Las Matemáticas y las comunicaciones seguras
Algoritmo de Elgamal

Trabajo Fin de Grado

Autor:
María Dolores Gómez Olvera

Tutor:
Juan Antonio López Ramos

Grado en Matemáticas

Septiembre, 2017
Universidad de Almería

Índice general

Abstract in English

Cryptography (from greek, *kryptos*, 'hidden', and *logos*, 'writing') is the science of information security. It is the practice and study of techniques for constructing algorithms, protocols and systems to prevent third parties or the public from reading private messages. In this case, we considerate the study of public key cryptography, which was created to solve the key distribution problem, and communication through insecure channels. We talk more specifically about the ElGamal Cryptosystem.

This work is a compedium of knowledge in areas such as Number Theory, Computational Algebra, and Programming, in which, from some basic notions of Cryptography, we study the ElGamal Cryptosystem, and develope a mobile application that implements this cryptosystem, whose objective is the increase of comunication security through any messaging application.

The structure of this work is: In the first chapter, we talk about the algebraic basis and Group Theory basis we need to understand and prove what comes after this. It is specially interesting the study of finite cyclic groups and Lagrange theorem, because they are fundamental for the cryptosystem. In the second chapter, we go through some important cryptographic concepts, and study the ElGamal cryptosystem. By last, in the third chapter, we introduce the Android Development environment, go through the application structure, and focus on the programation of the algorithm, which is the most important part from the mathematical point of view.

Resumen en español

La Criptografía (del griego *kryptos*, 'escondido', y *logos*, 'escritura') es la ciencia de las comunicaciones seguras. Esta se encarga del estudio de algoritmos, protocolos y sistemas que se utilizan para dotar de seguridad a las comunicaciones, a la información y a las entidades que se comunican. En este caso consideraremos el estudio de la criptografía de clave pública, la cual se creó para solucionar el problema del intercambio de claves y la comunicación a través de canales inseguros, y más concretamente, nos centraremos en el criptosistema de ElGamal.

El presente trabajo es un compendio de conocimientos de teoría de números, álgebra computacional y programación, en el cual, a partir de unas nociones básicas de criptografía, se procede a un estudio del criptosistema que consideramos de interés, y al desarrollo de una aplicación móvil en la cual se implementa el mismo, y cuyo objetivo es aumentar la seguridad de las comunicaciones a través de cualquier aplicación de mensajería.

La estructura del proyecto es la siguiente: En el primer capítulo, asentaremos las bases algebraicas y de teoría de grupos necesarias para justificar todo lo que vendrá a continuación. Es de especial interés el estudio de los grupos cíclicos finitos y el teorema de Lagrange, ya que son fundamentales para el criptosistema. En el segundo capítulo, repasaremos algunos conceptos criptografía importantes, y procederemos al estudio del criptosistema de ElGamal. Por último, en el tercer capítulo, pasaremos a presentar el entorno de programación de Android, y veremos la estructura de la aplicación, resaltando principalmente la parte de programación del algoritmo, ya que es la que tiene más interés desde el punto de vista matemático.

1

Introducción

Desde el principio de la historia, las personas hemos intercambiado mensajes cifrados con la intención de ocultar su información a terceros. El cifrado de mensajes se lleva practicando desde hace más de 4.000 años y, precisamente, el origen de la palabra criptografía lo encontramos en el griego.

En la era de la información, como conocemos a la época en la que vivimos, la protección de la misma es uno de los retos más importantes para la informática, las matemáticas y la telemática. Y es que, aunque el término criptografía nos suele hacer pensar en el mundo del espionaje, los ejércitos y la diplomacia, está muy presente en nuestro día a día. Cuando nos conectamos a servicios como Gmail, estamos estableciendo una comunicación segura y, por tanto, cifrada entre nuestro ordenador (o dispositivo móvil) y los servidores de Google. Y esto ocurre con cualquier tipo de aplicación o página web que requiera información nuestra de carácter sensible.

El correcto cifrado de los datos es una de las obligaciones que impone la normativa española para una inmensa cantidad de empresas, incluída en la Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal (LOPD). Según un informe más reciente, 494/2009, "La seguridad en el intercambio de información de carácter personal en la que hay que adoptar medidas de seguridad de nivel alto, en particular los requisitos de cifrado de datos, no es un tema baladí, ni un mero trámite administrativo, ni una cuestión de comodidad. Es el medio técnico por el cual se garantiza la protección de un derecho fundamental y al que hay que dedicar el tiempo y los recursos que sean necesarios para su correcta implementación".

A la vista está que se trata un tema de vital importancia, y en el cual es necesario invertir muchos recursos en materias de investigación. El primer gran problema que se planteó en la criptografía actual, era el problema del intercambio de clave. Históricamente, se había buscado un canal seguro de intercambio, el cual requería irremediablemente, estar en contacto físicamente con la otra parte, de manera que no fuese accesible para terceros. Sin embargo, los avances en informática hicieron deseable la posibilidad del intercambio de clave a través de un medio tan inseguro como la red. La solución a este problema vino de la mano de Diffie y Hellman en 1976, con el primer protocolo de intercambio de clave a través de un canal inseguro que revolucionó el mundo de la criptografía.

En relación a la idea subyacente a este intercambio de clave, la seguridad proporcionada por el problema del logaritmo discreto, surge el criptosistema de Elgamal, elemento fundamental de este trabajo. Veremos a continuación, por tanto, toda la base matemática necesaria para estudiar este criptosistema, y lo implementaremos en el lenguaje de programación Java, con el objetivo de crear una aplicación móvil para el encriptado de mensajes viable en cualquier aplicación de mensajería.

2 Resultados generales sobre Grupos

En primer lugar, se definen algunos conceptos previos necesarios relacionados con Álgebra. Se trata de unas definiciones básicas y unos resultados que nos ayudarán a entender lo que veremos posteriormente.

2.1 Resultados algebraicos básicos

Veamos en primer lugar la definición de congruencia, y algunas de sus propiedades, que requeriremos más tarde.

Definición 2.1. *Sea m un número entero positivo, decimos que dos enteros a y b son* ***congruentes*** *modulo m, si existe un $k \in \mathbb{Z}$ tal que $a - b = km$. Usaremos la notación $a \equiv b\,(mod\,m)$.*

Proposición 2.1. *La relación de congruencia módulo m en $\mathbb{Z}$ es de equivalencia, y divide a $\mathbb{Z}$ en clases de equivalencia, de manera que dos diferentes de ellas son disjuntas.*

Demostración:

Para comprobar que es una relación de equivalencia, comprobemos que satisface las propiedades:

- Reflexiva. $\forall a \in \mathbb{Z}, a - a = 0 = 0 \cdot m$.
- Simétrica. $\forall a, b \in \mathbb{Z}$, si $a - b = km, b - a = (-k)m$.
- Transitiva. $\forall a, b, c \in \mathbb{Z}$, si tenemos $a - b = km$ y $b - c = hm$,

entonces $a - c = (a - b) + (b - c) = km + hm = (k + h)m$.

Para demostrar que dos clases de quivalencia diferentes son disjuntas, basta probar que si dos de ellas no tienen intersección vacía, son iguales. Sean $[a]$ y $[b]$ dos clases de equivalencia módulo m, tales que $[a] \cap [b] \neq \phi$. Tomando $c \in [a] \cup [b]$,tenemos que $c \equiv a(mod\,m)$ y $c \equiv b(mod\,m)$. Por la definición de congruencia, $c - a = km$ y $c - b = hm$, con k, h enteros. Por tanto, $a - b = (c - b) - (c - a) = (h - k)m$, de donde se deduce que $a \equiv b(mod\,m)$.

■

Proposición 2.2. *Sea m entero positivo, y $a, b, c, d \in \mathbb{Z}$*

a. Si $a \equiv b(mod\,m)$ y $c \equiv d(mod\,m)$, entonces se tiene que $a + c \equiv b + d(mod\,m)$.

b. Si $a \equiv b(mod\,m)$ y $c \equiv d(mod\,m)$, entonces se tiene que $a \cdot c \equiv b \cdot d(mod\,m)$.

Demostración:

Si $a \equiv b(mod\,m)$ y $c \equiv d(mod\,m)$, de la definición de congruencia se deduce que existen números enteros r y s tales que $a - b = rm$ y $c - d = sm$. Por tanto,

$$(a + c) - (b + d) = (a - b) + (c - d) = (r + s)m$$

De aquí se deduce que $a + c \equiv b + d(mod\,m)$.

En las mismas condiciones,

$$ac - bd = ac - bc + bc - bd = (a - b)c + b(c - d) = rdm + sbm = (rd + sb)m$$

De aquí se deduce que $a \cdot c \equiv b \cdot d(mod\,m)$.

■

Definición 2.2. *El conjunto de las clases de equivalencia en $\mathbb{Z}$ relativas a la congruencia módulo m se representa mediante $\mathbb{Z}_m$.*

A continuación, se definen algunos resultados de divisibilidad que también nos resultarán útiles. El siguiente algoritmo es consecuencia del hecho de que todo conjunto de números enteros acotado superiormente tiene máximo.

Algoritmo 2.1. ***(Algoritmo de la división)*** *Sean dos números enteros a y b, con $b \neq 0$. Entonces existen dos números enteros c y r tal quea $= cb + r$, con $0 \leq r \leq |b|$.*

Lema 2.1. *Sea a y b números enteros no nulos, y p un número primo tal que $p \mid ab$. Entonces, o bien $p \mid a$, o bien $p \mid b$.*

Demostración:

Sean $a, b \in \mathbb{Z}$, tales que $p|ab$. Entonces podemos escribir $ab = pr$, para algún entero r. Supongamos que p no es divisor de a; probaremos que $p|b$. Al ser p primo, como p no es divisor de $a, mcd(a, p) = 1$. Por tanto, como consecuencia del conocido Teorema de Bezout, podemos encontrar dos enteros u y v tales que:

$$1 = ua + vp$$

Entonces tenemos $b = b \cdot 1 = b(ua + vp) = uab + vbp = upr + vpb = (ur + vb)p$. Por tanto, $p|b$.

■

Y como último apunte de esta sección, veamos el Pequeño Teorema de Fermat, resultado imprescindible en este ámbito.

Teorema 2.1. ***(Pequeño teorema de Fermat).*** *Sea p un número primo, y a un número natural, tal que p no divide a a. Entonces $a^{p-1} \equiv 1(mod p)$.*

Demostración:

Puesto que p no divide a a y p es primo, se tiene que $(a, p) = 1$, y por tanto el conjunto

$$\{0, a \cdot 1, a \cdot 2, ..., a \cdot (p-1)\}$$

es un sistema completo de restos módulo p. Por tanto, para cada i, tal que $1 \leq i \leq p-1$, i es congruente con algún $j \cdot a$, $1 \leq j \leq p-1$. Así pues,

$$1 \cdot 2 \cdot ... \cdot (p-1) \equiv a(a \cdot 2)...a(p-1)(mod p)$$

esto es,

$$p|a(a \cdot 2)...a(p-1) = (a^{p-1} - 1) \cdot 1 \cdot 2 \cdot ... \cdot (p-1)$$

Como p no divide a $1 \cdot 2...(p-1)$, p divide a a^{p-1}, y por tanto, $a^{p-1} - 1 \equiv 0(mod p)$, que era lo que queríamos demostrar.

■

2.2 Resultados de Teoría de grupos

A continuación, se definen algunos conceptos previos necesarios relacionados con Teoría de Grupos. Se define qué es un grupo abeliano, un subgrupo, y varios conceptos necesarios para ver posteriormente el teorema de Lagrange, imprescindible en el algoritmo que vamos a tratar.

Definición 2.3. *Un **grupo** es un par $(G, *)$, donde G es un conjunto no vacío, y $*$ es una ley de composición interna definida en G,*

$$\begin{aligned} G \times G &\longrightarrow G \\ (x, y) &\longmapsto x * y \end{aligned}$$

tal que verifica las siguientes propiedades:

(1) Asociativa: $(x * y) * z = x * (y * z), \quad \forall x, y, z \in G.$

(2) Elemento neutro: Existe $e \in G$ tal que $x * e = e * x = x, \quad \forall x \in G.$

(3) Elemento simétrico: Para cada $x \in G$, existe $x^{-1} \in G$ tal que $x * x^{-1} = x^{-1} * x = e.$

Cuando no exista riesgo de confusión con la operación interna, diremos simplemente que G es un grupo, y escribiremos xy en lugar de $x * y$.

Definición 2.4. *Si en un grupo G se verifica la propiedad conmutativa, es decir, que $xy = yx$ para todo $x, y \in G$, diremos que G es un grupo conmutativo o **abeliano**.*

Algunos ejemplos de grupos abelianos y no abelianos son:

Ejemplo 2.1. *(Grupos abelianos) Algunos ejemplos son: $(\mathbb{Z}, +)$ y $(\mathbb{R}, +)$. En estos casos el elemento neutro es el cero, y el simétrico es el opuesto del número dado.*

Sean $\mathbb{Q}^* = \mathbb{Q}\backslash\{0\}$, y $\mathbb{R}^* = \mathbb{R}\backslash\{0\}$, entonces se verifica que $(\mathbb{Q}^*, \cdot)$ y $(\mathbb{R}^*, \cdot)$ son abelianos, donde $\cdot$ representa el producto usual. El elemento neutro es el número 1 y el simétrico es el inverso del número dado.

Ejemplo 2.2. *(Grupos no abelianos) Por otro lado, tenemos como ejemplo de grupos no abelianos el conjunto de las matrices regulares, $\mathcal{M}_n$. Podemos ver que, en el caso n=2:*

$$A \cdot B = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} a_1 b_1 + a_2 b_3 & a_1 b_2 + a_2 b_4 \\ a_3 b_1 + a_4 b_3 & a_3 b_2 + a_4 + b_4 \end{pmatrix}$$

$$B \cdot A = \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} \cdot \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} = \begin{pmatrix} a_1 b_1 + a_3 b_2 & a_2 b_1 + a_4 b_2 \\ a_1 b_3 + a_4 b_3 & a_2 b_3 + a_4 + b_4 \end{pmatrix}$$

De forma que $A \cdot B \neq B \cdot A$.

Otro ejemplo de grupo no abeliano seria el de los cuaternios. Estos son una extensión de los números reales, añadiendo las unidades imaginarias i, j y k a los números reales, tal que $i^2 = j^2 = k^2 = ijk = -1$.

El producto en este caso se realiza de la siguiente forma. Sean dos cuaternios $a = a_1 + a_2 i + a_3 j + a_4 k$, y $b = b_1 + b_2 i + b_3 j + b_4 k$.

$a \cdot b = (a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4) + (a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3)i + (a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2)j +$
$+ (a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1)k$

$b \cdot a = (a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4) + (a_1b_2 + a_2b_1 - a_3b_4 + a_4b_3)i + (a_1b_3 + a_2b_4 + a_3b_1 - a_4b_2)j +$
$+ (a_1b_4 - a_2b_3 + a_3b_2 + a_4b_1)k$
Con lo que podemos observar que $a \cdot b \neq b \cdot a$, el producto no es conmutativo.

Definición 2.5. *Dados (G,∗) un grupo, y H un subconjunto no vacío de G, diremos que H es un* ***subgrupo*** *de G si H es un grupo respecto de la misma operación que dota a G de estructura de grupo.*

Proposición 2.3. *Dados (G, ∗) un grupo y H un subconjunto no vacío de G, entonces H es un subgrupo de G si y solo si:*
(1) Para cualesquiera $x, y \in H$ se tiene $xy \in H$.
(2) $1 \in H$.
(3) Para todo $x \in H$ se tiene $x^{-1} \in H$.

Demostración:

⇐: Es inmediato, ya que basta observar que la condición (1) nos dice que la operación ∗ es interna en H, la condición (2) afirma que 1 es el elemento neutro de H, y la condición (3) dice que todo elemento de H tiene inverso perteneciente también a H. Por tanto, solo nos falta ver que se verifica la propiedad asociativa. Pero si $x, y, z \in H$, entonces $x, y, z \in G$, y por tanto,

$$x(yz) = (xy)z$$

⇒: Si H es un subgrupo de $(G, *)$, tenemos que $(H, *)$ es un grupo, y por tanto se verifica:

(1) La operación ∗ es interna en H, y asi para cualquier par de elementos $x, y \in H$ se tiene que $x * y \in H$.

(2) Sea 1_H el elemento neutro en H, por tanto para todo $x \in H$ se tiene

$$x1_H = 1_H x = x.$$

Pero, por otra parte, como $x \in G$, también se tiene

$$x1 = 1x = x$$

Por tanto, para todo $x \in H$ se verifica $x1_H = x1$, y de ahí que $1_H = 1$.

(3) Sea $x' \in H$ el simétrico de H, entonces se tiene que $xx' = x'x = 1$. Por tanto, x' es el simétrico de x en G, y así, por la unicidad del elemento simétrico, $x^{-1} = x' \in H$.

■

Como ejemplo, podemos ver que los subgrupos del grupo $(\mathbb{Z}, +)$ son los conjuntos de la forma

$$m\mathbb{Z} = \{mx : x \in \mathbb{Z}\}$$

para cada entero m no negativo. Dado que $\mathbb{Z}$ es un grupo cíclico, definiremos qué significa esto primero, y procederemos a demostrar que sus subgrupos son de esta forma. Pero antes, definamos el concepto de orden.

Definición 2.6. *Sea G un grupo. Al cardinal de un subgrupo H de G se llama* ***orden*** *de H, y lo denotamos por |H|. En particular, al número de elementos de G se llama orden de G. Un grupo es finito cuando $|G| < \infty$. En caso contrario, se dice que el grupo G es infinito.*

Tanto $(\mathbb{Z}, +)$ como sus subgrupos no triviales son grupos infinitos.

Definición 2.7. *Sea G un grupo, y a un elemento de G. Si el subgrupo <a> es finito, llamamos* ***orden*** *de a, y lo denotamos por ord(a), al orden del subgrupo <a>.*

Grupos cíclicos

Definición 2.8. *Diremos que un grupo G es* ***cíclico*** *si existe un elemento $g \in G$ tal que el* ***subgrupo generado*** *por g es G, es decir <g>, G = <g>. En este caso g se denomina* ***generador*** *de G.*

El grupo $(\mathbb{Z}, +)$ de los números enteros es un grupo cíclico, ya que

$$\mathbb{Z} =< 1 >=< -1 > .$$

En relación a los grupos cíclicos tenemos también el siguiente resultado.

Proposición 2.4. *Todo subgrupo de un grupo cíclico es cíclico.*

Demostración:

Sea $G =< g >$ un grupo cíclico con a un generador, y sea $H \leq G$. Es claro que si H es {1} o G, entonces es cíclico. Supongamos que esto no es así. Sea $g^k \in H$ tal que $g^m \notin H$ para $m < k$, y sea g^s otro elemento de H. Utilizando el algoritmo de división de Euclides (2.1), podemos escribir $s = ck + r$, con $o \leq r < k$. Como H es un subgrupo, $(g^k)^{-1} = g^{-k} \in H$. Por tanto,

$$g^s(g^{-k})^c = g^{s-ck} = g^r \in H$$

en contra de la definición de k, a menos que $r = 0$.

Así, cada elemento de H es de la forma $(g^k)^n$ para algún $n \in \mathbb{Z}$, y H es cíclico generado por g^k.

■

Esto nos resultara útil ahora para ver que:

Proposición 2.5. *Todo subgrupo H de $(\mathbb{Z}, +)$, es de la forma $H = m\mathbb{Z}$, para algún entero no negativo m.*

Demostración:

Sea $H \leq \mathbb{Z}$. Por la proposición 2.4, existe $n \in \mathbb{Z}$, tal que

$$H =< m >= \{mx : x \in \mathbb{Z}\} = m\mathbb{Z}$$

■

Con lo cual queda probado que todo subgrupo de $\mathbb{Z}$ es de la forma $H = m\mathbb{Z}$.

El grupo $\mathbb{Z}_p^*$

En esta sección veremos el grupo $\mathbb{Z}_p^*$, necesario para el algoritmo criptográfico que pretendemos estudiar en este trabajo, y definiremos lo necesario para encontrar un generador de este grupo, y utilizarlo posteriormente.

El siguiente resultado nos muestra cómo definir una suma y un producto en $\mathbb{Z}_m$ y enuncia algunas de sus propiedades.

Teorema 2.2. *(a) Si m es un entero positivo, y $[a],[b] \in \mathbb{Z}_m$, se pueden definir las operaciones suma y multiplicación en $\mathbb{Z}_m$ mediante*

$$[a]+[b]=[a+b] \qquad [a][b]=[ab]$$

(b) Ambas operaciones tienen las propiedades asociativa y conmutativa, y se relacionan mediante la propiedad distributiva. La clase $[0]$ es el elemento neutro para la suma, y la clase $[1]$ lo es para el producto.

(c) Todo elemento $[a] \in \mathbb{Z}_m$ tiene su opuesto respecto a la suma, $[m-a]$.

Demostración:

En primer lugar, probamos que las operaciones están bien definidas. Esto lo asegura la proposición 2.2, ya que si $[a]=[a']$ y $[b]=[b']$, $a'+b' \equiv a+b(mod\, m)$, y por tanto, $[a']+[b']=[a]+[b]$. De manera similar, tenemos que $a'b' \equiv ab(mod\, m)$, y por tanto, $[a'b']=[ab]$.

Las propiedades asociativa, conmutativa, y de existencia de elemento neutro también se siguen de dicha proposición y de las propiedades verificadas por los números enteros.

Por otro lado, para demostrar la última parte, basta observar que

$$[a]+[m-a]=[a+m-a]=[m]=[0]$$

Con lo cual $[m-a]=[-a]$ es el opuesto de $[a]$ con respecto a la suma.

■

Con la operación de suma, $(\mathbb{Z}_m,+)$ es un grupo abeliano. Sin embargo, con la operación de multiplicación $(\mathbb{Z}_m,\cdot)$ no es un grupo, basta observar que el elemento $[0]$ no tiene inverso. Definimos, por tanto:

Definición 2.9. *El conjunto de clases de equivalencia $\mathbb{Z}_m^* = \mathbb{Z}_m \backslash 0$.*

Sin embargo, en general tampoco se trata de un grupo. Podemos verlo, por ejemplo, en el caso de $(\mathbb{Z}_6^*,\cdot)$, en el cual $[2]$, $[3]$ y $[4]$ no poseen inverso. En general, si $m=r\cdot s$, con $s,r \in \mathbb{Z}$, $s,r>1$, se tiene que $[s]\cdot[r]=[m]=[0]$ en $\mathbb{Z}_m$; esto sugiere que $(\mathbb{Z}_m^*,\cdot)$ sera un grupo solamente cuando m sea primo.

Proposición 2.6. *Para todo número primo positivo p, $\mathbb{Z}_p^*$ es un grupo abeliano con $p-1$ elementos.*

Demostración:

Ya se ha visto en el teorema 2.2 que la operación $\cdot$ está bien definida en $\mathbb{Z}_p$, y en particular, en $\mathbb{Z}_p^*$. Para demostrar que es cerrada, es suficiente probar que dados $[a],[b] \in \mathbb{Z}_p^*$, el caso $[a]\cdot[b]=[0]$ es imposible.

En efecto, si suponemos que $[a]\cdot[b]=[0]$, deducimos que $ab=kp$ para algun $k\in\mathbb{Z}$. Entonces p divide a $a\cdot b$, y como p es primo, el lema 2.1 nos permite deducir que $p|a$ o $p|b$. Se tendría entonces que $[a]=[0]$ o que $[b]=[0]$, en contra de la hipótesis de que $[a],[b]\in\mathbb{Z}_p^*$. La clase residual $[1]$ es el elemento neutro. El resto de las afirmaciones de la proposición son ostensibles, siendo quizás la existencia de inverso la menos evidente de ellas. Para probar que cualquier elemento del conjunto tiene un elemento inverso, sea $[a]\in\mathbb{Z}_p^*$, observamos que el maximo comun divisor de a y p es 1; por el Teorema de Bezout, existen $m,n\in\mathbb{Z}$ tales que $1=ma+np$; de aquí se deduce que

$$[1]=[m][a]+[n][p]=[m][a]$$

y por tanto m es el inverso de a en $(\mathbb{Z}_p^*,\cdot)$, puesto que $(\mathbb{Z}_p^*,\cdot)$ tiene la propiedad conmutativa.

■

Teorema de Lagrange

En esta sección, veremos en el caso de grupos finitos, la relación entre el orden de un subgrupo y el del grupo que lo contiene, a través del teorema de Lagrange. Para entender el teorema y su demostración, necesitamos algunos conceptos, que definimos a continuación.

Definición 2.10. *Sean G un grupo, y $H\subseteq G$ un subgrupo. Definimos en G las siguientes* ***relaciones binarias****:*

$\forall x,y\in G \quad x\mathcal{R}_H y \iff xy^{-1}\in H.$

$\forall x,y\in G \quad x\mathcal{R}^H y \iff x^{-1}y\in H.$

Lema 2.2. *Con la notación anterior, las relaciones binarias $\mathcal{R}_H$ y $\mathcal{R}^H$, son relaciones de equivalencia sobre G.*

Demostración:

Veamos que $\mathcal{R}_H$ es una relación de equivalencia sobre G. Para ello tiene que verificar los axiomas:

- Reflexiva. Para todo $x\in G$, se tiene que $xx^{-1}=1\in H$, por tanto $x\mathcal{R}_H x$.
- Simétrica Sean $x,y\in G$, tales que $x\mathcal{R}_H y$, entonces $xy^{-1}\in H$. Pero al ser H un subgrupo, se tiene que $(xy^{-1})^{-1}\in H$. Por tanto, y $x^{-1}\in H$, es decir, $y\mathcal{R}_H x$
- Transitiva. Sean $x,y,z\in G$, tales que $x\mathcal{R}_H y$ e $y\mathcal{R}_H z$. Entonces se tiene $xy^{-1}\in H$. e $yz^{-1}\in H$. Por tanto, al ser H un subgrupo, $xz^{-1}=(xy^{-1})(yz^{-1})\in H$, y de ahi que $x\mathcal{R}_H z$.

Para la relacion $\mathcal{R}^H$ el procedimiento es análogo.

■

Notación. Dado $a\in G$, denotemos por $[a]_H$ y $[a]^H$ respectivamente las clases de equivalencia que las relaciones $\mathcal{R}_H$ y $\mathcal{R}^H$ determinan en G. Asimismo, los conjuntos cocientes seran denotados por $G/\mathcal{R}_H$ y $G/\mathcal{R}^H$ respectivamente.

Lema 2.3. *Con la notación anterior, se verifica que*

$$[a]_H=\{ha : h\in H\} \quad y \quad [a]^H=\{ah : h\in H\}$$

Demostración:

Notemos

$$Ha = \{ha : h \in H\}.$$

donde $a \in G$. Se trata de demostrar que $[a]_H = Ha$. Si $y \in [a]_H$, entonces $y\mathcal{R}_H a$. Por tanto, $ya^{-1} \in H$. Así, existe $h \in H$ tal que $ya^{-1} = y$, y de aquí que $y = ha$, con $h \in H$. Por tanto, $y \in Ha$.

Recíprocamente, sea $y \in H_a$. Entonces existe $h \in H$, tal que $y = ha$, luego $ya^{-1} = h \in H$. Asi, y $\mathcal{R}_H$ a. Por tanto, $y \in [a]_H$.

El caso $[a]^H = aH$ es análogo.

■

Definición 2.11. *Las clases de equivalencia Ha y aH se llaman respectivamente clases adjuntas por la derecha y por la izquierda de G módulo H.*

Definamos a continuación el concepto de índice, esencial en esta sección, y algunas propiedades en relación con él.

Definición 2.12. *Sea G un grupo y sea H un subgrupo de G.*

(a) Si $G/\mathcal{R}_H$ es un conjunto infinito, decimos que H es un subgrupo de G de índice infinito.

(b) Si $G/\mathcal{R}_H$ es finito, se llama ***índice de H en G****, y lo denotamos por [G:H], al cardinal del conjunto $G/\mathcal{R}_H$. En este caso decimos que H es un subgrupo de G de índice finito.*

Corolario 2.2.1. *Si G es un grupo de orden finito, entonces todo subgrupo H de G es de índice finito.*

Demostración:

Consideremos la aplicación

$$\begin{aligned} G &\longrightarrow G/\mathcal{R}_H \\ a &\longmapsto Ha \end{aligned}$$

Esta aplicación es sobreyectiva de forma evidente. Por tanto, $card(G/\mathcal{R}_H) \leq card(G)$, y de ahí que $G/\mathcal{R}_H$ sea finito.

■

Lema 2.4. *Sea H un subgrupo de G, y sea $x \in G$. Entonces las aplicaciones:*

$$\begin{aligned} f : H &\longrightarrow Hx \quad y \quad g : H \longrightarrow xH \\ h &\longmapsto hx \qquad\qquad h \longmapsto xh \end{aligned}$$

son biyectivas.

Demostración:

Veamos que la aplicación f es biyectiva.

- f es inyectiva, ya que si $f(h_1) = f(h_2)$, entonces $h_1 x = h_2 x$. Por tanto, $h_1 = h_2$.
- f es sobreyectiva, ya que para todo $hx \in Hx$, existe $h \in H$ tal que $f(h) = hx$.

De igual forma se demuestra que g es biyectiva.

■

Teorema 2.3. ***(Teorema de Lagrange)*** *. Sea G un grupo finito, y sea H un subgrupo de G. Entonces se verifica que:*

$$|G| = |H| \cdot [G : H]$$

Demostración:

Consideramos la relación $\mathcal{R}_H$ definida en G. Al ser $\mathcal{R}_H$ una relación de equivalencia, G es una unión disjunta de las clases de equivalencia Hx, y al ser G un grupo finito, habrá solo un número finito. Sean estas

$$Hx_1, Hx_2, ..., Hx_r,$$

donde se verificara que el número de elementos de G es la suma de los cardinales de estas clases, es decir

$$card(G) = \sum_{i=1}^{r} card(Hx_i), \quad Hx_i \in G/\mathcal{R}_H$$

Por el lema 2.4, se verifica que

$$card(Hx_i) = card(H) = |H|$$

entonces

$$card(G) = card(H) \cdot card(G/\mathcal{R}_H).$$

Por tanto,

$$|G| = |H| \cdot [G : H]$$

ya que el número de clases que la relación $G/\mathcal{R}_H$ determina en G es, por definición, el índice de H en G.

■

A partir de este teorema, podemos probar el siguiente resultado:

Proposición 2.7. *Sea G un grupo con elemento neutro e y cuyo orden es* $n = p_1^{e_1} \dots p_k^{e_k}$ *y sea* $g \in G$*. Entonces g es un generador para G si y solo si* $g^{n/p_i} \neq e$*.*

Demostración:

$\Rightarrow$: Si g es generador, entonces $g^n = e$, siendo n el menor entero que verifica esta expresión. Por tanto, ya que $1 < \frac{n}{p_i} < n$ tenemos que $g^{n/p_i} \neq e$.

$\Leftarrow$: Sea $g \in G$, podemos considerar el subgrupo generado por g, $< g >$. Por el teorema de Lagrange, el orden de este subrupo divide al orden del grupo,

$$m = | < g > | \, | |G| = n$$

Con lo cual, el orden de g, m, es un divisor de n. Sin embargo, tenemos que $g^{n/p_i} \neq e$. Por tanto, la única posibilidad es que $m = n$, es decir, que $g^n \equiv e$. Es decir, que g es un generador de G.

■

A partir de esta proposición (2.7), podemos describir un algoritmo para encontrar el generador de un grupo cíclico, que nos resultara útil para hallar posteriormente un generador en el algoritmo de Elgamal.

Algoritmo 2.2. ***(Encontrar el generador de un grupo cíclico)*** *La entrada del algoritmo será un grupo cíclico G de orden n, con n expresado en sus factores primos, $n = p_1^{e_1} p_2^{e_2} ... p_k^{e_k}$. La salida sera el generador α de G. El procedimiento es el siguiente:*

1. *Se elige un elemento aleatorio a de G.*
2. *Para cada i, con $1 \leq i \leq k$, se hace lo siguiente:*
   2.1. *Se calcula $b_i \equiv a^{n/p_i} (mod n)$.*
   2.2. *Si $b \equiv e (mod n)$, se vuelve de nuevo a (1).*
3. *El algoritmo devuelve a.*

Veamos a continuación unos ejemplos de ejecución de este algoritmo.

Ejemplo 2.3. *Sea el grupo aditivo $G = (\mathbb{Z}_9, +)$, con $n = |\mathbb{Z}_9| = 9 = 3^2$, y $e = 0$.*

1. Tomamos $a = 3$.
2. Calculamos $b_1 = 3^{9/3} = 3^3 = 9 \equiv 0 (mod 9)$.
Como obtenemos $b_1 \equiv 0 (mod 9)$., volvemos a (1).
1. Tomamos $a = 4$.
2. Calculamos $b_1 = 4^{9/3} = 4^3 = 64 \equiv 1 \neq 0 (mod 9)$.
Entonces $a = 4$ es generador del grupo aditivo $G = \mathbb{Z}_9$. Podemos ver que:

$$\mathbb{Z}_9 =< 4 >= \{4^1, 4^2, 4^3, 4^4, 4^5, 4^6, 4^7, 4^8, 4^9\} = \{4, 8, 3, 7, 2, 6, 1, 5\}$$

Ejemplo 2.4. *Sea el grupo multiplicativo $G = \mathbb{Z}_{123}$, con $n = |\mathbb{Z}_{123}| = 122 = 2 \cdot 61$, por 2.6.*

1. Tomamos $a = 3$.
2. Calculamos b_i

$$b_1 = 3^{122/2} 3^{61} = (3^5)^{12} \cdot 3 \equiv (-1)^1 2 \cdot 3 \equiv 3 (mod 122)$$

$$b_2 = 3^{122/61} = 3^2 \equiv 9 (mod 122)$$

Como obtenemos $b_i \not\equiv 1 (mod 122)$, tenemos que 3 es generador del grupo $\mathbb{Z}_{123}$.

3

Criptografía

En este capítulo vamos a ver, en primer lugar, algunos conceptos básicos de criptografía. También se estudiará el problema del cálculo de números primos, y en relación a esto se detallará todo lo necesario para llegar al test de Miller-Rabin, el más utlizado actualmente en este contexto. Y a continuación, se presentará el tema central del trabajo, que es criptosistema de Elgamal, y se estudiarán distintos aspectos teóricos en relación al mismo.

3.1 Introducción a la Criptografía

Definimos a continuación los conceptos previos necesarios de criptografía.

Definición 3.1. *Un **criptosistema** es una 5-tupla (M, C, K, E, D), donde:*
M es el conjunto de posibles mensajes m para ser cifrados.
C es el conjunto de posibles mensajes cifrados c.
K es el conjunto de posibles claves para ser utilizadas.
E: $M \times K \longrightarrow C$ es una aplicación denominada algoritmo de encriptado o cifrado.
D: $C \times M \longrightarrow M$ es una aplicación denominada algoritmo de descifrado.

El proceso de cifrado puede realizarse de dos formas, dependiendo del conjunto K de claves utilizadas.

Definición 3.2. *Un **criptosistema simétrico** es aquel que utiliza la misma clave tanto para el proceso de encriptado como para el de desencriptado.*

Definición 3.3. *Un **criptosistema asimétrico** o de clave pública, es aquel que utiliza distintas claves para cada uno de los procesos. Utiliza una clave pública para el proceso de encriptado, y una clave privada para el desencriptado.*

Si realizamos una comparación de ambos tipos, podemos concluir que:
- Los criptosistemas simétricos son mucho más eficientes en tiempo de cómputo. Esto se debe a que utilizan funciones básicas y muy adecuadas para su implementación en hardware.
- En general, el texto cifrado resultante mediante un criptosistema simétrico es igual en tamaño al mensaje original.
- Los criptosistemas de clave pública aumentan considerablemente el tamaño de la información encriptada con respecto a la original, son más lentos, pero ofrecen un nivel de seguridad mucho mayor que los de clave simétrica.

Esto se debe a que los algoritmos simétricos basan su fortaleza en el número de combinaciones posibles a la hora de trabajar sobre un texto. Sin embargo, en el caso de los algoritmos de clave pública, su robustez se debe a problemas de índole matemática cuya solución se ha demostrado ser extremadamente compleja.

Veamos en qué consiste esta idea de la criptografía de clave pública.

Definición 3.4. *Una **función de una sola vía** es una función cuyo cálculo directo es viable, mientras que el cálculo de la función inversa es de tal complejidad que resulta imposible, con los conocimientos matemáticos actuales ni aún con las capacidades de cálculo más avanzadas.*

Definición 3.5. *Una **función con puerta trasera** es una función f tal que*

1. El cálculo de $y = f(x)$ es viable (complejidad de orden polinomial).

2. Existe cierta información adicional, cuyo conocimiento hace viable el cálculo de $x = f^{-1}(y)$.

3. Sin el conocimiento de la información adicional, f es una función de una sola vía.

4. El cálculo de la información adicional, aún conociendo su naturaleza, es irrealizable para los que no conozcan la clave secreta del sistema.

El primer criptosistema que hace uso de una función de este tipo es el introducido por Rivest, Shamir y Addleman en 1977, y conocido por las iniciales de sus inventores como *RSA*. La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa el en producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto.

Junto con este, el otro criptosistema más relevante entre los conocidos de clave pública es el que motiva nuestro interés, y se trata del criptosistema de *ElGamal*. En este caso, tenemos que este criptosistema se basa en el problema del logaritmo discreto, del cual hablaremos más adelante.

En cualquier caso, para este tipo de métodos nos encontramos un problema, y es el cálculo de números primos. La cuestión de los números primos es un tema recurrente de investigación en teoría de números. Su distribución no está determinada por ningún algoritmo conocido (lo cual los hace interesantes en el ámbito de la criptografía, pero necesitamos aportar métodos que nos faciliten la utilización de números primos en los algoritmos.

3.2 Cálculo de números primos

Supongamos que tenemos un entero de 129 cifras y queremos ver si es primo o no. Este suceso ocurrio en abril de 1994, y se le denomino como el derribo del RSA129. Se trataba de un número de 129 cifras que los autores del sistema habian hecho público, planteandolo como un reto. Un grupo de 600 matemáticos, con la ayuda de 1.600 voluntarios reclutados a traves de Internet, consiguieron factorizar el número. Sin embargo, se calcula que poniendo a trabajar todos los ordenadores del mundo en paralelo, una clave de 1.024 digitos tardaría un tiempo equivalente a la edad del universo (13.700 millones de años) en romperse.

Claramente, son necesarios mejores métodos para la comprobación de la primalidad de un número. Aunque pueda resultar sorprendente a primera vista, la factorización y la comprobación de la primalidad no son lo mismo. Es mucho mas fácil probar que un número es compuesto que factorizarlo. Hay muchos enteros grandes que se sabe que son compuestos, pero que no han sido factorizados. Un ejemplo de esto sería el

siguiente.

Sea $n = 15$. Tomamos un entero $a = 2$. Tenemos que:

$$2^4 \equiv 16 \equiv 1(mod 15)$$
$$2^{12} \equiv (2^4)^3 \equiv 1^3 \equiv 1(mod 15)$$
$$2^{14} \equiv 2^{12} \cdot 2^2 \equiv 4 \not\equiv 1(mod 15)$$

Con lo cual, por el Pequeño Teorema de Fermat (2.1), $n = 15$ no es primo. Podemos ver, pues, que no es necesaria la factorización del número para ver que es primo.

Test de primalidad

A partir del Pequeño Teorema de Fermat (2.1), podemos establecer un primer test de primalidad. Aunque se le denomina de esta forma, rigurosamente podríamos decir que se trata de test de factorización, ya que lo que da es una respuesta certera únicamente en el caso de que n sea compuesto.

Proposición 3.1. ***(Test de primalidad de Fermat)*** *Sea $n > 1$ un entero. Se elige un entero aleatorio a con $1 < a < n-1$. Si $a^{n-1} \not\equiv 1(mod\, n)$, entonces n es compuesto Si $a^{n-1} \equiv 1(mod\, n)$, entonces n es probablemente primo.*

El test de Fermat es bastante preciso para numeros grandes. Si establece que un numero es compuesto, entonces esta garantizando que lo es. Si establece que un numero es probablemente primo, entonces los resultados empíricos muestran que esto es probablemente cierto. Es más, ya que la exponenciación modular es rápida, el test de Fermat puede ejecutarse bastante rápido.

Sin embargo, el criterio que sigue para ver que es primo es el del Pequeño Teorema de Fermat (2.1) en sentido inverso, implicación que no se verifica en general. Existen números compuestos n tales que $a^{n-1} \equiv 1(mod\, n)$ para cualquier primo relativo con n.

Definición 3.6. *Los números de Carmichael son los números compuestos n que satisfacen la congruencia $a^{n-1} \equiv 1(mod\, n)$ para todo entero a primo relativo con n.*

El primer número de Carmichael es

$$n = 561 = 3 \cdot 11 \cdot 17$$

siendo $a^{560} - 1$ divisible por 561 para cualquier a coprimo con 561.
Los numeros de Carmichael son infinitos, siendo los primeros:

$$561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, ...$$

A causa de esto, se han investigado nuevos test de primalidad, que fueran mas precisos, es decir, que dieran menos falsos positivos.

Teniendo en cuenta que la exposición modular se realiza elevando al cuadrado sucesivamente, si tenemos cuidado con el procedimiento por el cual se eleva al cuadrado, el test de Fermat puede combinarse con el siguiente resultado para obtener para obtener un nuevo test, más fuerte que el anterior.

Proposición 3.2. *Sea n un entero, y sean x e y, con* $x^2 \equiv y^2 (mod n)$, *pero* $x \not\equiv \pm y\,(mod n)$. *Entonces n es compuesto. Ademas,* $gcd(x-y, n)$ *es un factor no trivial de n.*

Demostración:

Sea $d = gcd(x-y, n)$. Si $d = n$, entonces $x \equiv y (mod n)$, lo cual contradice la hipótesis.

Supongamos que $d = 1$. Un resultado básico de divisibilidad nos dice que, si $a|bc$, y $gcd(a, b) = 1$, entonces $a|c$. En nuestro caso, ya que n divide a $x^2 - y^2 = (x-y)(x+y)$ y $d = 1$, tendríamos que n divide a $x+y$, lo cual contradice la hipótesis de que $x \not\equiv -y (mod n)$. En consecuencia, $d \neq 1$, asi que d es un factor no trivial de n.

■

Veamos también cómo aplicaremos el test de Fermat, ya que no lo aplicaremos directamente, sino que lo utilizaremos como base para el siguiente resultado, que es el que aplicaremos.

Proposición 3.3. *Sea n primo impar,* $n-1 = 2^k m$, *m impar. Sea a un entero tal que* $gcd(a, n) = 1$. *Entonces tenemos*

$$a^m \equiv 1 (mod n) \qquad o \qquad a^{2^k m} \equiv -1 (mod n), \quad j \in \{0, 1, ..., s-1\}$$

Demostración:

Para ver que esto ocurre, vemos que el polinomio $x^{n-1} = x^{2^k m}$ puede factorizarse tantas veces como potencias de 2 haya en el exponente:

$$x^{2^k m} - 1 = (x^{2^{k-1} m})^2 - 1 = (x^{2^{k-1} m} - 1)(x^{2^{k-1} m} + 1) = (x^{2^{k-2} m} - 1)(x^{2^{k-2} m} + 1)(x^{2^{k-1} m} + 1) =$$

$$= ... = (x^m - 1)(x^m + 1)(x^{2m} + 1) = (x^{4m} + 1)...(x^{2^{k-1} m} + 1)$$

Como n es primo, y $1 < a < n-1$, entonces $a^{n-1} - 1 \equiv 0 (mod n)$ por el Pequeño Teorema de Fermat, con lo cual la factoriacion nos queda:

$$x^{2^k m} - 1 = (x^m - 1)(x^m + 1)(x^{2m} + 1) = (x^{4m} + 1)...(x^{2^{k-1} m} + 1) \equiv 0 (mod m)$$

con lo que uno de los factores debe ser $0 (mod m)$. Asi, tenemos que:

$$a^m \equiv 1 \qquad o \qquad a^{2^k m} \equiv -1 (mod n), \quad j \in \{0, 1, ..., s-1\}$$

■

Algoritmo 3.1. ***(Test de primalidad de Miller-Rabin)*** *Sea* $n > 1$ *un entero impar. Dado* $n - 1 = 2^k m$, *con m impar. Se elige un entero aleatorio a con* $1 < a < n - 1$.

(1) Se computa $b_0 \equiv a^m (mod\ n)$. *Si* $b_0 \equiv \pm 1 (mod\ n)$, *y entonces se para y declara que n es probablemente primo. De otro modo, (2).*

(2) Sea $b_1 \equiv b_0^2 (mod\ n)$. *Si* $b_1 \equiv 1 (mod\ n)$, *entonces n es compuesto (y mcd(*b_0 *- 1, n) da un factor no trivial de n). Si* $b_1 \equiv$ *-1 (mod n), entonces para y declara que n es probablemente primo. De otro modo, (3).*

(3) Sea $b_2 \equiv b_1^2 (mod\ n)$. *Si* $b_2 \equiv 1 (mod\ n)$, *entonces n es compuesto. Si* $b_2 \equiv$ *-1 (mod n), entonces para y declara que n es probablemente primo. Continua de esta forma hasta parar o alcanzar* b_{k-1}. *Si* $b_{k-1} \not\equiv$ *-1 (mod n), entonces n es compuesto.*

Apliquemos este método a algunos ejemplos.

Ejemplo 3.1. *Sea* $n = 337$. *Tenemos que* $n - 1 = 336 = 2^4 \cdot 21$. *Se toma aleatoriamente un entero,* $a = 15$.

(1) Se computa $b_0 \equiv 15^{21} (mod\, 337)$. *Entonces* $b_0 \equiv 278 (mod 337) \equiv -59 (mod 337) \not\equiv \pm 1 (mod 337)$, *asi que pasamos a (2).*

(2) Sea $b_1 \equiv (-59)^2 (mod\, 337)$. *Entonces* $b_1 \equiv 111 (mod 337) \equiv -59 \not\equiv \pm 1 (mod 337)$, *de manera que pasamos a (3).*

(3) Sea $b_2 \equiv 111^2 (mod 337)$. *Entonces* $b_2 \equiv 189 (mod 337) \not\equiv \pm 1 (mod 337)$, *de manera que seguimos hasta parar o alcanzar* b_3.

Sea $b_3 \equiv 189^2 (mod 337)$. *Entonces* $b_3 \equiv 336 \equiv -1 (mod 337)$. *Asi, tenemos que* $n = 337$ *es primo.*

Ejemplo 3.2. *Por otro lado, sea* $n = 377$. *Tenemos que* $n - 1 = 376 = 2^3 \cdot 47$. *Se toma aleatoriamente un entero,* $a = 11$, *y procedemos igual que en el caso anterior.*

$$b_0 = 11^{47} \equiv 305 \equiv -72 \not\equiv \pm 1 (mod 377).$$

$$b_1 = (-72)^2 \equiv -94 \not\equiv \pm 1 (mod 377).$$

$$b_2 = (-94)^2 \equiv 165 \not\equiv -1 (mod 377)$$

Dado que $b_{k-1} = b_2 \not\equiv -1 (mod 377)$, *tenemos que* $n = 337$ *es compuesto. De hecho,* $337 = 13 \cdot 29$.

Por último, veamos un ejemplo en el cual falla el algoritmo.

Ejemplo 3.3. *Sea* $n = 85$. *Tenemos* $n - 1 = 84 = 2^2 \cdot 21$. *Si tomamos el entero* $a = 13$*:*

$$b_0 = 13^{21} \equiv 13 \not\equiv \pm 1 (mod 85).$$

$$b_1 = 13^2 \equiv -1 (mod 85).$$

Con lo cual el test estableceria que 85 *es primo. Sin embargo tomando un entero aleatorio, sea* $a = 4$.

$$b_0 = 4^{21} \equiv 4 \not\equiv \pm 1 (mod 85).$$

$$b_1 = 4^2 \equiv 16 (mod 85) \not\equiv \pm 1 (mod 85).$$

Y dado que $b_{k-1} = b_1 \not\equiv -1 (mod 85)$, *tenemos que* $n = 85$ *es compuesto.*

Por este motivo, dado que hay excepciones a la hora de tomar a de forma segura, la forma de evitar estos errores es aplicar el test varias veces con varios enteros aleatorios diferentes, de forma que el resultado obtenido sea lo más seguro posible.

Búsqueda de números primos

A partir del test de primalidad que acabamos de ver (3.1) , podemos definir un algoritmo para la búsqueda de primos de la siguiente forma:

Algoritmo 3.2. *La entrada del algoritmo es un entero, k. La salida es un entero p probablemente primo de k bits. Se procede de la siguiente forma:*

1. Se genera un entero n de k bits de forma aleatoria.

2. Se realiza una breve comprobación previa para comprobar que si n es divisible por un primo de orden bajo. En caso afirmativo, se vuelve al paso (1). En caso contrario, (3).

3. Se aplica el test de Miller-Rabin a n. Si el algoritmo determina que es primo, esta es la salida del mismo. En caso contrario, se vuelve al paso (1).

Y con esto, ya tenemos las herramientas necesarias para definir el algoritmo en el cual se basa la generación de claves de ElGamal:

Algoritmo 3.3. ***(Seleccionar un primo y un generador de*** $\mathbb{Z}_p^*$***).*** *La entrada del algoritmo es la longitud k, en bits, de p,y un parametro de seguridad t. La salida es un primo de longitud k bits, tal que p-1 tenga un factor primo* $\geq t$*, un generador* α *de* $\mathbb{Z}_p^*$ *.*

El procedimiento es el siguiente:

1. Se repite el siguiente bucle hasta que p - 1 tenga un factor primo $\geq t$*.*

1.1. Se selecciona aleatoriamente un primo p, de longitud k bits (procediendo segun el algoritmo 3.2, por ejemplo).

1.2. Se factoriza p - 1.

2. Se utiliza el algoritmo 2.2 con $G = \mathbb{Z}_p^*$ *y* $n = p\text{-}1$ *para encontrar un generador* α *de* $\mathbb{Z}_p^*$*.*

3. El algoritmo devuelve (p, α)*.*

3.3 El criptosistema de ElGamal

Como decíamos anteriormente, el criptosistema de ElGamal es un criptosistema de clave pública basado en el problema del logaritmo discreto.

Definición 3.7. *Sean* a, b *elementos de un grupo cíclico finito G, se conoce como* ***logaritmo discreto de*** b ***en base*** a *, a la solución* x *de la ecuación* $a^x = b$*.*

$$x = log_a(b) \Longleftrightarrow a^x = b$$

Por tanto, en nuestro caso el problema del logaritmo discreto consiste en: Dados $a, b \in \mathbb{Z}_n$, tales que $b = a^x (mod n)$ para algún entero x, el cálculo de dicho exponente x.

En 1976, Diffie y Hellman inventaron un método para el intercambio de claves secretas en canales abiertos basado en el problema del logaritmo discreto. Todos los criptosistemas de clave pública posteriores seguirán la idea de este algoritmo.

Algoritmo 3.4. ***(Intercambio de clave de Diffie-Hellman).*** *Sean p un primo grande, y* α *un generador de* $\mathbb{Z}_p^*$*. Ambos valores son públicos. Supongamos que A y B quieren intercambiar una clave secreta a través de un canal público. El procedimiento sería:*

1. A escoge un número aleatorio x, con $1 < x < p - 1$*, lo mantiene en secreto y envía a B el valor público* $z_A = \alpha^x (mod p)$*.*

2. B elige otro número aleatorio y, con $1 < y < p-1$, lo mantiene en secreto y envía a A el valor público $z_B = \alpha^y (mod p)$.

3. A calcula $z_{AB} = z_B^x (mod p) = \alpha^{yx} (mod p)$.

4. B calcula $z_{BA} = z_A^y (mod p) = \alpha^{xy} (mod p)$.

En la misma línea de la idea de este algoritmo, podemos situar el algoritmo de ElGamal.

El criptosistema

A continuación se presenta el algoritmo central sobre el que trabajaremos. En primer lugar, presentamos el algorimo de generación de claves, y posteriormente el algoritmo de cifrado.

Algoritmo 3.5. ***(Algoritmo de generación de clave de Elgamal)*** *Generación de las claves publica y privada de A y B. Cada uno de ellos debe hacer lo siguiente:*

1. Se genera un primo grande p, y un generador α del grupo multiplicativo $\mathbb{Z}_p^*$.

2. Se selecciona un entero aleatorio a, $1 \leq a \leq p-2$, y se calcula $\alpha^a (mod p)$.

3. La clave pública generada es (p, α, α^a), y la clave privada es a.

Algoritmo 3.6. ***(Algoritmo de Elgamal)*** *El algoritmo se basa en que B encripta un mensaje m para A, y A lo desencripta.*

1. Encriptación. B sigue los siguientes pasos:

(a) Obtiene la clave pública de A, que es (p, α, α^a).

(b) Representa el mensaje como un entero m en el rango $\{0, 1, ..., p-1\}$.

(c) Selecciona un entero aleatorio k, con $1 \leq k \leq p-2$.

(d) Computa $\gamma = \alpha^k mod p$, y $\delta = m \cdot (\alpha^a)^k mod p$.

(e) Finalmente, manda el mensaje cifrado, $c = (\gamma, \delta)$ a A.

2. Desencriptación. Para recuperar el texto original m de c, A sigue los siguientes pasos:

(a) Usa su clave privada a para computar $\gamma^{p-1-a} (mod p)$. Notese que $\gamma^{p-1-a} (mod p) = \gamma^{-a} = \alpha^{-ak}$.

(b) Entonces halla m calculando $(\gamma^{-a}) \cdot \delta (mod p)$.

Veamos un ejemplo de aplicación del algoritmo para números muy pequeños.

Ejemplo 3.4. *(Generación de la clave). Sean A y B dos individuos. El individuo A selecciona el primo $p = 2357$ y el generador $\alpha = 2$ de $\mathbb{Z}_{2357}^*$. A elige una clave privada $a = 1751$, y calcula:*

$$\alpha^a (mod p) = 2^{2357} (mod 2357) = 1185$$

Con lo cual la clave pública de A es (2357, 2, 1185).

Ejemplo 3.5. *(Encriptación). Para encriptar el mensaje $m = 2035$, B selecciona un entero aleatorio $k = 1520$, y calcula*

$$\gamma = 2^{1520} (mod 2357) = 1430$$

y

$$\delta = 2035 \cdot 1185^{1520} (mod 2357) = 697$$

Así, B envía $\gamma = 1430$ *y* $\delta = 697$ *a A.*

(Desencriptación). Para desencriptar el mensaje recibido, A calcula

$$\gamma^{p-1-a} = 1430^{605}(mod\,2357) = 852$$

y recupera el mensaje mediante

$$m = 872 \cdot 697(mod\,2357) = 2035$$

Con lo cual obtiene m =2035, el mensaje enviado originalmente por A.

Cuestiones relevantes a tener en cuenta

Algunos aspectos a tener en cuenta con respecto al criptosistema de ElGamal son los siguientes:

a. Elección de los parámetros

Dependiendo del carácter de los individuos y el medio a través del cual se comunican, es posible realizar ciertas variaciones en el algoritmo. Una variación podría ser que todos los individuos tuviesen el mismo primo p y el mismo generador α, de manera que estos no necesitan ser publicados como parte de la clave. En resultado sería de claves públicas de menor tamaño, lo cual puede ser interesante el algunos casos.

b. Eficiencia del algoritmo

- El proceso de encriptación requiere dos exponenciaciones $\alpha^k(mod p)$ y $(\alpha^a)^k(mod p)$. Estas pueden ser aceleradas eligiendo k de forma aleatoria con cualidades adicionales.
- Una desventaja de ElGamal es la expasión que se produce del mensaje. El mensaje cifrado queda el doble de largo que su correspondiente mensaje original.

c. Seguridad

Un aspecto muy interesante del criptosistema de ElGamal es que un mensaje no es encriptado dos veces de la misma forma, puesto que dicho encriptado depende de un número aleatorio k que se genera a lo largo del proceso de encriptado.

Es crucial que cada vez que se utilice el algoritmo, se utilicen k aleatorios distintos. Si el mismo k se utilizase para encriptar dos mensajes m_1 y m_2, cuyos pares correspondientes serían $(\gamma_1, \delta_1), (\gamma_2, \delta_2)$, tendríamos $\delta_1/\delta_2 = m_1/m_2$, con lo que conociendo uno de los mensajes originales, podríamos obtener el otro.

d. ElGamal generalizado

Aunque el criptosistema de ElGamal es habitualmente descrito en el grupo multiplicativo $\mathbb{Z}_p^*$, puede ser generalizado a cualquier grupo cíclico finito. Dado que la seguridad del algoritmo está basada en la intratabilidad del problema del logaritmo discreto, esto debe tenerse en cuenta a la hora de escoger otro G. Este debería satisfacer las dos condiciones siguientes:

- Ser eficiente. La operación sobre el grupo G debería ser relativamente sencilla de realizar.
- Ser seguro. El problema del logaritmo discreto en G debería ser computacionalmente inviable.

4 Implementación de ElGamal en Android

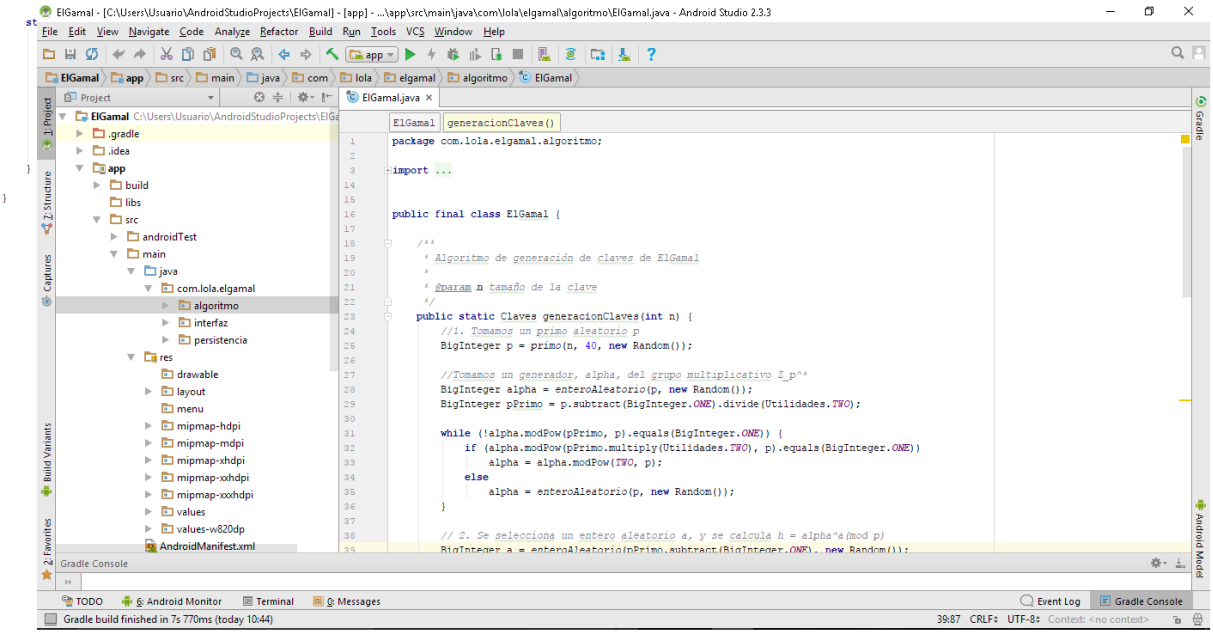
A continuación, se presenta la labor de programación realizada. En primer lugar, veremos de forma concisa el entorno de programación. Entonces procederemos a ver la estructura de nuestra aplicación, que ha sido implementada en Java (código fuente) y XML (interfaz gráfica). Por último, pasamos a ver de forma detallada cómo ha sido programado el algoritmo en Java.

4.1 Entorno de programación. Android Studio

Veamos en primer lugar una breve descripción del entorno:

Android es un sistema operativo diseñado principalemente para dispositivos móviles con pantalla táctil, tales como smartphones o tabletas. Inicialmente fue desarrollado por Android Inc, pero actualmente es propiedad de Google.

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado en 2013 por Google, y reemplazó a Eclipse como entorno oficial para el desarrollo de aplicaciones Android.



A continuación se describen los elementos principales:

- Manifest. El manifiesto es un elemento imprescindible para cualquier aplicación, y proporciona la información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la aplicación.

- Java. Esta carpeta contendrá todo el código fuente de la aplicación, clases auxiliares, etc.

- Res. Esta contendrá todos los ficheros de recursos necesarios para el proyecto: imágenes, layouts, cadenas de texto, etc.

Los diferentes tipos de recursos se distribuyen en las siguientes subcarpetas:

- Drawable. Contiene las imágenes y otros elementos gráficos usados por la aplicación.

- Mipmap. Contiene el icono de lanzamiento de la aplicación, es decir, el icono que aparecerá en el menú de aplicaciones del dispositivo.

- Layout. Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica.

- Values. Contiene ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto, estilos, definición colores, etc.

4.2 Estructura de la aplicación

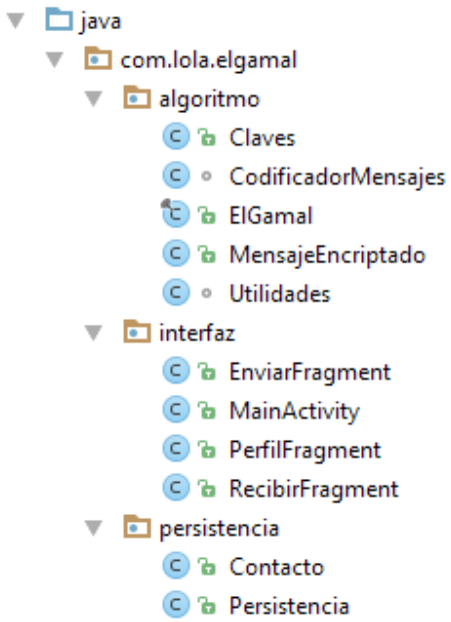
Para comenzar, vamos a proceder a definir de forma breve los elementos presentes en la aplicación.

En primer lugar, en lenguaje Java, encontramos las partes más relevantes, que serían:

1. Algoritmo. Se trata de la parte principal de la aplicación. Es la que determina cómo se realiza el cifrado, y que por tanto asegura la seguridad de la comunicación.

2. Interfaz. En ella se detallan todos los elementos que el usuario utiliza para realizar el cifrado y descifrado de los mensajes.

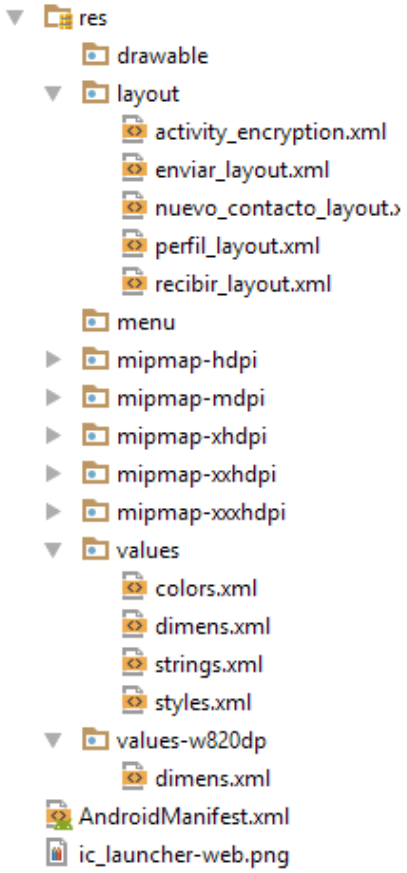
3. Persistencia. En esta parte se almacenan todos los datos necesarios para la comunicación, siendo estos las claves propias, tanto la pública como la privada, y las claves públicas de los contactos que deseemos.



Por otro lado, los elementos en la aplicación que no pertenecen a esta categoría, que están en formato XML, están divididos en:

4. Layout. Aquí vemos la descripción de la interfaz gráfica de cada uno de los menús y pantallas de la aplicación.

5. Values. En esta parte se almacenan básicamente los estilos y colores presentes en la interfaz gráfica.



4.3 Implementación del algoritmo en Java

La parte más relevante de la implementación es la que ha sido realizada en Java, y dentro de todo lo que se ha realizado en Java, desde el punto de vista matemático, lo que tiene mayor interés es el algoritmo.

Los pasos realizados durante el diseño del algoritmo han sido:

1. Creamos una clase Utilidades, en la cual se encontrarán algunas funciones imprescindibles para la realización del algoritmo, que sin embargo, no se encuentran dentro del mismo. En un primer momento definimos:

- Una función *enteroAleatorio*, que generará un entero grande (BigInteger). Los valores de entrada son un entero N, y un generador de aleatorios *generadorAleatorios*, que nos servirá para definir un entero aleatorio módulo N.

- Una función *primo*, en la cual se genera un primo de la forma $p = 2p' + 1$. Los valores de entrada son *nb_bits* (número de bits), *certeza* (un valor de seguridad para determinar si el número es primo o no), y *generadorAleatorios* (para generar los enteros aleatorios cuya primalidad comprobamos posteriormente).

- Una función TWO, que define de forma directa un entero grande cuyo valor sea 2, que nos será útil para generar un primo de la forma $p = 2p' + 1$, y también más adelante en otras funciones que definiremos posteriormente.

```
class Utilidades {


    static BigInteger TWO = BigInteger.valueOf(2);

    static BigInteger enteroAleatorio(BigInteger N, Random generadorAleatorios) {
        return new BigInteger(N.bitLength() + 100, generadorAleatorios).mod(N);
    }


    static BigInteger primo(int nb_bits, int certeza, Random generadorAleatorios) {
        BigInteger pPrimo = new BigInteger(nb_bits, certeza, generadorAleatorios);
        BigInteger p = pPrimo.multiply(TWO).add(BigInteger.ONE);

        while (!p.isProbablePrime(certeza)) {
            pPrimo = new BigInteger(nb_bits, certeza, generadorAleatorios);
            p = pPrimo.multiply(TWO).add(BigInteger.ONE);
        }
        return p;
    }
```

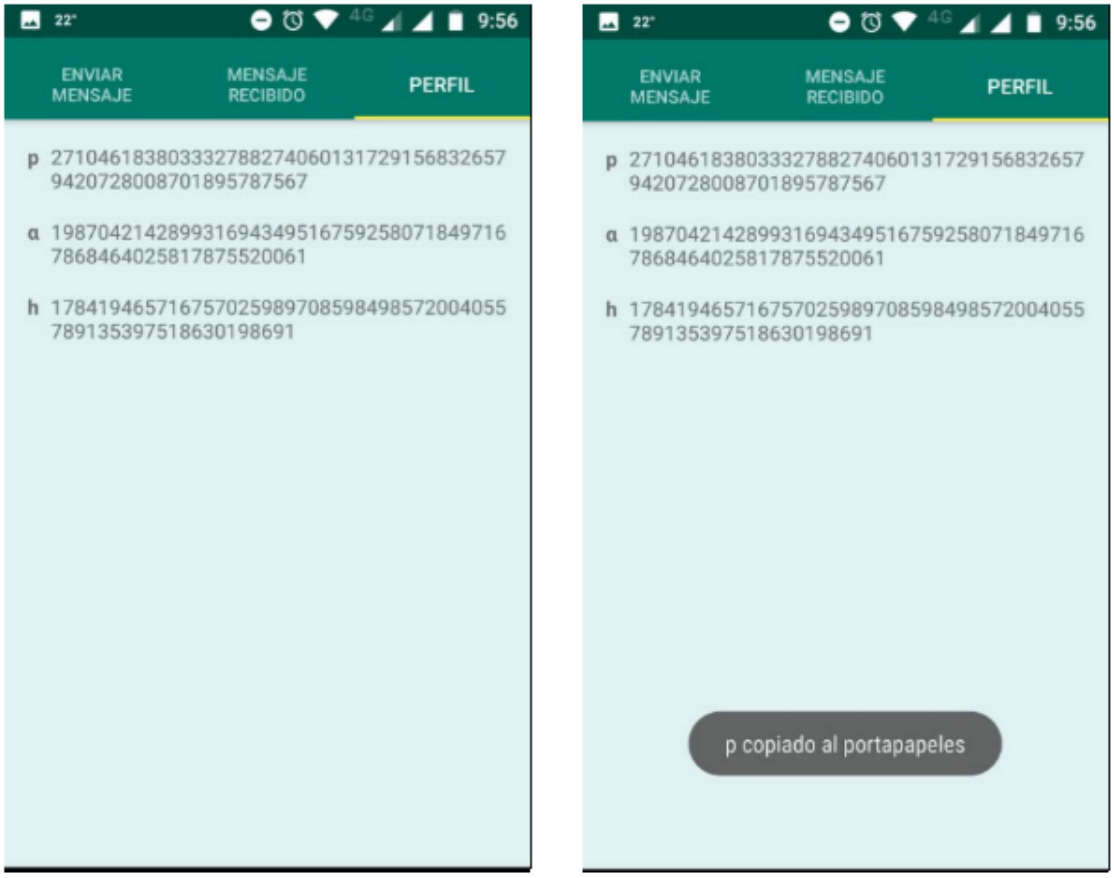
A Utilidades se añadirá otra función posteriormente, pero no tiene sentido definirla todavía.

2. Diseñamos una clase ElGamal, en la cual se encontrarán los algoritmos de generación de claves, encriptación y desencriptación. Lo hacemos de la siguiente forma:

- En primer lugar, definimos una función *generacionClaves*, la cual, a partir de un entero n, genera un primo con la función que definimos anteriormente (*primo*), y un generador del grupo multiplicativo $\mathbb{Z}_p^*$ según el algoritmo 3.3. A partir de estos elementos, se generan las claves pública y secreta, según el algoritmo 3.5.

```
    public static Claves generacionClaves(int n) {
        //1. Tomamos un primo aleatorio p
        BigInteger p = primo(n, 40, new Random());

        //Tomamos un generador, alpha, del grupo multiplicativo Z_p^*
        BigInteger alpha = enteroAleatorio(p, new Random());
        BigInteger pPrimo = p.subtract(BigInteger.ONE).divide(Utilidades.TWO);

        while (!alpha.modPow(pPrimo, p).equals(BigInteger.ONE)) {
            if (alpha.modPow(pPrimo.multiply(Utilidades.TWO), p).equals(BigInteger.ONE))
                alpha = alpha.modPow(TWO, p);
            else
                alpha = enteroAleatorio(p, new Random());
        }

        // 2. Se selecciona un entero aleatorio a, y se calcula h = alpha^a(mod p)
        BigInteger a = enteroAleatorio(pPrimo.subtract(BigInteger.ONE), new Random());
        BigInteger h = alpha.modPow(a, p);

        // 3. la clave secreta es a y la clave pública es (p, alpha, h)
        return new Claves(new Publica(p, alpha, h), new Privada(p, a));
    }
```

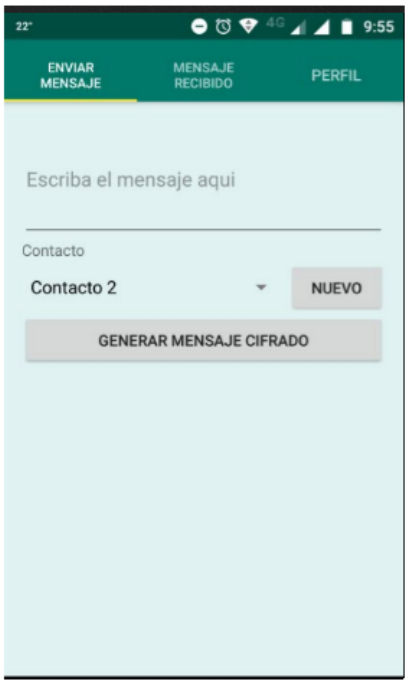
Y lo que vemos por pantalla en este caso es únicamente nuestra clave pública generada. La clave privada no se muestra en ningún momento, ya que esto hace más segura la aplicación, y además, de la forma en que está diseñada la aplicación, no hay ninguna necesidad de ello ni tendría ninguna utilidad práctica mostrarla.

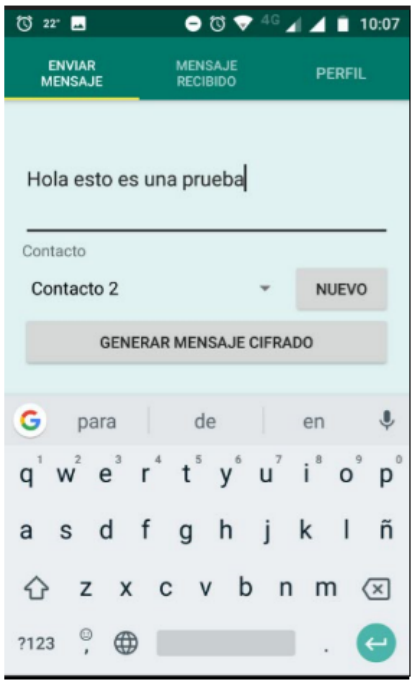


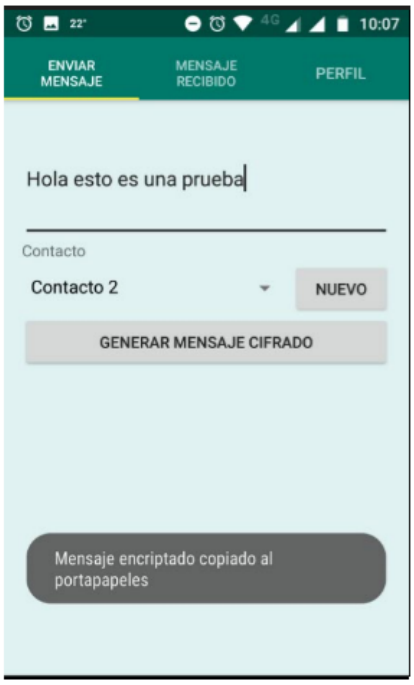
La parte del diseño de la interfaz se ha realizado de forma que al presionar sobre cualquiera de los componentes de la clave, esta se copie al portapapeles, lo cual hará más fácil compartir nuestra clave pública con los usuarios con los cuales deseemos comunicarnos.

- En segundo lugar, definimos una función *encriptacion*, la cual se encargará de generar el mensaje encriptado a partir del mensaje original, y la clave pública del usuario receptor.

```
public static MensajeEncriptado encriptacion(Publica clave, String mensaje) {
    List<Trozo> trozos = new ArrayList<>();
    for (String trozo : dividirEnSubcadenas(mensaje, 5)) {
        BigInteger mensajeCodificado = CodificadorMensajes.codificar(trozo);
        BigInteger pPrimo = clave.p.subtract(BigInteger.ONE).divide(TWO);
        BigInteger k = enteroAleatorio(pPrimo, new Random());
        BigInteger gamma = clave.alpha.modPow(k, clave.p);
        gamma.add(gamma);
        // encriptamos alpha^k, trozo * h^k
        BigInteger delta = mensajeCodificado.multiply(clave.h.modPow(k, clave.p));
        trozos.add(new Trozo(gamma, delta));
    }
    return new MensajeEncriptado(trozos);
}
```

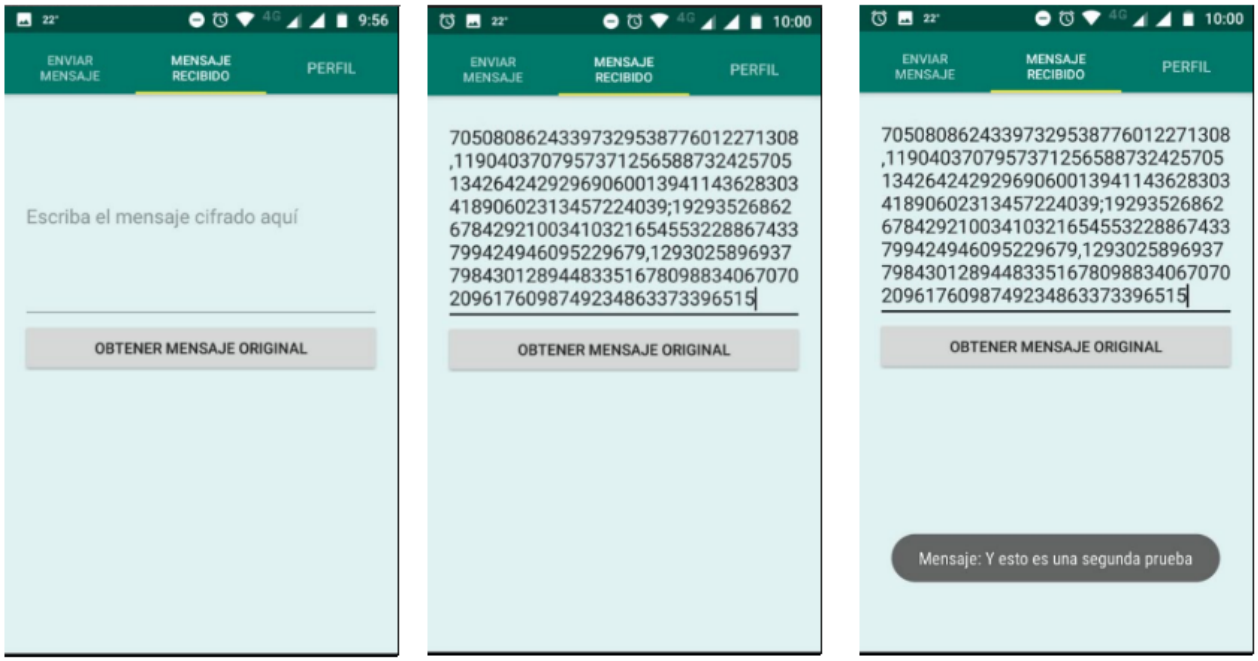






- En tercer lugar, definimos una función *desencriptacion*, cuyo comentido será desencriptar un mensaje encriptado de cualquier remintente que conozca nuestra clave pública. Los valores de entrada son el mensaje recibido y la clave privada del usuario.

```
    public static String desencriptacion(Privada clave, MensajeEncriptado encriptado) {
        StringBuilder mensaje = new StringBuilder();
        for (Trozo trozo : encriptado.trozos) {
            BigInteger hk = trozo.gamma.modPow(clave.a, clave.p);
            BigInteger mensajeDecodificado = trozo.delta.multiply(hk.modInverse(clave.p)).mod(clave.p);
            mensaje.append(CodificadorMensajes.decode(mensajeDecodificado));
        }
        return mensaje.toString();
    }
}
```



Como podemos ver en el código, ha sido necesario dividir en trozos los mensajes que se realizan. Mediante la experimentación hemos podido comprobar que la complejidad de cálculo del algoritmo hace inviable enviar mensajes relativamente largos de forma directa. Esto tiene sentido, ya que la criptografía asimétrica (3.3) ha sido pensada para las comunicaciones breves, como vimos anteriormente. Sin embargo, resulta interesante enviar mensajes un poco más largos de lo que en principio se nos permite. Por tanto, mediante la función *dividirEnSubcadenas* podemos llevar a cabo esta tarea. Esta es la función que nos faltaba por definir en Utilidades. Veámosla en este momento:

- Función *dividirEnSubcadenas* (Utilidades). En este caso, tenemos como valores de entrada el mensaje, y un entero *longitud* que indicará la longitud que queremos que tengan los trozos. Este valor será un múltiplo del número de bits que tenga cada caracter. Dentro tenemos un *if* que comprueba si el mensaje tiene una longitud inferior a *longitud*, en cuyo caso el 'conjunto' de trozos estaría formado por un único trozo. Posteriormente, tenemos un bucle *for* que genera un ArrayList *subcadenas* de

trozos mientras que la longitud del mensaje sera superior a *longitud*. La salida de esta función es el array de trozos de mensaje.

```
static List<String> dividirEnSubcadenas(String mensaje, int longitud) {
    if (mensaje.length() < longitud) {
        Collections.singletonList(mensaje);
    }
    List<String> subcadenas = new ArrayList<>((mensaje.length() + longitud - 1) / longitud);
    for (int indiceInicio = 0; indiceInicio < mensaje.length(); indiceInicio += longitud) {
        int indiceFin = Math.min(mensaje.length(), indiceInicio + longitud);
        subcadenas.add(mensaje.substring(indiceInicio, indiceFin));
    }
    return subcadenas;
}
```

3. En tercer lugar, junto con la función *subcadenas*, se ha definido todo lo necesario para dividir el mensaje en una nueva clase, *mensajeEncriptado*. En ella podemos ver:

- En primar lugar, una función que define la clase *trozo*, y un String para imprimirla cuando sea necesario.

```
public static class Trozo {
public final BigInteger gamma;
public final BigInteger delta;

public Trozo(BigInteger gamma, BigInteger delta) {
    this.gamma = gamma;
    this.delta = delta;
}

@Override
public String toString() {
    return "Trozo{" +
            "gamma=" + gamma +
            ", delta=" + delta +
            '}';
}
```

- En segundo lugar, se presenta la clase *trozos* como lista, y tenemos la clase *guardarComoTexto*. Esta clase la utilizaremos para guardar como texto la lista de enteros producida tras la encriptación del mensaje, de forma que se presente como un texto para enviarlo a través de la aplicación de mensajería deseada. En ella se produce la unión de los trozos mediante un bucle *for*.

```
public final List<Trozo> trozos;

public MensajeEncriptado(List<Trozo> trozos) { this.trozos = trozos; }

@Override
public String toString() {
    return "MensajeEncriptado{" +
            "trozos=" + trozos +
            '}';
}

public static String guardarComoTexto(MensajeEncriptado mensajeEncriptado){
    StringBuilder texto = new StringBuilder();
    for (Trozo trozo : mensajeEncriptado.trozos) {
        if (texto.length() > 0) {
            texto.append(';');
        }
        texto.append(trozo.gamma).append(',')
                .append(trozo.delta);
    }
   return texto.toString();
}
```

- A continuación, se presenta la clase *cargarDeTexto*, que realiza el proceso inverso. A un mensaje recibido en forma de texto lo transforma en dos listas de trozos (en formato texto). Mediante un bucle *for* se produce la separación de cada uno de los trozos (que recordemos que está formada por un γ, y un δ), y a continuación se definen los String de gammas y deltas.

Como podemos ver, incluye un *try – catch*, por si el mensaje no fuese introducido en el formato correcto.

```
public static MensajeEncriptado cargarDeTexto(String mensajeEncriptado) throws IllegalFormatException {
    try {
        List<Trozo> trozos = new ArrayList<>();
        for (String trozo : mensajeEncriptado.split(";")) {
            int posicionSeparador = trozo.indexOf(',');
            String gamma = trozo.substring(0, posicionSeparador);
            String delta = trozo.substring(posicionSeparador + 1);
            trozos.add(new Trozo(new BigInteger(gamma), new BigInteger(delta)));
        }
        return new MensajeEncriptado(trozos);
    } catch (Exception ex) {
        throw new IllegalArgumentException("Formato incorrecto de mensaje: " + mensajeEncriptado, ex);
    }
}
```

4. A continuación, se define una clase imprescindible para el funcionamiento del algoritmo, la clase *CodificadorMensajes*. Esta clase se encarga de la conversión del mensaje de texto a cifras, y de cifras a texto.

- En primer lugar, se define *MAX_CHARACTER_LENGTH*, que indica la longitud máxima que tiene un carácter de texto codificado en forma de número. Los caracteres codificados incluyen letras de cualquier alfabeto, incluyendo la ñ, o carácteres chinos, por ejemplo.

- En segundo lugar, se realiza la función *codificar*, que codifica cada carácter, y les añade ceros de relleno en caso de ser necesario, para que todos tengan la misma longitud y puedan decodificarse fácilmente.

- En tercer lugar, tenemos la función *decodificar*, que realiza el proceso inverso, recibe todos los valores numéricos generados tras la encriptación, y se encarga de transformarlos en el texto del mensaje original.

```
class CodificadorMensajes {

    /*
     Los dígitos se codifican con su valor numérico, utilizando ceros de relleno
     para hacer que todos tengan la misma longitud y se puedan decodificar fácilmente
     */
    private static final int MAX_CHARACTER_LENGTH = String
            .valueOf((int) Character.MAX_VALUE).length();

    public static BigInteger codificar(String string) {
        StringBuilder codificado = new StringBuilder();
        for (int i = string.length() - 1; i >= 0; i--) {
            int digitValue = string.charAt(i);
            String caracterCodificado = String
                    .format(Locale.ENGLISH, "%0" + MAX_CHARACTER_LENGTH + "d", digitValue);
            codificado.append(caracterCodificado);
        }
        return new BigInteger(codificado.toString());
    }

    public static String decodificar(BigInteger number) {
        String valor = number.toString();
        int ultimoIndice = valor.length();
        StringBuilder decodificado = new StringBuilder();
        while (ultimoIndice > 0) {
            int primerIndice = Math.max(0, ultimoIndice - MAX_CHARACTER_LENGTH);
            String caracterUnicode = valor.substring(primerIndice, ultimoIndice);
            decodificado.append((char) Integer.parseInt(caracterUnicode));
            ultimoIndice -= MAX_CHARACTER_LENGTH;
        }
        return decodificado.toString();
    }
```

5. Por último, se ha creado una clase cuyo único objetivo es el almacenamiento de las claves, denominada *Claves*, y un String cuando sea necesario su uso.

```
package com.lola.elgamal.algoritmo;

import java.math.BigInteger;

public class Claves {

    public final Publica publica;
    public final Privada privada;

    Claves(Publica publica, Privada privada) {
        this.publica = publica;
        this.privada = privada;
    }

    @Override
    public String toString() {
        return "Claves{" +
                "publica=" + publica +
                ", privada=" + privada +
                '}';
    }


  public static class Privada{
      public final BigInteger p;
      public final BigInteger a;

      public Privada(BigInteger p, BigInteger a) {
          this.p = p;
          this.a = a;
      }

      @Override
      public String toString() {
          return "Privada{" +
                  "p=" + p +
                  ", a=" + a +
                  '}';
      }
  }
```

```
public static class Publica {
    public final BigInteger p;
    public final BigInteger alpha;
    public final BigInteger h;

    public Publica(BigInteger p, BigInteger alpha, BigInteger h) {
        this.p = p;
        this.alpha = alpha;
        this.h = h;
    }

    @Override
    public String toString() {
        return "Publica{" +
                "p=" + p +
                ", alpha=" + alpha +
                ", h=" + h +
                '}';
    }
}
```

Con esto, y la realización de la interfaz gráfica, tenemos una aplicación totalmente funcional que permite encriptar y desencriptar mensajes mediante el uso de ElGamal. Los mensajes encriptados a través de esta aplicación pueden enviarse a través de cualquier aplicación de mensajería, y será totalmente imposible su descifrado sin el uso de la aplicación, que es la que puede acceder a la clave privada.

Aunque un mensaje fuese interceptado, y también el dispositivo con el cual fue enviado, sería imposible acceder al contenido original del mensaje, ya que ha sido encriptado con la clave pública del receptor, y la persona que ha interceptado el mensaje y el dispositivo, solo tendría acceso a esta clave pública y a los datos del emisario, y no es posible desencriptar el mensaje original sin la clave privada del receptor.

5 Conclusiones

Actualmente ya existen soluciones tecnológicas que proporcionan confidencialidad al envío de información a través de la red. El año pasado, tras ver cómo algunos de sus usuarios comenzaban a decantarse por Telegram (entre otras aplicaciones), por tener un mayor grado de seguridad, WhatsApp aplicó el cifrado de extremo a extremo que conocemos actualmente. Sin embargo, tanto en Whatsapp, como en Telegram y otras aplicaciones de mensajería, la confidencialidad se ofrece únicamente durante el proceso de envío de la información, es decir, que cualquier persona con acceso al teléfono, independientemente de que sea el usuario o no, puede observar la información almacenada en el mismo, y puede leer los mensajes.

La solución propuesta con esta aplicación va más allá, y almacena los mensajes de forma segura en tanto en cuanto no se conozca la clave privada, la cual puede almacenarse en un repositorio o tarjeta externa, y solo acceder a ella cuando se quiera acceder a la información que protege.

La gran ventaja con respecto a otras soluciones como una contraseña o un pin, es que la seguridad es mucho más fuerte en este caso, ya que se produce por el uso de un algoritmo cuya fortaleza reside en un problema computacional de complejidad muy alta, como es el logaritmo discreto, y no existen herramientas o aplicaciones que puedan hackear aplicaciones basadas en este tipo de seguridad.

Como conclusión, este trabajo pone de manifiesto que el conocimiento de las matemáticas, combinadas con nociones de programación, sirve para llevar a cabo una aplicación original al mundo real, y que ofrece solución a un problema específico. Se transfiere, de este modo, el conocimiento aprendido durante estos años a la sociedad, que es el fin último de la Ciencia.

Bibliografía

[1] A. Menezes, P. van Oorschot, S. Vanstone *Handbook of Applied Cryptography*, CRC Press, 1996.

[2] B. Philips, C. Stewart, B. Hardy, K. Marsicano, *Programación con Android. Edición 2016*, Ediciones Anaya Multimedia, 2016.

[3] J. Dorronsoro, E. Hernandez *Números, grupos y anillos*, Addison-Wesley / Universidad Autónoma de Madrid, 1996.

[4] J. Pastor Franco, M.A. Sarasa, *Criptografía Digital. Fundamentos y aplicaciones*, Prensas Universitarias de Zaragoza, 1998.

[5] J. Ribas Lequerica, *Desarrollo de aplicaciones para Android. Edición 2017*, Ediciones Anaya Multimedia, 2017.

[6] J. Rif, L.L. Hugget, *Comunicación digital*, Masson S.A., 1991.

[7] M. Moreno, E. Pardo, *Teoría de grupos*, Universidad de Cádiz, 2003.

[8] R.P. Grimaldi, *Matemática Discreta y sus aplicaciones*, McGraw-Hill, 2004.

[9] W. Trappe, L. C. Washington, *Introduction to Cryptography with Coding Theory*, Pearson Prentice Hall, 2006.

[10] Página web oficial de desarrolladores Android. URL: https://developer.android.com/develop/