Criptosistemas de clave pública

Josep Domingo Ferrer

PID_00214911
Módulo 5

UOC
Universitat Oberta de Catalunya

Índice

Introducción

En este módulo didáctico presentamos el concepto de **criptografía de clave pública**, que ha supuesto una revolución en la criptografía moderna. Para estudiarlo, abordaremos los aspectos siguientes:

a) Empezamos con unas nociones preliminares de teoría de los números y de teoría de la complejidad, imprescindibles para entender el funcionamiento de los criptosistemas de clave pública.

b) A continuación explicamos los conceptos de *clave pública*, *funciones unidireccionales* y *funciones unidireccionales con trampa*.

c) Después introduciremos el intercambio de claves de Diffie-Hellman.

d) Finalmente, describimos los dos criptosistemas de clave pública principales: el RSA y el ElGamal, que son los más usados actualmente.

Objetivos

Los materiales didácticos de este módulo os tienen que permitir alcanzar los objetivos siguientes:

1. Entender los conceptos de *clave pública, función unidireccional* y *función unidireccional con trampa*.

2. Adquirir o recuperar los conocimientos de aritmética modular y de matemática discreta necesarios para entender el funcionamiento de los criptosistemas de clave pública.

3. Conocer los dos criptosistemas de clave pública más importantes.

1. Conceptos preliminares

En este apartado vamos a resumir los conceptos básicos de la teoría de los números que nos hacen falta para entender el resto del módulo. El estudiante que ya tenga un cierto nivel de conocimientos de aritmética modular o de matemática discreta puede pasar por alto este apartado.

1.1. Aritmética modular

Dados los enteros a, b y $n \neq 0$, se dice que a es congruente con b módulo n, que se escribe:

$$a \equiv_n b,$$

si y sólo si $a - b = kn$ para algún k entero.

$20 \equiv_7 6$, porque $(20 - 6) = 2 \cdot 7$.

Una definición alternativa sería: $a \equiv_n b$ si y sólo si n divide $(a - b)$, que se escribe $n|(a - b)$.

Si $a \equiv_n b$, entonces b se llama **residuo de *a* módulo *n***. Recíprocamente, a es un residuo de b módulo n. Escribiremos ***a* mod *n*** para representar el residuo de a módulo n en el rango $[0, n - 1]$.

Un conjunto de enteros $r_1, ..., r_n$ es un **conjunto completo de residuos módulo *n*** si para cada entero a, hay exactamente un r_i en el conjunto tal que $a \equiv_n r_i$.

Por ejemplo, 13 mod 7 = 6.

Para cualquier módulo, el conjunto de enteros $\{0, 1, ..., n - 1\}$ es un conjunto completo de residuos módulo n.

Igual como sucede en el conjunto de los enteros, los elementos "enteros mod n" con las operaciones de suma y de multiplicación forman un **anillo conmutativo**. Esto significa que la suma cumple las propiedades asociativa y conmutativa y además existen el elemento neutro y el elemento inverso, y para el producto se cumple la propiedad asociativa respecto de la suma y también tiene elemento neutro.

Además, calcular con la aritmética modular (es decir, reducir cada resultado intermedio módulo n) da el mismo resultado que calcular con la aritmética entera ordinaria y reducir el resultado final módulo n. De una manera más formal, se puede expresar este hecho con el teorema de la aritmética modular.

Teorema 1: principio de la aritmética modular. Sean a_1 y a_2 enteros, y sea OP uno de los operadores binarios +, – o · . La reducción módulo n es un homomorfismo del conunto de los enteros al conjunto los enteros módulo n, es decir:

$$(a_1 \text{ OP } a_2) \bmod n = [(a_1 \bmod n) \text{ OP } (a_2 \bmod n)] \bmod n$$

Equivalencia de las operaciones con aritmética entera y modular

Podemos calcular el resultado de la operación 9.739 + 7.777 + 5.345 módulo 7 de dos maneras equivalentes:

a) Primero sumamos de la manera habitual, 9.739 + 7.777 + 5.345 = 22.861, y a continuación reducimos el resultado: 22.861 mod 7 = 6.

b) Primero reducimos cada sumando, 9.739 mod 7 = 2; 7.777 mod 7 = 0; 5.345 mod 7 = 4, y después sumamos los resultados de las reducciones: (2 + 0 + 4) mod 7 = 6.

Operar con aritmética modular presenta la ventaja de que reduce el rango de los valores intermedios. Para un módulo n de k bits, el valor de cualquier suma, resta o multiplicación cabrá en $2k$ bits, a lo sumo. Esto nos permite, por ejemplo, calcular exponenciaciones modulares del estilo $a^z \bmod n$ sin generar resultados intermedios monstruosos.

El cálculo informatizado

Observad que para el cálculo informatizado es más adecuado operar con aritmética modular; de este modo nos aseguramos de que los resultados intermedios sean pequeños y prevendremos posibles desbordamientos (en inglés, *overflows*) intermedios.

Dado que algunos de los algoritmos de cifrado que vamos a describir en este módulo se basan en la exponenciación mod n, en primer lugar vamos a esbozar un método de exponenciación rápida para calcular $a^z \bmod n$:

exp_rapid(*a*,*z*,*n*)

begin "Devuelve $x = a^z \bmod n$"
  $a_1 := a$; $z_1 := z$;
  $x := 1$;
  while $z_1 \neq 0$ **do** "$x(a_1^{z_1} \bmod n) = a^z \bmod n$"
  begin
    while $z_1 \bmod 2 = 0$ **do**
    begin "elevar al cuadrado a_1 mientras z_1 par"
      $z_1 := \lfloor z_1/2 \rfloor$;
      $a_1 := (a_1 * a_1) \bmod n$
    end;
    $z_1 := z_1 - 1$;
    $x := (x * a_1) \bmod n$ "Multiplicación"
  end;
  exp_rapid := x
end

Si $(z_{k-1}, ..., z_1, z_0)$ es la representación binaria del exponente z, el algoritmo procesa los bits en orden $z_0, z_1, ..., z_{k-1}$ (de menos a más significativo). Si

$z_i = 0$, calcula un cuadrado; si $z_i = 1$, multiplica y calcula un cuadrado. Para cada bit de z se hacen, pues, hasta dos multiplicaciones, excepto para el más significativo, que sólo da lugar a una multiplicación. Puesto que la longitud de z en bits es $\lfloor \log_2 z \rfloor$, el número total de multiplicaciones es, como mucho:

$$2 \cdot (\lfloor \log_2 z \rfloor - 1) + 1 = 2 \cdot \lfloor \log_2 z \rfloor + 1$$

Hay que tener presente que exponenciar por el método "inocente" de las multiplicaciones sucesivas requeriría z multiplicaciones.

Cálculo de a^{37} con el algoritmo *exp_rapid*(*a*,*z*,*n*)

Para ilustrar el algoritmo *exp_rapid*(*a*,*z*,*n*) tomamos, por ejemplo, z = 37. La tabla siguiente refleja los valores intermedios de las variables (sólo se presentan los cambios de valor):

a_1	z_1	x
a	37	1
_	36	a
a^2	18	_
a^4	9	_
_	8	a^5
a^8	4	_
a^{16}	2	_
a^{32}	1	_
_	0	a^{37}

En total hay ocho multiplicaciones, valor inferior a la cota $2\lfloor \log_2 37 \rfloor + 1 = 11$. De hecho, el algoritmo se basa en la descomposición siguiente:

$$a^{37} = a^{32+4+1} = ((((a^2)^2)^2)^2)^2(a^2)^2a$$

Inversos multiplicativos

Los enteros 11 y 3 son inversos multiplicativos mod 16 porque:
$11 \cdot 3 \bmod 16 = 33 \bmod 16 = 1$.

1.2. Cálculo de inversos en aritmética modular

Si trabajamos con aritmética entera ordinaria, no hay inversos multiplicativos. Es decir, para un entero $a \neq 1$ no hay ningún otro entero x tal que se verifique $ax = 1$. En cambio, si trabajamos con aritmética modular, para un entero a en el rango de valores $[0, n-1]$ a veces es posible hallar un único entero en el mismo rango que verifique $ax \bmod n = 1$.

En los algoritmos de cifrado descritos en este apartado se utiliza a menudo el inverso multiplicativo. El teorema siguiente da la condición que se tiene que cumplir para que haya un inverso multiplicativo.

Teorema 2. Dado $a \in [0, n-1]$, a tiene un único inverso multiplicativo módulo n si a es coprimo con n, es decir, $\text{mcd}(a, n) = 1$.

Pero no sirve de mucho saber que hay inverso si no disponemos de algoritmos para calcularlo. Para avanzar hay que presentar algunos conceptos adicionales.

Leonhard Euler

Matemático suizo del siglo XVIII cuya obra abarca toda la matemática de su época.

La **función phi de Euler** $\phi(n)$ es el número de elementos del conjunto $\{0..., n\}$ que son coprimos con n.

Teorema 3. La función phi de Euler se puede calcular con las fórmulas siguientes:

1) Para p primo, $\phi(p) = p - 1$ y $\phi(1) = 1$.

2) Para $n = pq$ con p, q primos: $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$.

3) Para n arbitrario tenemos:

$$\phi(n) = \prod_{i=1}^{t} p_i^{e_t - 1}(p_i - 1),$$

en que $n = p_1^{e_1} p_2^{e_1} \ldots p_t^{e_t}$ es la descomposición de n en factores primos $p_1..., p_t$, donde p_i tiene multiplicidad e_i.

Elementos de $\phi(10)$

$\phi(10) = 4$ porque hay cuatro elementos en {0, ..., 10} que son coprimos con 10; concretamente son el 1, el 3, el 7 y el 9.

Los teoremas siguientes relacionados con la función de Euler son importantes en la teoría de los números.

Teorema 4: teorema pequeño de Fermat. Sea p primo, entonces para todo a tal que $\text{mcd}(a, p) = 1$ se cumple:

$$a^{p-1} \bmod p = 1.$$

Teorema 5: teorema de Euler. Para todo a y n tales que $\text{mcd}(a, n) = 1$ se cumple:

$$a^{\phi(n)} \bmod n = 1.$$

Pierre de Fermat

Matemático francés del siglo XVII que desempeñó el papel de precursor en cálculo diferencial, en geometría analítica, en probabilidad y también en teoría de números. El teorema pequeño de Fermat no es el célebre teorema i ndemostrado de Fermat. Este último no ha sido demostrado hasta 1995.

El teorema de Euler proporciona una primera manera de calcular el inverso multiplicativo de a mod n resolviendo la ecuación:

$$ax \bmod n = 1.$$

Cuando $\text{mcd}(a, n) = 1$, la solución se obtiene a partir de la expresión siguiente:

$$x = a^{\phi(n)-1} \bmod n.$$

Cálculo del inverso multiplicativo de 7 mod 15

Supongamos $a = 7$ y $n = 15$; entonces la función phi de Euler de 15 es:

$$\phi(15) = \phi(3)\phi(5) = 2 \cdot 4 = 8$$

Si aplicamos el teorema de Euler para encontrar el inverso multiplicativo obtenemos:

$$x = 7^{8-1} \bmod 15 = 13$$

En efecto, 13 es el inverso de 7, ya que $7 \cdot 13 \bmod 15 = 1$.

Si se conoce $\phi(n)$, el inverso de a módulo n se puede calcular a partir de la expresión anterior y del algoritmo *exp_rapid*(*a*,*z*,*n*). El problema es que si el módulo n es muy grande (como en los criptosistemas que vamos a ver), es difícil hallar $\phi(n)$. El algoritmo de Euclides extendido permite calcular inversos sin conocer $\phi(n)$.

Euclides

Matemático griego del siglo III a.C. su obra Elementos es considerada el libro de geometría por excelencia y resume todas las aportaciones griegas anteriores a Arquímedes.

El **algoritmo de Euclides básico** sirve para calcular el máximo común divisor de dos números, a y n.

Algoritmo mcd(*a*, *n*)

begin
  $g_0 := n$;
  $g_1 := a$;
  $i := 1$;
  while $g_i \neq 0$ **do**
  begin
    $g_{i+1} := g_{i-1} \bmod g_i$;
    $i := i + 1$;
  end;
  mcd := g_{i-1}
end

La **versión extendida del algoritmo de Euclides** permite encontrar el inverso de a módulo n cuando mcd(a, n) = 1:

Algoritmo inv(*a*, *n*)

begin "Devuelve x tal que $ax \bmod n = 1$, donde $0 < a < n$"
  $g_0 := n$; $g_1 := a$;
  $u_0 := 1$; $v_0 := 0$;
  $u_1 := 0$; $v_1 := 1$;
  $i := 1$;
  while $g_i \neq 0$ **do** "$g_i = u_i n + v_i a$"

begin
$y := \lfloor g_{i-1}/g_i \rfloor$;
$g_{i+1} := g_{i-1} - y * g_i$;
$u_{i+1} := u_{i-1} - y * u_i$;
$v_{i+1} := v_{i-1} - y * v_i$;
$i := i + 1$;
end;
$x := v_{i-1}$
if $x \geq 0$ **then** inv := x **else** inv := $x + n$
end

Si el módulo con respecto al cual se invierte es n, entonces el algoritmo $inv(a,n)$ requiere de el orden de $\ln(n)$ divisiones; es bastante eficiente, pues.

Utilización del algoritmo de Euclides extendido

Ilustraremos el funcionamiento del algoritmo de Euclides extendido calculando el inverso de 3 módulo 7, es decir, resolviendo la ecuación $3x \bmod 7 = 1$.

La tabla siguiente refleja los valores intermedios de las variables (sólo se presentan los cambios de valor):

i	g_i	u_i	v_i	y
0	7	1	0	–
1	3	0	1	2
2	1	1	–2	3
3	0	–	–	–

Como $v_2 = -2$ es negativo, la solución es $x = -2 + 7 = 5$.

Finalmente mencionaremos un último resultado (ya milenario) que es útil en algunos protocolos criptográficos. La idea es resolver un determinado tipo de sistemas de ecuaciones modulares.

Teorema 6: teorema chino del resto. Sean $d_1, ..., d_t$ enteros coprimos dos a dos, y sea $n = d_1 d_2 ... d_t$. El sistema de ecuaciones:

$$(x \bmod d_i) = x_i; \text{ para } i = 1, ..., t$$

tiene una solución común x en el rango $[0, n - 1]$.

Demostración: para cada $i = 1, ..., t$ tenemos $\text{mcd}(d_i, n/d_i) = 1$. Por lo tanto, existe el inverso de n/d_i módulo d_i, es decir, y_i tal que $(n/d_i)y_i \bmod d_i = 1$. Por otra parte $(n/d_i)y_i \bmod d_j = 0$ para $j \neq i$, porque d_j es un factor de n/d_i. Sea:

$$x = \left[\sum_{i=1}^{t} \left(\frac{n}{d_i} \right) y_i x_i \right] \bmod n,$$

Luego x es la solución de la ecuación siguiente:

$$(x \bmod d_i) = x_i; \text{ para } y = 1, \ldots, t$$

dado que:

$$x \bmod d_i = \left(\frac{n}{d_i}\right) y_i x_i \bmod d_i = x_i$$

Utilización del teorema chino del resto

Utilizaremos el teorema chino del resto para resolver la ecuación $3x \bmod 10 = 1$ a base de ir resolviendo ecuaciones con módulos más pequeños. Observamos que $10 = 2 \cdot 5$; por consiguiente, $d_1 = 2$ y $d_2 = 5$. Entonces tenemos que encontrar las soluciones x_1 y x_2 de las ecuaciones:

- $3x \bmod 2 = 1$,
- $3x \bmod 5 = 1$.

Aplicamos el algoritmo $inv(a,n)$ y obtenemos $x_1 = 1$ y $x_2 = 2$. A continuación, el teorema chino del resto permite encontrar una solución común x de las ecuaciones:

- $x \bmod 2 = x_1 = 1$,
- $x \bmod 5 = x_2 = 2$.

Encontramos y_1 y y_2 tales que:

- $\left(\frac{10}{2}\right) y_1 \bmod 2 = 1$,
- $\left(\frac{10}{5}\right) y_2 \bmod 5 = 1$.

Obtenemos $y_1 = 1$ y $y_2 = 3$, con lo cual:

$$x = \left[\left(\frac{10}{2}\right) y_1 x_1 + \left(\frac{10}{5}\right) y_2 x_2\right] \bmod 10 = [5 \cdot 1 \cdot 1 + 2 \cdot 3 \cdot 2] \bmod 10 = 7.$$

Por lo tanto, 7 es el inverso de 3 módulo 10.

1.3. Terminología básica de la teoría de la complejidad

La complejidad computacional permite fundamentar el análisis de los requisitos computacionales de las técnicas de criptoanálisis, es decir, la dificultad que comporta romper una cifra.

1.3.1. Complejidad de un algoritmo

La fortaleza de una cifra computacionalmente* segura viene determinada por la complejidad de cálculo de los algoritmos necesarios para romperla. Esta complejidad del cálculo se mide por el tiempo T y el espacio E que se invierte en romper el criptograma, y T y E se expresan como funciones de la medida n de la entrada del algoritmo. Más que utilizar complejidades exactas $f(n)$, se

*** Categoría a la cual pertenecen todas las cifras utilizadas en la práctica.**

suelen utilizar órdenes de magnitud O($g(n)$), de modo que $f(n) = O(g(n))$ quiere decir que hay constantes c y n_0 tales que:

$$f(n) \leq c|g(n)|; \text{ para } n \geq n_0.$$

El propósito que yace tras el uso de órdenes de magnitud es que $g(n)$ sea más simple que $f(n)$.

Cálculo del orden de magnitud de la complejidad de un algoritmo

Si la complejidad exacta de un algoritmo es $f(n) = 24n + 16$, podemos decir que $f(n) = O(n)$, ya que:

$$24n + 16 \leq 25n, \text{ para } n \geq 16.$$

De la misma manera, si $f(n)$ es un polinomio de grado t en n, podemos escribir $f(n) = O(n^t)$.

Un **algoritmo** es **polinómico** o, más exactamente, **de tiempo polinómico**, si su tiempo de ejecución es $T = O(n^t)$ para alguna constante t. Un algoritmo polinómico es: **constante**, si $t = 0$; **lineal**, si $t = 1$; **cuadrático**, si $t = 2$; etc. Asimismo, un **algoritmo** es **exponencial**, o **de tiempo exponencial**, si $T = O(t^{h(n)})$ para una constante t y un polinomio $h(n)$.

Tiempo de ejecución de diferentes algoritmos

Para n grande, las diferentes clases de complejidad temporal dan lugar a tiempos muy diferentes. Por ejemplo, supongamos que tenemos una máquina capaz de procesar 10^8 instrucciones por segundo. La tabla siguiente da el tiempo de ejecución para las diferentes clases de algoritmos mencionadas más arriba.

Clase	Complejidad	Operaciones para $n = 10^8$	Tiempo
Pol. constante	$O(1)$	1	10^{-8} segundos
Pol. lineal	$O(n)$	10^8	1 segundo
Pol. cuadrático	$O(n^2)$	10^{16}	1.000 días
Pol. cúbico	$O(n^3)$	10^{24}	$2{,}7 \cdot 10^8$ años
Exponencial	$O(2^n)$	$10^{9.030.900}$	$10^{9.030.886}$ años

Una manera de romper cifras criptográficas consiste en efectuar búsquedas exhaustivas en el espacio de claves y probar todas las claves posibles para saber si el descifrado es correcto (con sentido) o no. Si la medida del espacio de claves es $n = 2^{H(K)}$, el tiempo de ejecución de esta estrategia es $T = O(n) = O(2^{H(K)})$; es decir, el tiempo es lineal en el número de claves n, pero exponencial en la longitud de las claves $H(K)$. Por este motivo, el hecho de doblar la longitud de la clave del criptograma DES (*Data Encryption Standard*), de 56 a 112 bits, puede tener un impacto enorme en su fortaleza (aunque la distancia de unicidad sólo se doble).

Ved el DES en el subapartado 2.1 del módulo didáctico "Criptosistemas de clave compartida: cifrado en bloque" de esta asignatura.

1.3.2. Complejidad de un problema

La teoría de la complejidad clasifica un problema según el espacio y el tiempo mínimos que se requieren para resolver las instancias* más difíciles con una máquina de Turing. Una máquina de Turing es una máquina de estados finitos con una cinta infinita de lectura/escritura. Los problemas que una máquina de Turing pueda resolver en un orden de complejidad polinómico, en un sistema real serán de orden polinómico, y a la inversa.

*** Una instancia es el problema para un valor concreto de la entrada.**

Los **problemas tratables** son los que se pueden resolver en tiempo polinómico en una máquina de Turing. El resto son problemas intratables o difíciles. Un cierto tipo de problemas son tan difíciles que ni siquiera se puede escribir ningún algoritmo capaz de resolverlos; son los **problemas indecidibles**.

El conjunto de los problemas resolubles en tiempo polinómico se denomina **clase P.** La clase polinómica no determinista, **clase NP**, engloba todos los problemas que una máquina de Turing no determinista pueden resolver en tiempo polinómico, es decir, que si la máquina halla la solución, puede comprobar su corrección en tiempo polinómico. Observad que ello no significa que el problema se pueda resolver, pues no hay ninguna garantía de que la máquina acierte la respuesta correcta.

Es evidente que la clase NP incluye la clase P. Ahora bien, hay problemas en NP que parecen exigir un tiempo exponencial. Sin embargo, no se ha podido demostrar que $P \neq NP$; de momento, pues, no se puede excluir que algún día se encuentren algoritmos polinómicos para resolver todos los problemas de la clase NP.

Un **problema NP-difícil** es aquel que no se puede resolver en tiempo polinómico, si no es que $P = NP$.

Un **problema NP-completo** es aquel en el que cualquier otro problema de NP se puede reducir a un tiempo polinómico.

Si se encontrara un algoritmo polinómico para un problema NP-completo, entonces se habría demostrado que $P = NP$.

1.4. Problemas difíciles: logaritmo discreto y factorización

La seguridad de los criptosistemas de clave pública que funcionan en la práctica se basa en la dificultad computacional de algunos problemas NP de la teoría de los números. Mencionaremos dos de ellos: el problema del logaritmo

discreto, sobre el cual se sostiene el criptosistema de ElGamal, y el problema de la factorización, sobre el cual se basa el criptosistema RSA. Para poder entender la formulación de ambos problemas, hay que tener unas nociones básicas previas de teoría de grupos.

1.4.1. Grupos y elementos primitivos

Un **grupo** es un conjunto G con una operación definida, que representaremos por $*$. La operación hace corresponder a cada pareja de elementos a y b un tercer elemento $a * b$, resultado de aplicar la operación. Las cuatro propiedades de un grupo son las siguientes:

- La operación es interna: si $a, b \in$ G, entonces $a * b \in G$.
- La operación es asociativa: $(a * b) * c = a * (b * c)$.
- Existe un elemento identidad I tal que $I * a = a * I = a$ para todo $a \in G$.
- Todo $a \in G$ tiene elemento inverso, representado por a^{-1}, tal que $a * a^{-1} = a^{-1} * a = I$.

Ejemplos de grupos

Algunos ejemplos de grupos son los siguientes:

- Los enteros $\mathbb{Z}$ con la operación suma. El elemento identidad es $I = 0$ y $a^{-1} = -a$.
- Los reales $\mathbb{R}$ con la operación multiplicación. El elemento identidad es $I = 1$ y $a^{-1} = 1/a$.
- Los enteros módulo n, representados por $\mathbb{Z}_n = \{0, ..., n - 1\}$, con la suma módulo n. El elemento identidad es $I = 0$ y $a^{-1} = n - a$.
- Los enteros $\mathbb{Z}_n^* = \{m : 1 \leq m \leq n$ y $\text{mcd}(m,n) = 1\}$ con la operación multiplicación. El elemento identidad es $I = 1$ y, por otra parte, hemos visto que todo entero coprimo con n tiene un inverso multiplicativo módulo n.

En un grupo finito G, el **orden del grupo** es su cardinal $|G|$.

Sea G un grupo y $a \in G$; con las potencias de a, es decir: a^0, a^1, a^2, ..., podemos formar el conjunto:

$$< a > = \{a^i : i \geq 0\}.$$

Si $m = |G|$ es el orden de G, se cumple que $a^m = I$. Por consiguiente, la secuencia se repite al cabo de m pasos, es decir, $a^{m+1} = a$; aunque también se podría repetir antes.

Si G es un grupo y $a \in G$, el subgrupo $< a > = \{a^i: i \geq 0\}$ se denomina **subgrupo generado** por a y tiene orden $t = | < a > |$ que divide el orden $m = |G|$.

Subgrupos generados por elementos del grupo $\mathbb{Z}_9^*$

Si $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$, el orden del grupo (es decir, el tamaño del conjunto) es el número de enteros menores de 9 que son coprimos, es decir, $\phi(9) = 6$. Algunos de los subgrupos generados por los elementos de este grupo son los siguientes:

- $< 1 > = \{1\}$.
- $< 2 > = \{1, 2, 4, 8, 7, 5\}$.
- $< 4 > = \{1, 4, 7\}$.
- $< 5 > = \{1, 5, 7, 8, 4, 2\}$.

Así, pues, según qué elemento tomemos, podemos hacer la vuelta sin que lleguen a salir todos los elementos de $\mathbb{Z}_9^*$.

Un elemento $g \in G$ se denomina **primitivo** o **generador de *G*** si las potencias de g generan G. Es decir, $< g > = G$ es todo el grupo G. Igualmente, un grupo se llama **cíclico** si tiene un elemento primitivo.

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

Se sabe que si p es primo, el grupo siguiente es cíclico:

$$\mathbb{Z}_p^* = \{1, 2..., p - 1\}.$$

Observad que el orden del grupo es el número de enteros menores que p que son coprimos, es decir, $\phi(p) = p - 1$.

1.4.2. El problema del logaritmo discreto

En un grupo cíclico de orden m y generador g, para cualquier $y \in G$ hay un único $y \in \{0..., m - 1\}$ tal que $g^i = y$. Esta única i se representa por $\log_g y$ y se denomina **logaritmo discreto de *y* en la base *g***.

En criptografía es especialmente interesante el caso $G = \mathbb{Z}_p^*$ cuando p es primo y g es un generador de $\mathbb{Z}_p^*$. Entonces se pueden calcular potencias módulo p de manera rápida. Por ejemplo, con el algoritmo *exp_rapid*(*a*,*z*,*n*).

Dado $y = g^i \bmod p$, hallar el logaritmo discreto i es difícil (problema del logaritmo discreto). El mejor algoritmo actual tarda un tiempo $O(\exp[\sqrt{\ln p \ln \ln p}])$, es decir, un tiempo exponencial. Una variante reciente de este algoritmo parece requerir sólo un tiempo $O(\exp[\ln p \ln \ln p]^{1/3})$.

1.4.3. El problema de la factorización

Factorizar un entero *n* significa hallar la descomposición en factores primos. Esta descomposición existe siempre y es única.

En criptografía es especialmente interesante la factorización de enteros de la forma $n = pq$, donde p y q son primos. En este caso alguien conoce n, pero no conoce ni p ni q y querría hallarlos (problema de la factorización).

El problema de la factorización también es difícil. Los mejores algoritmos, que tardan un tiempo $O(\exp[\sqrt{\ln n \ln\ln n}])$, son el de Dixon y el de la criba cuadrática*. Si p es el divisor primo más pequeño de n, hay un algoritmo llamado *de curva elíptica* que tarda un tiempo $O(\exp[\sqrt{2\ln p \ln\ln p}])$. Este último algoritmo, pues, sólo se aconseja cuando se sospecha que uno de los dos factores de n es mucho más pequeño que el otro. Una propuesta reciente de algoritmo de criba parece tardar sólo un tiempo $O(\exp[\ln n \ln\ln n]^{1/3})$.

*** En inglés, *quadratic sieve*.**

La complejidad del cálculo de la factorización de $n = pq$ es, pues, exponencial con los mejores algoritmos que se conocen actualmente. De hecho, es sorprendente ver la semejanza existente entre los mejores algoritmos conocidos para calcular logaritmos discretos y para factorizar, teniendo en cuenta que los algoritmos utilizados para resolver un problema no tienen mucho en común con los usados para el otro problema.

2. Fundamentos de los criptosistemas de clave pública

La criptografía de clave compartida presenta tres inconvenientes:

Los criptosistemas de clave compartida se tratan en los módulos "Criptosistemas de clave compartida: cifrado en bloque" y "Criptosistemas de clave compartida: cifrado en flujo" de esta asignatura.

1) La distribución de claves: antes de empezar a comunicarse, dos usuarios tienen que elegir una clave secreta. Por lo tanto, o bien se tienen que encontrar personalmente o bien han de confiar en un canal seguro para pasarse las claves. Pero también es perfectamente posible que no se puedan encontrar personalmente y que no dispongan de ningún canal seguro.

2) La gestión de claves: en una red de n usuarios, cada par de usuarios tiene que tener su clave compartida particular, lo cual implica un total de $n(n-1)/2$ claves* para toda la red.

*** Combinaciones de n elementos tomados de dos en dos.**

3) No hay firma digital: la firma digital es el equivalente de las firmas manuales en el caso de la información electrónica; con un criptosistema de clave compartida, generalmente no existe la posibilidad de firmar los mensajes, por el hecho mismo de que todas las claves son compartidas al menos por dos usuarios.

El concepto de *criptosistema de clave pública*, que permite superar los inconvenientes anteriores, fue propuesto por W. Diffie y M.E. Hellman en el artículo "New directions in cryptography" aparecido en 1976. La idea, que se puede considerar revolucionaria, era permitir un intercambio seguro de mensajes entre emisor y receptor sin ni siquiera tener que encontrarse previamente para acordar una clave secreta común.

2.1. Concepto de *clave pública*

Un criptosistema de clave pública se orienta más a menudo a una red de usuarios que a una sola pareja. En este criptosistema, cada usuario u tiene asociada una pareja de claves $< P_u, S_u >$: la **clave pública**, P_u, que se publica con el nombre del usuario en un directorio público que todo el mundo puede leer, y la clave privada, S_u, que sólo conoce u. Las parejas de claves se generan mediante un algoritmo de generación de claves.

Para enviar un mensaje secreto de m a u, todos los integrantes de la red utilizan el mismo método:

1) Buscar P_u.

2) Calcular $c = E(P_u, m)$, en donde E es un **algoritmo público de cifrado**.

3) Enviar c al usuario u.

Al recibir el texto cifrado c, el usuario u lo puede descifrar de la manera siguiente:

1) Buscar su clave privada S_u.

2) Calcular $D(S_u,c)$, en donde D es un **algoritmo público de descifrado**.

Para que este procedimiento funcione, es preciso que $D(S_u,E(P_u,m)) = m$. De este modo, la gestión de claves se simplifica porque ahora sólo hay n pares de claves para n usuarios, en vez de las $n(n-1)/2$ claves que hacían falta con la criptografía de clave compartida.

2.2. Funciones unidireccional y unidireccional con trampa

Los criptosistemas de clave pública se basan en la existencia de algunos tipos de funciones que son difíciles de invertir.

Una **función unidireccional*** $f: M \to C$ es una función invertible que resulta fácil de calcular $f(m) = c$, pero cuya inversa $f^{-1}(c) = m$ es difícil de calcular.

Una **función unidireccional con trampa**** es una función unidireccional que puede ser fácilmente invertida cuando se conoce cierta información adicional.

*** En inglés, *one-way*.**
**** En inglés, *trapdoor one-way*.**

En las definiciones anteriores, el término "fácil" debe entenderse en el sentido de tiempo polinómico, y "difícil" en el sentido de tiempo exponencial. Curiosamente, no se sabe a ciencia cierta si hay funciones unidireccionales, y menos aún, funciones unidireccionales con trampa, aunque sí hay funciones susceptibles de ser consideradas como tales.

Posibles funciones unidireccionales

A partir de lo que hemos explicado acerca de los problemas difíciles, podrían ser unidireccionales las funciones siguientes:

- Exponencial discreta. Con el algoritmo *exp_rapid*(a,z,n) o uno similar, resulta fácil exponenciar. En cambio, se hace difícil volver atrás calculando logaritmos discretos.
- Producto de primos. Es fácil multiplicar números primos. En cambio, es difícil volver atrás factorizando el producto obtenido.

3. Intercambio de claves de Diffie-Hellman

En un artículo de 1976, W. Diffie y M.E. Hellman describieron un protocolo de intercambio de claves que evita el inconveniente de la distribución de claves. El hecho innovador es que dos usuarios pueden pactar una clave secreta compartida sobre un canal inseguro.

Hemos estudiado el problema de la distribución de claves en el apartado 2 de este módulo.

Protocolo 1: Intercambio Diffie-Hellman

Para llevar a cabo correctamente el protocolo de intercambio de Diffie-Hellman, hay que seguir los pasos siguientes:

La idea del intercambio...

... la tuvo Hellman, según explica W. Diffie:

"Finalmente, Marty [Hellman] dio el gran paso una mañana de mayo de 1976. [...] Marty me llamó y me explicó el intercambio de claves exponencial: era de una simplicidad desconcertante. Mientras lo escuchaba me di cuenta de que hacía tiempo que ya lo sabía, pero no era consciente de ello."

1) Los dos usuarios, A y B, escogen públicamente un grupo multiplicativo finito G de orden n y un generador $\alpha \in G$.

2) A genera un número aleatorio a, calcula α^a dentro de G y transmite el resultado a B.

3) B genera un número aleatorio b, calcula α^b dentro de G y transmite el resultado a A.

4) A recibe α^b y calcula $(\alpha^b)^a$ dentro de G.

5) B recibe α^a y calcula $(\alpha^a)^b$ dentro de G.

Al final del protocolo 1, A y B han pactado un elemento α^{ab} del grupo G que es común a ambos y secreto para el resto de usuarios. La seguridad del intercambio de Diffie-Hellman se basa en la dificultad del problema de Diffie-Hellman generalizado.

Problema de Diffie-Hellman generalizado (**PDHG**): el criptoanalista puede conocer toda la ejecución del protocolo 1, con lo cual obtiene G, n, α, α^a y α^b. Con esta información quiere calcular α^{ab}.

Si $G = \mathbb{Z}_p^*$, con p primo, el PDHG se llama simplemente *problema de Diffie-Hellman* (PDH). Destacamos la semejanza de este problema con el del logaritmo discreto generalizado, que enunciamos a continuación.

Problema del logaritmo discreto generalizado (**PLDG**): un criptoanalista obtiene G, n, α y α^a, y quiere calcular a.

Aunque hasta la fecha no se ha demostrado nunca que PDHG y PLDG sean equivalentes, por otra parte, no se ha encontrado cómo resolver PDHG sin resolver PLDG. Lo que está claro es que si el criptoanalista supiera resolver PLDG, entonces podría encontrar a (respectivamente, b) a partir de α^a (respectivamente, α^b) y, por consiguiente, α^{ab}.

Funcionamiento del protocolo de Diffie-Hellman

Ilustramos el funcionamiento del protocolo de Diffie-Hellman con el ejemplo siguiente:

1) A y B eligen públicamente $G = \mathbb{Z}_{53}^*$ y el generador $\alpha = 2$ (podéis comprobar que las potencias de 2 generan G).

2) A elige $a = 29$, calcula $\alpha^a = 2^{29} \bmod 53 = 45$ y envía 45 a B.

3) B escoge $b = 19$, calcula $\alpha^b = 2^{19} \bmod 53 = 12$ y envía 12 a A.

4) A recibe 12 y calcula $12^{29} \bmod 53 = 21$.

5) B recibe 45 y calcula $45^{19} \bmod 53 = 21$.

Sólo A y B saben que el número compartido es 21. Un criptoanalista enemigo ve $\mathbb{Z}_{53}^*$, 2, 45 y 12.

4. Criptosistemas de clave pública

En este apartado tratamos de los dos criptosistemas de clave pública más utilizados actualmente: el RSA y el ElGamal. El primero se basa en el problema de la factorización, mientras que el segundo lo hace en el problema del logaritmo discreto.

Lectura complementaria

l estudiante interesado en hallar más información sobre otros criptosistemas de clave pública puede consultar el libro siguiente:

D. Stinson (1995). *Cryptography: Theory and Practice*. Boca Ratón: CRC Press.

Cabe decir que existen otros criptosistemas de clave pública, como el de McEliece, el de Rabin, los de mochila, etc. Sin embargo, ninguno de ellos es una alternativa práctica al RSA y al de ElGamal, ya sea por razones de poca eficiencia (McEliece y Rabin) o bien de poca seguridad (los criptosistemas de mochila o *knapsack*).

Por simplicidad, los ejemplos que vamos a exponer, tanto de RSA como de ElGamal, se basarán en grupos multiplicativos enteros del tipo $\mathbb{Z}_n^*$. No obstante, se pueden utilizar las llamadas *curvas elípticas* como alternativa a los grupos multiplicativos enteros. La ventaja de utilizar estas curvas es que se obtienen implementaciones más rápidas de los criptosistemas.

Lectura complementaria

El estudiante interesado en las curvas elípticas puede consultar los libros siguientes:

A. Fuster, D. De la Guía, L. Hernández, F. Montoya, J. Muñoz (1997). T*écnicas criptográficas de protección de datos*. Madrid: Ra-ma.

D. Stinson (1995). *Cryptography: Theory and Practice*. Boca Ratón: CRC Press.

4.1. El criptosistema RSA

En su artículo inicial de 1976, Diffie y Hellman formularon la idea de un criptosistema de clave pública, pero no dieron ningún ejemplo práctico. La primera aplicación de la propuesta fue el criptosistema RSA, publicado por Rivest, Shamir y Adleman en 1978 en el célebre artículo "A method for obtaining digital firms and public-key cryptosystems". A continuación exponemos su algoritmo de generación de claves.

Algoritmo de generación de claves del RSA

La generación de las claves consiste en los pasos siguientes:

1) Cada usuario u escoge dos números primos p y q*. Luego calcula $n = pq$, con lo cual el grupo multiplicativo que utilizará u es $\mathbb{Z}_n^*$, de orden $\phi(n) = \phi(pq) = (p-1)(q-1)$. Para u resulta fácil calcular este orden, ya que conoce p y q.

*** Actualmente se recomienda que cada uno de éstos primos tenga más de 200 dígitos.**

2) El usuario u elige un entero positivo e que cumpla las relaciones $1 \leq e < \phi(n)$ y $\text{mcd}(e, \phi(n)) = 1$.

3) El usuario u calcula d, el inverso de e, en $\mathbb{Z}_{\phi(n)}$* con el algoritmo de Euclides extendido. Obtenemos $ed \bmod \phi(n) = 1$.

*** El inverso existe por la manera como se ha elegido *e*.**

4) La clave pública de u es el par (n, e), y su clave privada es d. Por descontado, p, q y $\phi(n)$ también se mantienen en secreto.

Si un usuario A quiere enviar un mensaje $m \in \mathbb{Z}_{n_b}^*$ a otro usuario B, utiliza la clave pública de B, (n_b, e_b) para calcular el valor $m^{e_b} \bmod n_b = c$, que se envía a B.

Para recuperar el mensaje original, B calcula:

$$c^{d_b} \bmod n_b = (m^{e_b})^{d_b} \bmod n_b = m^{e_b d_b} \bmod n_b =$$
$$= m^{1+t\phi(n_b)} \bmod n_b = (m \bmod n_b)(m^{\phi(n_b)} \bmod n_b)^t \bmod n_b = m$$

En la última igualdad se ha utilizado el teorema de Euler, que garantiza que $m^{\phi(n_b)} \bmod n_b = 1$ si $\text{mcd}(m, n_b) = 1$ (y esto es casi seguro que pase porque n_b es muy grande y sólo tiene dos divisores, p_b y q_b).

Codificación y descodificación de un mensaje utilizando una clave pública

Consideremos una codificación del alfabeto que transforma las letras A-Z en los números del 0 al 25. Supongamos que:

a) El usuario B ha elegido los primos $p_b = 281$ y $q_b = 167$, y ha obtenido $n_b = 281 \cdot 167 = 46.927$ y utiliza el grupo $\mathbb{Z}_{46.927}^*$.

b) El orden de este grupo es $\phi(46.927) = 280 \cdot 166 = 46.480$. B elige $e_b = 39.423$ y comprueba que $\text{mcd}(39.423, 46.480) = 1$. Entonces determina el inverso de 39.423 módulo 46.480 y obtiene $d_b = 26.767$.

c) La clave pública de B es $(n_b, e_b) = (46.927, 39.423)$ y el resto de valores se mantienen secretos.

Para enviar un mensaje a B, hay que mirar primero su longitud: el mensaje tiene que pertenecer al grupo en que trabajamos; en este caso, no puede superar $n_b = 46.927$. Por otra parte, codificamos cada letra en base 26 y como $26^3 = 17.576 < n_b < 456.976 = 26^4$, el mensaje puede tener hasta tres letras. Para enviar un mensaje más largo, habría que partirlo en bloques de tres letras. Naturalmente, las n_b utilizadas en la práctica son mucho mayores (tienen al menos 400 dígitos decimales).

Imaginemos que enviamos a B el mensaje YES. Codificado en base 26, adquiere la forma siguiente:

$$Y \cdot 26^2 + E \cdot 26 + S = 24 \cdot 26^2 + 4 \cdot 26 + 18 = 16.346 = m$$

A continuación ciframos m con la clave pública de B, y obtenemos:

$$c = m^{e_b} \bmod n_b = 16.346^{39.423} \bmod 46.927 = 21.166$$

Y 21.166 es la codificación en base 26 de las letras BFIC. Éste es el mensaje cifrado expresado con letras.

Supongamos que B ha recibido BFIC. Para volver al mensaje original, codifica el texto recibido en base 26 y obtiene 21.166. A continuación recupera:

$$m = c^{d_b} \bmod n_b = 21.166^{26.767} \bmod 46.927 = 16.346.$$

Y, por último, descodifica m y obtiene el texto original: YES.

Una cuestión técnica es que el algoritmo de generación de claves RSA requiere encontrar primos muy grandes (200 dígitos decimales). Hay diversos métodos eficientes para confirmar que un número grande es primo; por ejemplo, los

tests de primalidad de Solovay-Strassen, Miller-Rabin y Goldwasser-Kilian. Aquí detallamos el primero de estos tests y remitimos al estudiante interesado por conocer otros métodos a los libros que se indican en la bibliografía.

Obtención de números primos con el test de primalidad de Solovay-Strassen

Para encontrar un número primo suficientemente elevado, los autores del RSA recomiendan generar números aleatorios hasta hallar uno que sea probablemente primo. Una forma de comprobar la primalidad de un número b, consiste en generar cien números aleatorios $a \in [1, b-1]$ y verificar si todos ellos pasan el test de Solovay-Strassen:

$$\text{mcd}(a, b) = 1 \text{ y } J(a,b) \bmod b = a^{(b-1)/2} \bmod b,$$

donde $J(a,b)$ es el **símbolo de Jacobi**.

Si b es primo, la igualdad anterior se cumple para todo $a \in [1, b-1]$. Si b no es primo, la ecuación se cumple con una probabilidad máxima de 1/2 para cada a, de modo que la probabilidad de que cien números a diferentes pasen el test si b no es primo es de sólo $1/2^{100}$.

Cuando mcd(a, b) = 1, el símbolo de Jacobi $J(a,b)$ se puede calcular de manera eficiente con el algoritmo recursivo siguiente:

Algoritmo *J*(*a*,*b*)

```
begin
    if a = 1 then J := 1
    else if a mod 2 = 0 then
    begin
        if (b * b – 1) / 8 mod 2 = 0 then
        J := J(a/2,b) else J := –J(a/2,b)
    end
    else if (a – 1) * (b – 1) / 4 mod 2 = 0 then
    J := J(b mod a,a) else J := –J(b mod a,a)
end
```

4.1.1. Velocidad del RSA

En la práctica, para que el proceso de cifrado con el RSA sea más rápido, se suele escoger un exponente público e pequeño; a menudo se toma $e = 3$. Esta elección no compromete la seguridad del sistema, que descansa en el secreto de la clave privada d.

Si nos fijamos en la velocidad que se consigue en las implementaciones RSA, debemos reconocer dos hechos:

a) En *software*, el criptosistema de clave compartida DES llega a funcionar cien veces más deprisa que las mejores implementaciones de RSA.

b) En *hardware*, el DES llega a ser entre 1.000 y 10.000 veces más rápido que el RSA.

A pesar de su relativa lentitud, el RSA y los otros criptosistemas de clave pública se utilizan porque también permiten firmar digitalmente los mensajes enviados. Normalmente se hace por medio de una combinación de clave pública y de clave secreta conocida como **sobre digital**, que ofrece la velocidad de la clave compartida y la flexibilidad de la clave pública. Para trabajar en esta modalidad hay que seguir los pasos siguientes:

La firma digital se estudia en el apartado 1 del módulo "Firmas digitales" de esta asignatura.

1) El usuario *A* cifra el mensaje *m* con un criptosistema de clave compartida (por ejemplo, el DES) y una clave aleatoria.

2) *A* envía a *B* el mensaje cifrado *c*. A continuación, *A* envía a *B* la clave aleatoria cifrada con el RSA bajo la clave pública de *B*.

3) Gracias a su clave privada RSA, *B* recupera la clave aleatoria DES y la utiliza para descifrar el mensaje *c* y recuperar *m*.

4.1.2. Seguridad del RSA

Para romper el criptosistema RSA, un criptoanalista tiene que encontrar el valor d, que es la trampa de la función unidireccional $f(m) = m^e \bmod n$, pero para ello hay que conocer $\phi(n)$, que no es fácil de encontrar a partir de n, a no ser que se conozca la factorización $n = pq$.

De acuerdo con lo dicho, podemos intuir que la seguridad del criptosistema RSA se basa en el problema de la factorización.

Sin embargo, no se ha podido demostrar que la única manera de hallar d a partir de e pase por factorizar n.

Como la seguridad del RSA está relacionada con el problema de la factorización, conviene elegir los dos primos p y q que compongan n de modo que la factorización sea tan difícil como se pueda. Por lo general, se suele escoger p y q aleatoriamente entre los primos que cumplen las condiciones siguientes:

1) p y q deben ser de magnitud parecida, pero sin estar demasiado próximos.

2) $(p-1)$ y $(q-1)$ han de tener factores primos grandes.

3) mcd$(p-1, q-1)$ tiene que ser pequeño.

Justifiquemos cada una de estas condiciones:

1) Si p y q fueran demasiado próximos, con $p > q$, entonces $(p-q)/2$ sería pequeño y $(p+q)/2$ sólo sería un poco mayor que $\sqrt{n}$. Por otra parte:

$$\frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{4}$$

El miembro de la izquierda de la ecuación es un cuadrado perfecto. Para factorizar n, basta probar los enteros $x > \sqrt{n}$ hasta hallar uno tal que $x^2 - n$ sea un cuadrado perfecto y^2. Entonces, $p = x + y$ y $q = x - y$. El ataque no tarda mucho en triunfar, porque la x que buscamos es $(p+q)/2$ y sabemos que no es mucho mayor que $\sqrt{n}$.

2) La segunda condición sobre p y q exige que $(p-1)$ y $(q-1)$ tengan factores primos grandes. Si esta condición no se diera, todos los factores de $\phi(n) = (p-1)(q-1)$ serían pequeños. Si k es una cota superior de los factores primos de $\phi(n)$, es fácil probar todos los valores v candidatos a ser $\phi(n)$: hay que probar combinaciones de factores primos menores que k en que cada factor r va elevado a un exponente entre 1 y $\lfloor \log_v n \rfloor$. Para comprobar si lo hemos hecho bien, podemos elevar un criptograma m^e a $(v+1)/e$ si este exponente es entero. Si $v = \phi(n)$, al elevar m^e a $(\phi(n)+1)/e$, deberemos recuperar el mensaje en claro m (teorema de Euler) y habremos confirmado el acierto.

3) La tercera condición (sobre el mcd(p, q)) se justifica porque, al ser $p-1$ y $q-1$ pares, $\phi(n)$ es divisible por 4. Si mcd$(p-1, q-1)$ fuera grande, entonces mcm$(p-1, q-1) = mcm$ sería pequeño en comparación con $\phi(n) = (p-1)(q-1)$. Conviene que nos demos cuenta de que cualquiera inverso de e módulo mcm sirve para descifrar, y que encontrar mcm es fácil si se sabe que es pequeño.

Una manera de elegir p y q de modo que cumplan las tres condiciones anteriores es escoger dos primos r y r' grandes y del mismo orden de magnitud, con $r' \sim 2r$, y tales que $p = 2r + 1$ y $q = 2r' + 1$ sean primos.

La primera condición se cumple: p y q son de magnitud parecida, pero q es el doble de p. La segunda condición también se cumple, ya que $p - 1 = 2r$ y $q - 1 = 2r'$ tienen factores primos grandes, r y r' respectivamente. Finalmente, vemos que mcd$(p-1, q-1) = 2$ es pequeño, cosa que satisface la tercera condición.

4.2. El criptosistema ElGamal

En 1985, T. ElGamal publicó un criptosistema de clave pública basado en la exponenciación discreta sobre un grupo multiplicativo $\mathbb{Z}_p^*$, en que p es un número primo grande (400 dígitos decimales, por lo menos). Es fácil generalizar la propuesta de ElGamal a un grupo finito G.

Protocolo de generación de claves de ElGamal

Para generar las claves que se utilizan en el ElGamal hay que seguir los pasos siguientes:

1) Se escoge un grupo finito G y un elemento $\alpha \in G$ que no tiene que ser necesariamente un generador.

2) Cada usuario A elige un número aleatorio a, que será su clave privada, y calcula α^a en G, que será su clave pública.

Si un usuario A quiere enviar un mensaje $m \in G$ a un usuario B, entonces hace lo siguiente:

1) A genera un número aleatorio v y calcula α^v en G.

2) A mira la clave pública de B, α^b, y calcula $(\alpha^b)^v$ y $m \cdot \alpha^{bv}$ en G.

3) A envía a B el par $(\alpha^v, m \cdot \alpha^{bv})$.

Para recuperar el mensaje original, B tiene que hacer lo siguiente:

1) B calcula $(\alpha^v)^b$ en G.

2) B obtiene m sólo dividiendo la segunda parte del criptograma:

$$\frac{m \cdot \alpha^{vb}}{\alpha^{vb}}$$

Codificación y descodificación de un mensaje con el criptosistema de ElGamal

Para ilustrar el funcionamiento del ElGamal, supongamos que se ha elegido el grupo $G = \mathbb{Z}^*_{15.485.863}$ y $\alpha = 7$ (de hecho, las potencias de 7 generan todo G).

El usuario B ha escogido $b = 21.702$ y ha calculado su clave pública de la manera correspondiente:

$$\alpha^b = 7^{21.702} \bmod 15.485.863 = 8.890.431.$$

Supongamos que A quiere enviar a B el mensaje PACO. El mensaje codificado en base 26 tiene la forma siguiente:

$$\text{PACO} = 15 \cdot 26^3 + 0 \cdot 26^2 + 2 \cdot 26 + 14 = 263.706 = m.$$

Después, A elige el número aleatorio $v = 480$ y calcula:

$$\alpha^v = 7^{480} \bmod 15.485.863 = 12.001.315.$$

A continuación, *A* calcula:

$$(\alpha^b)^v = 8.890.431^{480} \bmod 15.485.863 = 9.846.598$$

y codifica el mensaje:

$$m \cdot (\alpha^b)^v = 263.706 \cdot 9.846.598 \bmod 15.485.863 = 14.893.663.$$

Luego, *A* envía a *B* los números 12.001.315 y 14.893.663 o las letras del alfabeto resultantes de descodificar estos números en base 26. *B* recibe el mensaje (si está en formato letra, tendrá que codificarlo en base 26), y a continuación calcula:

$$(\alpha^v)^b = 12.001.315^{21.702} \bmod 15.485.863 = 9.846.598.$$

Después *B* ha de calcular $(m \cdot \alpha^{vb})/\alpha^{vb}$. Para ello, debe encontrar el inverso de α^{vb} módulo 15.485.863. Con el algoritmo de Euclides extendido hallamos $\alpha^{-vb} = 14.823.281$.

Finalmente, descodifica el mensaje:

$$m = \frac{m \cdot \alpha^{vb}}{\alpha^{vb}} = (14.893.663 \cdot 14.823.281) \bmod 15.485.863 = 263.706.$$

4.2.1. Velocidad del ElGamal

Igual que el RSA, el criptosistema de ElGamal tiene que hacer exponenciaciones. La diferencia es que las exponenciaciones de cifrado no dependen del mensaje *m*, sino de un exponente aleatorio *v* elegido por el emisor. Por lo tanto, se pueden calcular antes de enviar el mensaje. De hecho, la única operación que hay que hacer forzosamente en el mismo momento en que se cifra *m* es un producto (mucho más rápido que una exponenciación). En consecuencia, el cifrado del ElGamal puede ser más rápido que el cifrado del RSA.

El descifrado del ElGamal requiere una exponenciación y un producto, es decir, prácticamente el mismo tiempo que un descifrado del RSA, que requería una exponenciación. Un pequeño inconveniente del ElGamal es que el criptosistema es una pareja de enteros, que suelen ocupar el doble de bits que el mensaje en claro (por lo tanto, una ampliación del texto cifrado)

4.2.2. Seguridad del ElGamal

El criptosistema de ElGamal utiliza la exponenciación discreta como función unidireccional. Más en concreto, la función unidireccional para enviar al usuario *B* es $f(m) = (\alpha^v, m \cdot \alpha^{bv})$ y la trampa es la clave privada *b*. La seguridad del criptosistema de ElGamal se basa en la dificultad del problema que exponemos a continuación.

Problema de ElGamal (PEG): el criptoanalista que escucha la transmisión cifrada de *A* hacia *B* conoce *G*, *n*, α, α^a, α^b, α^v y $m \cdot \alpha^{vb}$. Él quiere calcular *m*.

Es fácil darse cuenta de que el PEG equivale al problema de Diffie-Hellman. Y, como en el caso del PDHG, no se ha demostrado que el PEG sea equivalente al problema del logaritmo discreto (PLDG), pero tampoco se ha hallado hasta ahora ningún ataque general contra el ElGamal que no pase por calcular logaritmos discretos. Es evidente que si el criptoanalista supiera resolver logaritmos discretos, podría hallar la clave privada b del destinatario B a partir de la clave pública α^b, descifrar el criptograma con la clave b y recuperar m.

El problema de Diffie-Hellman se explica en el apartado 3 de este módulo.

Cabe destacar que la propuesta original de ElGamal utilizaba $G = \mathbb{Z}_p^*$ para p muy grande y α un generador de G. Si lo generalizamos a cualquier grupo G y cualquier elemento $\alpha \in G$, tenemos que asegurarnos de que el subgrupo cíclico $< \alpha >$, generado por las potencias de α, sea suficientemente grande para que continúe resultando difícil calcular logaritmos.

Resumen

La **aritmética modular** y, más generalmente, la **teoría de los números** son la base sobre la cual se implementan los criptosistemas de clave pública.

El **intercambio de claves de** Diffie-Hellman permite a dos usuarios pactar una clave secreta compartida sobre un canal inseguro.

Si utilizamos criptografía de clave pública en una red de n usuarios, basta con que cada usuario tenga dos claves (una pública y otra privada) para permitir la comunicación confidencial entre cualquier par de usuarios. Recordemos que con criptografía de clave compartida son necesarias $n(n-1)/2$ claves compartidas. En efecto, para enviar un mensaje confidencial a un usuario B, cualquier otro usuario A tiene que cifrar el mensaje bajo la clave pública de B. Sólo B podrá recuperar el mensaje original después de descifrar el criptograma recibido con su clave privada.

Los criptosistemas de clave pública prácticos ofrecen seguridad computacional. La **fortaleza** de una cifra segura computacionalmente queda determinada por la complejidad de cálculo de los algoritmos necesarios para romperla.

Los dos criptosistemas más utilizados son el **RSA** y el **ElGamal**, que fundamentan su seguridad en los problemas de la factorización y del logaritmo discreto, respectivamente.

A pesar de que se puede incrementar la rapidez del RSA y del ElGamal si se implementan con curvas elípticas, el hecho es que son considerablemente más lentos que los criptosistemas de clave compartida de tipo DES. A pesar de ello, su simplicidad en la gestión de claves y la posibilidad de usarlos para hacer firmas digitales los hacen atractivos. El **sobre digital** es una técnica que combina las ventajas de la clave pública y la clave compartida: el mensaje se envía cifrado con un criptosistema de clave compartida bajo una clave aleatoria, y a continucación se envía la clave aleatoria cifrada con la clave pública del destinatario.

Actividades

1. Buscad en Internet implementaciones de RSA y de ElGamal.

2. Programad el algoritmo *exp_rapid*(*n*,*z*,*a*).

3. Programad el algoritmo de Euclides extendido.

4. Programad el test de primalidad de Solovay-Strassen.

5. Escribid un programa que simule el intercambio de Diffie-Hellman con números pequeños (por ejemplo, p de 8 dígitos decimales). Simulad la ejecución de este protocolo para dos usuarios.

6. Escribid programas que implementen el cifrado/descifrado del RSA con números pequeños (por ejemplo, un módulo n de 8 dígitos decimales). Utilizad el algoritmo de exponenciación rápida, el algoritmo de Euclides extendido y el test de primalidad que habéis programado en las actividades 3, 4 y 5, respectivamente.

7. Escribid programas que implementen el cifrado/descifrado del ElGamal con números pequeños (por ejemplo, un p de 8 dígitos decimales). Utilizad el algoritmo de exponenciación rápida y el algoritmo de Euclides extendido que habéis programado en las actividades 2 y 3, respectivamente.

8. Buscad en Internet información sobre librerías multiprecisión. Podéis hacer una búsqueda del término 'multiprecision' o bien buscar librerías concretas como *gpm* o *LiDIA*. Se trata de librerías que contienen rutinas aritméticas para sumar, restar, exponenciar, etc., números de longitud arbitrariamente grande (centenares de dígitos decimales). Estas librerías son necesarias para crear aplicaciones de *software* serias de los criptosistemas RSA y ElGamal.

Ejercicios de autoevaluación

1. Demostrad el principio de la aritmética modular (teorema 1).

2. Con el algoritmo de Euclides básico (algoritmo mcd(a, n)), calculad paso a paso el máximo común divisor de 2.678 y 346.

3. Hallad el inverso de 24 en $\mathbb{Z}_{37}^*$.

4. Comprobad que $80 = \sum_{d|80} \phi(d)$.

5. Para cada ecuación de la forma $ax \bmod n = b$ indicada a continuación, hallad el valor x en el rango $[0, n-1]$.
- $5x \bmod 17 = 1$.
- $19x \bmod 26 = 1$.
- $17x \bmod 100 = 1$.
- $17x \bmod 100 = 10$.

6. A partir de las congruencias $x \bmod 3 = 2$ y $x \bmod 5 = 4$, calculad x utilizando el teorema chino del residuo. ¿En qué rango se halla x?

7. En el criptosistema de ElGamal, tomad $p = 23$, $\alpha = 5$ (α es primitivo en $\mathbb{Z}_{23}^*$). Detallad cómo un usuario B generaría su par de claves, pública y privada. Detallad paso a paso cómo (usuario A) cifraríais vuestra edad en años módulo 23 para enviarla al usuario B.

Solucionario

Ejercicios de autoevaluación

1. Para empezar, escribimos:
- $a_1 = k_1 n + r_1$,
- $a_2 = k_2 n + r_2$,

donde $r_1, r_2 \in [0, n-1]$.

a) Para la suma tenemos:

$$\begin{aligned}(a_1 + a_2) \bmod n &= [(k_1 n + r_1) + (k_2 n + r_2)] \bmod n = \\ &= [(k_1 + k_2)n + (r_1 + r_2)] \bmod n = [r_1 + r_2] \bmod n = \\ &= [(a_1 \bmod n) + (a_2 \bmod n)] \bmod n\end{aligned}$$

b) La sustracción se demuestra igual que la suma.

c) Finalmente, para la multiplicación tenemos:

$$\begin{aligned}(a_1 \cdot a_2) \bmod n &= [(k_1 n + r_1) \cdot (k_2 n + r_2)] \bmod n = \\ &= [(k_1 k_2 n + r_1 k_2 + r_2 k_1)n + r_1 r_2] \bmod n = [r_1 \cdot r_2] \bmod n = \\ &= [(a_1 \bmod n) \cdot (a_2 \bmod n)] \bmod n\end{aligned}$$

2. Hay que hacer una tabla con el estado de las variables a cada paso del algoritmo mcd(a, n), y al final obtenemos que el máximo común divisor es 2.

3. Hay que hacer una tabla con el estado de las variables a cada paso del algoritmo *inv*(a, n), y al final obtenemos que el inverso de 24 módulo 37 es 17.

4. Los divisores de $80 = 2^4 \cdot 5$ son 1, 2, 4, 5, 8, 10, 16, 20, 40 y 80.
Por otra parte, $\phi(1) = 1$, $\phi(2) = 2 - 1 = 1$, $\phi(4) = 2(2-1) = 2$, $\phi(5) = 5 - 1 = 4$, $\phi(8) = 2^2(2-1) = 4$, $\phi(10) = (2-1)(5-1) = 4$, $\phi(16) = 2^3(2-1) = 8$, $\phi(20) = 2(2-1)(5-1) = 8$, $\phi(40) = 2^2(2-1)(5-1) = 16$ y $\phi(80) = 2^3(2-1)(5-1) = 32$

Si sumamos todos los valores de ϕ obtenemos:

$$\sum_{d|80} \phi(d) = 1 + 1 + 2 + 4 + 4 + 4 + 8 + 8 + 16 + 32 = 80.$$

5. Para resolver $5x \bmod 17 = 1$ hay que encontrar el inverso de 5 módulo 17. Este inverso existe, ya que mcd(5, 17) = 1, y con el algoritmo de Euclides extendido hallamos $x = 7$.
La segunda ecuación y la tercera se resuelven igual si nos damos cuenta de que los inversos respectivos existen, porque mcd(19, 26) = 1 y mcd(17, 100) = 1.
Para resolver la ecuación $17x \bmod 100 = 10$, primero dividimos el miembro de la derecha por 10 y lo dejamos a 1. A continuación, resolvemos $17y \bmod 100 = 1$, lo cual implica hallar el inverso y de 17 módulo 100. Con el algoritmo de Euclides extendido obtenemos $y = 53$. Multiplicamos y por 10 módulo 100 y encontramos la solución x de la ecuación original:

$$x = 10y \bmod 100 = 10 \cdot 53 \bmod 100 = 30.$$

6. Los módulos de las ecuaciones son coprimos, de modo que podemos utilizar el teorema chino del residuo. Encontramos y_1 como el inverso de $(3 \cdot 5)/3 = 5$ módulo 3; tenemos $y_1 = 2$. Encontramos y_2 como el inverso de $(3 \cdot 5)/5 = 3$ módulo 5; tenemos $y_2 = 2$. Entonces:

$$x = \left[\left(\frac{n}{d_1}\right) y_1 x_1 + \left(\frac{n}{d_2}\right) y_2 x_2\right] \bmod n = \left[\frac{15}{3} \cdot 2 \cdot 2 + \frac{15}{5} \cdot 2 \cdot 4\right] \bmod 15 = 14,$$

x tenía que estar en el rango [1, 15 − 1], es decir, no podía ser superior a 14.

7. Tenemos $p = 23$ y $\alpha = 5$. Supongamos que el usuario B escoge su clave privada $b = 3$. Entonces calcula su clave pública como $\alpha^b = 53 \bmod 23 = 10$. Supongamos que nuestra edad en años módulo 23 es $m = 11$. Para cifrar m escogemos aleatoriamente v, por ejemplo $v = 6$, y calculamos $\alpha^v = 5^6 \bmod 23 = 8$. Tomamos la clave pública de B y calculamos $(\alpha^b)^v = 106 \bmod 23 = 6$. Después calculamos $m\,\alpha^{bv} = (11 \cdot 6) \bmod 23 = 20$. El criptograma que se tiene que enviar a B es (8, 20).

Glosario

algoritmo de tiempo exponencial *m* Algoritmo cuyo tiempo de ejecución es $T = O(t^{h(n)})$ para una constante t y un polinomio $h(n)$.

algoritmo de tiempo polinómico *m* Algoritmo cuyo tiempo de ejecución es $T = O(n^t)$ para una constante t.

cíclico *m* Grupo que contiene un elemento primitivo o generador.

complejidad de un algoritmo *f* Tiempo de cálculo y espacio de almacenaje que requiere un algoritmo determinado.

complejidad de un problema *f* Espacio y tiempos mínimos que son necesarios para resolver las instancias más difíciles del problema con una máquina de Turing.

criptosistema de clave pública *m* Criptosistema en donde cada usuario tiene una clave pública y una privada; con este criptosistema una pareja de usuarios se puede comunicar confidencialmente sin compartir una clave secreta.

elemento primitivo *m* Elemento generador.

ElGamal Criptosistema de clave pública basado en el problema del logaritmo discreto.

factorización *f* Problema que consiste en encontrar los factores primos de un entero y que actualmente se considera intratable.

función unidireccional *f* Función unidireccional invertible que es fácil de calcular en sentido directo y difícil de calcular en sentido inverso.

función unidireccional con trampa *f* Función que es fácil de calcular en sentido inverso sólo si se conoce una información adicional.

generador *m* Elemento de un grupo cuyas potencias sucesivas generan todo el grupo.

grupo *m* Conjunto cerrado bajo una cierta operación asociativa, dentro del cual todos los elementos tienen inverso.

orden de un grupo *m* Número de elementos, finito o infinito, que contiene un grupo.

primo seguro *m* Primo que cumple los requisitos de seguridad establecidos por el RSA.

problema del logaritmo discreto *m* Problema que consiste en invertir la exponenciación modular y que actualmente se considera intratable.

problema difícil *m* Manera de aludir a un problema intratable.

problema indecidible *m* Problema para el cual no es posible encontrar un algoritmo que lo resuelva.

problema intratable *m* Problema que no se puede resolver en tiempo polinómico con una máquina de Turing.

problema tratable *m* Problema que se puede resolver en tiempo polinómico con una máquina de Turing.

RSA Criptosistema de clave pública muy utilizado, publicado por Rivest, Shamir y Adleman en 1978; se basa en el problema de la factorización.

sobre digital *m* Técnica que combina la criptografía de clave compartida y la de clave pública con el fin de aprovechar la velocidad de la primera y la flexibilidad de la segunda.

subgrupo generado *m* Conjunto de potencias diferentes del elemento, calculadas con la operación del grupo.

test de primalidad *m* Procedimiento para determinar, a menudo de manera probabilística, si un entero es primo.

Bibliografía

Denning, D. E. (1982). *Cryptography and data security*. Reading: Addison-Wesley.

Diffie, W.; Hellman, M. (1976). "New directions in cryptography". *IEEE transactions on informatio theory* (núm. 6, vol. 22, pág. 644-654).

ElGamal, T. (1985). "A public-key cryptosystem and a signature scheme based on discrete logarithms". *IEEE transactions on information theory* (vol. 31, pág. 469-472).

Fuster, A.; de la Guía, D.; Hernández, L.; Montoya, F.; Muñoz, J. (1997). *Técnicas criptográficas de protección de datos*. Madrid: Ra-ma.

Rivest, R. L.; Shamir, A.; Adleman, L. (1978). "A method for obtaining digital signatures and public-key cryptosystems". *Communications of the ACM* (núm. 2, vol. 21, pág. 120-126).

Stinson, D. (1995). *cryptography: theory and practice*. Boca Ratón: CRC Press.