El problema del logaritmo discreto en criptografía

José Galaviz Casas

Facultad de Ciencias, UNAM

Índice general

1. Introducción

1.1. Motivación

A lo largo de toda la historia de la criptografía persistió un problema aparentemente insoluble: el del intercambio de clave. En todo sistema criptográfico existen un emisor y un receptor de los mensajes, aquellos que pertenecen al mismo bando en el conflicto que presupone la criptografía. Emisor y receptor intercambian mensajes cifrados mediante algún mecanismo que permite ocultar al enemigo el significado de dichos mensajes, aún cuando estos fueran interceptados. Asociada con el sistema criptográfico debe haber una clave que determina el mapeo existente entre el mensaje en texto claro y el criptograma que le corresponde. Es el desconocimiento de esta clave lo que hace seguro al sistema, es la ausencia de este elemento lo que impide al enemigo comprender los mensajes que se intercambian el emisor y el receptor.

En los sistemas simétricos, históricamente los más comunes, la clave para cifrar y para descifrar son la misma, así que este único elemento debe ser compartido exclusivamente por el emisor y el receptor, deben acordarlo en algún momento y además, deben hacerlo fuera del sistema criptográfico, ya que el canal del comunicación que comparten dentro de este sistema está permanentemente intervenido por el enemigo. Las partes deben entonces acordar una reunión o usar un mensajero autorizado o comunicarse por un medio seguro (cuya existencia haría innecesaria, *de facto*, la comunicación criptográfica); soluciones, todas ellas, imprácticas y riesgosas, cuando no imposibles. Este es el llamado *problema de intercambio de clave*.

Fue hasta 1976 que el añejo problema del intercambio de clave pudo resolverse. Whitfield Diffie, Martin Hellman y Ralph Merkle idearon el protocolo que hoy lleva su nombre: Diffie-Hellman o Diffie-Hellman-Merkle y que permite al emisor y al receptor acordar una clave secreta usando el canal inseguro del sistema criptográfico. El intruso, a pesar de conocer todo lo que se dice en el canal, es incapaz de descubrir la clave secreta acordada

por aquellos o al menos es incapaz de hacerlo en un tiempo razonable.

Para formular el protocolo se requirió la invención de un concepto puramente computacional: el de función de un sólo sentido. Una función invertible que es fácil de calcular sobre algún elemento de su dominio, pero cuya inversa es difícil de calcular. La clave aquí es el uso de la *dificultad*. El concepto de función invertible es matemático: toda función biyectiva es invertible, no se menciona nunca qué tan difícil es calcular la función o la inversa, ese detalle es computacional ya que involucra la complejidad del algoritmo que calcula. Que un cálculo sea *difícil* significa que el mejor algoritmo que se puede tener para llevarlo a cabo, consume tanto tiempo en general, que carece de sentido pensar en ejecutarlo para resolver un caso práctico. Estamos ante un problema en el que se aplica perfectamente la frase popular de *encontrar una aguja en un pajar*.

Diffie, Hellman y Merkle aprovecharon la existencia de un problema de este tipo para construir un protocolo que permite, al emisor y al receptor de un sistema criptográfico, acordar una clave secreta sólo conocida por ellos a pesar de usar un canal inseguro para acordarla. Este es, sin duda, uno de los más importantes avances en el campo de la criptografía ya que ha permitido en el mundo moderno implementar, por ejemplo, el esquema usual de comercio electrónico, en el que el consumidor puede enviar al vendedor la información confidencial que lo identifica y lo hace sujeto de crédito, por ejemplo, sin que esta información esté, en principio, en riesgo de ser usada indebidamente por terceros.

En este documento revisaremos el trasfondo del problema en el que se basaron Diffie, Hellman y Merkle par construir su protocolo y analizaremos, tanto el protocolo mismo, como los medios usados para tratar de romper la seguridad de los sistemas basados en él.

1.2. Grupos

Uno de los conceptos más importantes y útiles en matemáticas es el de *grupo*. Un grupo es una estructura algebraica (es decir, un conjunto con una o varias operaciones definidas entre sus elementos y que satisface ciertos requisitos establecidos) con una operación que puede ser denotada como una suma (en cuyo caso hablamos de un grupo aditivo) o bien puede denotarse como un producto (en cuyo caso se denomina grupo multiplicativo). Realmente, como siempre sucede en la matemática formal, no importa si la operación es pensada como suma o producto, bastaría con cambiar de nombre algunas cosas y la estructura del grupo permanece invariante.

Definición 1. Un grupo $(G, +)$ es un conjunto no vacío G, junto con una operación binaria '+', en el que se satisfacen los siguientes requisitos:

- Cerradura: para cualesquiera $a, b \in G$, $a + b$ es un único elemento G.
- Asociatividad: dados $a, b, c \in G$, se tiene que:

$$a + (b + c) = (a + b) + c$$

- Identidad: existe un único elemento distinguido $e \in G$ llamado identidad tal que, para toda $a \in G$: $a + e = e + a = a$.
- Inverso: para cada elemento $a \in G$ existe un elemento distinguido $a' \in G$ tal que: $a + a' = e$

En un grupo aditivo suele usarse el símbolo 0 (cero) para denotar al neutro aditivo y $-a$ para denotar al inverso aditivo de a; en un grupo multiplicativo se escoge normalmente a 1 (uno) para el neutro multiplicativo y con a^{-1} se denota al inverso multiplicativo. Cuando la operación en el grupo es conmutativa, se dice que el grupo es *abeliano*.

El conjunto de los enteros $\mathbb{Z}$ junto con la suma, forma un grupo aditivo, igual que los números racionales con la suma, ambos casos son de grupos infinitos. Los números reales son un grupo (de hecho más que eso) con la suma y también, si les quitamos el cero, lo son con el producto. También es un caso de grupos infinitos. En cambio el conjunto de los enteros módulo 4 con la suma, por ejemplo, es un caso de grupo finito. Cuando un grupo es finito, al número de elementos en él se le denomina *orden del grupo*.

No podemos, claro está, cambiar arbitrariamente de conjunto o de operación y pretender que el resultado siga siendo un grupo. Si nuestro conjunto es el de los números naturales $\mathbb{N}$ y la operación a considerar es la resta, por ejemplo, entonces no necesariamente $a - b$ es un elemento de $\mathbb{N}$ (no lo es si $a > b$), lo que viola la cerradura; así que se debe tener cuidado con la elección del conjunto y la operación. En algunas definiciones se omite la propiedad de cerradura explícita y se prefiere calificar a la operación como *binaria interna*, lo que contempla, de hecho, la cerradura, en contraste con las operaciones *externas* como la resta en nuestro ejemplo con los naturales.

Ejemplo 1.2.1. Algunos ejemplos rápidos:

- Ni $(\mathbb{N}, +)$, ni $(\mathbb{N}, \cdot)$ son un grupo.
- $(\mathbb{Z}, +)$ sí lo es pero $(\mathbb{Z}, \cdot)$ no.

- $(\mathbb{Q}, +)$ también es grupo y $(\mathbb{Q}, \cdot)$ lo sería si quitamos al cero.
- $(\mathbb{R}, +)$ es grupo y $(\mathbb{R}, \cdot)$, igual que en el caso anterior, sólo si quitamos al cero.

$\triangleleft$

En general $(\mathbb{Z}_m, +)$ es grupo para toda m, pero $(\mathbb{Z}_m, \cdot)$ no lo es, a menos que m sea primo y le quitemos al cero, porque en ese caso todos los elementos en $\mathbb{Z}_m$ son primos relativos con el módulo y por tanto tienen inverso multiplicativo.

Ejemplo 1.2.2. El conjunto de los enteros no negativos menores que n junto con la operación de suma módulo n forman un grupo para toda n. Por ejemplo, usando $\mathbb{Z}_4$:

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\triangleleft$

Ejemplo 1.2.3. El otro posible grupo de orden 4 es el llamado 4-grupo de Klein:

$*$	**1**	**a**	**b**	**c**
1	1	a	b	c
a	a	1	c	b
b	b	c	1	a
c	c	b	a	1

$\triangleleft$

1.3. Grupos cíclicos

Existe un tipo especial de grupo, en el que todos los elementos pueden ser generados a partir de uno sólo de ellos usando la operación del grupo,

es decir en un grupo multiplicativo de este tipo, G habrá un elemento g tal que cualquier otro elemento e puede escribirse como $e = g \cdot g \cdots \cdot g = g^k$ para alguna k; si se tratara de un grupo aditivo entonces diríamos que $e = g + g \cdots + g = k\, g$.

En el caso de $(\mathbb{Z}_m, +)$, sabemos que el 1 es un generador ya que sumándolo consigo mismo módulo m obtenemos a todos los demás miembros del grupo.

También $(\mathbb{Z}_p \setminus \{0\}, \cdot)$ es un grupo cíclico si p es primo.

Ejemplo 1.3.1. Sea $p = 7$, entonces:

$(\mathbb{Z}_7 \setminus \{0\}, \cdot)$ es un grupo cíclico generado por 3, ya que:

$$3^0 = 1,\ 3^1 = 3,\ 3^2 = 2,\ 3^3 = 6,\ 3^4 = 4,\ 3^5 = 5,\ 3^6 = 1$$

$\lhd$

A diferencia de $\mathbb{Z}_4$ con la suma, el 4-grupo de Klein no es cíclico, no existe ningún elemento que, operado consigo mismo repetidas veces nos de como resultado los demás elementos del grupo.

2. El problema del logaritmo discreto

En el apartado 57 de la tercera sección de sus *Disquisitiones Arithmeticae*, Gauss habla acerca de lo que Euler había llamado *raíces primitivas módulo n*: un número g es una raíz primitiva módulo n si cualquier otro número q, primo relativo con n, puede expresarse como una potencia de g, es decir, si para cada q existe k tal que $g^k \equiv q \pmod{n}$. Gauss enfatiza la ventaja del concepto al hacer operaciones con congruencias, las simplifica "casi de igual modo como la introducción de los logaritmos simplifica las operaciones de la aritmética común". El símil del logaritmo va más allá. A continuación elige una raíz primitiva a como base para referir a ella todos los números no divisibles por p, es decir, si b es no divisible por p entonces se puede escribir como $b \equiv a^e \pmod{p}$ y llama a e el *índice* de b. Gauss añade luego un ejemplo que reproducimos a continuación.

Ejemplo 2.0.1. En su ejemplo Gauss utiliza el módulo 19 y toma como raíz primitiva al 2, de forma que:

Número	1	2	3	4	5	6	7	8	9
Índice	0	1	13	2	16	14	6	3	8
Número	10	11	12	13	14	15	16	17	18
Índice	17	12	15	5	7	11	4	10	9

◁

Así pues, el exponente, al que Gauss llama índice, es el análogo del logaritmo cuando la base es el elemento elegido como raíz primitiva. Tratándose del exponente en un conjunto discreto de elementos, se suele usar el término *logaritmo discreto*.

En un campo finito $\mathbb{Z}_p$ (p primo por supuesto) decimos que un elemento g es un *generador*, una *raíz primitiva*, o *primitivo* módulo p si para cualquier elemento $x \in \mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ existe un número n tal que $x = g^n$, es decir $g^n \equiv x \ (mod\ p)$.

Ejemplo 2.0.2. Por ejemplo en $\mathbb{Z}_{11}$, 2 es un generador módulo 11 porque todo elemento de $\mathbb{Z}_{11}^* = \{1, \ldots, 10\}$ puede escribirse como una potencia de 2 (véase la tabla siguiente).

$x \in \mathbb{Z}_{11}^*$	Expresión	Valor nominal
1	2^{10}	1024
2	2^1	2
3	2^8	256
4	2^2	4
5	2^4	16
6	2^9	512
7	2^7	128
8	2^3	8
9	2^6	64
10	2^5	32

$\triangleleft$

Conviene notar que la relación entre los números y su índice parece aleatoria: si se ordenan los índices crecientemente, los números obtenidos al elevar la base al índice módulo p no poseen un orden. Esto es algo que, como se verá, resulta provechoso desde el punto de vista criptográfico.

Si un elemento a no es raíz primitiva, entonces al elevarlo a diferentes potencias se obtienen *algunos* elementos de $\mathbb{Z}_p^*$, es decir el conjunto de posibles resultados de a^n es un subconjunto propio de $\mathbb{Z}_p^*$, pero si es raíz primitiva entonces el conjunto de posibles resultados de a^n es exactamente todo $\mathbb{Z}_p^*$.

Ejemplo 2.0.3. Si retomamos $\mathbb{Z}_{11}$ del ejemplo previo pero ahora usamos 3 como base ocurre lo siguiente (abusaremos de la notación usando "=" en vez de "≡" y omitiendo el módulo).

$$3^0 = 1,\ 3^1 = 3,\ 3^2 = 9,\ 3^3 = 5,\ 3^4 = 4,\ 3^5 = 1,\ \ldots$$

Como se puede ver el 3 no es un generador de todo $\mathbb{Z}_{11}$, sólo de un subgrupo cíclico de orden 5.

$\triangleleft$

Supongamos que se nos proporciona un elemento primitivo g y el valor del módulo m de un campo finito o de un grupo cíclico y que se nos da también el valor s que resulta de elevar g a un cierto exponente e módulo m. Es decir en la expresión:

$$s \equiv g^e \pmod{m}$$

conocemos los valores de g, m y s y supongamos que se nos pide el valor de e. Dadas las condiciones que hemos mostrado:

1. Que la secuencia de resultados de g^e parece aleatoria y
2. Que si se elige bien g esa secuencia es del tamaño del orden del grupo.

entonces para saber el valor de e que determina s deberemos probar, en el peor de los casos, con todos y cada uno de los valores posibles de e hasta obtener s. Es decir el proceso algorítmico de probar metódicamente uno por uno los posibles exponentes es $O(\|G\|)$ donde $\|G\| = m$ es el orden del grupo. Tanto peor cuanto más grande sea el grupo.

Definición 2. Dados un grupo cíclico $(G, \cdot)$, un elemento generador $g \in G$ y un elemento $s \in G$, el *problema del logaritmo discreto en* G consiste en encontrar el valor de $e \in \mathbb{Z}$ tal que:

$$s = g^e$$

El problema del logaritmo discreto es un problema computacional. Hasta la fecha no se ha encontrado un algoritmo clásico que lo resuelva en tiempo polinomial[1]. Se cree que, si resulta cierto que $P \neq NP$ entonces el problema del logaritmo discreto podría estar en una clase de problemas que, están en NP, pero no en P ni entre los NP-completos, por lo que se les ha llamado problemas NPI (NP-intermedios). Con todo y no pertenecer a la máxima categoría de complejidad dentro de NP, el problema del logaritmo discreto luce bastante intratable computacionalmente.

Dados g y un exponente e es fácil calcular $s = g^e$, pero dado s y g no es trivial calcular e. Ir en un sentido es fácil, pero ir en sentido contrario, calcular la función inversa, es difícil. Esto es lo que en computación se conoce como *función de un sólo sentido* o *one-way function*, algo que, como veremos, se puede aprovechar para construir un sistema criptográfico.

[1]Sin embargo, el algoritmo de Shor [4] lo resuelve en una computadora cuántica en tiempo polinomial.

Conviene aclarar qué tan difícil es encontrar un generador del grupo. Podría pensarse que es muy difícil hacerlo, pero no es así, resulta que los generadores, es decir las raices primitivas están bien caracterizadas y no son tan escasas.

Teorema 1. *Si t es una raíz primitiva en el grupo cíclico de orden m, $\mathbb{Z}_m$, entonces*

$$t^{\frac{m-1}{2}} \equiv m-1 \pmod{m} \equiv -1 \pmod{m}$$

Así que es posible encontrar una raíz primitiva generando un número aleatorio y luego probando si es generador elevándolo a la $(m-1)/2$ y viendo si se obtiene algo congruente con $m-1$. El número de raíces primitivas módulo m es $\varphi(\varphi(m))$, donde φ es la llamada *función indicatriz de Euler*. En particular si m es primo: $\varphi(\varphi(m)) = \varphi(m-1)$ lo que es, por cierto, la cantidad total de números menores que $m-1$ que son primos relativos con él. En general no son pocos, así que no es generalmente tan costoso encontrar un generador.

Ejemplo 2.0.4. 6 es raíz primitiva módulo 761:

$6^{760/2} \equiv 760 \pmod{761}$

Por otra parte $\varphi(760) = 288$, así que ese es el número de raíces primitivas módulo 761

$\triangleleft$

3. Diffie-Helman

Como lo mencionamos en la introducción, en 1976 dos ingenieros eléctricos de la Universidad de Stanford, Whitfield Diffie y Martin Hellman, junto con Ralph Merkle, publicaron un artículo titulado *New directions in cryptography* en el que exponían un mecanismo mediante el cual es posible que dos personas puedan ponerse de acuerdo en una palabra clave secreta comunicándose a gritos entre una multitud que no debe conocer su secreto (es decir comunicándose mediante un canal inseguro). Le llamaron *"criptosistema de llave pública."*

3.1. Intercambio de llaves de Diffie-Hellman

Imaginemos que queremos usar un sistema criptográfico simétrico. Hay una sola palabra clave que es utilizada para cifrar y descifrar, y ésta no debe conocerla el enemigo. Así que ambas partes deben acordar una clave sin que nadie más se entere de ella. El canal de comunicación es inseguro, es escuchado por todo mundo, así que las partes tienen una de dos opciones:

1. Ponerse de acuerdo por adelantado: enviarse la clave a través de un mensajero autorizado y seguro, o citarse en algún lugar para ponerse de acuerdo sin que nadie escuche.

2. Ponerse de acuerdo usando el canal: de tal manera que nadie pueda obtener la clave (o al menos obtenerla en un tiempo razonable) a partir de la información que es enviada a través del canal por ambas partes.

El esquema de intercambio de llaves de Diffie-Hellman hace posible la segunda alternativa. Para explicar el esquema, presentamos la siguiente analogía:

Imaginemos que tanto A como B tienen sendos botes con capacidad para tres litros de pintura. A y B se ponen de acuerdo en un color básico que se dicen a gritos; C, un intruso que pretende enterarse de las conversaciones privadas de A y B también sabe ese color. A y B ponen un litro de

pintura de color básico en sus botes. Además A añade un litro de su color favorito, que nadie, salvo ella conoce. B hace lo propio, añade también un litro exacto de su color secreto y luego, ante la vista de todos, incluyendo a C, se intercambian los botes. Ni A sabe el color secreto de B, ni B sabe el de A, ambos sólo reciben la pintura mezclada. C tampoco puede saber ninguno de los dos colores secretos. Luego A añade al bote que recibió de B un litro de su propio color secreto, y B añade al bote que recibió de A un litro de su propio color. Ahora tanto A como B tienen exactamente el mismo color en sus botes y nadie más puede conocer su composición exacta. Este ejemplo es una buena ilustración del esquema de intercambio de llave de Diffie-Hellman.

Sean Alicia y Bartolo[1] (A y B respectivamente) las dos partes que desean obtener una clave secreta. El algoritmo de intercambio de llaves de Diffie-Hellman es el siguiente:

1. Alicia y Bartolo se ponen de acuerdo en un par de números: un primo grande p y una raíz primitiva $g \in \mathbb{Z}_p^*$. Se pueden poner de acuerdo en un canal inseguro.

2. Alicia elige un entero aleatorio grande α, con $0 < \alpha < p-1$, y le envía a Bartolo

$$X = g^{\alpha} \pmod{p}.$$

3. Bartolo elige un entero aleatorio grande β, con $0 < \beta < p-1$, y le envía a Alicia

$$Y = g^{\beta} \pmod{p}.$$

4. Alicia calcula $K = Y^{\alpha} \pmod{p}$.

5. Bartolo calcula $K' = X^{\beta} \pmod{p}$.

Tenemos que $K = K' = g^{\alpha\beta} \pmod{p}$. Nadie que haya estado escuchando la conversación entre Alicia y Bartolo puede calcular K. Sólo conocería p, g, X, y Y, y para calcular K tendría que poder hacer alguna de las siguientes cosas:

1. Obtener el logaritmo discreto de X ó Y en base g módulo p para obtener α ó β, respectivamente, y poder calcular $g^{\alpha\beta} \pmod{p}$.

[1] Generalmente en los libros y artículos de criptografía se usan *Alice* y *Bob*.

2. Calcular $g^{\alpha\beta}$ (mód p) de alguna otra manera diferente a la opción anterior.

La conjetura de Diffie-Hellman es que la segunda opción no puede hacerse sin haber llevado a cabo la primera[2], es decir, forzosamente se tiene que optar por 1. Como veremos, esto es en general difícil. Para asegurarse de que lo sea realmente debe elegirse p grande y de tal forma que $\frac{p-1}{2}$ sea también primo, g puede elegirse arbitrariamente.

[2]Hasta la fecha no se ha probado que la segunda opción no es posible sin haber resuelto antes la primera, pero no hay nada que asegure que en efecto sea imposible.

4. Criptosistema de ElGamal

En 1984 Taher ElGamal propuso un esquema que permite tanto el cifrado de mensajes como la implementación de firmas digitales. El procedimiento común a ambos usos es el siguiente:

1. Elegir los siguientes números:
   - Un primo p (grande). Esto determina el campo finito donde se efectuarán las operaciones.
   - Un número aleatorio $g < p$, generador en $\mathbb{Z}_p$.
   - Cada usuario u elige un número aleatorio x_u, con $0 < x_u < p$.
2. Calcular
$$y_u = g^{x_u} \pmod{p}. \tag{4.1}$$
3. Pueden hacerse públicos y_u, g y p. Debe mantenerse en secreto x_u. De hecho p y g son públicos y comunes para todos los usuarios del sistema de llave pública (todo el directorio de personas y llaves), pero y_u es particular de cada usuario.

4.1. Cifrado de mensajes

Imaginemos que Alicia desea enviar un mensaje a Bartolo usando el criptosistema de ElGamal. Alicia conoce g, p, y y_B, y procede a hacer lo siguiente:

1. Elige un entero k aleatoriamente, primo relativo a $p-1$. Alicia mantiene secreto el valor de k.
2. Calcula
$$a \equiv g^k \pmod{p}, \tag{4.2}$$

y

$$b \equiv y_B^k M \pmod{p} \tag{4.3}$$

3. El mensaje cifrado es la pareja ordenada (a, b).

La expresión (4.3) implica que

$$\frac{b}{a^{x_B}} \equiv \frac{y_B^k M}{a^{x_B}} \pmod{p}.$$

Usando (4.1) obtenemos:

$$\frac{y_B^k M}{a^{x_B}} \equiv \frac{g^{kx_B} M}{a^{x_B}} \pmod{p}. \tag{4.4}$$

Por otra parte, de la expresión (4.2) tenemos:

$$a^{x_B} \equiv g^{kx_B} \pmod{p}.$$

Sustituyendo en (4.4) tenemos finalmente:

$$\frac{y_B^k M}{a^{x_B}} \equiv \frac{g^{kx_B} M}{a^{x_B}} \equiv \frac{g^{kx_B} M}{g^{kx_B}} \equiv M \pmod{p}.$$

Para descifrar el mensaje se calcula

$$M = \frac{b}{a^{x_B}} \pmod{p},$$

que sólo lo puede efectuar quien posea la clave secreta x_B. Así que este esquema de cifrado se basa en la hipótesis de que el logaritmo discreto es difícil de calcular.

4.2. Firma digital

Sea M el mensaje que se desea firmar y supongamos que Alicia desea firmarlo para enviarlo a Bartolo, de tal manera que Bartolo quede convencido que el mensaje viene de Alicia y no de otra persona. Para hacer esto utilizando el criptosistema de ElGamal, Alicia hace lo siguiente:

1. Alicia elige un entero aleatorio k, primo relativo a $p-1$, que mantendrá en secreto.

2. Calcula $a \equiv g^k \pmod{p}$.

3. Encuentra, usando el algoritmo de Euclides extendido, un entero positivo b tal que $M \equiv (x_A a + kb) \pmod{(p-1)}$.

4. La "firma" del mensaje M es la pareja ordenada (a, b).

Para confirmar que la firma es válida, Bartolo calcula: $y_A^a a^b \pmod{p}$ cuyo valor debe coincidir con: $g^M \pmod{p}$. Notemos que Bartolo conoce a y b, pues fueron enviados como la firma del mensaje M, conoce M (pues el mensaje fue enviado); y el valor de g y y_A son públicos y por ende conocidos.

Para verificar que en efecto $y_A^a a^b \equiv g^M \pmod{p}$, recordemos que $y_A \equiv g^{x_A} \pmod{p}$, y que $a \equiv g^k \pmod{p}$. De manera que

$$\begin{aligned} y_A^a a^b &\equiv (g^{x_A})^a \left(g^k\right)^b \pmod{p} \\ &\equiv g^{ax_A} g^{kb} \pmod{p} \\ &\equiv g^{ax_A + kb} \pmod{p} \\ &\equiv g^M \pmod{p} \end{aligned}$$

por la elección de b.

¿Quien posee la información necesaria para calcular a y b de tal manera que $g^M \equiv y_A^a a^b \pmod{p}$? El valor de b depende del valor de x_A, que suponemos es conocido sólo por Alicia. De manera que al recibir (M, a, b), estamos tan seguros de que el mensaje fue envíado por Alicia como estamos de la seguridad del criptosistema mismo.

Conviene aclarar qué tan difícil es encontrar un par de números primos relativos. Resulta que la probabilidad de que al elegir dos números al azar estos resulten primos relativos es [5]:

$$\frac{6}{\pi^2} \approx 0{,}6$$

o sea suficientemente probable para llevarlo a cabo sin un costo excesivo.

5. Criptoanálisis

5.1. Paso pequeño, paso grande

En inglés el método se denomina *baby-step giant-step*. La intención es encontrar, claro, la x tal que:

$$\beta = \alpha^x \pmod{n} \tag{5.1}$$

El método, en el peor caso, no es mejor que la búsqueda exhaustiva, pero se procura acelerar el proceso comprando tiempo pagado con memoria. En efecto, haremos uso de una tabla y, pensando en una implementación seria, tendría que ser una tabla de dispersión (hash) para hacer las búsquedas lo más rápido posible.

La observación crucial del método es la siguiente: sea $t = \lceil \sqrt{n} \rceil$. Si $\beta = \alpha^x$, entonces existen i, j, $0 \leq i, j < t$ tales que $x = it + j$. Así que tenemos:

$$\beta = \alpha^x = \alpha^{it}\alpha^j \tag{5.2}$$

o equivalentemente:

$$\log_\alpha \beta = x = i\,t + j \tag{5.3}$$

El algoritmo de paso grande, paso chico, comienza con un par de pasos de inicialización:

- Se construye una tabla, cuyas entradas son (j, α^j), con $0 \leq j \leq t$, los *pasos chicos*. Esta es una tabla hash que debe ser accedida usando como llave la segunda componente (α^j).
- Se calcula α^{-t}, el *paso grande*.
- La llave inicial de búsqueda se define como β.

El procedimiento consiste en buscar la llave en la segunda componente de la tabla, si se encuentra en la primera iteración (que indicaremos con $i = 0$), hemos hallado un valor de j tal que $\beta = \alpha^j$, así que hemos terminado y $\log_\alpha \beta = j$. Si no es así debemos iterar: incrementar el valor de i y cambiar la llave por $\beta\alpha^{-it}$. Cuando encontremos, en alguna iteración, el valor de la llave habremos hallado la solución a la ecuación 5.2. La complejidad esperada del algoritmo es $O(\sqrt{n})$.

Algoritmo 1 Algoritmo de paso pequeño, paso grande.

Require: $n > 0$, el orden de un grupo cíclico G con generador α y $\beta \in G$
Ensure: x tal que $\beta = \alpha^x \pmod{n}$

```
1: function PASO-PEQ-PASO-GDE(n, α, β)
2:     m ← ⌈√n⌉
3:     T ← CONSTRUYETABLA(j, α^j); j ∈ {0, ..., m − 1}
4:     α^{−m} ← (α^{−1})^m
5:     llave ← β
6:     for i ← 0 to m − 1 do
7:         if ENTABLA(llave,T) then
8:             j ← BUSCA (llave, T)
9:             return x = i m + j
10:        end if
11:        llave ← llave · α^{−m}
12:    end for
13: end function
```

Ejemplo 5.1.1. Ilustraremos con un ejemplo el método de paso pequeño, paso grande.

Sea $m = 127$ el tamaño del módulo, así que el orden máximo de un elemento es 126 y por supuesto la raíz es $r = \frac{126}{2} = 63$, así que un elemento primitivo g se distingue porque:

$$g^r \equiv m - 1 \equiv -1 \pmod{m}$$

Sea $\alpha = 6$ el generador elegido. En efecto $6^{63} \equiv 126 \pmod{127}$.

Calculando la raíz del módulo: $t = \lceil \sqrt{127} \rceil = 12$.

El inverso de α por su parte es: $\alpha^{-1} \equiv 106 \pmod{127}$, de donde: $\alpha^{-t} = \alpha^{-12} \equiv 94 \pmod{127}$

Supongamos que se requiere encontrar el logaritmo discreto de $\beta = 25$. La tabla inicial es:

j	0	1	2	3	4	5	6	7	8	9	10	11
α^j	1	6	36	89	26	29	47	28	41	119	79	93

En la primera iteración ($i = 0$) buscamos en el segundo renglón de la tabla al 25 y no lo encontramos. Ahora ($i = 1$) calculamos: $\beta\alpha^{-t} = 25 \cdot 94 \equiv 64 \pmod{127}$, pero el 64 tampoco está en la tabla. Procedemos a calcular ($i = 2$): $\beta\alpha^{-2t} = 25 \cdot 94^2 \equiv 47 \pmod{127}$, que sí está en la tabla en la posición $j = 6$, por lo que el valor regresado por el algoritmo sería:

$$x = i\,t + j = 2 \cdot 12 + 6 = 30$$

En efecto $6^{30} \equiv 25 \pmod{127}$

$\lhd$

5.2. Colisiones

Otra manera de encontrar la x en la expresión 5.1, en un grupo cíclico de orden n, es buscar una colisión:

$$\alpha^{i_1} \cdot \beta^{j_1} \equiv \alpha^{i_2} \cdot \beta^{j_2} \pmod{n} \tag{5.4}$$

Si substituimos $\alpha^x = \beta$ en esta expresión:

$$\begin{aligned} \alpha^{i_1} \cdot \beta^{j_1} &= \alpha^{i_1} \cdot (\alpha^x)^{j_1} = \alpha^{i_1 + x\,j_1} \\ &\equiv \alpha^{i_2} \cdot \beta^{j_2} \pmod{n} \\ &= \alpha^{i_2} \cdot (\alpha^x)^{j_2} = \alpha^{i_2 + x\,j_2} \end{aligned}$$

de donde: $i_1 + x\,j_1 = i_2 + x\,j_2$. Por lo que:

$$x = \frac{i_2 - i_1}{j_1 - j_2} \pmod{n} \tag{5.5}$$

Ejemplo 5.2.1. En el grupo cíclico multiplicativo $\mathbb{Z}_{11}^*$.

Sean: $\alpha = 2$ (generador del grupo, como puede verse en las tabla siguiente) y $\beta = 9$. Hay que encontrar x tal que:

$$2^x \equiv 9 \pmod{11}$$

$t \in \mathbb{Z}_{11}^*$	α^i	Valor nominal
1	2^{10}	1024
2	2^1	2
4	2^2	4
8	2^3	8
5	2^4	16
10	2^5	32
9	2^6	64
7	2^7	128
3	2^8	256
6	2^9	512

Si hiciéramos una tabla similar para β^j tendríamos que:

$$\begin{aligned}
\beta^0 &= \beta^5 &= 1 \\
\beta^1 &= \beta^6 &= 9 \\
\beta^2 &= \beta^7 &= 4 \\
\beta^3 &= \beta^8 &= 3 \\
\beta^4 &= \beta^9 &= 5
\end{aligned}$$

Existe una colisión entre:

$$2^2 \cdot 9^3 \equiv 2^8 \cdot 9^2 \pmod{11}$$

de donde, en la expresión 5.5: $i_1 = 2$, $j_1 = 3$, $i_2 = 8$, $j_2 = 2$,

$$x = \frac{8-2}{3-2} = 6$$

En efecto: $2^6 \equiv 9 \pmod{11}$.

$\triangleleft$

5.3. ϱ de Pollard

Pensemos en un grupo cíclico como los que hemos estado usando. Tiene un número posiblemente grande, pero finito de elementos. Si, iniciando en uno de sus elementos hacemos una caminata digamos, aleatoria y muy larga, al estar permanentemente trabajando módulo el orden del grupo, terminaríamos reencontrando algún elemento visitado previamente. De hecho podemos pensar en definir los pasos de la caminata como una función que, dado el elemento visitado en el paso i nos diga cuál elemento le sigue, incluso podríamos modelar esto como una gráfica cuyos vértices sean los elementos del grupo y cuyas aristas sean los pasos que nos llevan de uno a otro. Esta es la idea detrás del algoritmo de que John Pollard ideo en dos variantes: la de 1975 para factorizar números y la de 1978 para calcular el logaritmo discreto. La idea es encontrar, como hemos dicho previamente, colisiones como la especificada por la expresión 5.4.

Sea G un grupo cíclico de orden n. Lo partiremos en tres subconjuntos ajenos, digamos: G_0, G_1 y G_2. No es muy relevante como partimos a G, pero debe poder comprobarse rápidamente, dado un elemento $x \in G$ a que subconjunto pertenece (por ejemplo $x \in G_k$ donde $x \equiv k \pmod 3$). Así pues:

$$G = G_0 \cup G_1 \cup G_2 \text{ donde: } G_k \cap G_t = \emptyset \text{ si } k \neq t;\ k, t \in \{0, 1, 2\}$$

La sucesión de elementos de G visitados por la "caminata" la determina la función:

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i, & x \in G_0 \\ x_i^2, & x \in G_1, x \neq 1 \\ \alpha \cdot x_i, & x \in G_2 \cup \{1\} \end{cases} \tag{5.6}$$

hemos excluido artificialmente al 1 de G_1 porque, dada la expresión para las $x_i \in G_1$, si allí estuviera el 1 y cayéramos en él, ya no nos moveríamos.

Recordemos que queremos encontrar una colisión y por tanto x_i debe tener la forma:

$$x_i = \alpha^{a_i}\ \beta^{b_i}$$

Siendo a_i el exponente de α, resulta claro que debemos incrementarlo en 1, cuando x_i cae en G_2, como lo indica la función 5.6. Asimismo debemos duplicarlo cuando x_i cae en G_1 y dejarlo tal y como esté cuando x_i caiga en G_0. Análogamente debemos incrementar el exponente de β, es decir, b_i cuando x_i cae en G_0, duplicarlo cuando cae en G_1 y dejarlo tal cual en G_2. En síntesis:

$$a_{i+1} = \begin{cases} a_i, & x \in G_0 \\ 2\,a_i \pmod{n}, & x \in G_1, x \neq 1 \\ a_i + 1 \pmod{n}, & x \in G_2 \cup \{1\} \end{cases} \tag{5.7}$$

$$b_{i+1} = \begin{cases} b_i + 1 \pmod{n}, & x \in G_0 \\ 2\,b_i \pmod{n}, & x \in G_1, x \neq 1 \\ b_i, & x \in G_2 \cup \{1\} \end{cases} \tag{5.8}$$

Igual que en el caso del algoritmo de paso pequeño, paso grande, la complejidad del algoritmo de Pollard en tiempo de ejecución es $O(\sqrt{n})$, pero tiene la ventaja de usar $O(1)$ de memoria.

Cuando el algoritmo 3 calcula x_{2i} en la línea 10, establece una carrera entre dos sucesiones: la de x_i y la de x_{2i} que va más adelante de la primera. A partir de algún momento ambas secuencias van a quedar atrapadas en un ciclo de longitud $\sigma \approx \sqrt{n}$, así que en algún momento coincidirán, como los automóviles en un circuito de carreras. Esto generará tarde o temprano una colisión. Esta parte del algoritmo aparentemente *no* se la debemos a Robert Floyd[1], de donde proviene el nombre con el que se le conoce tradicionalmente: *Algoritmo de detección de ciclo de Floyd.*

Ejemplo 5.3.1. $G = \mathbb{Z}^*_{383}$, $\alpha = 2$, $n = 191$, $\beta = 228$.

Partimos el conjunto G en tres subconjuntos de acuerdo a la siguiente regla: $x \in S_1$ si $x \equiv 1 \pmod 3$; $x \in S_2$ si $x \equiv 0 \pmod 3$; y $x \in S_3$ si $x \equiv 2 \pmod 3$. La Tabla 5.1 da los valores de x_i, a_i, b_i, x_{2i}, a_{2i}, y b_{2i} al final de cada paso del algoritmo. Entonces tenemos que $x_{14} = x_{28} = 144$. Calculamos $r = b_{14} - b_{28} \mod 191 = 125$, $r^{-1} = 136 \mod 191$, y entonces

$$\log_\alpha(\beta) = r^{-1}(a_{28} - a_{14}) \mod 191 = 110.$$

◁

[1] Al parecer el algoritmo le fue atribuido erróneamente a Floyd por Donald Knuth en 1969.

Algoritmo 2 Funciones auxiliares del algoritmo ϱ de Pollard.

Require: $x_i \in G$, $G = G_0 \cup G_1 \cup G_2$ disjuntos

function F($x_i \in G$, α, β)
    if $x_i \in G_0$ **then**
        return $\beta\, x_i$
    else if $(x_i \in G_1) \wedge (x_i \neq 1)$ **then**
        return x_i^2
    else ▷ $x_i \in G_2 \cup \{1\}$
        return $\alpha\, x_i$
    end if
end function
function A($x_i \in G$, a_i)
    if $x_i \in G_0$ **then**
        return a_i
    else if $(x_i \in G_1) \wedge (x_i \neq 1)$ **then**
        return $2\, a_i \pmod{n}$
    else ▷ $x_i \in G_2 \cup \{1\}$
        return $a_i + 1 \pmod{n}$
    end if
end function
function B($x_i \in G$, b_i)
    if $x_i \in G_0$ **then**
        return $b_i + 1 \pmod{n}$
    else if $(x_i \in G_1) \wedge (x_i \neq 1)$ **then**
        return $2\, b_i \pmod{n}$
    else ▷ $x_i \in G_2 \cup \{1\}$
        return b_i
    end if
end function

Algoritmo 3 Algoritmo de ϱ de Pollard.

Require: $x_i \in G$, $G = G_0 \cup G_1 \cup G_2$ disjuntos

function POLLARD-LOG($n = |G|$)
    $x_0 \leftarrow 1$
    $a_0 \leftarrow 0$
    $b_0 \leftarrow 0$
    $i \leftarrow 1$
    loop
        $x_i \leftarrow f(x_{i-1}, \alpha, \beta)$
        $a_i \leftarrow \text{A}(x_{i-1}, a_{i-1})$
        $b_i \leftarrow \text{B}(x_{i-1}, b_{i-1})$
        $x_{2i} \leftarrow f(f(x_{2i-2}, \alpha, \beta), \alpha, \beta)$
        $a_{2i} \leftarrow \text{A}(f(x_{2i-2}, \alpha, \beta), \text{A}(x_{2i-2}, a_{2i-2})$
        $b_{2i} \leftarrow \text{B}(f(x_{2i-2}, \alpha, \beta), \text{B}(x_{2i-2}, a_{2i-2})$
        if $x_i = x_{2i}$ **then**
            if $(b_i - b_{2i}) = 0$ **then**
                exit(error)
            end if
            return $\frac{a_{2i}-a_i}{b_i-b_{2i}} \pmod{n}$
        end if
        $i \leftarrow i + 1$
    end loop
end function

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
1	228	0	1	279	0	2
2	279	0	2	184	1	4
3	92	0	4	14	1	6
4	184	1	4	256	2	7
5	205	1	5	304	3	8
6	14	1	6	121	6	18
7	28	2	6	144	12	38
8	256	2	7	235	48	152
9	152	2	8	72	47	54
10	304	3	8	14	96	118
11	372	3	9	256	97	119
12	121	6	18	304	98	120
13	12	6	19	121	5	51
14	144	12	38	144	10	104

Cuadro 5.1: Pasos intermedios del algoritmo ro de Pollard

Bibliografía

[1] Katz, J. y Lindell Y., *Introduction to Modern Cryptography*, 2a Ed., Chapman & Hall / CRC, 2014.

[2] Menezes, A.J., van Oorschot P.C. y Vanstone S.A., *Handbook of Applied Cryptography*, CRC Press, 1996.

[3] Paar, C. y Pelzl J., *Understanding Cryptography*, Springer, 2010.

[4] Shor, Peter, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", *SIAM Journal on Computing*, Vol. 26, No.5, 1997, pp. 1484–1509.

[5] Weisstein, Eric W. Relatively Prime.", *MathWorld–A Wolfram Web Resource*. `http://mathworld.wolfram.com/RelativelyPrime.html`