un polinomio por si mismo:

Observación 8. *Algoritmo de la potencia rápida:*

El siguiente algoritmo recursivo calcula x^n para un natural n dado.

$$x^n = \begin{cases} x & si\ n = 1 \\ (x^{\frac{n}{2}})^2 & si\ n\ es\ par \\ x * x^{n-1} & si\ n\ es\ impar \end{cases}$$

Comparado con el método original de multiplicar x por sí mismo $n-1$ veces, este algoritmo sólo utiliza $\log n$ multiplicaciones y acelera el cálculo de x^n tremendamente; más o menos de la misma forma que el algoritmo de la multiplicación acelera una multiplicación sobre el método más lento de realizar una suma repetida.

```
potrapida[p_, n_, rp_, a_, b_]:=Module[{ac}, ac = 1;
If[b == 1, Return[a], If[EvenQ[b], ac = multGF[p, n, rp, (potrapida[p, n, rp, a, (b/2)]),
    (potrapida[p, n, rp, a, (b/2)])],
ac = multGF[p, n, rp, a, (potrapida[p, n, rp, a, (b - 1)])]]];
Return[ac]
]
```

potrapida[p, n, pprim, a, 4]

$1 + x^3 + x^5$

Una vez definidas todas las funciones y herramientas que vamos a utilizar, procedemos a la aplicación de el algoritmo de ElGamal para grupos finitos.

Primero definimos la función que genere la clave pública $gap \in GF(p^n)$ y la clave privada a.

- La clave privada a será un número aleatorio entre 1 y y el número de elementos del grupo finito $GF(p^n)$, $p^n - 1$.
- La clave pública $gap \in GF(p^n)$ se calcula de la siguiente forma:

$$gap = x^a \bmod pprim$$

donde x es el generador de $GF(p^n)$ y *pprim* un polinomio primitivo de dicho grupo finito.

```
Fun2[p_, n_, pprim_]:=Module[{a, gappol, gap}, a = 0;
a = RandomInteger[{1, p^n - 1}];
gappol = potrapida[p, n, pprim, x, a];
gap = PolynomialToElement[GF[p, n], gappol];
Return[{gap, a}]]
```

Cifrado ElGamal:

Definición 10. *Dado el mensaje m tal que $m \in GF(p^n)$ que queremos enviar a un receptor y $gap \in GF(p^n)$ la clave pública. Si escogemos al azar k tal que $1 < k < p^n - 1$, el mensaje codificado viene dado por la siguiente tupla:*

$$C_k(m) = (h, c) \in (GF(p^n) \times GF(p^n)) \quad donde,$$

$$\begin{aligned} h &= x^k \ mod \ pprim \\ c &= m * (gap)^k \ mod \ pprim. \end{aligned}$$

```
CifradoGAMALGF[m_, p_, n_, pprim_, gap_]:=Module[{c, h, k},
k = RandomInteger[{1, p^n - 1}];
h = PolynomialToElement[GF[p, n], potrapida[p, n, pprim, x, k]];
c = PolynomialToElement[GF[p, n], multGF[p, n, pprim, potrapida[
    p, n, pprim, ElementToPolynomial[GF[p, n][gap], x], k],
    PolynomialMod[ElementToPolynomial[GF[p, n][m], x], pprim]]];
Return[{h, c}]]
```

Descifrado ElGamal:

Definición 11. *Dada la tupla de cifrado $(h, c) \in (GF(p^n) \times GF(p^n))$ y conocida la clave privada a, se calcula $s = h^a \ mod \ pprim$ y $s^{-1} \in GF(p^n)$ usando el algoritmo extendido de Euclides. El mensaje inicial $m \in GF(p^n)$ se calcula de la siguiente forma:*

$$D(h, c) = s^{-a} * c \ mod \ pprim.$$

```
DescifradoGAMAL[hc_, p_, n_, pprim_, a_]:=Module[{h, c, s, s1, s2, mpol, m},
h = ElementToPolynomial[GF[p, n][First[hc]], x];
c = ElementToPolynomial[GF[p, n][Last[hc]], x];
```

s = potrapida[p, n, pprim, h, a];

s1 = First[Last[PolynomialExtendedGCD[s, pprim, x, Modulus → p]]];

mpol = multGF[p, n, pprim, s1, c];

m = PolynomialToElement[GF[p, n], mpol];

Return[m]]

6. Aplicación en un caso real

En nuestro caso vamos a cifrar el mensaje "Ejemplo prueba" que una vez pasado el mensaje a binario con el programa introducido en el capítulo anterior queda de la siguiente forma :

- Mensaje["Ejemplo prueba."]
- Salida: 1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000010010111 0

mensajeL = {1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000010010111 0}

Vamos a aplicar el criptosistema de ElGamal en el grupo finito $GF(2^{15})$.

<< FiniteFields`

p = 2 n = 15 GF[p, n]

Primero debemos dividir el mensaje en bloques $b_i \in GF(2^{15})$.

mensajeLb = bloques[mensajeL, n]

$\{\{1,0,0,0,1,0,1,0,1,1,0,1,0,1,0\},\{0,1,1,0,0,1,0,1,0,1,1,0,1,1,0\},$

$\{1,0,1,1,1,0,0,0,0,0,1,1,0,1,1\},\{0,0,0,1,1,0,1,1,1,1,0,0,1,0,0\},$

$\{0,0,0,0,1,1,1,0,0,0,0,0,1,1,1\},\{0,0,1,0,0,1,1,1,0,1,0,1,0,1,1\},$

$\{0,0,1,0,1,0,1,1,0,0,0,1,0,0,1\},\{0,1,0,0,0,0,1,0,0,1,0,1,1,1,0\}\}$

Seguidamente, calculamos el polinomio primitivo $pprim$, para ello primero deberemos obtener la factorización de los $x^{p^n} - x \in \mathbb{Z}_p[x]$ de grado igual n.

facts = FactoresGF[p, n]

$\{1+x+x^6, 1+x^3+x^6, 1+x+x^2+x^4+x^6, 1+x+x^3+x^4+x^6, 1+x^5+x^6,$
$1+x+x^2+x^5+x^6, 1+x^2+x^3+x^5+x^6, 1+x+x^4+x^5+x^6\}$

pprim = PolinomioPrimitivo[*p*, *n*, facts]

$1+x+x^3+x^4+x^6$

Obtenemos la clave pública *gap* y la clave privada a.

Fun2[*p*, *n*, pprim]

$\{\{0,1,0,0,0,0,1,1,1,0,1,0,1,1,0\}_2, 10244\}$

gap = {0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0} (polinomio de $GF(p^n)$)

***a* = 10244**

Ciframos cada uno de los elementos (que son polinomios de $GF(p^n)$) del mensaje *mensajeLb* introduciendo la clave pública *gap*.

Map[CifradoGAMALGF[#, *p*, *n*, pprim, gap]&, mensajeLb]

$\{\{\{0,1,1,1,1,0,1,0,0,1,1,1,1,0,1\}_2, \{1,0,0,0,0,0,1,0,0,0,0,1,0,1,1\}_2\},$
$\{\{1,1,0,0,1,0,0,1,0,0,1,1,0,1,1\}_2, \{1,1,1,0,0,0,0,0,1,1,0,1,1,0,0\}_2\},$
$\{\{1,0,0,1,1,0,1,0,0,1,1,1,0,0,1\}_2, \{0,0,1,1,0,0,1,1,1,1,1,1,1,1,0\}_2\},$
$\{\{1,0,1,1,1,1,1,0,1,1,0,1,1,0,1\}_2, \{0,0,1,0,1,1,0,0,1,0,1,0,1,0,1\}_2\},$
$\{\{1,1,1,1,1,0,0,0,0,0,0,1,1,0,0\}_2, \{0,1,1,0,1,0,0,1,1,1,1,1,0,1,1\}_2\},$
$\{\{1,1,0,0,0,0,1,1,0,1,1,1,0,0,1\}_2, \{1,0,0,1,1,0,1,0,1,0,1,0,0,0,0\}_2\},$
$\{\{0,1,1,0,0,0,1,0,1,0,0,0,0,1,1\}_2, \{0,1,1,0,0,0,1,0,0,1,1,0,0,1,0\}_2\},$
$\{\{0,1,0,0,0,0,0,0,0,0,0,1,0,0,1\}_2, \{0,1,1,1,1,0,1,1,0,0,0,0,0,1,1\}_2\}\}$

Así obtenemos el código par enviar al receptor, formado por las tuplas $(h,c) \in (GF(p^n) \times GF(p^n))$.

Codigo = {{{0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1}, {1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1}},
{{1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1}, {1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0}},
{{1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1}, {0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0}},
{{1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1}, {0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1}},
{{1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0}, {0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1}},

{{1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1}, {1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0}},

{{0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1}, {0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0}},

{{0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1}, {0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1}}

Una vez que el receptor obtiene el código y dispone de la clave privada a, utiliza la función "DesdifradoGAMAL" para desencriptar y obtener el mensaje inicial.

Map[DescifradoGAMAL[#, *p*, *n*, pprim, *a*]&, Codigo]

$\{\{1,0,0,0,1,0,1,0,1,1,0,1,0,1,0\}_2, \{0,1,1,0,0,1,0,1,0,1,1,0,1,1,0\}_2,$

$\{1,0,1,1,1,0,0,0,0,0,1,1,0,1,1\}_2, \{0,0,0,1,1,0,1,1,1,1,0,0,1,0,0\}_2,$

$\{0,0,0,0,1,1,1,0,0,0,0,0,1,1,1\}_2, \{0,0,1,0,0,1,1,1,0,1,0,1,0,1,1\}_2,$

$\{0,0,1,0,1,0,1,1,0,0,0,1,0,0,1\}_2, \{0,1,0,0,0,0,1,0,0,1,0,1,1,1,0\}_2\}$

Una vez que el receptor obtiene el mensaje descifrado en binario solo quedaría unirlo como una sola cadena de 0 y 1 y pasarlo a caracteres con la función explicada en el primer capítulo. De este modo obtendríamos el mensaje inicial:

- Codigo[1000101011010100110010101101101011100000110110001101111001000000111000011100100111010101100101011000100110000100101110]
- "E", "j", "e", "m", "p", "l", "o", " ", "p", "r", "u", "e", "b", "a", "."

Capítulo 5

Criptosistema de ElGamal en matrices circulantes

1. Introducción

Al igual que en el capítulo anterior, el objetivo de este es volver a desarrollar el sitema de cifrado y descifrado de ElGamal pero de nuevo en un ámbito diferente, en matrices con coeficientes en un cuerpo finito. Para ello definiremos una matriz cuadrada con elementos pertenecientes a un cuerpo finito, consideraremos simplemente el caso de $\mathbb{Z}_p$, obteniendo así el grupo cíclico que nos dan las potencias de dicha matriz. Probaremos que la matriz elegida tiene inversa mediante la construcción de una matriz circulante y a partir de ella generaremos el grupo cíclico.

Una vez introducida la fundamentación teórica veremos como tratar el mensaje que se quiere enviar al receptor y desarrollaremos el algoritmo de encriptación de ElGamal para matrices con coeficientes en $\mathbb{Z}_p$ y su implementación en Mathematica, además de un ejemplo de un caso real.

2. Grupo cíclico. Matriz circulante

Para generar el grupo cíclico G a partir de las potencias de una matriz cuadrada N, necesitamos comprobar que esta posee inversa, para ello consideraremos lo que se conoce como matrices circulantes.

Definición 12. *Dados $n \in \mathbb{Z}$ y $\{c_1, c_2, \cdots, c_n\}$, la matriz $n \times n$ definida como*

$$C = circ(c_1, c_2, c_3, \cdots, c_n) = \begin{pmatrix} c_1 & c_2 & c_3 & \cdots & c_n \\ c_n & c_1 & c_2 & \cdots & c_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_3 & \cdots & c_n & c_1 & c_2 \\ c_2 & \cdots & c_{n-1} & c_n & c_1 \end{pmatrix} \tag{5.1}$$

es llamada matriz circulante de orden n.

Nótese que los elementos de cada fila de C son idénticos a los de la fila anterior, pero han sido movidos una posición hacia la derecha. Es fácil ver que la suma de matrices circulantes es de nuevo una matriz circulante; lo mismo ocurre cuando se multiplica por un escalar.

Ejemplo 5. *La forma general de una matriz circulante de orden 5 compuesta por los valores de* $a = (a_0, a_1, a_2, a_3, a_4)$ *es la siguiente:*

$$C_5(a) = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & a_4 \\ a_4 & a_0 & a_1 & a_2 & a_3 \\ a_3 & a_4 & a_0 & a_1 & a_2 \\ a_2 & a_3 & a_4 & a_0 & a_1 \\ a_1 & a_2 & a_3 & a_4 & a_0 \end{pmatrix} \tag{5.2}$$

En consecuencia la matriz queda determinada completamente por su primera fila.

En nuestro caso, para formar la matriz circulante H tomaremos los elementos $a_0, a_1, ...,$ $a_{n-1} \in \mathbb{Z}_p$ con p primo. Para formar el grupo cíclico G a partir de la matriz H necesitaremos que esta sea invertible y para ello veremos que su rango es n, es decir, que su rango coincide con su orden. Vamos a utilizar el siguiente teorema:

Teorema 8. *Sea* H *una matriz circulante de grado* n *formada por los coeficientes del polinomio* $f(x) = c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1}$. *El rango de la matriz circulante* H *es* $n - d$, *donde* d *es el grado del polinomio resultante de calcular el máximo común divisor* $M.C.D(f(x), x^{n-1})$.

La demostración de dicho teorema la podemos encontrar en el artículo [4] de la bibliografía.

Así pues, para obtener una matriz circulante H y que sea invertible, generamos n elementos de Z_p, calculamos el polinomio $f(x)$ asociado y calculamos el $M.C.D(f(x), x^{n-1})$. En caso de que el máximo común divisor sea un número (polinomio de grado 0), según el teorema 8, el rango de H será $n - 0 = n$, con lo que ya podremos formar la matriz H invertible.

Una vez que hemos obtenido la matriz circulante H invertible, ya podemos generar el grupo cíclico G a partir de las potencias de dicha matriz H en el que vamos a realizar todas las operaciones necesarias para encriptar y desencriptar un mensaje mediante el sistema de ElGamal.

Atendiendo a [2] que podemos encontrar en las referencias donde se demuestra el siguiente teorema, vamos a realizar una serie de observaciones con respecto a este resultado.

Teorema 9. *Toda matriz circulante es diagonalizable.*

Así pues, sea C una matriz circulante. Como C es diagonalizable por el teorema anterior, entonces $C = PDP^{-1}$, con D diagonal. De este modo $C^n = PD^nP^{-1}$. Nuestro objetivo es calcular $n \in \mathbb{N}$ tal que C^n sea la identidad.

Como consecuencia directa del siguiente corolario:

Corolario 2. $C^n = I$ *si y solo si* $D^n = I$

En consecuencia, calcularemos la matriz D formada por los valores propios, que serán valores de $\mathbb{Z}_p$. Seguidamente, para obtener $n \in \mathbb{N}$ tal que $C^n = I$ utilizaremos la proposición anterior, calcular n tal que $D^n = I$.

Como D^n es elevar cada elemento de la diagonal a n, para calcular n de tal modo que $D^n = I$ nos limitaremos a calcular el mínimo común múltiplo de los órdenes de los elementos de la diagonal en $\mathbb{Z}_p$. Por el teorema de Lagrange, estos órdenes deben ser menores que $p-1$ y en consecuencia el orden de una matriz circulante sobre Z_p sería $p-1$ o un divisor de $p-1$ y, de este modo, tanto la clave privada a, como el factor aleatorio k, pueden ser tomados entre 2 y $p-2$, siempre que uno de los valores propios sea in generador de $\mathbb{Z}_p$.

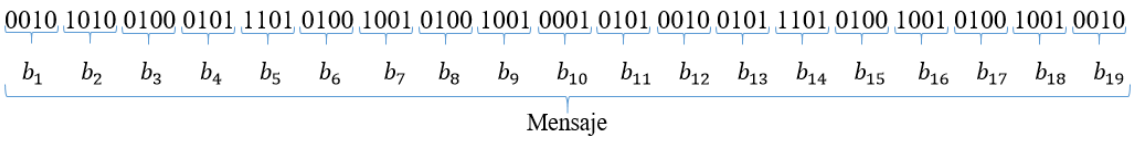
3. Tratamiento de la información

Para el tratamiento de la información usando el criptosistema de ElGamal con matrices con coeficientes en $\mathbb{Z}_p$ hay que realizar algunas modificaciones en cuanto a la división del mensaje.

En primer lugar vamos a fijar la notación, la matriz circulante obtenida H de grado n en el cuerpo finito $\mathbb{Z}_p$ con p primo de longitud k, mediante los coeficientes del polinomio $f(x) = c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1}$ es la que genera el grupo cíclico G. Para poder realizar operaciones entre el mensaje M y esta matriz circulante H, deberemos tratar el mensaje en forma de matriz.

Una vez convertido el mensaje a binario lo dividimos en bloques b_i de longitud $k-1$ para asegurarnos que cada bloque pertenece a $\mathbb{Z}_p$. Cada bloque que será un número decimal compuesto de ceros y unos, será un elemento de la matriz cuadrada M que deberá ser de orden n. Si faltan elementos para completar dicha matriz H, añadiremos un uno en cada lugar, es decir, si la matriz H es de tamaño 7×7 y tenemos solo 30 bloques, los 19 elementos que faltan serán añadidos como unos para formar la matriz completa M.

Ejemplo 6. *Tratamiento de un mensaje.*
Si tomamos un primo $p = 12553$ y un natural $n = 5$, tendríamos que cada bloque $b_i \in \mathbb{Z}_p$ tendría longitud $k = 4$ bits. El mensaje se dividiría en bloques de la siguiente forma:



Por tanto la matriz M del mensaje quedaría de la siguiente forma:

$$M = \begin{pmatrix} b_1 & b_2 & b_3 & b_4 & b_5 \\ b_6 & b_7 & b_8 & b_9 & b_{10} \\ b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{16} & b_{17} & b_{18} & b_{19} & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{5.3}$$

Observación 9. *En la implementación de un caso real, vamos a elegir un primo p sobre el que trabajar de longitud $k = 1024$ bits. Por lo que el mensaje lo dividiremos en bloques de longitud menor de 1024, para asegurarnos que los números que pertenecen a cada unos de los bloques b_i son elementos de $\mathbb{Z}_p$.*

Veamos ahora como implementar este tratamiento del mensaje en Mathematica. Para ello vamos a definir dos funciones: ElemMatrix y Mensaje. La primera función dados el mensaje m, un número p primo y un natural n creará una matriz de la forma explicada anteriormente dividiendo dicho mensaje en los bloques b_i y rellenando con unos para completar la matriz M cuadrada de rango n. La segunda función introduciremos la matriz y nos devolverá el mensaje inicial.

```
ElemMatrix[m_, p_, n_]:=Module[{k, lm, l, ult, M},
k = Length[IntegerDigits[p]] - 1;
lm = Flatten[IntegerDigits[m]];
While[Mod[Length[lm], k]!= 0, lm = Prepend[lm, 0]];
l = Partition[lm, k];
ult = Length[l];
While[Length[l] < (n * n), l = Append[l, 1]];
M = Partition[l, n];
Return[M]]

Mensaje[matrix_, p_, n_]:=Module[{list, l, c},
list = Flatten[List[matrix], 2];
l = {};
For[i = 1, i ≤ Length[list], i++,
If[Length[list[[i]]] > 1, l = AppendTo[l, list[[i]]]]];
c = FromDigits[Flatten[l]];
Return[c]]
```

Veamos un ejemplo de su funcionamiento, si tomamos el mensaje "Ejemplo prueba."

```
mensajeL = {1000101011010100110010101101101011100000110110001101111001000000111
0000011100100111010101100101011000100110000100101110}
```

Vamos a tomar al igual que en el ejemplo 6 $p = 12553$ por lo que $k = 4$ y esta vez $n = 6$ para ver cómo rellena con unos los elementos de la matriz que faltan.

```
codigo = ElemMatrix[mensajeL, 12553, 6]   (MatrixForm)
```

$$\begin{pmatrix} \{0,1,0,0\} & \{0,1,0,1\} & \{0,1,1,0\} & \{1,0,1,0\} & \{0,1,1,0\} & \{0,1,0,1\} \\ \{0,1,1,0\} & \{1,1,0,1\} & \{0,1,1,1\} & \{0,0,0,0\} & \{0,1,1,0\} & \{1,1,0,0\} \\ \{0,1,1,0\} & \{1,1,1,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,1,1\} & \{0,0,0,0\} \\ \{0,1,1,1\} & \{0,0,1,0\} & \{0,1,1,1\} & \{0,1,0,1\} & \{0,1,1,0\} & \{0,1,0,1\} \\ \{0,1,1,0\} & \{0,0,1,0\} & \{0,1,1,0\} & \{0,0,0,1\} & \{0,0,1,0\} & \{1,1,1,0\} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Mensaje[codigo, 12553, 6]

100010101101010011001010110110101110000011011000110111100100000011100000111000

10011101010110010101100010011000010010101110

4. Implementación en Mathematica

Veamos la implementación del algoritmo del ElGamal en Mathematica, pero esta vez usando matrices con coeficientes en $\mathbb{Z}_p$.

Primero será necesario generar un p primo, para ello introducimos el intervalo (a, b) en el que se desea obtener aleatoriamente dicho primo.

primoentre[a_, b_]:=Module[{*p*},

***p* = RandomInteger[{*a*, *b*}]; While[PrimeQ[*p*]==False, *p* = *p* + 1];**

Return[*p*]]

Una vez hemos generado un número p primo y elegido n natural que será el número de coeficientes en $\mathbb{Z}_p$ que marcará el orden de las matrices que utilizaremos, vamos a construir la matriz H utilizando matrices circulantes. Primero obtenemos el polinomio $f(x)$ con coeficientes en $\mathbb{Z}_p$ tal que $M.C.D(f(x), x^{n-1}) = 1$:

Fun1[p_, n_]:=Module[{*H*, coef, fx},

coef = RandomInteger[*p* − 1, *n*];

fx = ElementToPolynomial[GF[*p*, *n*][coef], *x*];

While[Exponent[PolynomialGCD[fx, *x*^*n* − 1], *x*]!=0,

coef = RandomInteger[*p* − 1, *n*];

fx = ElementToPolynomial[GF[*p*, *n*][coef], *x*]];

Return[fx]]

Para una mayor eficiencia en los cálculos, las potencias calculadas al igual que en el

capítulo anterior serán realizadas mediante el método de la potencia rápida:

potrapida[A_, n_]:=Module[{ac}, ac = 1;

If[*n* == 1, Return[*A*],

If[EvenQ[*n*], ac = (potrapida[*A*, (*n*/2)]).(potrapida[*A*, (*n*/2)]),

ac = *A*.(potrapida[*A*, (*n* − 1)])]];

Return[ac]]

Construimos la matriz circulante H a partir de los coeficientes de $f(x)$ para que al aplicar el teorema 8, obtengamos que su rango es n y por tanto asegurarnos que es invertible.

MatrixH[p_, f_]:=Module[{*H*, coef},

coef = CoefficientList[*f*, *x*];

***H* = ToeplitzMatrix[coef];**

Return[*H*]]

Una vez construida la matriz H con la que generamos el grupo cíclico G, procedemos a desarrollar el algoritmo de ElGamal, para ello generamos la clave privada a y la clave pública Ha.

Observación 10. *Como hemos explicado en la sección 2, tanto la elección de la clave privada a y el número aleatorio k en el cifrado, van a ser en el rango* $(2, p-2)$ *y esto es debido a que el orden del grupo G, es decir, el n tal que* $H^n = I$ *con H matriz circulante que genera a G será* $p-1$ *o un divisor de este. Veamos un pequeño ejemplo con una matriz circulante que ilustre este resultado:*

***p* = 12553**

MatrixForm[*H*]

$$\begin{pmatrix} 4145 & 4413 & 3080 & 1279 & 4669 & 6146 \\ 4413 & 4145 & 4413 & 3080 & 1279 & 4669 \\ 3080 & 4413 & 4145 & 4413 & 3080 & 1279 \\ 1279 & 3080 & 4413 & 4145 & 4413 & 3080 \\ 4669 & 1279 & 3080 & 4413 & 4145 & 4413 \\ 6146 & 4669 & 1279 & 3080 & 4413 & 4145 \end{pmatrix}$$

Id = IdentityMatrix[6]

$\{\{1,0,0,0,0,0\},\{0,1,0,0,0,0\},\{0,0,1,0,0,0\},\{0,0,0,1,0,0\},\{0,0,0,0,1,0\},$

$\{0,0,0,0,0,1\}\}$

Solve[Det[$H - x$Id] == 0, x, Modulus → p] $= \{\{x \to 8918\}\}$

FactorInteger[$p-1$] $= \{\{2,3\},\{3,1\},\{523,1\}\}$

Mod[8918^2, p] $= 7469$

Mod[8918^3, p] $= 2324$

Mod[8918^523, p] $= 8725$

Luego el orden de G es $p-1$.

ClavesPyP[p_, H_]:=Module[{a, Ha},

a = RandomInteger[{2, $p-2$}];

Ha = potrapida[H, a];

Return[{a, Ha}]]

A partir de la clave pública Ha y la matriz circulante H, ciframos usando el algoritmo de ElGamal el mensaje M que ha sido reordenado en forma de matriz como hemos explicado en el apartado anterior. La salida de está función obtendremos la tupla $\{Hk, c\}$que será el código cifrado que deberemos enviar al receptor que disponga de la clave privada para descrifrarlo.

Cifrado ElGamal:

Definición 13. *Dado el mensaje M tal que $M \in G(H)$ que queremos enviar a un receptor y $Ha \in G(H)$ la clave pública. Si escogemos al azar k tal que $1 < k < n$, el mensaje codificado viene dado por la siguiente tupla:*

$$C_k(M) = (Hk, c) \in (G(H) \times G(H)) \quad \textit{donde,}$$

$$\begin{aligned} Hk &= H^k \textit{(potencia de matrices)} \\ c &= M * Ha^k. \end{aligned}$$

CifradoELGAMAL[p_, H_, Ha_, M_]:=Module[{k, Hk, Hak, c},

k = RandomInteger[{2, $p-2$}];

Hk = potrapida[H, k];

Hak = potrapida[Ha, k];

$c = M$.Hak;

Return[{Hk, c}]]

Una vez disponemos del código c cifrado y de la clave privada a, aplicamos esta función para obtener el mensaje inicial M aplicando el algoritmo de ElGamal.

Descifrado ElGamal:

Definición 14. *Dada la tupla de cifrado $(Hk, c) \in (G(H) \times G(H))$ y conocida la clave privada a, se calcula $Hka = Hk^a$ y $H^{-ka} \in G(H)$ mediante potencias de matrices. El mensaje inicial $M \in G(H)$ se calcula de la siguiente forma:*

$$D(Hk, c) = c * H^{-ka}.$$

DescifradoELGAMAL[c_, a_]:=Module[{M, inv, Hk, MHak, Hka},

Hk = First[c];

MHak = Last[c];

Hka = potrapida[Hk, a];

inv = Inverse[Hka];

M = MHak.inv;

Return[M]]

5. Aplicación en un caso real

En nuestro caso vamos a cifrar el mensaje "Ejemplo prueba" que una vez pasado el mensaje a binario con el programa introducido en el capítulo anterior queda de la siguiente forma :

- Mensaje["Ejemplo prueba."]
- Salida: 1000101011010100110010101101101011100000110110001101111001000000111000001110010011101010110010101100010011000010010111 0

mensajeL = {1000101011010100110010101101101011100000110110001101111001000000111

0000011100100111010101100101011000100110000100101110}

Vamos a aplicar el criptosistema de ElGamal en matrices con coeficientes en $\mathbb{Z}_p$.

Observación 11. *Gracias a la utilización de matrices, el tratamiento del mensaje nos permite acumular una mayor cantidad de mensaje dentro de una matriz y codificarla toda junta, sin embargo para una mayor estética de este trabajo realizaremos un ejemplo sencillo para no llenar páginas de matrices.*

Vamos tomar:

$\boldsymbol{p = 12553 \quad n = 6}$

Calculamos el polinomio $f(x)$ de $n = 6$ coeficientes en $\mathbb{Z}_p$.

$\mathbf{funx = Fun1[}\boldsymbol{p}\mathbf{,}\boldsymbol{n}\mathbf{]}$

$5305 + 5077x + 2617x^2 + 10739x^3 + 8412x^4 + 4620x^5$

Construimos la matriz circulante H a partir de $f(x)$.

$\boldsymbol{H} = \mathbf{MatrixH[}\boldsymbol{p}\mathbf{, funx]}$

{{5305, 5077, 2617, 10739, 8412, 4620},

{5077, 5305, 5077, 2617, 10739, 8412},

{2617, 5077, 5305, 5077, 2617, 10739},

{10739, 2617, 5077, 5305, 5077, 2617},

{8412, 10739, 2617, 5077, 5305, 5077},

{4620, 8412, 10739, 2617, 5077, 5305}}

Generamos la clave privada a y la clave pública Ha.

$\mathbf{\{}\boldsymbol{a}\mathbf{, Ha\}} = \mathbf{ClavesPyP[}\boldsymbol{p}\mathbf{,}\boldsymbol{H}\mathbf{]}$

{4,

{{281350489835927045, 288413482497120182, 236461989191789221, 238858265312867543,

289151243001879217, 275074068457905268}, {288413482497120182, 301641482817130072,

247644485720174751, 245883700114456731, 299239309703099186, 289151243001879217},

{236461989191789221, 247644485720174751, 206302941462060942, 200660899219805622,

245883700114456731, 238858265312867543}, {238858265312867543, 245883700114456731,

200660899219805622, 206302941462060942, 247644485720174751, 236461989191789221},

{289151243001879217, 299239309703099186, 245883700114456731, 247644485720174751,

301641482817130072, 288413482497120182}, {275074068457905268, 289151243001879217,

238858265312867543, 236461989191789221, 288413482497120182, 281350489835927045}}}

Como hemos dicho el mensaje que queremos enviar en forma binaria es "Ejemplo prueba.", lo pasamos a forma de matriz para poder encriptarlo.

$\boldsymbol{M} = \mathbf{Map[FromDigits, ElemMatrix[mensajeL,}\boldsymbol{p}\mathbf{], \{2\}]}$

{{100, 101, 110, 1010, 110, 101}, {110, 1101, 111, 0, 110, 1100}, {110, 1111, 10, 0, 111, 0},

{111, 10, 111, 101, 110, 101}, {110, 10, 110, 1, 10, 1110}, {1, 1, 1, 1, 1, 1}}

Ciframos el mensaje M introduciendo la clave pública Ha y la matriz H.

c = CifradoELGAMAL[p, H, Ha, M]

{{{5305, 5077, 2617, 10739, 8412, 4620}, {5077, 5305, 5077, 2617, 10739, 8412},

{2617, 5077, 5305, 5077, 2617, 10739}, {10739, 2617, 5077, 5305, 5077, 2617},

{8412, 10739, 2617, 5077, 5305, 5077}, {4620, 8412, 10739, 2617, 5077, 5305}},

{{384111595137350021560, 397011168169593387769, 325191015558936816044,

330282304371296987902, 398616757012261124708, 381954583205214722222},

{709127190945472407533, 742303984940690875813, 611358322948042517433,

606614804364018080313, 738997601282740122917, 716337955419369864190},

{385836340801378982449, 402541178718760487168, 330499962573536269232,

328946346918714311452, 400202551404195914818, 385907657715766405199},

{144075122438102475127, 149473702622381879491, 123061994790169345678,

123205615689529377698, 149703689682775378241, 143962544935057137077},

{368306094201626498273, 386178947861305122811, 318972759626772611102,

315961500901288459222, 385249261725522823631, 374843709676845812371},

{1609309538297488476, 1671973703853860139, 1375812281021154810,

1375812281021154810, 1671973703853860139, 1609309538297488476}}}

Una vez que el receptor obtiene el código y dispone de la clave privada a, utiliza la función "DesdifradoElGAMAL" para desencriptar y obtener el mensaje inicial.

DescifradoELGAMAL[c, a]

{{100, 101, 110, 1010, 110, 101}, {110, 1101, 111, 0, 110, 1100}, {110, 1111, 10, 0, 111, 0},

{111, 10, 111, 101, 110, 101}, {110, 10, 110, 1, 10, 1110}, {1, 1, 1, 1, 1, 1}}

Una vez que el receptor obtiene el mensaje descifrado en binario solo quedaría unirlo como una sola cadena de 0 y 1 y pasarlo a caracteres con la función explicada en el primer capítulo. De este modo obtendríamos el mensaje inicial:

- Codigo[100010101101010011001010110110101110000011011000110111100100000011110000011100100111010101100101011000100110000100101110]
- "E", "j", "e", "m", "p", "l", "o", " ", "p", "r", "u", "e", "b", "a", "."

Observación 12. *Podemos ver que el mensaje descifrado tiene unos al final que quedan como residuo ya que fueron añadidos para completar la matriz M, pero que no nos darán mayor problema debido a que al descifrar en caracteres bastará con quitar la parte final del mensaje que no sirva.*

Capítulo 6

Criptosistema ElGamal sobre curvas elípticas

1. Introducción

La criptografía en curvas elípticas está basada en el algoritmo de clave pública. Posee un nivel de seguridad similar al de el criptosistema RSA o al del logaritmo discreto. Centraremos este capítulo de nuestro trabajo en explicar la cimentación básica del criptosistema de curvas elípticas, tanto la teoría matemática que lo forma como la implementación de su algoritmo para la realización de casos sencillos.

Primero vamos a definir el concepto de curva elíptica y las operaciones dentro del grupo de curvas elípticas. Una vez definidos los elementos del grupo y sus operaciones, cifraremos y descifraremos puntos en dicho grupo generado por una curva elíptica.

2. Curvas elípticas

Empezaremos dando una breve introducción sobre el concepto matemático de las curvas elípticas. La criptografía en curvas elípticas esta basada en la generalización del problema del logaritmo discreto, que para hacerlo computacionalmente difícil de resolver, necesitamos un grupo con ciertas propiedades.

Las curvas elípticas forman unas ecuaciones de polinomios con propiedades especiales para la criptografía, por lo que necesitamos considerar dichs curvas no sobre los números reales sino sobre un grupo finito. La elección más popular es el grupo finito $GF(p)$ donde toda la aritmética es considerada en módulo p primo.

Definición 15. *La curva elíptica sobre $\mathbb{Z}_p$, con p primo, $p > 2$, es el conjunto de pares $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ que cumplen*

$$y^2 \equiv x^3 + a \cdot x + b \ mod\ p \tag{6.1}$$

con el punto imaginario en el infinito θ, donde

$$a, b \in \mathbb{Z}_p$$

y la condición $4 \cdot a^3 + 27 \cdot b^2 \neq 0 \ mod\ p$.

Para uso criptográfico, estamos interesados en estudiar la curva sobre un grupo finito como en la definición. Si trazamos una curva elíptica sobre $\mathbb{Z}_p$, no obtenemos nada remotamente parecido a una curva. Sin embargo, nada nos impide tomar una ecuación de curva elíptica y su trazado sobre el conjunto de números reales para mostrar sencillos ejemplos de propiedades.

Ejemplo 7. *Curva elíptica* $y^2 = x^3 - 3x + 3$ *en* $\mathbb{R}$.

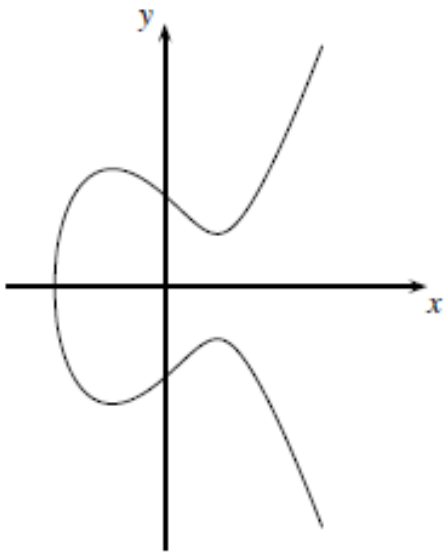


Figura 6.1: $y^2 = x^3 - 3x + 3$ en $\mathbb{R}$

Podemos observar las curvas elípticas son simétricas con respecto al eje x y por tanto para los valores x_i pertenecientes a la curva, $y_i = \sqrt{x_i^3 + a \cdot x_i + b}$ y $-\sqrt{x_i^3 + a \cdot x_i + b}$ son soluciones.

2.1. Grupo de operaciones en curvas elípticas

Vamos a denotar el grupo de operaciones con el símbolo de la suma "+", es decir, dados dos puntos coordenados $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ calculamos un tercer punto R de la siguiente forma:

$$\begin{aligned} P + Q &= R \\ (x_1, y_1) + (x_2, y_2) &= (x_3, y_3) \end{aligned}$$

Sin embargo esta operación esta definida no como la suma arbitraria y usual de dos puntos, sino que tiene otra interpretación geométrica. Vamos a distinguir dos casos: la suma de dos puntos distintos y la suma de un punto consigo mismo.

- **Suma** $P+Q$: Para calcular $R = P+Q$ con $P \neq Q$ trazamos una recta entre P y Q en la que obtenemos un tercer punto en la intersección de dicha recta con la curva elíptica. El punto simétrico a este con respecto al eje x, por definición, el punto R. En la figura 6.2 se muestra la suma de puntos sobre una curva elípticas en $\mathbb{R}$.

- **Suma** $P + P$: En este caso calculamos un punto $R = P + P$ que podemos denotar como $2P$. Trazamos la tangente a P y obtenemos un punto de corte con la curva elíptica. El punto simétrico a dicho a este con respecto al eje x da como resultado el punto R que queríamos calcular, En la figura 6.3 muestra el doble de un punto sobre una curva elíptica en $\mathbb{R}$.

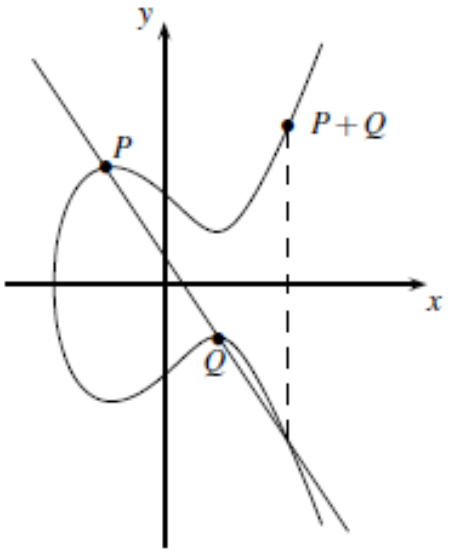


Figura 6.2: Suma de puntos sobre una curva elípticas en $\mathbb{R}$

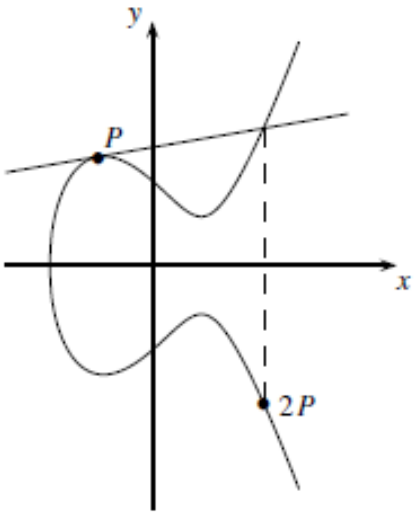


Figura 6.3: Doble de un punto sobre una curva elíptica en $\mathbb{R}$

Sin embargo, en criptografía no podemos realizar estas construcciones geométricas. Por tanto, calcularemos estas coordenadas analíticamente con las siguientes expresiones, ya que pueden ser realizadas en cualquier cuerpo, no solo en el de los números reales. En particular, vamos a tomar las curvas elípticas definidas en el cuerpo finito $GF(p)$.

Definición 16. *Sean $P = (x_1, y_1), Q = (x_2, y_2)$ puntos definidos en la curva elíptica sobre un grupo finito $GF(p)$ con p primo. Definimos $R = (x_3, y_3)$ como la suma de $P + Q$ de la siguiente forma:*

$$\begin{aligned} x_3 &= s^2 - x_1 - x_2 \ mod\ p \\ y_3 &= s(x_1 - x_3) - y_1 \ mod\ p \end{aligned}$$

donde,

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \ mod\ p & si\ P \neq Q\ (Suma\ de\ puntos) \\ \frac{3x_1^2 + a}{2y_1} \ mod\ p & si\ P = Q\ (Punto\ doble) \end{cases}$$

Para establecer completamente nuestro grupo finito debemos definir la identidad o elemento finito, que vendrá representado por θ:

$$P + \theta = P$$

para todos los elementos de la curva. Este punto no es de la forma (x, y) sino que es una definición abstracta, un punto en el infinito. De acuerdo con la definición de grupo, podemos definir el elemento inverso $-P$ de P de la siguiente forma:

$$P + (-P) = \theta$$

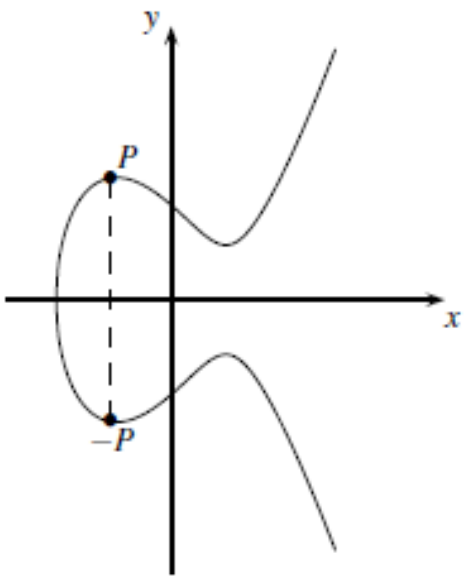


Figura 6.4: Inverso de un punto sobre una curva elíptica en $\mathbb{R}$

De este modo, si $P = (x_1, y_1)$ el punto $-P$ vendrá dado por las siguientes coordenadas $(x_p, -y_p)$ como podemos ver en la figura 6.4. Para el caso de las curvas elípticas en grupos finitos $GF(p)$ vendrá dado por $y_p \equiv p - y_p \bmod p$ es decir,

$$-P = (x_p, p - y_p) \tag{6.2}$$

3. Implementación en Mathematica

Como hemos explicado en la sección anterior, lo primero que vamos a definir es el grupo de operaciones en curvas elípticas. Para ello vamos a construir las funciones "Suma" e "Inverso" basadas en la definición 16 y la ecuación 6.2 respectivamente, además de una potencia rápida para optimizar el cálculo de grandes operaciones.

```
Suma[P_, Q_, p_, a_]:=Module[{x1, x2, y1, y2, s, x3, y3},
If[Inverso[P, p] == Q, Return[Infinity]];
If[P == Infinity, Return[Q]];
If[Q == Infinity, Return[P]];
x1 = First[P];
y1 = Last[P];
x2 = First[Q];
y2 = Last[Q];
s = If[P == Q, Mod[(3 * x1^2 + a) * PowerMod[(2 * y1), -1, p], p],
Mod[(y2 - y1) * PowerMod[(x2 - x1), -1, p], p]];
x3 = Mod[s^2 - x1 - x2, p];
y3 = Mod[s * (x1 - x3) - y1, p];
Return[{x3, y3}]]

Inverso[P_, p_]:=Module[{xp, yp},
xp = First[P];
yp = Last[P];
Return[{xp, p - yp}]]

PotRapida[n_, P_, p_, a_]:=Module[{ac}, ac = 1;
If[n == 1, Return[P],
If[EvenQ[n],
ac = Suma[(PotRapida[(n/2), P, p, a]),
(PotRapida[(n/2), P, p, a]), p, a],
ac = Suma[P, (PotRapida[(n - 1), P, p, a]), p, a]]];
Return[ac]]
```

Veamos su funcionamiento en un sencillo ejemplo:

Ejemplo 8. *Vamos a calcular los puntos de la curva* $E : y^2 \equiv x^3 + 2 * x + 2$ *mod* 17. *El orden del grupo cíclico formado por la curva elíptica es* $\#E = 19$, *por lo que todo elemento es primitivo. Si tomamos como generador* $P = (5, 1) \in E$*:*

$2P = (5,1) + (5,1) = (6,3)$	$8P = (13,7)$	$14P = (9,1)$
$3P = 2P + P = (10,6)$	$9P = (7,6)$	$15P = (3,16)$
$4P = (3,1)$	$10P = (7,11)$	$16P = (10,11)$
$5P = (9,16)$	$11P = (13,10)$	$17P = (6,14)$
$6P = (16,13)$	$12P = (0,11)$	$18P = (5,16)$
$7P = (0,6)$	$13P = (16,4)$	$19P = \theta$

Cuadro 6.1: Puntos de la curva $E : y^2 \equiv x^3 + 2 * x + 2$ mod 17.

En los siguientes puntos, la estructura cíclica se hace visible:

$$\begin{aligned} 20P &= 19P + P = \theta + P = P \\ 21 &= P \\ &\vdots \end{aligned}$$

Veamos ahora algunos de estos mismos cálculos haciendo usos de las funciones definidas:

$\boldsymbol{P = \{5, 1\}}$

$\boldsymbol{-P = Inverso[P, p]} = \{5, 16\}$

$\boldsymbol{2P = Suma[P, P, p, a]} = \{6, 3\}$

$\boldsymbol{4P = Suma[2P, 2P, p, a]} = \{3, 1\}$

$\boldsymbol{8P = Suma[4P, 4P, p, a]} = \{13, 7\}$

$\boldsymbol{16P = Suma[8P, 8P, p, a]} = \{10, 11\}$

$\boldsymbol{18P = Suma[16P, 2P, p, a]} = \{5, 16\}$

$\boldsymbol{19P = Suma[18P, P, p, a]} = \theta$

$\boldsymbol{20P = 19P + P = \theta + P = Suma[19P, P, p, a]} = \{5, 1\}$

$\boldsymbol{18P = PotRapida[18, P, p, a]} = \{5, 16\}$

Una vez definido el grupo de operaciones en curvas elípticas, vamos a desarrollar el algoritmo de criptografía del ElGamal explicado en capítulos anteriores, para puntos del

grupo cíclico generado por una curva elíptica. Para ello necesitaremos los parámetros: p primo, una curva E (con sus respectivos a y b) y un punto G generador de la misma.

En primer lugar, vamos a generar la clave pública y la clave privada. La clave privada a que en este caso llamaremos "*clavea*" para no confundir con la definición de la curva elíptica será un número aleatorio entre 2 y el número de puntos de la curva. La clave pública será aG.

ClavesPyP[n_, G_, p_, a_]:=Module[{clavea, aG},

clavea = RandomInteger[{2, *n*}];

aG = PotRapida[clavea, *G*, *p*, *a*];

Return[{clavea, aG}]]

A partir de la clave pública aG y el generador G, ciframos usando el algoritmo de ElGamal el mensaje M que será un punto perteneciente a la curva elíptica. La salida de está función obtendremos la tupla $\{kG, C\}$que será el código cifrado que deberemos enviar al receptor que disponga de la clave privada para descrifrarlo.

Cifrado ElGamal:

Definición 17. *Dado el mensaje M tal que $M \in E$ que queremos enviar a un receptor y $aG \in E$ la clave pública. Si escogemos al azar k tal que $2 < k < p$, el mensaje codificado viene dado por la siguiente tupla:*

$$C_k(M) = (kG, C) \in (E \times E) \quad \textit{donde,}$$

$$\begin{array}{rcl} & kG & \textit{(Suma k veces el punto G)} \\ C & = & M + k(aG). \end{array}$$

Cifrado[M_, G_, aG_, p_, a_]:=Module[{*k*, kG, *C*},

***k* = RandomInteger[{2, *p*}];**

kG = PotRapida[*k*, *G*, *p*, *a*];

***C* = Suma[*M*, (PotRapida[*k*, aG, *p*, *a*]), *p*, *a*];**

Return[{kG, *C*}]]

Una vez disponemos del código C_k cifrado y de la clave privada *clavea*, aplicamos esta función para obtener el mensaje inicial M aplicando el algoritmo de ElGamal.

Descifrado ElGamal:

Definición 18. *Dada la tupla de cifrado $(kG, C) \in (E \times E)$ y conocida la clave privada clavea, se calcula $S = clavea(kG)$ y $-S \in E$ la función potencia rápida e inverso. El mensaje inicial $M \in E$ se calcula de la siguiente forma:*

$$D(kG, C) = C * -S = (M + kaG) + (-S) = M.$$

Descifrado[Cod_, clavea_, p_, a_]:=Module[{*M*, *S*, invS, kG, *C*},

kG = First[Cod];

***C* = Last[Cod];**

***S* = PotRapida[clavea, kG, *p*, *a*];**

invS = Inverso[*S*, *p*];

***M* = Suma[*C*, invS, *p*, *a*];**

Return[*M*]]

Veamos un pequeño ejemplo de la utilización de este algoritmo: Tomamos como curva elíptica E la del ejemplo anterior y como generador de esta P.

Ejemplo 9. ***ECurv* = *y*^2 == *x*^3 + 2 * *x* + 2**

$y^2 == 2 + 2x + x^3$

***n* = 19**

{*clavea*, *aG*} = *ClavesPyP*[*n*, *P*, *p*, *a*]

$\{5, \{9, 16\}\}$

***M* = {13, 7}**

***Cod* = *Cifrado*[*M*, *P*, *aG*, *p*, *a*]**

$\{\{10, 6\}, \{3, 1\}\}$

***Descifrado*[*Cod*, *clavea*, *p*, *a*]**

$\{13, 7\}$

4. Aplicación en un caso real

La criptografía de curvas elípticas es un sistema criptográfico basado en las matemáticas de las curvas elípticas. Debido a esto, es un sistema mas eficiente y complejo que el del RSA utilizando claves más pequeñas. En la actualidad su uso no se dedica a la encriptación de mensajes como en los ejemplos que hemos visto en capítulos anteriores, sino que se utiliza para cifrar puntos.

En la criptografía es de vital importancia la seguridad en el intercambio de claves, y en este sentido es de gran utilidad dicho criptosistema que estamos tratando. Al cifrar puntos de la curva elíptica, estos pueden ser claves que se desean enviar. Grandes institutos tecnológicos han establecido unos requisitos mínimos de tamaño de clave de 1024 bits para RSA y DSA y de 160 bits para criptografía en curvas elípticas. Habiéndose publicado una lista de curvas elípticas recomendadas de 5 tamaños distintos de claves (80, 112, 160, 192, 256) [5].

Ejemplo 10. *(Representación hexadecimal)*
Domain Parameters for 160-Bit Curves

Curve-ID: brainpoolP160r1
p = *E95E4A5F737059DC60DFC7AD95B3D8139515620F*
a = *340E7BE2A280EB74E2BE61BADA745D97E8F7C300*
b = *1E589A85954234121 34FAA2DBDEC95C8D8675E58*
x = *BED5AF16EA3F6A4F62938C4631EB5AF7BDBCDBC3*
y = *1667CB477A1A8EC338F94741669C976316DA6321*
n = *E95E4A5F737059DC60DF5991D45029409E60FC09*

Donde p primo, n orden del grupo finito, a y b parámetros de la curva y $G = (x, y)$ generador del grupo finito.

Ejemplo 11. *(Representación hexadecimal)*
Domain Parameters for 192-Bit Curves

Curve-ID: brainpoolP192r1
p = *C302F41D932A36CDA7A3463093D18DB78FCE476DE1A86297*
a = *6A91174076B1E0E19C39C031FE8685C1CAE040E5C69A28EF*
b = *469A28EF7C28CCA3DC721D044F4496BCCA7EF4146FBF25C9*
x = *C0A0647EAAB6A48753B033C56CB0F0900A2F5C4853375FD6*
y = *14B690866ABD5BB88B5F4828C1490002E6773FA2FA299B8F*
n = *C302F41D932A36CDA7A3462F9E9E916B5BE8F1029AC4ACC1*
Donde p primo, n orden del grupo finito, a y b parámetros de la curva y $G = (x, y)$ generador del grupo finito.

Observación 13. *Para llevar a cabo un ejemplo real con los parámetros ofrecidos sobre curvas elípticas sería necesario un ordenador de mayor capacidad que de los que se dispone en el hogar, por lo que se deja planteado la implementación del programa para un caso real.*

Bibliografía

[1] ElGamal, T. «A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms». IEEE TRANSACTIONS ON INFORMATION THEORY, Vol 31 (4), pp. 469-472. 1985.

[2] Ingleton, A. W. (1956). The rank of circulant matrices. Journal of the London Mathematical Society, Vol 1(4), pp. 445-460.

[3] Koblitz, N., Elliptic Curve Cryptosystems. Math. Comput., 48, no. 177, pp. 203-209, 1987.

[4] Lara, T. (2001). Circulant matrices. Divulgaciones Matemáticas, Vol 9(1), pp. 85-102.

[5] Lochter, M., and Merkle, J.,Elliptic curve cryptography (ECC) brainpool standard curves and curve generation (pp.2070 - 1721).RFC 5639, March. 2010

[6] Miller, V.S., Use of Elliptic Curves in Cryptography. Advances in Cryptology - CRYPTO '85, LNCS 218, pp. 417-426, 1986.

[7] Paar, C., and Pelzl, J., Understanding cryptography: a textbook for students and practitioners., Springer Science and Business Media, 2009.

[8] Paredes, G. G. (2006). Introducción a la criptografía.

[9] Rivest, R., Shamir, A., Adleman, L., A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, Vol. 21 (2), pp. 120–126. 1978.

[10] Wolfram Language and System Documentation Center. (2020). Wolfram Language and System. https://reference.wolfram.com/language/