Trabajo Dirigido en Estadística y Computación

Licenciatura en Matemáticas

Curso 2012-2013

El logaritmo discreto y sus aplicaciones en Criptografía

Alumna: Jennifer Santamaría Fernández

Director: Daniel Sadornil Renedo

Facultad de Ciencias

Universidad de Cantabria

Índice general

Capítulo 1

Introducción

En la actualidad, mantener en secreto ciertas informaciones es una parte fundamental e imprescindible de prácticamente cualquier acción cotidiana, ya sea pagar una compra con tarjeta de crédito, consultar el correo electrónico o enviar un mensaje de texto. Precisamente ese es uno de los objetivos de la Criptografía y, para ello, se necesitan herramientas que permitan llevar a cabo dicha tarea. Dichas herramientas son aportadas, principalmente, por las Matemáticas. El uso de problemas matemáticos difíciles o imposibles de resolver bajo ciertas condiciones, con las herramientas de cálculo disponibles hoy en día, es algo habitual en Criptografía. Uno de los problemas matemáticos en los que se basan algunos sistemas criptográficos es el tópico de este trabajo: el logaritmo discreto.

Sea G un grupo abeliano finito (multiplicativo) y sea g un elemento de orden n de G. Dado un elemento a perteneciente al subgrupo generado por g, se define el logaritmo discreto de a en base g como el entero k, $0 \leq k \leq n-1$, tal que:

$$g^k = a$$

Aunque el logaritmo discreto se ha definido en un grupo multiplicativo y en este trabajo sólo se consideran grupos de este tipo, se puede definir de forma general en un grupo. Es posible definir el logaritmo discreto en grupos aditivos como el conjunto de puntos de una curva elíptica. En tal caso, se habla de logaritmo discreto elíptico. Los algoritmos de cálculo explicados a lo largo de este trabajo son todos adaptables a este caso salvo el Index-Calculus. Además, existen otros algoritmos específicos.

Para abordar no sólo la resolución, sino también la utilidad del logaritmo discreto, el trabajo se estructura como sigue. En el capítulo 2, se exponen los pilares básicos de la Criptografía con el objetivo de situar el área de aplicación del tema del trabajo. Se presenta en dicho capítulo una visión general de la Criptografía: objetivos, aplicaciones, tipos de ataques, tipos de seguridad, etc. Además, se expone la diferencia entre la Criptografía de clave privada y de clave pública, siendo esta última donde el logaritmo discreto juega un papel fundamental. Se cierra el capítulo con un pequeño resumen del desarrollo de la Criptografía a lo largo de la historia.

En el capítulo 3, se hace un breve repaso de aquellos resultados fundamentales del Álgebra y de la Teoría de Números que estarán presentes a lo largo de todo el trabajo y que serán necesarios para comprender algunas aplicaciones del logaritmo discreto en Criptografía y diversos métodos de resolución del problema.

Una vez tratadas las cuestiones preliminares, se introduce el problema del logaritmo discreto en el capítulo 4. Se pone de manifiesto la dificultad que entraña su resolución y se presentan algunos sistemas criptográficos basados en el mismo como, por ejemplo, el criptosistema de Massey-Omura o el de ElGamal. Asimismo se estudian el algoritmo de intercambio de claves de Diffie-Hellman y dos algoritmos de firma digital (DSA y de ElGamal) que también basan su seguridad en el problema del logaritmo discreto.

En el capítulo 5, se estudian algunos de los métodos más importantes y conocidos para resolver el problema del logaritmo discreto. Se comienza por los métodos genéricos: la fuerza bruta, el algoritmo de Shanks y los algoritmos ρ y λ de Pollard. Estos algoritmos no aprovechan ninguna característica del grupo. A continuación se presenta el algoritmo de Silver-Pohlig-Hellman que resuelve el problema del logaritmo discreto de forma eficiente si el cardinal del grupo tiene todos sus factores primos pequeños (menores que una cuota adecuada). Se concluye el capítulo con la exposición del Index-Calculus.

El Index-Calculus es el más "potente" de los métodos presentados y, a diferencia del algoritmo de Silver-Pohlig-Hellman, no necesita que el grupo donde se trabaja cumpla ninguna propiedad para ser eficaz. Sin embargo, para poder utilizar el Index-Calculus, es imprescindible disponer de algoritmos de factorización en el grupo. De hecho, en el capítulo 5 se incluyen algunos métodos eficaces para factorizar polinomios necesarios para aplicar el Index-Calculus en ciertos grupos.

Capítulo 2

Criptografía. Definición y desarrollo.

La palabra Criptografía tiene su origen etimológico en las palabras griegas *krypto* ("oculto") y *graphos* ("escribir"), lo que da una idea clara de su definición clásica. La Criptografía se definía como el arte de escribir mensajes en clave secreta o de modo enigmático para evitar que su contenido fuese inteligible por un posible intruso en la comunicación. Es decir, el único objetivo de la Criptografía era conseguir la confidencialidad de los mensajes. En 1949, Shannon publicó el artículo *Una teoría matemática de las comunicaciones* [26] y la Criptografía comenzó a ser considerada una ciencia aplicada que requiere conocimientos de otras ciencias como las Matemáticas o la Teoría de la Información.

A finales del siglo XX, el desarrollo de la informática e Internet dieron lugar a grandes cambios en en Criptografía. Por ejemplo, la gran cantidad de información a disposición de muchos usuarios hace necesario que los datos estén protegidos durante su almacenamiento, no sólo durante la transmisión. Es por eso que, actualmente, la Criptografia tiene otros objetivos aparte de la transmisión secreta de la información. Este tipo de aplicaciones se engloba dentro de lo que se denominan protocolos criptográficos. Un protocolo criptográfico es un conjunto bien definido de etapas, implicando a dos o más partes y acordado por ellas, designado para realizar una tarea específica que utiliza como herramienta algún algoritmo criptográfico. He aquí algunos ejemplos:

- Protocolos de autentificación. Hay dos tipos de autentificación: autentificación de mensaje y autentificación de usuario. La primera asegura que el contenido del mensaje no ha sido alterado. La segunda, certifica la identidad del remitente.
- Protocolos para compartir secretos. La finalidad de este tipo de protocolos es distribuir un cierto secreto entre un conjunto $\mathcal{P}$ de participantes de manera que ciertos subconjuntos prefijados de $\mathcal{P}$ puedan, uniendo sus participaciones, recuperar dicho secreto.
- Pruebas de conocimiento cero: hacen posible que un individuo pueda convencer a otro de que posee cierta información sin revelarle nada sobre el contenido de la misma.
- Transacciones electrónicas seguras: permiten realizar de manera electrónica segura las operaciones bancarias habituales, firma electrónica de contratos...
- Votaciones electrónicas: permiten realizar un proceso electoral electrónicamente, garantizando la privacidad de cada votante y la imposibilidad de fraude.

Así pues, la Critografía ya no sólo se utiliza para preservar la confidencialidad, sino que con ella se buscan también otros objetivos:

- Autentificación. Es la propiedad que me permite identificar el generador de la información (y de esta manera evitar posibles suplantaciones de identidad).
- Integridad. Es la propiedad que busca mantener los datos libres de modificaciones no autorizadas.
- No repudio. Proporciona protección contra la interrupción, por parte de alguna de las entidades implicadas en la comunicación, de haber participado en toda o parte de la comunicación. Esto evita que el emisor niegue el envío de cierta información o que el receptor niegue que la recibió.

Sin embargo, la Criptografía corresponde sólo a una parte de la comunicación secreta. Si se quiere mantener en secreto la comunicación es porque existe la posibilidad de que un "enemigo" intercepte el mensaje y este sea revelado. El conjunto de técnicas utilizadas por el enemigo para descifrar los mensajes secretos es una ciencia conocida como Criptoanálisis. La Criptología es el conjunto la Criptografía y el Criptoanálisis.

Contra un mensaje secreto es posible encontrar dos tipos básicos de amenazas: la pasiva y la activa. En la primera de ellas, el enemigo simplemente quiere acceder a la información, mientras que en la segunda pretende llevar a cabo una modificación o falsificación de la misma o incluso una interrupción de la comunicación. Es decir, el primer tipo de amenaza compromete la confidencialidad y el segundo, la integridad y autenticidad de los mensajes. En cuanto a la seguridad de la comunicación, es conveniente conocer la situación del enemigo y de qué herramientas dispone. En este contexto podrían distinguirse cuatro tipos de ataque:

- Ataque sólo con texto cifrado. Se trata de la peor situación posible para el criptoanalista pues, a partir del mensaje cifrado, éste desea conocer el mensaje original.
- Ataque con texto original conocido. El criptoanalista tiene acceso a una correspondencia entre un texto original y su cifrado.
- Ataque con texto original escogido. El enemigo puede obtener el cifrado de cualquier texto original que cumpla ciertas condiciones prefijadas por él.
- Ataque con texto cifrado escogido. El criptoanalista puede obtener los mensajes originales correspondientes a determinados mensajes cifrados que elija (pero no el mensaje que desea descifrar).

Ante el ataque de un criptoanalista, un criptosistema puede presentar uno de los siguientes tipos de seguridad:

- Seguridad teórica o perfecta. El criptosistema es seguro contra cualquier enemigo que tenga recursos y tiempo ilimitados.
- Seguridad práctica o computacional. El criptosistema es seguro contra aquellos enemigos que tengan insuficiente tiempo y/o recursos.

- Seguridad probable. La seguridad del criptosistema aún no ha sido demostrada, pero por el momento no ha sido roto.
- Seguridad condicional. El criptosistema es seguro hasta que se desarrollen nuevos o mejores métodos de criptoanálisis.

La forma en que la Criptografía protege la información es variando su forma. Dado un texto original se llama cifrado a una transformación del mismo en el llamado texto cifrado. La transformación que permite recuperar el texto original a partir del texto cifrado se llama descifrado. La familia de transformaciones posibles se denomina sistema criptográfico o criptosistema y cada una de ellas está determinada por un parámetro llamado clave. De manera más formal:

Definición 2.1. *Un sistema criptográfico o criptosistema es una terna* $(\mathcal{M}, \mathcal{C}, \mathcal{K})$*, donde* $\mathcal{M}$ *es el conjunto de mensajes originales,* $\mathcal{C}$ *es un conjunto de mensajes cifrados y* $\mathcal{K}$ *es un conjunto finito de claves, junto con dos funciones:*

$$c : \mathcal{M} \times \mathcal{K} \longrightarrow \mathcal{C} \quad y \quad d : \mathcal{C} \times \mathcal{K} \longrightarrow \mathcal{M}$$

tales que $d(c(\mathcal{M}, k), k) = M$ *para todo* $(M, k) \in \mathcal{M} \times \mathcal{K}$*.*
Las funciones c *y* d *reciben el nombre de funciones de cifrado y descifrado respectivamente.*

Los criptosistemas pueden dividirse en sistemas de clave privada y sistemas de clave pública. En los sistemas de clave privada las personas que comparten el criptosistema comparten o guardan en secreto las funciones c y d (o, al menos, la clave de que dependen). Por el contrario, en los sistemas de clave pública cada usuario i del criptosistema tiene un par de claves (c_i,d_i). La primera de ellas, c_i, es la clave pública y es la que utiliza cualquier otro usuario j que quiera transmitir un mensaje M a i. Esto quiere decir que j cifra el mensaje en la forma $C = c_i(M)$. La segunda clave, d_i, es privada y empleada por i para recuperar los mensajes cifrados que recibe. Esto es, $M = d_i(C) = d_i(c_i(M))$.

En criptografía de clave pública es importante notar que la clave pública se obtiene a partir de la privada (o viceversa). Por tanto, para que realmente el secreto sea tal, es necesario que c_i venga definida por una función conocida pero de la que sea computacionalmente imposible deducir d_i sin el conocimiento de cierta información suplementaria que sólo posee i. Este tipo de funciones se denominan funciones trampa y están basadas en la dificultad computacional de ciertos problemas matemáticos. Tal y como se verá en el capítulo 4, algunos criptosistemas de clave pública utilizan la exponenciación en un grupo multiplicativo finito como función trampa y en su criptoanálisis es necesario resolver el problema del logaritmo discreto.

Los primeros usos conocidos de Criptografia se remontan a la antigüedad. El primero data del siglo V a.C, durante la guerra entre Atenas y Esparta. El método utilizado es concido como la *escítala* y consistía en lo siguiente. El mensaje se escribía sobre una cinta enrollada en un rodillo de manera que, al desenrollarla, las letras quedaban descolocadas y el mensaje sólo podía ser leído en otro rodillo de igual grosor. Asimismo se sabe que los romanos utilizaban un cifrado consistente en sustituir unas letras por otras según una regla fija, conocida como *cifrado César*. Dicha regla era trasladar la letra a cifrar unas posiciones en el alfabeto. También aparecen textos cifrados por sustitución en la Biblia.

La obra más antigua que existe sobre Criptografía se titula *Liber Zifrorum* y fue escrita por Cicco Simoneta en el siglo XV. También en este siglo, Alberti hace grandes aportaciones al

campo del criptoanálisis. En el siglo XVI el uso de la Criptografía se generalizó en los ambientes diplomáticos y Vigenère reunió diversos métodos de la época en su libro *Traicté des Chiffres*.

Todos los criptosistemas utilizados hasta entonces eran muy simples pues usaban sustitución, transposición o una combinación de ambas cosas y, por lo tanto, fueron fáciles de romper. En 1883, Kerckhoffs en su trabajo titulado *La Criptografía militar* [11] recomendó que los sistemas criptográficos cumpliesen las siguientes reglas:

1. No debe existir ninguna forma de recuperar mediante un texto cifrado el mensaje original o la clave.
2. Todo sistema criptográfico debe estar compuesto por dos tipos de información:
   - Pública, como es la familia de algoritmos que lo definen.
   - Privada, como la clave que se usa en cada cifrado en particular.
3. La forma de escoger la clave ha de ser fácil de recordar y de modificar.
4. Debe ser factible la comunicación del mensaje cifrado con los medios de transmisión habituales.
5. La complejidad del proceso de recuperación del texto original debe corresponderse con el beneficio obtenido.

Dichas reglas han sido adoptadas por gran parte de la comunidad criptográfica y se resumen en el conocido como principio de Kerckhoffs: la seguridad de un criptosistema se mide suponiendo que el enemigo conoce completamente los procesos de cifrado y descifrado.

En el siglo XX se produjo un gran desarrollo en Criptografía debido a las guerras mundiales, en las cuales era necesario establecer comunicaciones secretas a través del telégrafo y la radio. En la Primera Guerra Mundial, el descifrado del conocido como telegrama Zimmermann, en el que el ministro alemán pretendía convencer a Japón y México de invadir EE.UU, fue clave para que EE.UU entrase en guerra. Durante la Segunda Guerra Mundial, se construyó la máquina *Colossus*, precursora de los ordenadores modernos. Ésta permitió a la oficina criptoanalítica británica capitaneada por Alan Turing romper la seguridad de la máquina de cifrado alemana *Enigma*.

Hasta 1976, todos los criptosistemas eran de clave privada. En ese momento, Diffie y Hellman [8] establecieron los principios teóricos que debería satisfacer un criptosistema de clave pública. Estas son las conocidas como condiciones de Diffie-Hellman:

1. El cálculo de las claves pública y privada debe ser computacionalmente sencillo, es decir, dado por un algoritmo de complejidad polinómica.
2. El proceso de cifrado ha de ser computacionalmente sencillo.
3. El proceso de descifrado, conociendo la clave secreta, debe ser computacionalmente sencillo.
4. La obtención de la clave secreta a partir de la pública ha de ser un problema computacionalmente imposible, es decir, dado por un algoritmo de complejidad exponencial.

5. La obtención del mensaje original, conociendo el mensaje cifrado y la clave pública, debe ser computacionalmente imposible.

Nacieron a partir de entonces nuevos criptosistemas como el RSA o ElGamal. El primero basa su seguridad en la dificultad de factorizar un número natural compuesto. El sistema criptográfico ElGamal, por su parte, utiliza como función trampa la exponenciación modular, cuya función inversa es el logaritmo discreto.

Los criptosistemas de clave pública tienen el defecto de ser demasiado lentos. Esta desventaja puede ser solventada utilizándolos únicamente para distribuir las claves para un sistema de clave privada con el que llevar a cabo una comunicación más rápida. En esa línea, Diffie y Hellman propusieron el primer protocolo de intercambio de clave basado en la exponenciación en cuerpos finitos. Además, estos autores también introdujeron el concepto de firma digital. La firma digital es básicamente un conjunto de datos asociados a un mensaje que permiten asegurar la identidad del firmante y la integridad del mensaje. La firma digital debe ser:

1. Única, pudiendo generarla solamente el usuario legítimo;
2. No falsificable, el intento de falsificación debe llevar asociada la resolución de un problema numérico intratable;
3. Fácil de autentificar, esto es, cualquier receptor puede establecer su autenticidad;
4. Irrevocable, el autor de una firma no puede negar su autoría;
5. Fácil de generar.

Otra característica que deben tener las firmas digitales es que deben depender tanto del mensaje como del autor. Si esto no fuese así, el receptor podría modificar el mensaje y mantener la firma, produciendo así un fraude. Los criptosistemas de clave pública pueden ser fácilmente utilizados para generar firmas digitales. Un cierto usuario i con clave (c_i,d_i) procede de la siguiente manera para firmar sus mensajes. A cada mensaje $M \in \mathcal{M}$, le asocia la firma $s = d_i(M)$. Entonces, cualquier usuario puede calcular $c_i(s)$ y verificar que coincide con m. Sin embargo, sólo i puede deducir el valor de s para el que $c_i(s) = m$, esto es, sólo i puede calcular la firma. En el capítulo 4 se verán algunos algoritmos de firma basados en exponenciación que basan sus seguridad en el problema del logaritmo discreto.

Capítulo 3

Preliminares algebraicos

En este capítulo se presentan aquellas definiciones y resultados matemáticos básicos que aparecerán con frecuencia a lo largo del trabajo.

El problema del logaritmo discreto será planteado y resuelto en grupos cíclicos finitos por lo que conviene tener claros todos los conceptos relacionados con estas estructuras algebraicas. Salvo que se especifique otra cosa, se trabajará con grupos multiplicativos.

Definición 3.1. *Sea G un grupo y sea a un elemento de G. El orden de a, $ord_G(a)$, es el menor entero positivo r tal que $a^r = 1$. Si $a^r = 1$ implica $r = 0$, entonces se dice que a tiene orden infinito.*

Si G es un grupo finito, no hay elementos de orden infinito. En tal caso se tiene también que $a^{|G|} = 1$ para todo elemento a de G.

Proposición 3.2. *Sea a un elemento de G de orden r. Entonces:*

1. *El grupo generado por a es $< a >= \{1, a, a^2, ..., a^{r-1}\}$.*
2. *Si t es un entero tal que $a^t = 1$ se cumple que $r \mid t$.*
3. *Si $a^n = a^m$ para ciertos enteros n, m se tiene que $n \equiv m \pmod{r}$.*

Definición 3.3. *Un grupo G es cíclico si existe algún elemento $a \in G$ tal que $G =< a >$.*

Corolario 3.4. *Un grupo finito G es cíclico si y sólo si existe un elemento $a \in G$ tal que $O(a) =\mid G \mid$.*

Proposición 3.5. *Sea G un grupo finito de orden n.*

1. *El orden de cada subgrupo de G y el orden de cada elemento de G es divisor de n.*
2. *Si n es primo, G es cíclico y no posee subgrupos propios.*
3. *Si G es cíclico, para cada divisor positivo d de n existe un único subgrupo de orden d de G.*

A continuación se presentan un teorema de estructura para grupos abelianos finitos.

Teorema 3.6. *Todo grupo abeliano finito $G \neq \varnothing$ es isomorfo a un producto de grupos cíclicos finitos de órdenes $m_1, ..., m_t$ donde los m_i son mayores que 1 y primos entre sí.*

$$G \approx \mathbb{Z}/m_1\mathbb{Z} \times ... \times \mathbb{Z}/m_t\mathbb{Z}$$

Este isomorfismo puede calcularse mediante el teorema chino de los restos:

Teorema 3.7 (Teorema chino de los restos)**.** *Sean $m_1, m_2, ..., m_r$ enteros positivos dos a dos primos entre sí. Sean $a_1, a_2, ..., a_r$ enteros cualesquiera. En esas condiciones, el sistema de congruencias*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

tiene solución. Además, existe un único valor de x satisfaciendo todas las congruencias y verificando que $0 \leq x < m$ donde $m = m_1 \cdot m_2 \cdot ... \cdot m_r$.

El grupo $\mathbb{Z}/m\mathbb{Z}$ es un grupo finito con la suma. El conjunto de las unidades de $\mathbb{Z}/m\mathbb{Z}$, $(\mathbb{Z}/m\mathbb{Z})^*$, es un grupo de cardinal $\varphi(m)$ donde φ es la función de Euler. Por lo tanto, para cada elemento a de $(\mathbb{Z}/m\mathbb{Z})^*$, $a^{\varphi(m)} = 1 \pmod{m}$. Dicho resultado es el conocido teorema de Euler.

Se utilizará también la versión para polinomios del teorema chino de los restos.

Teorema 3.8. *Sea $f(x) = \prod_{i=1}^{m} p_1(x)p_2(x)...p_m(x)$ un polinomio cuyos factores $p_i(x)$ sean primos entre sí dos a dos. La aplicación*

$$\begin{array}{rccc} \varphi: & \dfrac{\mathbb{F}_q[x]}{(f(x))} & \longrightarrow & \dfrac{\mathbb{F}_q[x]}{(p_1(x))} \oplus ... \oplus \dfrac{\mathbb{F}_q[x]}{(p_m(x))} \\ & g(x) & \longmapsto & (g_1(x), ..., g_m(x)) \end{array}$$

donde $g_i(x) \equiv g(x) \pmod{p_i(x)}$ para $i = 1, ..., m$, es un isomorfismo de anillos.

Por otra parte, en muchas ocasiones se trabajará en cuerpos finitos y se necesitará calcular el logaritmo discreto en el grupo multiplicativo del cuerpo. De ahí, que el siguiente resultado sea fundamental:

Proposición 3.9. *El grupo multiplicativo de un cuerpo finito es cíclico.*

El grupo multiplicativo de un cuerpo K se denota por K^* y se denomina *elemento primitivo* o *generador*,g, a cualquier elemento que lo genere. No existe un único elemento primitivo y si g es un generador, necesariamente su orden coincide con el orden del cuerpo.

También conviene tener presente la definición y propiedades de la característica de un cuerpo:

Definición 3.10. *La característica de un cuerpo K, $char(K)$, es el menor entero positivo r tal que $r \cdot 1 = 0$. Si la relación $r \cdot 1 = 0$ implica $r = 0$, entonces $char(K) = 0$.*

Proposición 3.11. *La caracterísitica de un cuerpo es siempre cero o un número primo.*

Proposición 3.12. *Sea K un cuerpo finito. Entonces $char(K)$ es un primo p y $|K| = p^m$ para algún entero $m \geq 1$.*

Por lo tanto, si $|K| = p^m$, el orden de K^* es $\varphi(p^m) = (p-1)p^{m-1}$.

Finalmente, es importante presentar la clasificación de cuerpos finitos:

Proposición 3.13. *Para cada primo p y cada entero positivo m, existe un cuerpo con p^m elementos y dos cuerpos con p^m elementos son isomorfos.*

Como consecuencia de los últimos resultados se tiene:

1. Dado un primo p, los enteros módulo p forman un cuerpo de p elementos y, por la proposición 3.13, todo cuerpo de p elementos es isomorfo a él. El único cuerpo (salvo isomorfismo) de p elementos es denotado por $\mathbb{F}_p$.

2. Sea $q = p^m$, esto es, q es una potencia de un primo p. En este caso, dado un polinomio irreducible cualquiera $f(x)$ de grado m, se tiene que $\mathbb{F}_p[X]/f(x)$ es un cuerpo de q elementos. Por tanto, por la proposición 3.13 todo cuerpo de q elementos es isomorfo a él. El único cuerpo (salvo isomorfismo) de q elementos se denota por $\mathbb{F}_q$.

3. Por la proposición 3.12, $\mathbb{F}_p$ y $\mathbb{F}_q$ son los únicos cuerpos finitos (salvo isomorfismo).

Capítulo 4

El problema del logaritmo discreto y criptosistemas basados en él

4.1. El problema del logaritmo discreto

Sea G un grupo abeliano finito y sea $g \in G$ de orden n. Dado $a \in < g > \subseteq G$, se define el logaritmo discreto de a en base g como el entero k, $0 \leq k \leq n-1$, tal que:

$$g^k = a$$

Se dice también que k es el *índice de a en base g*.

El problema del logaritmo discreto (PLD) consiste en, dados g y a, calcular k.

Ejemplo 4.1. Sea $\mathbb{F}_{2131}^*$ el grupo multiplicativo de los enteros módulo 2131. Se tiene que $\mathbb{F}_{2131}^* = < 37 >$. Como $1217 \equiv 37^5 \pmod{2131}$, el logaritmo discreto de 1217 en base 37 es 5.

La importancia del estudio de este problema radica en el interés del logaritmo discreto como operación inversa a la exponenciación en un grupo. La exponenciación modular es una operación sencilla y se conocen métodos eficientes de calcularla como el que se expone a continuación. En cambio, el cálculo del logaritmo discreto módulo un entero cualquiera no siempre puede realizarse de forma eficiente.

Supóngase que se desea calcular b^e en el grupo multiplicativo de los enteros módulo m. En primer lugar, hay que escribir el exponente en binario: $e = \sum_{i=0}^{n-1} a_i 2^i$. A continuación, han de seguirse los siguientes pasos:

- Se define $i := n-1$ y $v := 1$.
- Para $i = n-1$ hasta 0
  1. $v = \begin{cases} v = vb \pmod{m} & \text{si } a_i = 1 \\ v = v & \text{si } a_i \neq 1 \end{cases}$
  2. $v = \begin{cases} v = v & \text{si } i = 0 \\ v = v^2 \pmod{m} & \text{si } i \neq 0 \end{cases}$

Tras la última iteración se tiene $v = b$.

Ejemplo 4.2. Sea $\mathbb{F}_5{}^*$ el grupo multiplicativo de los enteros módulo 5. Se quiere calcular 3^{27}. Esto puede hacerse de cualquiera de las formas explicadas:

- Realizando las 26 multiplicaciones: $3^{27} \equiv 2 \pmod 5$
- Utilizando el método binario de exponenciación modular. Como $27 = 2^4 + 2^3 + 2 + 1$, la representación en binario de 27 es 11011. Entonces, se toma $i = 4$, $v = 1$ y siguiendo los pasos indicados:

i	a_i	v
4	1	$(1 \cdot 3)^2 \equiv 4 \pmod 5$
3	1	$(4 \cdot 3)^2 \equiv 4 \pmod 5$
2	0	$4^2 \equiv 1 \pmod 5$
1	1	$(1 \cdot 3)^2 \equiv 4 \pmod 5$
0	1	$4 \cdot 3 \equiv 2 \pmod 5$

  Por lo tanto, $3^{27} \equiv 2 \pmod 5$. Con este método, el número necesario de multiplicaciones pasa de 26 a 8.

Esto pone de manifiesto que la exponenciación podría ser una buena función trampa. Precisamente, este es el motivo por el que algunos criptosistemas y sistemas de autentificación de mensajes e intercambio de claves basan su seguridad en el PLD. En la siguiente sección se estudian algunos de ellos.

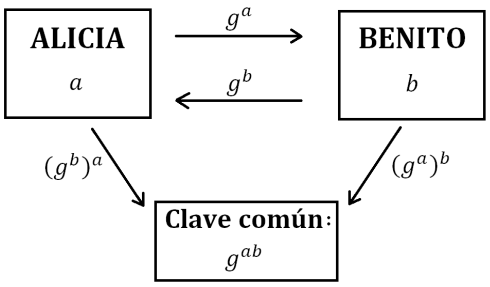
4.2. Criptosistemas basados en el logaritmo discreto

4.2.1. Intercambio de claves de Diffie-Hellman (DH)

En 1976, Whitfield Diffie y Martin Hellman [8] propusieron el primer protocolo de intercambio de clave basado en la exponenciación en cuerpos finitos. El proceso se detalla a continuación. En primer lugar, se eligen y hacen públicos un cuerpo finito $\mathbb{F}_q$ y un elemento primitivo $g \in \mathbb{F}_q$. Supóngase que dos personas Alicia (A) y Benito (B) quieren acordar una clave secreta común. Entonces proceden de la siguiente manera:

1. A y B eligen dos enteros, a y b respectivamente, con la única condición de que $2 \leq a, b \leq q - 2$ y los mantienen en secreto.
2. A transmite g^a a B y B transmite g^b a A.
3. A calcula $(g^b)^a$ y B calcula $(g^a)^b$.

La clave común será entonces g^{ab}.



Obsérvese que en la elección de a y b no se consideran los enteros 1 y $q-1$ ya que, en ambos casos, el algoritmo pierde toda su seguridad.

- Si $a = 1$ (o $b = 1$), entonces $g^a = g$ (o $g^b = g$). Por lo tanto, si se sabe que $g^a = g$ y $1 \leq a, b \leq q-1$, es fácil deducir que $a = 1$ y que la clave compartida será g^b.
- Si $a = q-1$ (o $b = q-1$), entonces $g^a = 1$ (o $g^b = 1$). Por lo tanto, si se sabe que $g^a = 1$ y que $1 \leq a, b \leq q-1$, es fácil deducir que $a = q-1$ y hallar la clave compartida $g^{ab} = (g^b)^{q-1}$.

Ejemplo 4.3. Alicia y Benito quieren establecer una clave común utilizando el método de intercambio de claves de Diffie-Hellman. Trabajan en el cuerpo $\mathbb{F}_{23}^*$ y toman 5 como elemento primitivo. Entonces Alicia, elige un entero $a = 7$ y Benito, otro $b = 13$. Alicia envía a Benito $5^a \equiv 17 \pmod{23}$ y él le envía a ella $5^b \equiv 21 \pmod{23}$. A continuación, Alicia calcula 21^7 y Benito, 17^{13}. Ambos obtienen la clave común $5^{7 \cdot 13} \equiv 10 \pmod{23}$.

Es obvio que si se dispusiese de un método eficaz para computar el logaritmo discreto, este procedimiento perdería toda seguridad. Cualquier persona que interceptase g^a y g^b podría obtener el entero a (o b) y después hallar g^{ab}, es decir, la clave común. Los autores del protocolo conjeturaron que romper este proceso es equivalente en dificultad al problema del logaritmo discreto pero aún no se ha demostrado. No se ha determinado si existen o no otras formas de calcular g^{ab} a partir de g^a y g^b, lo que se conoce como el problema de Diffie-Hellman (PDH). El estudio de estas cuestiones ha dado lugar a variantes del DHP como el problema de decisión de Diffie-Hellman, consistente en, dados g^a, g^b y un elemento $s \in \mathbb{F}_q$, decidir si $s = g^{ab}$.

A pesar de que no se ha demostrado que PDH y PLD son equivalentes en el caso general, existen resultados que prueban la equivalencia de ambos problemas bajo ciertas condiciones. Den Boer [4] probó la equivalencia en grupos $\mathbb{F}_p{}^*$ donde p es un primo tal que $\varphi(p-1)$ tiene únicamente factores primos menores que un cierto valor. Por su parte, Maurer [14] demostró la equivalencia en grupos cíclicos de orden $\prod_{i=1}^{r} p_i^{e_i}$ si para cada primo p_i se puede determinar una curva elíptica en $\mathbb{F}_{p_i}$ con ciertas propiedades cuyo orden sólo tenga divisores menores que una cierta cota.

En 1984, Varadharajan, Odoni y Sanders [20] idearon una variante del algoritmo de Diffie-Hellman utilizando la matriz compañera B de un polinomio irreducible de grado m sobre $\mathbb{F}_p$. El algoritmo consiste en los mismos pasos que el DH pero se trabaja con la matriz B en lugar de con el elemento primitivo. Sin embargo, este nuevo procedimiento no aporta mayor seguridad [21]. Incluso si la matriz B se obtuviese a partir de matrices compañeras de polinomios irreducibles de grados $m_1, ..., m_s$ el problema podría reducirse a calcular logaritmos discretos en los cuerpos $GF(p^{m_i})$.

4.2.2. Criptosistema de Massey-Omura

Este criptosistema de clave pública fue propuesto en 1982 por James Massey y Jim K. Omura [13] como caso particular del "protoloco de tres pasos" ideado por Adi Shamir alrededor de 1980. En dicho protocolo, se utiliza la conmutatividad de ciertas funciones para conseguir, en tres pasos, que dos personas intercambien un mensaje de forma segura sin compartir ninguna clave. El proceso es el siguiente:

Paso 1: El emisor del mensaje m, A, elige una clave de cifrado e_A y su correspondiente clave de descifrado d_A y envía el mensaje cifrado $E(e_A, m)$ al receptor.

Paso 2: El receptor, B, elige una clave de cifrado e_B y su correspondiente clave de descifrado d_B. A continuación, cifra el mensaje que ha recibido $E(e_B, E(e_A, m))$ y se lo envía a A.

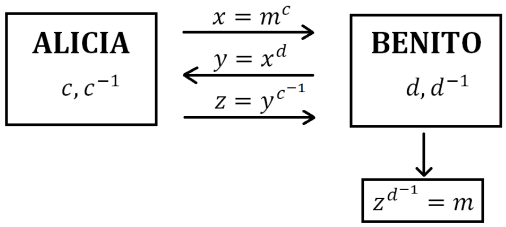
Paso 3: A descifra el mensaje recibido con su clave d_A y envía el resultado a B. Nótese que $D(d_A, E(e_B, E(e_A, m))) = E(e_B, m)$ porque la función de cifrado es conmutativa.

Finalmente, B obtiene el mensaje usando su clave de descifrado: $D(d_B, E(e_B, m)) = m$.

Se presenta ahora el criptosistema de Massey-Omura. El cuerpo en el que se trabaja, $\mathbb{F}_q$, es conocido.

Imagínese que un emisor A quiere enviar un mensaje $m \in \mathbb{F}_q{}^*$ al receptor B. En primer lugar, A elige un entero c tal que $1 \leq c \leq q-1$ con c y $q-1$ primos entre sí y calcula $c^{-1} \pmod{q-1}$. B realiza el mismo proceso, es decir, escoge un entero d con las mismas características que c y calcula $d^{-1} \pmod{q-1}$. A continuación, A y B comienzan el siguiente intercambio de mensajes cifrados:

1. A calcula $x \equiv m^c \pmod{q}$ y se lo transmite a B.
2. B calcula $y \equiv x^d \equiv (m^c)^d \pmod{q}$ y se lo envía a A.
3. A calcula $z \equiv y^{c^{-1}} \pmod{q}$ y se lo transmite a B. (Nótese que $z \equiv m^{cdc^{-1}} \equiv m^d \pmod{q}$)
4. B finalmente calcula $z^{d^{-1}} \equiv m \pmod{q}$.



Claramente, cualquier criptoanalista capaz de resolver el logaritmo discreto podría obtener el mensaje a partir de $x = m^c$, $y = m^{cd}$ y $z = m^d$:

1. Calcula d resolviendo $m^{cd} \equiv x^d \pmod{q}$ (el logaritmo discreto de m^{cd} en base x) y c resolviendo $m^{cd} \equiv z^c \pmod{q}$ (el logaritmo discreto de m^{cd} en base z).
2. Como dispone de c y d, calcula c^{-1} y $d^{-1} \pmod{q-1}$.
3. $m \equiv y^{c^{-1}d^{-1}} \pmod{q}$.

En [24], Sakurai y Shizuya probaron que, en $\mathbb{F}_p^*$ donde p es un primo tal que $p-1$ tiene sólo factores primos pequeños con respecto a una cierta cota, si se conoce la factorización de $p-1$ y g es un generador de $\mathbb{F}_p^*$, el PLD es equivalente a romper la seguridad del criptosistema Massey-Omura.

Ejemplo 4.4. Alicia quiere enviar a Benito el mensaje $m = 13$ en $\mathbb{F}_{53}^*$ utilizando el criptosistema de Massey-Omura. Para ello, Alicia elige $c = 3$ y calcula $c^{-1} = 35$. Benito escoge $d = 7$ y obtiene $d^{-1} = 15$. Entonces, comienzan el intercambio de mensajes cifrados:

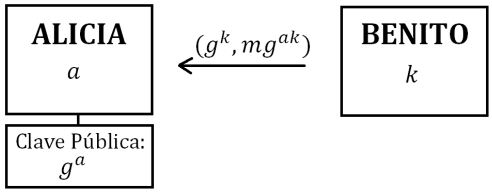
1. Alicia envía $13^3 \equiv 24 \pmod{53}$.
2. Benito calcula $24^7 \equiv 36 \pmod{53}$ y se lo envía a Alicia.
3. Alicia envía a Benito $36^{35} \equiv 15 \pmod{53}$.
4. Benito obtiene el mensaje m calculando $15^{15} \equiv 13 \pmod{53}$.

4.2.3. Criptosistema de ElGamal

Este criptosistema fue propuesto por Taher ElGamal [9] en 1985. Se conocen el cuerpo $\mathbb{F}_q$ y un elemento primitivo g del mismo.

Cierto usuario del sistema, A, elige un entero a tal que $2 \leq a \leq q-2$ y calcula g^a. El entero a es su clave privada y g^a es la clave pública. Si otro usuario, B, quiere mandar un mensaje m a A ha de hacer lo siguiente:

1. Elegir un entero k, $2 \leq k \leq q-2$
2. Enviar el par (g^k, mg^{ak}) a A.



Nota: en la elección de a y k, los enteros 1 y $q-1$ se descartan por razones similares a las expuestas en el intercambio de claves de Diffie-Hellman (ver apartado 4.2.1).

A partir del par (g^k, mg^{ak}), es fácil para A obtener el mensaje original m de la siguiente manera:

1. Calcula $g^{ak} = (g^k)^a$
2. Halla $(mg^{ak})/g^{ak} = m$

El segundo paso para recuperar el mensaje puede ser sustituido por:

1. Calcula $(g^k)^{q-1-a}$
2. Halla $(g^k)^{q-1-a} mg^{ak} = mg^{k(q-1)-ka+ak} = m(g^{q-1})^k = m$

Este método es más eficiente que el anterior puesto que evita calcular el inverso de g^{ak}.

Ejemplo 4.5. Alicia y Benito quieren intercambiar mensajes utilizando el criptosistema de ElGamal en el cuerpo $\mathbb{F}_{157}$ con generador $g = 5$. Para ello, Alicia escoge su clave privada $a = 25$ y comparte su clave pública $g^a = 34$. Supongamos que Benito quiere mandar el mensaje $m = 19$ a Alicia. Entonces elige un entero $k = 89$ y le envía el par $(5^{89}, 19 \cdot 5^{25 \cdot 89}) = (131, 45)$. Para obtener el mensaje original, Alicia halla $5^{25 \cdot 89} \equiv 85 \pmod{157}$ y calcula $45/85 \equiv 19 \pmod{157}$. Alicia también puede recuperar el mensaje a partir de $5^{89(157-1-25)} \equiv 133 \pmod{157}$ y calculando $133 \cdot 45 \equiv 19 \pmod{157}$.

Es importante señalar que el usuario B, debe elegir un entero k distinto para cada mensaje que quiera enviar a A. Supóngase que B utiliza el mismo k para codificar dos mensajes diferentes, m_1 y m_2. Esto es, B envía $(g^k, m_1 g^{ak})$ y $(g^k, m_2 g^{ak})$ a A. Entonces, cualquier criptoanalista que sepa de esta debilidad por parte de B y conozca el mensaje original m_1 correspondiente al mensaje cifrado $(g^k, m_1 g^{ak})$ puede calcular fácilmente la clave común $m_1 g^{ak}/m_1 = g^{ak}$ y proceder de la misma manera que si fuese A para obtener m_2.

Obviamente, cualquier persona capaz de resolver el PLD podría obtener a (o k) y estar en las mismas condiciones que el usuario A para obtener el mensaje original. Tsiounis y Yung [7] demostraron que la seguridad de ElGamal es equivalente al problema de decisión de Diffie-Hellman. Además, Sakurai y Shizuya [24] probaron que, en $\mathbb{F}_p^*$ donde p es un primo tal que $p-1$ tiene sólo factores primos pequeños con respecto a una cierta cota, dada la factorización de $p-1$ y si g es un generador de $\mathbb{F}_p^*$, la seguridad de este criptosistema es equivalente al PLD.

4.2.4. Algoritmo de firma de ElGamal

ElGamal [9] también propuso un algoritmo de firma basado en exponenciación en cuerpos finitos $\mathbb{F}_p$ con p primo. Imagínese que uno de los participantes, A, quiere firmar los mensajes. Para ello A publica un primo p, un elemento primitivo g módulo p y un entero y, $1 \leq y \leq p-1$, tal que $y = g^a$ donde a es un entero cualquiera que mantiene en secreto. El primo p y g pueden ser elegidas por cada usuario o ser las mismas para todos los participantes.

Clave Pública
(p, g, y)

Clave Privada
a

Para firmar un mensaje m, A procede así:

1. Elige un entero k con $(k, p-1) = 1$ y calcula $r = g^k$.
2. Calcula s a partir de la ecuación: $m \equiv ar + ks \ (mod\ p-1)$ (que tiene solución única puesto que $(k, p-1) = 1$)

La firma de A para el mensaje m es, por lo tanto, el par (r, s).

Obviamente, si alguien fuese capaz de calcular logaritmos discretos y obtener el entero a a partir de y, podría firmar mensajes como si fuese el usuario A. Es más, no se ha encontrado ninguna forma de romper la seguridad de este sistema de firma sin utilizar logaritmos discretos.

Si un usuario quiere verificar que un mensaje m procede efectivamente de A, debe verificar si:

$$g^m \equiv y^r r^s \ (mod\ p)$$

ya que, si efectivamente A es el emisor:

$$y^r r^s \equiv g^{ar} g^{ks} \equiv g^{ar+ks} \equiv g^m \ (mod\ p)$$

Ejemplo 4.6. Cierto usuario toma como clave privada $a = 33$ y comparte la siguiente clave pública: $(151, 6, 60)$. Si el usuario quiere firmar el mensaje $m = 79$ debe proceder como sigue:

1. Elige un cierto k, por ejemplo, $k = 11$ y calcula $r \equiv 6^{11} \equiv 77 \pmod{151}$
2. Calcula $s \equiv (79 - 33 \cdot 77)/11 \equiv 8 \pmod{150}$

La firma del mensaje $m = 79$ es $(77, 8)$. Si el destinatario del mensaje quiere verificar la autoría del mismo, ha de comprobar que $6^{79} \equiv 63 \equiv 60^{77} \cdot 77^8 \pmod{151}$.

4.2.5. Algoritmo de firma digital (DSA)

DSA es un estándar del Gobierno Federal de los Estados Unidos de América para firmas digitales. Fue propuesto por el NIST (National Institute of Standards and Technology) [17] en 1994 y el esquema de pasos a seguir para firmar un mensaje es similar al del algoritmo de firma ElGamal.

Con este algoritmo, si un usuario A quiere firmar un mensaje lo primero que debe hacer es establecer la clave pública (p, q, g, y) y la privada x. Para elegir dichas claves, ha de seguir las siguientes instrucciones[1]

1. Eligir un primo, p, de tamaño L, donde $512 \leq L \leq 1024$ y $64 \mid L$.
2. Escoger otro primo, q, de tamaño 160, tal que $p \equiv 1 \pmod{q}$.
3. Sea h un entero tal que $1 < h < p - 1$ y $h^{(p-1)/q} \neq 1 \pmod{p}$. Tomar $g \equiv h^{(p-1)/q} \pmod{p}$. Las condiciones impuestas sobre h garantizan que g es un generador del único subgrupo cíclico de orden q de $\mathbb{F}_p^*$. Como $g^q \equiv h^{p-1} \equiv 1 \pmod{p}$, el orden de g es divisor de q, esto es, 1 o q, pero no puede ser 1 ya que $g \neq 1$. Entonces el orden de g es q y, por lo tanto, genera el único subgrupo de orden q de $\mathbb{F}_p^*$.
4. Escoger x tal que $1 < x < q - 1$.
5. Calcular $y \equiv g^x \ (mod\ p)$

El usuario A está ahora en disposición de firmar su mensaje m. Debe obtener un par de enteros (r, s) a través de los siguientes cálculos:

1. Elige un entero k verificando $0 < k < q$.
2. Obtiene $r \equiv (g^k \pmod{p}) \pmod{q}$
3. Calcula $s \equiv k^{-1}(H(m) + xr) \pmod{q}$, donde H es la función hash SHA-1.[2].

Si el receptor del mensaje firmado quisiera asegurarse de que este ha sido realmente enviado por A, debería realizar los siguientes cálculos:

[1] Las condiciones de tamaño para los enteros p y q de los pasos 1 y 2, respectivamente, no tienen justificación matemática, sino que responden a cuestiones computacionales.

[2] Una función hash es una aplicación $h : \Sigma^* \longrightarrow \Sigma^n, n \in \mathbb{N}$ que transforma una cadena de longitud arbitraria en una de longitud fija. Dicha aplicación debe cumplir además ciertas características adicionales [27]

1. $w = s^{-1} \pmod q$
2. $u1 = H(m)w \pmod q$
3. $u2 = rw \pmod q$
4. Finalmente, verifica si $g^{u1}y^{u2} \equiv r \pmod p$.

Efectivamente, si A es el firmante, se tiene:

- $g^q \equiv h^{(p-1)} \equiv 1 \pmod p$
- $k \equiv H(m)s^{-1} + xrs^{-1} \equiv H(m)w + xrw \ (mod\ q)$

Por tanto,

$$g^k \equiv g^{H(m)w+xrw+zq} \equiv g^{H(m)w} g^{xrw} g^{qz} \equiv g^{u1}y^{u2} \pmod p$$

Al igual que en el algoritmo de firma de ElGamal, cualquier persona capaz de calcular logaritmos discretos podría calcular x y hacerse pasar por el usuario A.

4.2.6. Algoritmo de Blum-Micali

El algoritmo de Blum-Micali [3] es un generador de números pseudoaleatorios que basa su seguridad en la dificultad de calcular logaritmos discretos. Las secuencias de números pseudoaleatorios tienen aplicación en múltiples campos tales como la Criptografía, la simulación y la Estadística.

Sea p un primo impar y sea g un elemento primitivo módulo p. Dado un valor inicial x_0, se define la sucesión:

$$x_{i+1} \equiv g^{x_i} \ (mod\ p)$$

A partir de x_i se genera el bit pseudoaleatorio:

$$y_i = \begin{cases} 1 & \text{si } x_i \leq (p-1)/2 \\ 0 & \text{en otro caso} \end{cases}$$

Esto es, $y_i = 1$ cuando x_i es la raíz principal de x_i^2 módulo p, teniendo en cuenta que, dado un residuo cuadrático $T^2 = g^{2s}$ módulo p, se dice que g^s con $s \in [1, (p-1)/2]$ es la raíz principal de T y que $g^{s+(p-1)/2}$ es la raíz no principal.

Definida de esta forma, se tiene que $\{y_i\}_{i=1}^k$ es una secuencia pseudoaleatoria de k bits.

En [3], Blum y Micali demostraron que cualquier estrategia eficiente para predecir la secuencia puede ser transformada en una estrategia igualmente eficiente para resolver el PLD. Esto se debe a que disponer de un oráculo para la raíz principal permite construir un algoritmo que resuelva el PLD módulo p. Sin embargo, dicho problema es intratable para primos suficientemente grandes y, por lo tanto, la seguridad del algoritmo de Blum-Micali reside en la dficultad del PLD.

Ejemplo 4.7. Sea $p = 67$ y $g = 2$. Dado $x_0 = 3$ y $k = 15$, empleando el algoritmo de Blum-Micali se obtiene la siguiente secuencia pseudoaleatoria de 15 bits:

101111100100000

4.2.7. Criptosistema de Ciss-Cheikh-Sow

En [7], Ciss, Cheikh y Sow proponen un criptosistema de clave pública cuya seguridad se basa en los problemas de factorización y logaritmo discreto.

Cada usuario debe generar sus claves pública y privada, procediendo de la siguiente manera:

1. Elige dos primos p y q congruentes con 2 (mód 3) y calcula $pq = n$.
2. Toma g un elemento primitivo de $\mathbb{Z}_n$.
3. Toma un $k < \varphi(n)$ y calcula $y \equiv g^k \pmod{n}$

La clave privada de este usuario, A, es (p, q, k) y la pública,(n, g, y)

Si otro usuario, B, quiere mandar un mensaje m a A debe seguir los siguientes pasos:

1. Elige un entero $s < n$
2. Calcula $c_1 \equiv (my^s)^3 \pmod{n}$ y $c_2 \equiv g^s \pmod{n}$

B envía el par (c_1, c_2) a A.

Para descifrar el mensaje, A calcula:

$$(c_1)^{1/3}(c_2)^{-k} \equiv (mg^{ks})g^{-ks} \equiv m \pmod{n}$$

Es importante notar que A puede calcular la raíz cúbica de $c_1 \pmod{n}$ (y es única) porque la ecuación $x^3 \equiv c_1$ tiene solución única módulo p y módulo q. Esto último se debe a que, si p es un primo tal que $p \equiv 2 \pmod{3}$, entonces la función:

$$\begin{array}{ccc} \mathbb{Z}_p & \longrightarrow & \mathbb{Z}_p \\ x & \longmapsto & x^3 \end{array}$$

es biyectiva con inversa:

$$\begin{array}{ccc} \mathbb{Z}_p & \longrightarrow & \mathbb{Z}_p \\ x & \longmapsto & x^{1/3} \equiv x^{(2p-1)/3} \end{array}$$

La seguridad del algoritmo recae tanto en la factorización de n como en el PLD. Si un criptoanalista es capaz de factorizar n, puede obtener $(c_1)^{1/3} \equiv my^s \pmod{n}$ resolviendo la raíz cúbica módulo p y módulo q y utilizando el teorema chino de los restos. Sin embargo, también necesita obtener s, esto es, necesita calcular el logaritmo discreto de c_2 en base g módulo n, para poder calcular y^s y despejar m en la ecuación anterior. Por otra parte, si el criptoanalista es capaz de calcular logaritmos discretos, entonces, puede obtener la clave privada k y calcular $c_2 \equiv y^{ks} \pmod{n}$. No obstante, también necesita calcular $(c_1)^{1/3} \equiv my^{ks} \pmod{n}$ (para después hallar m) y para calcular esta raíz cúbica, es necesario conocer la factorización de n. Por lo tanto, queda probado que saber resolver uno de los dos problemas no permite obtener el mensaje original y que, en consecuencia, la seguridad del criptosistema se basa tanto en la factorización de n como en el PLD.

Ejemplo 4.8. Cierto usuario, Alicia, elige $p = 113$ y $q = 353$ y calcula $n = 39889$. A continuación toma un elemento primitivo de $\mathbb{Z}_{39889}$, $g = 3$. Elige $k = 145$ y calcula $y \equiv 3^{145} \equiv 29125 \pmod{39889}$. La clave privada de Alicia es $(113, 353, 145)$ y la clave pública, $(39889, 3, 29125)$. Si Benito quiere enviarle un mensaje, por ejemplo, $m = 3168$ debe hacer lo siguiente:

1. Elige $s = 5671$
2. Calcula $c_1 \equiv (3168 \cdot 29125^{5671})^3 \equiv 4914 \pmod{39889}$ y $c_2 \equiv 3^{5671} \equiv 38253 \pmod{39889}$

Benito envía el par $(4914, 38253)$ a Alicia. Esta, para recuperar el mensaje calcula:

$$4914^{1/3} \cdot 38253^{-145} \equiv 3168 \pmod{39889}$$

Para hallar $4914^{1/3} \pmod{39889}$ Alicia calcula $4914^{1/3} \equiv 4914^{(2\cdot 113-1)/2} \equiv 54 \pmod{113}$ y $4914^{1/3} \equiv 4914^{(2\cdot 353-1)/2} \equiv 264 \pmod{353}$ y utiliza el teorema chino de los restos.

Capítulo 5

Métodos de cálculo del logaritmo discreto

Hasta ahora se han presentado algunas de las aplicaciones del logaritmo discreto en Criptografía. Un método eficaz para resolver el PLD pondría en jaque la seguridad de todos los sistemas basados en él que son utilizados hoy en día. En este capítulo se estudian algunos de los algoritmos existentes para el cálculo del logaritmo discreto: Dichos algoritmos pueden clasificarse en tres tipos:

1. Algoritmos genéricos, esto es, algoritmos válidos en cualquier grupo.
2. Algoritmos para grupos cuyo cardinal tiene todos sus factores primos *pequeños*.
3. Algoritmos que utilizan propiedades particulares del grupo (en cuanto a su estructura).

En lo que sigue, salvo que se especifique otra cosa, se trabajará en un grupo cíclico $G =< g >$ de orden n y el objetivo será calcular el logaritmo discreto de $a \in G$ en base g, esto es, calcular k tal que $g^k = a$.

5.1. Algoritmos Genéricos

5.1.1. Fuerza Bruta

La manera más obvia de calcular el logaritmo discreto de un elemento a en base g es calcular las diferentes potencias de g, almacenarlas en una tabla y buscar el elemento k en dicha tabla. Claramente este método no es eficaz cuando el orden del grupo es grande.

Ejemplo 5.1. Se desea calcular en $\mathbb{F}_{97}^*$ el logaritmo discreto de 36 en base 5. Para ello, se construye la tabla de las sucesivas potencias de 5:

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5^j	1	5	25	28	43	21	8	40	6	30	53	71	64	29	48	46	36

Por lo tanto, $36 \equiv 5^{16} \pmod{97}$. Esto es, el logaritmo discreto de 36 en base 5 es 16.

5.1.2. Algoritmo de Shanks: "Baby step-Giant step"

Obviamente la fuerza bruta no es un buen método para calcular logaritmos discretos puesto que sólo es eficiente en grupos de orden pequeño. El método que se introduce en este apartado puede aplicarse de manera eficaz a un mayor número de grupos.

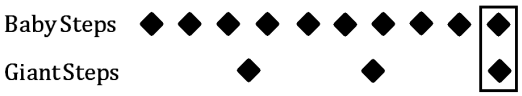
Todo elemento a de un grupo G de orden n puede ser expresado de manera única como $a = g^k$ donde k es un entero no negativo menor que $n-1$. Sea $m = \lceil \sqrt{n} \rceil$ (la parte entera de $\sqrt{n}$ por arriba), entonces, todo k puede expresarse como $k = mq + r$ con $0 \leq q, r \leq m-1$, simplemente escribiendo k en base m. Este hecho es la base del método de Shanks [25].

El método de Shanks para calcular el logaritmo discreto de a en base g consiste en los siguientes pasos:

1. Calcular los conjuntos:
   Baby Steps: $\{ag^{-i} : 0 \leq i \leq m-1\} = \{a, ag^{-1}, ag^{-2}, \dots, ag^{-(m-1)}\}$
   Giant Steps: $\{g^{jm} : 0 \leq j \leq m-1\} = \{1, g^m, g^{2m}, \dots, g^{(m-1)m}\}$

2. Buscar en esos conjuntos elementos que tengan el mismo valor. Es decir, buscar un elemento ag^{-i} en el primer conjunto y otro elemento g^{jm} en el segundo tales que $ag^{-i} = g^{jm}$.

3. Despejando a, se tiene que: $a = g^{jm+i}$ y, por lo tanto, el logaritmo discreto de a en base g es $k = jm + i$.



Nótese, atendiendo a las consideraciones iniciales que los elementos buscados en el paso 2 existen por la manera en que se han definido los conjuntos. De hecho, puede existir más de un par de elementos verificando la condición requerida y las soluciones obtenidas al considerar los distintos pares difieren en un múltiplo de n. Además, el método también proporciona la solución del PLD si n es una cota superior del orden del grupo.

Ejemplo 5.2. Se quiere calcular el logaritmo discreto de $a = 59$ en base $g = 7$ en el grupo ${\mathbb{F}_{71}}^*$ mediante el método de Shanks:

- $m = \lceil \sqrt{70} \rceil = 9$

Baby Steps	
i	$59 \cdot 7^{-i}$
0	59
1	49
2	7
3	1
4	61
5	29
6	65
7	60
8	39

Giant Steps	
j	7^{9j}
0	1
1	47
2	8
3	21
4	64
5	26
6	15
7	66
8	49

Se observa que $1 \equiv 59 \cdot 7^{-3} \equiv 7^0 \pmod{71}$. Entonces, $59 \equiv 7^3 \pmod{71}$, esto es, el logaritmo discreto de 59 en base 7 es 3. También se tiene que $59 \cdot 7^{-1} \equiv 7^{8 \cdot 9} \pmod{71}$, es decir, $59 \equiv 7^{73} \pmod{71}$ pero $73 \equiv 3 \pmod{\varphi(71)}$ y, por lo tanto, se obtiene la misma solución. En la segunda tabla se podrían haber omitido los cálculos para j mayor que 0, en cambio, estos se han realizado para comprobar que, efectivamente, pueden existir varias colisiones.

5.1.3. Algoritmo ρ de Pollard

El método ρ de Pollard [23] consiste en construir una secuencia pseudoaleatoria de elementos de G en la que existan dos términos iguales y, a partir de esos términos, calcular el logaritmo discreto. Se trata del algoritmo genérico más eficaz y, por ello, además de estudiar el método original, se presentan algunas variantes del mismo. A continuación se expone el método original en detalle.

En primer lugar, se toma una partición de G en tres conjuntos de aproximadamente el mismo tamaño S_1, S_2 y S_3 con la única condición de que 1 no pertenezca a S_2. La secuencia estará dada por:

$$x_0 = 1; \quad x_{i+1} = \begin{cases} x_i a & \text{si} \quad x_i \in S_1 \\ x_i^2 & \text{si} \quad x_i \in S_2 \\ x_i g & \text{si} \quad x_i \in S_3 \end{cases}$$

Si 1 está en S_2, entonces la secuencia es constante igual a 1. Esto explica la condición requerida para S_2 a la hora de elegir la partición de G.

Como $a = g^k$, se tiene que $x_i = g^{\alpha_i} a^{\beta_i}$ y los enteros α_i y β_i pueden calcularse independientemente del valor x_i de forma recurrente:

$$\alpha_0 = 0; \quad \alpha_{i+1} = \begin{cases} \alpha_i & \text{si} \quad x_i \in S_1 \\ 2\alpha_i & \text{si} \quad x_i \in S_2 \\ \alpha_i + 1 & \text{si} \quad x_i \in S_3 \end{cases}$$

$$\beta_0 = 0; \quad \beta_{i+1} = \begin{cases} \beta_i + 1 & \text{si} \quad x_i \in S_1 \\ 2\beta_i & \text{si} \quad x_i \in S_2 \\ \beta_i & \text{si} \quad x_i \in S_3 \end{cases}$$

Cuando se encuentran dos enteros i, j tales que $x_i = x_j$, se verifica que:

$$g^{\alpha_i} a^{\beta_i} = g^{\alpha_j} a^{\beta_j}$$

esto es,

$$g^{\alpha_i} g^{k\beta_i} = g^{\alpha_j} g^{k\beta_j}$$

y, por lo tanto,

$$g^{\alpha_i - \alpha_j} = g^{k(\beta_j - \beta_i)}$$

de donde se obtiene que:

$$\alpha_i - \alpha_j \equiv k(\beta_j - \beta_i) \pmod{n} \tag{5.1}$$

Denotando $u = \alpha_i - \alpha_j$ y $v = \beta_j - \beta_i$ se tiene que $g^u = g^{kv}$. Es claro que, si $v = 0$, el algoritmo no puede calcular el logaritmo discreto. En tal caso, se toma otra semilla, esto es, otro valor para x_0 y se repite el proceso. Si $v \neq 0$ se aplica el algoritmo extendido de Euclides para resolver la ecuación (5.1). Sea $d = (n, v)$, entonces:

$$d = vs + nt$$

Como

$$g^{us} = g^{kvs} = g^{k(d-nt)} = g^{kd}$$

se tiene que

$$us \equiv kd \pmod{n}$$

y por lo tanto

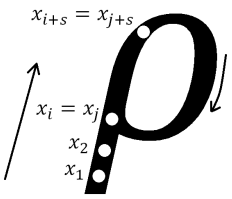
$$us - wn = kd$$

Puesto que $d \mid n$, $d \mid us$ y, finalmente, se obtiene que:

$$k = (us - wn)/d$$

donde w es un entero entre 0 y d que puede obtenerse por búsqueda exhaustiva.

Es fácil observar que, una vez que se da $x_i = x_j$, los elementos que se generan a partir del paso j son los mismos que los que se generan a partir del paso i y, por lo tanto, la secuencia entra en un bucle. Dibujando la situación se obtiene una forma parecida a la letra ρ y es este hecho el que da nombre al algoritmo.



Ejemplo 5.3. El entero $p = 809$ es primo y $g = 89$ tiene orden 101 en $\mathbb{F}^*_{809}$. El elemento 618 pertenece al subgrupo generado por g. Se desea calcular el logaritmo en base 89 de 618 en $\mathbb{F}^*_{809}$. En primer lugar, se toma la siguiente partición del grupo $\mathbb{F}^*_{809}$:

$S_1 = \{x \in \mathbb{F}^*_{809} : x \equiv 1 \pmod{3}\}$
$S_2 = \{x \in \mathbb{F}^*_{809} : x \equiv 0 \pmod{3}\}$
$S_3 = \{x \in \mathbb{F}^*_{809} : x \equiv 2 \pmod{3}\}$

En este momento se genera la secuencia aleatoria aplicando la función definida en la descripción del método y se busca una colisión en la misma.

i	(x_i, α_i, β_i)
1	(618,0,1)
2	(76,0,2)
3	(46,0,3)
4	(113,0,4)
5	(349,1,4)
6	(488,1,5)
7	(555,2,5)

i	(x_i, α_i, β_i)
8	(605,4,10)
9	(451,5,10)
10	(422,5,11)
11	(344,6,11)
12	(683,7,11)
13	(112,8,11)
14	(451,8,12)

Se tiene que $x_9 = x_{14} = 451$. Por tanto, el logaritmo discreto de 618 en base 89 es:

$$k = (8-5)(10-12)^{-1} = 3 \cdot 99^{-1} = 3 \cdot 50 = 49 \pmod{101}$$

Variantes del algoritmo

En este apartado se necesita la siguiente definición:

Definición 5.4. *Decimos que en una secuencia, $\{x_i\}_{i\in\mathbb{N}}$, existe un ciclo de longitud λ si existe μ tal que para todo índice i mayor o igual que μ se tiene que $x_i = x_{i+\lambda}$. En tal caso, se verifica también que $x_i = x_{i+k\lambda}$ para todo entero k no negativo.*

Nótese, que en el método de Pollard no se requiere encontrar los valores de λ y μ, basta con hallar dos elementos iguales de la secuencia.

En lo que sigue, siempre que no se diga nada, se considerará que λ y μ son los menores enteros positivos verificando las condiciones de la definición de ciclo. Conviene comentar que $\mu + \lambda$ mide el número máximo de iteraciones porque el elemento $x_{\mu+\lambda}$ es el primer elemento repetido de la secuencia. Así, cuanto menores sean ambos valores, más rápido se hallará una colisión y, por tanto, más rápido será el método. Se introduce a continuación un resultado que será útil para estudiar las mejoras en la eficacia de ejecución de las variantes del algorirmo de Rho.

Teorema 5.5 (Harris [10])**.** *Bajo la hipótesis de que una función de iteración, $f : G \to G$, se comporta como una función aleatoria, las esperanzas de λ y μ son la misma, $\sqrt{\pi n/8} \approx 0{,}63\sqrt{n}$, y la media del número máximo de evaluaciones necesarias antes de encontrar un colisión en la secuencia generada por f es $E(\mu + \lambda) = \sqrt{\pi n/2} \approx 1{,}25\sqrt{n}$, suponiendo que se guardan todos los elementos de la secuencia. (Recordemos que $n = |G|$)*

Es importante remarcar la importancia de la hipótesis sobre la función de iteración. Teske [29] probó empíricamente que la función iteración de Pollard no se comporta exactamente como una función aleatoria y, por lo tanto, los resultados obtenidos son peores que los del Teorema 5.5. Por ejemplo, en $\mathbb{F}_p^*$, donde p es primo, $E(\mu + \lambda) \approx 1{,}37\sqrt{n}$ y en subgrupos de orden primo de dicho grupo, $E(\mu + \lambda) \approx 1{,}55\sqrt{n}$.

Se pasan a presentar ahora diversas variantes del método ρ. En primer lugar, cabe mencionar que el algoritmo original no necesita almacenar toda la secuencia para encontrar un par de elementos iguales, sino que, basta con que se trabaje con los conjuntos:

$$(x_i, \alpha_i, \beta_i; x_{2i}, \alpha_{2i}, \beta_{2i}) \text{ para } i = 1, 2, ...$$

De hecho, este es el método que Pollard utilizó cuando presentó su algoritmo.

Teorema 5.6 (**Detección de ciclos de Floyd**). *Si en una secuencia, $\{x_i\}_{i\in\mathbb{N}}$, existe un ciclo, entonces existe un número natural i tal que $x_i = x_{2i}$. El menor i que verifica $x_i = x_{2i}$ cumple que $\mu \leq i \leq \mu+\lambda$, donde μ es el índice del primer elemento del ciclo y λ es la longitud del mismo.*

Demostración. Por definición de ciclo, existen λ y μ tales que para todo entero i mayor o igual que μ se tiene que $x_i = x_{i+k\lambda}$. Por tanto, siempre que $i = k\lambda \geq \mu$ se verifica que $x_i = x_{2i}$.
Es fácil ver que el menor i que verifica $x_i = x_{2i}$ cumple que $\mu \leq i \leq \mu + \lambda$. Efectivamente, i no puede ser menor que μ porque entonces x_i no sería un elemento dentro del ciclo. Además, si i fuese mayor que $\mu + \lambda$,

$$x_i = x_{\mu+\lambda+k} = x_{\mu+k} \text{ y } x_{2i} = x_{2(\mu+\lambda+k)} = x_{2(\mu+k)}$$

Entonces dicho i no sería el menor satisfaciendo la condición requerida ya que $x_{\mu+k} = x_{\mu+\lambda+k} = x_{2(\mu+\lambda+k)} = x_{2(\mu+k)}$. □

Ejemplo 5.7. Se considera la secuencia:

$$5, 7, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, ...$$

Se procede a hallar una colisión, utilizando el algoritmo de detección de ciclos de Floyd.

i	1	2	3	4	5	6	7	8	9	10
x_i	5	7	8	22	33	15	46	39	11	12
x_{2i}	7	22	15	39	12	26	22	15	39	12

Por lo tanto, $x_{10} = x_{20} = 12$. Obsérvese que el algoritmo no encuentra el primer elemento repetido de la secuencia, que sería $x_3 = x_{13} = 8$,sino que tan solo proporciona una colisión.

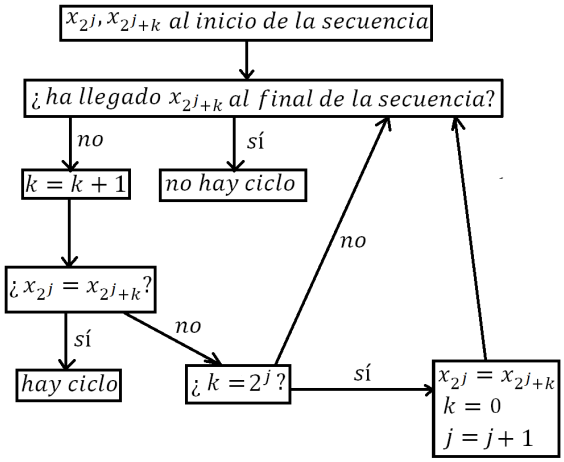
Este teorema explica que para encontrar dos elementos iguales en la sucesión de Pollard baste con comparar entre sí los elementos x_i y x_{2i}. Además, como $\mu \leq i \leq \mu+\lambda$, para encontrar una colisión, se necesitan μ iteraciones en el mejor de los casos y $\mu + \lambda$ en el peor. Aunque el método requiere muy poco espacio de almacenamiento, suponiendo que f se comporta como una función aleatoria, el número máximo de iteraciones necesarias es aproximadamente $1{,}03\sqrt{n}$ ([2]) y, como en cada iteración hay tres evaluaciones y una comparación, se precisan $1{,}03\sqrt{n}$ comparaciones y $3{,}09\sqrt{n}$ evaluaciones aproximadamente. Por tanto, la mejora de este método reside en la disminución de almacenaje ya que el número de evaluaciones es muy superior al del Teorema 5.5. El hecho de que en cada iteración haya tres evaluaciones se explica por lo siguiente: como en la iteración i, tenemos calculados x_i y x_{2i}, en la iteración $i+1$ necesitamos calcular $x_{i+1} = f(x_i)$, $x_{2i+1} = f(x_{2i})$ y $x_{2(i+1)} = x_{2i+2} = f(x_{2i+1})$.

Tal y como se ha visto en el ejemplo, el método de Floyd no proporciona necesariamente el primer elemento repetido. Tampoco tiene porqué proporcionar la longitud del ciclo. Sin embargo, es posible calcular λ y μ a partir de la colisión encontrada. Cuando se encuentra i tal que $x_i = x_{2i}$, se obtiene un periodo $\nu = 2i - i = k\lambda$ que es múltiplo de la longitud del ciclo. Una vez conocido ν, se traza la secuencia desde el primer término para encontrar el primer valor repetido x_μ sabiendo que, como λ es múltiplo de μ, $x_\mu = x_{\mu+\nu}$. Esto quiere decir, que sólo se necesita comparar los elementos x_j y $x_{j+\nu}$ hasta hallar un valor de j para el que se verifique la igualdad. Cuando ya se ha obtenido μ, es fácil hallar λ comparando los elementos x_μ y $x_{\mu+l}$. Entonces λ será el menor valor de l para el que se verifique la igualdad.

Se presentan a continuación algunas variantes del método que pueden ser agrupadas en dos tipos: las que proponen formas diferentes para hallar elementos repetidos en la secuencia y las que usan diferentes funciones de iteración.

En 1980, Brent [5] aumentó la velocidad del método de Rho al emplear otro algoritmo de detección de ciclos. El algoritmo de Brent utiliza, al igual que el algoritmo de Floyd, dos variables. Sin embargo, está basado en una idea diferente. La base de este algoritmo es que dado un elemento del ciclo, es posible encontrar la longitud del ciclo en λ pasos (de la misma manera en que se hallaba en el algoritmo de Floyd). Se van tomando elementos de la forma x_{2^j} y se comprueba si están o no en el ciclo hasta que uno de ellos verifica esa condición.

Se consideran las variables x_{2^j} y x_{2^j+k}. Inicialmente, estas variables toman los dos primeros valores de la secuencia. Entonces, se va incrementando el valor de k en una unidad, y comprobando en cada caso si $x_{2^j} = x_{2^j+k}$. Si ambas variables no coinciden para ningún valor de k menor o igual que 2^j, se actualiza $j = j + 1$, $k = 1$ y se repite el proceso. El valor de j indica el paso en que se encuentra el proceso. El siguiente esquema pretende clarificar el proceso:



Teorema 5.8 (**Detección de ciclos de Brent**). *Si en una secuencia $\{x_i\}_{i\in\mathbb{N}}$ existe un ciclo de longitud λ, entonces existe un número natural j tal que $x_{2^j} = x_{2^j+\lambda}$. Además, el algoritmo que se acaba de describir encuentra dicho valor j cuando $2^j \geq \max(\mu, \lambda)$, donde μ es el subíndice del primer elemento del ciclo.*

Demostración. Por definición, como la secuencia tiene un ciclo, existen λ y μ tales que $x_i = x_{i+\lambda}$ para todo i mayor o igual que μ. Entonces, siempre que $i = 2^j \geq \mu$ se verifica que $x_{2^j} = x_{2^j+\lambda}$.
Se prueba ahora la segunda parte del teorema. Si $2^j \geq \max(\mu, \lambda)$ es inmediato comprobar que el algoritmo halla una colisión pues $2^j \geq \mu$, esto es, x_{2^j} es un elemento del ciclo que se compara con x_{2^j+k} donde $1 \leq k \leq 2^j$. Es decir, se compara un elemento del ciclo con $2^i \geq \lambda$ elementos consecutivos, lo que forzosamente lleva a encontrar una colisión. Por otro lado, si el algoritmo encuentra una colisión, x_{2^j} ha de estar en el ciclo, esto es, $2^j \geq \mu$, y ha de ser comparado con, al menos, λ elementos consecutivos, es decir $k \geq \lambda$ y, por tanto, $2^j \geq \lambda$. Esto quiere decir, que no puede encontrar j con $2^j < \max(\lambda, \mu)$. □

Este algoritmo presenta dos ventajas con respecto al anterior. Aunque el espacio de almacenamiento requerido tanto en el método de Floyd como en el de Brent es similar y mínimo, el algoritmo de Brent necesita menos operaciones. El número máximo de iteraciones necesarias es aproximadamente $1{,}98\sqrt{n}$, suponiendo de f se comporta como una función aleatoria [2], y en cada iteración hay una evaluación y una comparación. Por tanto, si el coste de las comparaciones es insignificante, el algoritmo de Brent es más rápido que el de Floyd. Por otro lado, Brent encuentra la longitud del ciclo, λ, directamente. Si se desease hallar μ habría que proceder de manera análoga a como se comentó en el algoritmo de Floyd.

Ejemplo 5.9. Se va a hallar una colisión de la secuencia del ejemplo 5.7 mediante el algoritmo de detección de ciclos de Brent.

$$5, 7, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, ...$$

$x_1 = 5$	
k	1
x_{1+k}	7

$x_2 = 7$		
k	1	2
x_{2+k}	8	22

$x_4 = 22$				
k	1	2	3	4
x_{4+k}	33	15	46	39

$x_8 = 39$								
k	1	2	3	4	5	6	7	8
x_{8+k}	11	12	31	26	8	22	33	15

$x_{16} = 15$										
k	1	2	3	4	5	6	7	8	9	10
x_{16+k}	46	39	11	12	31	26	8	22	33	15

El algoritmo detecta la colisión $x_{16} = x_{26} = 15$ y proporciona la longitud del ciclo $\lambda = 10$.

Brent mejoró dicha variante en el mismo artículo demostrando que podían evitarse comparaciones innecesarias. Probó que basta con comparar cada x_{2^j} con los términos x_{2^j+k} donde $\frac{3}{2}2^j < 2^j + k \leq 2^{j+1}$, esto es, $2^{j-1} < k \leq 2^j$. Es decir, en esta versión solo se considera la mitad de valores posibles para k. Sea $\{x_{2^j+1}, x_{2^j+2}, ..., x_{\frac{3}{2}2^j}\}$ y $\{x_{\frac{3}{2}2^j+1}, x_{\frac{3}{2}2^j+2}, ..., x_{2^j+2^j}\}$ una partición de los elementos considerados en cada iteración del algoritmo original. Cada uno de los conjuntos tiene 2^{j-1} elementos. Si $\lambda \leq 2^{j-1}$, esto es $x_{2^j+\lambda}$ está en el primer conjunto, entonces x_{2^j} también es igual a un elemento del segundo conjunto (porque $x_{2^j+2\lambda}$ pertenece a él). Entonces, basta con buscar colisiones en el segundo conjunto, pero en este caso no siempre se halla el valor de λ directamente. En esta variante del algoritmo, el número máximo de evaluaciones está acotado por $2{,}24\sqrt{n}$ y el de comparaciones por $0{,}88\sqrt{n}$.

Por otra parte, Teske [28] introdujo una variante del algoritmo que reduce el número de iteraciones a cambio de almacenar más elementos de la secuencia y utilizar más comparaciones. El algoritmo de Teske utiliza un vector de tamaño t, $(x_{\sigma_1}, ..., x_{\sigma_t})$ con $t \geq 2$, cuyas componentes albergan inicialmente el valor de x_0 y se actualizan de la siguiente manera. En la i-ésima iteración, se compara el valor de x_i con el de cada componente del vector. Si no hay ningún valor igual, se comprueba si i es mayor o igual que v veces el índice del elemento en la primera componente para un cierto valor de v previamente fijado. Si es así, se elimina el elemento de la primera componente, se mueve el elemento de cada componente a la anterior ($x_{\sigma_{k-1}} = x_{\sigma_k}$), se guarda x_i en la última componente ($x_{\sigma_t} = x_i$) y se pasa a la siguiente iteración. Si no, el vector no se modifica y continúa con el proceso.

Teorema 5.10 (**Detección de ciclos de Teske**). *Sea $\{x_i\}_{i\in\mathbb{N}}$ una secuencia con un ciclo de longitud λ y sean v,t enteros positivos fijos tales que $v\mu \geq \lambda+\mu$ donde μ es el subíndice del primer elemento del ciclo. En estas condiciones, el algoritmo anterior encuentra una colisión.*

Demostración. Como la secuencia presenta un ciclo, para todo i mayor o igual que μ se tiene que $x_i = x_{i+\lambda}$. Además, si $k \geq \mu$ se verifica que $vk \geq \lambda+k$. Como $k \geq \mu$, existe s con $k = \mu+s$ y, por tanto,

$$vk = v(\mu+s) = v\mu + vs \geq \lambda+\mu+vs \geq \lambda+\mu+s = \lambda+k$$

Entonces, si se almacena un cierto x_k con $k \geq \mu$ y se compara con los sucesivos términos de la secuencia, se garantiza que se encuentra un elemento x_l con $x_k = x_l$ y $k < l \leq vk$, por ejemplo, $l = k+\lambda$.
Ha de probarse entonces que en algún momento el vector contiene un elemento x_k con $k \geq \mu$. Es claro que el algoritmo no puede hallar ninguna colisión antes de llegar a la iteración μ. Supóngase que en este momento se tiene el vector $(x_{\sigma_1}, ..., x_{\sigma_t})$. Entonces, se ha de verificar si $\mu+h \geq v\sigma_1$ para $h \geq 0$. Como $v\sigma_1$ es un valor fijo, la desigualdad será cierta para algún h y, en ese momento, el vector almacenará el valor $x_{\mu+h} = x_k$ con $k \geq \mu$.
Sólo falta ver que el elemento x_k aún está almacenado en el vector cuando se llega a la iteración $l \leq vk$. Si no fuese así, habría sido eliminado en cierta iteración $j < l$. Esto quiere decir que j verificaría $j \geq vk$ pero se tenía que $j < l \leq vk$ y, por lo tanto, x_k aún está en el vector al llegar a la iteración l. □

Una cuestión delicada es la elección de los parámetros v y t. Cuanto mayor sea t, mayor espacio de almacenamiento y más comparaciones son necesarios. En [28] también se muestra que $\sigma_{i+1} \approx R\sigma_i$ para algún R y que cuanto mayor es v mayor es R y, por tanto, más iteraciones son necesarias entre una sustitución del primer término y la siguiente. Por otro lado, para hallar una colisión ha de darse que $\sigma_i \geq \mu$ y $\sigma_i + \lambda \leq v\sigma_i$, lo que implica que $\lambda \leq (v-1)\sigma_i$. Entonces, cuanto mayor es v, más probabilidad hay de encontrar la primera colisión de la forma $x_{\sigma_i} = x_{\sigma_i+\lambda}$. Tras un análisis más exhaustivo de la influencia de ambos parámetros en el coste computacional del algoritmo y a la vista de resultados experimentales, Teske decide utilizar $v = 3$ y $t = 8$. Para estos valores, si la función de iteración se comporta como una función aleatoria el número de iteraciones está acotado por aproximadamente $1{,}42\sqrt{n}$ y, como para cada iteración hay una evaluación y 8 iteraciones, si el coste de las comparaciones es depreciable, se trata de un método mejor que el de Brent.

Ejemplo 5.11. Utilización del algoritmo de Teske para hallar una colisión en la secuencia del ejemplo 5.7:

$$5, 7, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, 12, 31, 26, 8, 22, 33, 15, 46, 39, 11, ...$$

i	x_i	$\exists j : x_i = x_{\sigma_j}$	$i \geq 3\sigma_1$	$(x_{\sigma_1}, ..., x_{\sigma_8})$	σ_1	$3\sigma_1$
1	5	–	–	(5,5,5,5,5,5,5,5)	1	3
2	7	no	no	(5,5,5,5,5,5,5,5)	1	3
3	8	no	sí	(5,5,5,5,5,5,5,8)	1	3
4	22	no	sí	(5,5,5,5,5,5,8,22)	1	3
5	33	no	sí	(5,5,5,5,5,8,22,33)	1	3
6	15	no	sí	(5,5,5,5,8,22,33,15)	1	3
7	46	no	sí	(5,5,5,8,22,33,15,46)	1	3
8	39	no	sí	(5,5,8,22,33,15,46,39)	1	3
9	11	no	sí	(5,8,22,33,15,46,39,11)	1	3
10	12	no	sí	(8,22,33,15,46,39,11,12)	3	9
11	31	no	sí	(22,33,15,46,39,11,12,31)	4	12
12	26	no	sí	(33,15,46,39,11,12,31,26)	5	15
13	8	no	no	(33,15,46,39,11,12,31,26)	5	15
14	22	no	no	(33,15,46,39,11,12,31,26)	5	15
15	33	sí	–	–	–	–

El algoritmo ha encontrado la colisión $x_5 = x_{15}$.

Las siguientes son variantes del método ρ basadas en funciones de iteración diferentes debidas a Teske [29].

- Función de Pollard generalizada: consiste en modificar ligeramente la definición de la función utilizada por Pollard. Tomamos $M = g^m$ y $K = a^k$, donde m y k son enteros elegidos al azar tales que $1 \leq m, k \leq n$, y una partición de G que verifique los mismos requisitos que en el método original. Definimos, entonces:

$$x_{i+1} = f_{PG}(x_i) = \begin{cases} x_i K & \text{si} \quad x_i \in S_1 \\ x_i^2 & \text{si} \quad x_i \in S_2 \\ x_i M & \text{si} \quad x_i \in S_3 \end{cases}$$

  La varianza de $\mu + \lambda$ en este caso es menor que en el algoritmo de Pollard original. Esto quiere decir que los valores de $\mu + \lambda$ obtenidos en distintos ejemplos se alejan menos de $E(\mu + \lambda)$ que con la función original. Es por eso que esta versión puede ser considerada como una versión controlada del método de Pollard.

- Teske's Adding-walk: se trata de una mejora basada en la utilización de r particiones en lugar de 3 que reduce el número de iteraciones. Se toman $2r$ enteros aleatorios m_i, k_i con $1 \leq m_i, k_i \leq n$ para $i = 1, ..., r$ y a partir de ellos se calculan los r multiplicadores $M_i = g^{m_i} a^{k_i}$ para $i = 1, ..., r$. Se define también una función hash:

$$v : G \longrightarrow \{1, 2, ..., r\}$$

  Entonces, la función de iteración se define de la siguiente manera:

$$x_{i+1} = f_{TA}(x_i) = x_i M_{v(x_i)}$$

  En este caso, es fácil ver que los exponentes se actualizan como sigue:

$$\alpha_{i+1} = \alpha_i + m_{v(x_i)}$$
$$\beta_{i+1} = \beta_i + k_{v(x_i)}$$

En grupos de orden primo, tomando una partición con más de 16 subconjuntos y utilizando el algoritmo de detección de ciclos de Teske, se tiene que $E(\mu+\lambda) \approx 1{,}45\sqrt{n}$ (Teske [29]). Se mejora, por tanto, la media del número de iteraciones obtenida para los subgrupos de orden primo de $\mathbb{F}_p^*$ con la función de iteración original.

- Teske's Mixed-walk: su estructura es una mezcla entre el "Teske's Adding-walk" y pasos consistentes en elevar al cuadrado. La función iteración se define de la siguiente manera:

$$x_{i+1} = f_{TM}(x_i) = \begin{cases} x_i M_{v(x_i)} & \text{si } v(x_i) \in \{1,2,...,r\} \\ x_i^2 & \text{en otro caso} \end{cases}$$

En [29], resultados experimentales muestran que tomando $r \geq 16$ y $q/r \approx 0{,}25$, donde q es el número de pasos consistentes en elevar al cuadrado, f_{TM} se comporta como una verdadera función aleatoria. Si tomamos $r = 16$ y $q = 4$, se tiene que $E(\mu+\lambda) \approx 1{,}3\sqrt{n}$, un valor cercano a la cota óptima $1{,}25\sqrt{n}$ aportada en el Teorema 5.5.

5.1.4. Algoritmo del canguro de Pollard

El algoritmo del canguro de Pollard [23], también conocido como algoritmo λ de Pollard, busca el logaritmo discreto, k, en un intervalo $[c,d] \subseteq \mathbb{Z}_n$. En caso de no conocer el intervalo al que pertenece el logaritmo discreto, se pueden establecer $c = 0$ y $d = n-1$, pero en tal caso es más eficiente el algortimo ρ.

En primer lugar, se elige un conjunto de enteros S y se define una función pseudoaleatoria $f : G \longrightarrow S$. A continuación, se escoge un entero N y se calcula la secuencia $\{x_i\}_{i=0}^{N}$ (el camino trazado por un *canguro domesticado* en N saltos) de la siguiente manera:

$$x_0 = g^d; \quad x_{i+1} = x_i g^{f(x_i)}$$

Seguidamente se calcula: $t = \sum_{i=0}^{N-1} f(x_i)$. Obsérvese que $x_N = x_0 g^t = g^{d+t}$. En este punto, el canguro domesticado tiende una trampa.

Se determina después la sucesión $\{y_i\}_{i\in\mathbb{N}}$ (el camino trazado por un *canguro salvaje*) como sigue:

$$y_0 = a; \quad y_{i+1} = y_i g^{f(y_i)}$$

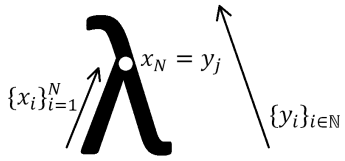
y una secuencia de enteros $\{t_j\}_{j\in\mathbb{N}}$: $t_j = \sum_{i=0}^{j-1} f(y_i)$. Nótese que $y_i = y_0 g^{t_i} = a g^{t_i}$.

Se dejan de calcular términos de las secuencias cuando se da una de las siguientes opciones:

1. $y_j = x_N$ para algún j, esto es, el canguro salvaje cae en la trampa tendida por el canguro domesticado. En este caso se tiene que $g^{d+t} = x_N = y_j = a g^{t_j}$ y, por tanto, $g^{d+t-t_j} = a = g^k$. En consecuencia, $k \equiv d + t - t_j \pmod{n}$.

2. $t_j > d - c + t$. Si esto sucede, no es posible hallar k puesto que $d + t - t_j < c$, lo que contradice la hipótesis de que $k \in [c,d]$. En este caso, hay que aplicar el método de nuevo, con S y/o f diferentes.

Este método es conocido como *método del canguro de Pollard*. Dicho nombre surge de la analogía establecida entre el hallazgo de un punto común de las sucesiones y la trampa tendida por un canguro domesticado a un canguro salvaje. También recibe el nombre de *método λ de Pollard* como alusión a la similitud entre la visualización del algoritmo y dicha letra griega. El trazo corto corresponde a la secuencia $\{x_i\}_{i=0}^{N}$ y el largo a $\{y_i\}_{i\in\mathbb{N}}$ que coincide con la primera secuencia en x_N y después continúa.



Ejemplo 5.12. Considérese el grupo $\mathbb{F}_{23}^*$. Aplicando el algoritmo del canguro de Pollard, es posible calcular el logaritmo discreto de 17 en base 7, sabiendo que pertenece al intervalo $[0, 7]$.

1. Se toma $f(x) = x^2$ como función pseudoaleatoria.
2. Tomando $N = 4$, se genera la secuencia $\{x_i\}_{i=0}^{4}$:
   - $x_0 \equiv 7^7 \equiv 5 \pmod{23}$
   - $x_1 \equiv 5 \cdot 7^{5^2} \equiv 13 \pmod{23}$
   - $x_2 \equiv 13 \cdot 7^{13^2} \equiv 21 \pmod{23}$
   - $x_3 \equiv 21 \cdot 7^{21^2} \equiv 9 \pmod{23}$
   - $x_4 \equiv 9 \cdot 7^{9^2} \equiv 11 \pmod{23}$

   Entonces, $d + t = 7 + 5^2 + 13^2 + 21^2 + 9^2 \equiv 19 \pmod{\varphi(23)}$
3. Se generan términos de la secuencia $\{y_i\}_{i\in\mathbb{N}}$ hasta hallar una colisión:
   - $y_0 \equiv 17 \pmod{23}$
   - $y_1 \equiv 17 \cdot 7^{17^2} \equiv 12 \pmod{23}$
   - $y_2 \equiv 12 \cdot 7^{12^2} \equiv 8 \pmod{23}$
   - $y_3 \equiv 8 \cdot 7^{8^2} \equiv 18 \pmod{23}$
   - $y_4 \equiv 18 \cdot 7^{18^2} \equiv 16 \pmod{23}$
   - $y_5 \equiv 16 \cdot 7^{16^2} \equiv 9 \pmod{23}$
   - $y_6 \equiv 9 \cdot 7^{9^2} \equiv 11 \pmod{23}$

   Por tanto, $k + t_6 = k + 17^2 + 12^2 + 8^2 + 18^2 + 16^2 + 9^2 \equiv 14 \pmod{\varphi(23)}$.

Como $x_4 = y_6$, se tiene que $k + t_6 \equiv d + t \pmod{\varphi(23)}$ y, en consecuencia, $k = 19 - 14 = 5$. Es decir, $7^5 \equiv 17 \pmod{23}$.

5.2. Algoritmo de Silver-Pohlig-Hellman

El algoritmo de Silver-Pohlig-Hellman [22] es un método utilizado para resolver el problema del logaritmo discreto en grupos cuyo orden, n, tiene divisores primos pequeños. Por lo tanto, no puede aplicarse en cualquier grupo, pero en aquellos en los que puede utilizarse resulta más eficiente que los métodos genéricos.

El primer paso de este algoritmo consiste en calcular, para cada primo p divisor de n, las p-ésimas raíces de la unidad:

$$r_{p,j} = g^{jn/p} \text{ para } j = 0, ..., p-1$$

Nuestro objetivo es hallar k, tal que $g^k = a$. Dada la descomposición en factores primos de n, $n = \prod_p p^\alpha$, para hallar k basta calcular $k \pmod{p^\alpha}$ para cada primo y aplicar después el teorema chino de los restos para obtener $k \pmod{n}$.

Veamos cómo calcular $k \pmod{p^\alpha}$ para cada p. Supongamos que tenemos la escritura de k en base p:

$$k \equiv k_0 + k_1 p + ... + k_{\alpha-1} p^{\alpha-1} \pmod{p^\alpha} \text{ con } 0 \leq k_i < p$$

Para determinar k debemos hallar $k_0, k_1, ..., k_{\alpha-1}$.

Para hallar k_0 calculamos $a^{n/p}$. Como $a^n \equiv 1 \pmod{n}$, se obtiene una raíz p-ésima de la unidad. Además,

$$a^{n/p} \equiv g^{kn/p} \equiv g^{k_0 n/p} g^{(k_1 + ... + k_{\alpha-1} p^{\alpha-2})n} \equiv g^{k_0 n/p} \equiv r_{p,k_0} \pmod{n}$$

y, por lo tanto, para determinar k_0 basta con comparar $a^{n/p}$ con las raíces p-ésimas de la unidad previamente calculadas y establecer $k_0 = j$ para j tal que $a^{n/p} = r_{p,j}$.

A continuación, para obtener k_1, reemplazamos a por $a_1 = a/g^{k_0}$ que tiene logaritmo discreto $k - k_0 \equiv k_1 p + ... + k_{\alpha-1} p^{\alpha-1} \pmod{p^\alpha}$. Como a_1 es una potencia de orden p, se tiene que $a_1^{n/p} \equiv 1 \pmod{n}$ y

$$a_1^{n/p^2} \equiv g^{(k-k_0)n/p^2} \equiv g^{(k_1 + ... + k_{\alpha-1} p^{\alpha-2})n/p} \equiv g^{k_1 n/p} \equiv r_{p,k_1} \pmod{n}$$

Luego, para hallar k_1 debemos comparar a_1^{n/p^2} con las raíces p-ésimas de la unidad y tomar $k_1 = j$ para j tal que $a_1^{n/p^2} = r_{p,j}$.

Se procede de la misma manera para hallar $k_2, k_3, ..., k_{\alpha-1}$. En general, para obtener k_i se toma

$$a_i = a/g^{k_0 + k_1 p + ... + k_{i-1} p^{i-1}}$$

que tiene logaritmo discreto $k - (k_0 + k_1 p + ... + k_{i-1} p^{i-1}) \equiv k_i p^i + ... + k_{\alpha-1} p^{\alpha-1} \pmod{p^\alpha}$. Como a_i es una potencia de orden p^i, se tiene que

$$a_i^{n/p^i} \equiv g^{(k_i p^i + ... + k_{\alpha-1} p^{\alpha-1})n/p^i} \equiv 1 \pmod{n}$$

y

$$a_i^{n/p^{i+1}} = g^{(k_i + k_{i+1} p + ... + k_{\alpha-1} p^{\alpha-i})n/p} = g^{k_i n/p} = r_{p,k_i} \pmod{n}$$

Entonces, tomamos k_i igual al valor j para el que $a_i^{n/p^{i+1}} = r_{p,j}$. Una vez determinados todos los coeficientes $k_0, k_1, ..., k_{\alpha-1}$, obtenemos $k \pmod{p^\alpha}$.

Hemos de repetir estos cálculos para cada factor primo de n y una vez dispongamos de $k \pmod{p^\alpha}$ para todo p, sólo resta aplicar el teorema chino de los restos para hallar $k \pmod{n}$.

Este algoritmo funciona bien cuando todos los factores primos de n son pequeños. Obviamente, si entre ellos hubiese algún primo grande (característica que depende de la capacidad de cálculo), la determinación de la tabla con las raíces p-ésimas de la unidad y las comparaciones de $y_i^{n/p^{i+1}}$ con dicha tabla llevarían mucho tiempo.

Ejemplo 5.13. Sea $q = 37$. El elemento $g = 2$ es un generador de $\mathbb{F}_q^*$. Además, $n = q - 1 = 2^2 3^2$. Se quiere calcular el logaritmo discreto en base 2 de 28 utilizando el algoritmo de Silver-Pohlig-Hellman. El primer paso es la precomputación de las raíces p-ésimas de la unidad para $p = 2$ y $p = 3$ tal y como se ha explicado.

p \ j	0	1	2
2	1	-1	-
3	1	26	10

Esto es, $r_{2,0} = 1$, $r_{2,1} = -1$, $r_{3,0} = 1$, $r_{3,1} = 26$ y $r_{3,2} = 10$.

Se pasa entonces a calcular el logaritmo discreto de $a = 28$ en bases 2^2 y 3^2.
Para $p = 2$, $k \equiv k_0 + k_1 2 \pmod{2^2}$.
$28^{36/2} \equiv 1 \equiv r_{2,0} \pmod{37}$. Entonces $k_0 = 0$.
$(28/2^0)^{36/4} \equiv -1 \equiv r_{2,1} \pmod{37}$. Por tanto, $k_1 = 1$.
Así, $k \equiv 2 \pmod{2^2}$.

Para $p = 3$, $k \equiv k_0 + k_1 3 \pmod{3^2}$.
$28^{36/3} \equiv 26 \equiv r_{3,1} \pmod{37}$. En consecuencia, $k_0 = 1$.
$(28/2^1)^{36/9} \equiv 10 \equiv r_{2,1} \pmod{37}$. Por consiguiente, $k_1 = 2$.
Entonces, $k \equiv 1 + 2 \cdot 3 \equiv 7 \pmod{3^2}$.

Se tiene el siguiente sistema de congruencias:

$$\begin{cases} k \equiv 2 \pmod 4 \\ k \equiv 7 \pmod 9 \end{cases}$$

Por el teorema chino de los restos, $k \equiv 34 \pmod{37}$ y, por tanto, el logaritmo discreto de 28 en base 2 es 34.

5.3. Index-Calculus

En las secciones anteriores se han visto algunos algoritmos genéricos y el algortimo de Silver-Pohlig-Hellman. Supongamos ahora que queremos resolver el problema del logaritmo discreto en el grupo $\mathbb{F}_{2^{127}}^*$. En tal caso, el orden del grupo es $2^{127} - 1$, un primo de Mersenne grande, así que no podemos aplicar Silver-Pohlig-Hellman y los algoritmos genéricos no

resultan eficientes por el gran tamaño del grupo. En esta sección se muestra el algoritmo denominado Index-Calculus, un método que sólo puede utilizarse en ciertos grupos entre los que se encuentran los grupos multiplicativos de los cuerpos finitos. La pérdida de generalidad de este método se compensa con una mayor eficiencia que surge de sacar provecho de las propiedades particulares del grupo.

El primero en introducir la idea básica del Index-Calculus fue Kraitchik en 1922. Merkle la redescubrió en 1977 cuando el PLD comenzó a adquirir importancia pero fue Adleman quien optimizó el algoritmo y lo presentó tal y como hoy lo conocemos en 1979 [1].

Para estudiar este método, se introducen primero los pasos generales del algoritmo y, a continuación, se estudiará su aplicación en los grupos multiplicativos de los cuerpos $\mathbb{F}_{p^m}$. Cabe mencionar que, en esta sección, $ind_g(x)$ denota el logaritmo discreto en base g de x.

La idea de este algoritmo consiste en explotar la representación de los elementos del grupo como producto de elementos de un subconjunto pequeño, la *base de factores*. Dado G de orden n, tomamos $B = \{p_1, p_2, ..., p_r\} \subseteq G$. El algoritmo consta, básicamente, de tres etapas:

1. Buscar identidades del tipo:

$$\prod_{i=1}^{r} p_i^{\alpha_i} = g^t,\ t \in \mathbb{Z}$$

   de donde se deduce que:

$$\sum_{i=1}^{r} \alpha_i ind_g(p_i) \equiv t \pmod{n} \tag{5.2}$$

2. Una vez halladas suficientes identidades del tipo (5.2), esto es, r linealmente independientes,tenemos un sistema compatible determinado cuyas incógnitas son los índices de los elementos de la base de factores. Resolviendo dicho sistema, se determina el logaritmo discreto de los elementos p_i.

   Estas dos etapas constituyen una *precomputación* que no depende del elemento del que queremos calcular el logaritmo discreto. Sólo debemos llevarlas a cabo una vez y podemos utilizar el resultado para calcular varios logaritmos discretos en base g.

3. Para obtener el logaritmo discreto de a se busca una relación de la forma:

$$\prod_{i=1}^{r} p_i^{\beta_i} = ag^e,\ e \in \mathbb{Z}$$

   que equivale a:

$$\prod_{i=1}^{r} g^{ind_g(p_i)\beta_i} = g^{ind_g(a)} g^e,\ e \in \mathbb{Z}$$

   De donde se deduce:

$$ind_g(a) \equiv \sum_{i=1}^{r} \beta_i ind_g(p_i) - e \pmod{n}$$

   Resolviendo esta última equivalencia se obtiene $k = ind_g(a)$.

Como ya hemos comentado, el Index-Calculus puede aplicarse al grupo multiplicativo de un cuerpo finito. En lo que sigue vamos a centrarnos en aplicar este algoritmo a $\mathbb{F}_q^*$ donde $q = p^m$ es una potencia de primo.

En el caso en que $m = 1$, esto es, cuando trabajamos en $\mathbb{F}_p$ las factorizaciones de las etapas 1 y 3 se pueden entender como factorizaciones de enteros menores que p. Como se dispone de métodos eficaces para factorizar enteros como producto de primos conocidos, se toma como base de factores un conjunto de números primos. En tal caso, el método no entraña mayor dificultad que calcular potencias de g y escribirlas como producto de los factores que conforman la base. Veamos un ejemplo ilustrativo.

Ejemplo 5.14. Supóngase que se quiere resolver $2^k \equiv 13 \pmod{2027}$. Lo primero que se debe hacer es escoger una base de factores, por ejemplo $B = \{2, 3, 5, 7, 11\}$. A continuación se calculan potencias aleatorias de $g \equiv 2 \pmod{2027}$ y se toman aquellas que pueden escribirse como producto de elementos de B:

$$\begin{aligned} 2^{293} &\equiv 63 \equiv 3^2 \cdot 7 \pmod{2027} \\ 2^{983} &\equiv 385 \equiv 5 \cdot 7 \cdot 11 \pmod{2027} \\ 2^{1318} &\equiv 1408 \equiv 2^7 \cdot 11 \pmod{2027} \\ 2^{1593} &\equiv 33 \equiv 3 \cdot 11 \pmod{2027} \\ 2^{1918} &\equiv 1600 \equiv 2^6 \cdot 5^2 \pmod{2027} \end{aligned}$$

Se tienen, por tanto, las siguientes equivalencias:

$$\begin{aligned} 293 &\equiv 2ind_2 3 + ind_2 7 \pmod{2026} \\ 983 &\equiv ind_2 5 + ind_2 7 + ind_2 11 \pmod{2026} \\ 1318 &\equiv 7ind_2 2 + ind_2 11 \pmod{2026} \\ 1593 &\equiv ind_2 3 + ind_2 11 \pmod{2026} \\ 1918 &\equiv 6ind_2 2 + 2ind_2 5 \pmod{2026} \end{aligned}$$

Como son 5 equivalencias linealmente independientes no es necesario buscar más y se puede pasar a resolver el sistema para obtener los índices de los elementos de la base. Dado que $2026 = 2 \cdot 1013$, se resuelve el sistema módulo 2 y módulo 1013 y se aplica el teorema chino de los restos. Se tiene entonces el siguiente sistema módulo 2:

$$\begin{aligned} ind_2 7 &\equiv 1 \pmod{2} \\ ind_2 5 + ind_2 7 + ind_2 11 &\equiv 1 \pmod{2} \\ ind_2 2 + ind_2 11 &\equiv 0 \pmod{2} \\ ind_2 3 + ind_2 11 &\equiv 1 \pmod{2} \end{aligned} \tag{5.3}$$

Nótese que $ind_2 2 = 1$ siempre. Entonces, $ind_2 2 \equiv ind_2 5 \equiv ind_2 7 \equiv ind_2 11 \equiv 1 \pmod{2}$ y

$ind_2 3 \equiv 0 \pmod 2$. Por otra parte se tiene el siguiente sistema módulo 1013:

$$\begin{aligned} 2ind_2 3 + ind_2 7 &\equiv 293 \pmod{1013} \\ ind_2 5 + ind_2 7 + ind_2 11 &\equiv 983 \pmod{1013} \\ ind_2 11 &\equiv 298 \pmod{1013} \\ ind_2 3 + ind_2 11 &\equiv 580 \pmod{1013} \\ 2ind_2 5 &\equiv 899 \pmod{1013} \end{aligned}$$

Entonces:

$$\begin{aligned} ind_2 2 \equiv 1 \pmod{1013} \quad &\text{y} \quad ind_2 2 \equiv 1 \pmod 2 \\ ind_2 3 \equiv 282 \pmod{1013} \quad &\text{y} \quad ind_2 3 \equiv 0 \pmod 2 \\ ind_2 5 \equiv 956 \pmod{1013} \quad &\text{y} \quad ind_2 5 \equiv 1 \pmod 2 \\ ind_2 7 \equiv 742 \pmod{1013} \quad &\text{y} \quad ind_2 7 \equiv 1 \pmod 2 \\ ind_2 11 \equiv 298 \pmod{1013} \quad &\text{y} \quad ind_2 11 \equiv 1 \pmod 2 \end{aligned}$$

Y, empleando el teorema chino de los restos para cada índice, finalmente se obtiene que:

$$\begin{aligned} ind_2 2 &\equiv 1 \pmod{2026} \\ ind_2 3 &\equiv 282 \pmod{2026} \\ ind_2 5 &\equiv 1969 \pmod{2026} \\ ind_2 7 &\equiv 1755 \pmod{2026} \\ ind_2 11 &\equiv 1311 \pmod{2026} \end{aligned}$$

Como ya son conocidos los índices de los elementos de la base, se puede pasar a la última etapa del algoritmo. Tomando $e = 1397$,

$$13 \cdot 2^{1397} \equiv 110 \equiv 2 \cdot 5 \cdot 11 \pmod{2026}$$

Se tiene entonces que:

$$ind_2 13 + 1397 ind_2 2 \equiv ind_2 2 + ind_2 5 + ind_2 11 \pmod{2026}$$

Y finalmente:

$$ind_2 13 \equiv -1397 + 1 + 1969 + 1311 \equiv 1884 \pmod{2026}$$

Supóngase ahora que $m \neq 1$, esto es, $q = p^m$ es una potencia de un primo pequeño para el que es posible resolver el logaritmo discreto de manera eficaz. Supóngase también que g es un generador de $\mathbb{F}_q^*$. Sea $f(x) \in \mathbb{F}_p[x]$ un polinomio irreducible de grado m. Entonces $\mathbb{F}_q$ es isomorfo a $\mathbb{F}_p[x]/f(x)$ y todo elemento $a \in \mathbb{F}_q$ puede escribirse como un polinomio $a(x) \in \mathbb{F}_p[x]$ de grado menor que m, en particular, $g = g(x)$ y $a = a(x)$. Las constantes son los elementos de $\mathbb{F}_p \subset \mathbb{F}_q$. Además, $g' = g^{(q-1)/(p-1)}$ genera $\mathbb{F}_p^*$. Para verlo ha de probarse que $g' \in \mathbb{F}_p^*$ y que su orden es $p - 1$:

- Para ver que g' es un elemento de $\mathbb{F}_p$ basta ver que $\sigma(g') = g'$ donde σ es el automorfismo de Frobenius de $\mathbb{F}_q$ que eleva cada elemento a la potencia p. Se debe probar entonces que $(g')^p = g'$.

$$(g')^p = g^{p(q-1)/(p-1)} = g^{p(p^m-1)/(p-1)} = g^{p(p-1)(p^{m-1}+p^{m-2}+p^{m-3}+\ldots+1)/(p-1)} = g^q g^{p^{m-1}+p^{m-2}+\ldots+p} = g \cdot g^{p^{m-1}+p^{m-2}+\ldots+p} = g^{p^{m-1}+p^{m-2}+\ldots+p+1} = g^{(q-1)/(p-1)} = g'$$

Obviamente $g' \neq 0$ ya que $g \neq 0$ y, por lo tanto, $g' \in \mathbb{F}_p^*$.

- Véase que el orden de g' es $p-1$. Obviamente,

$$(g')^{p-1} = g^{(p-1)(q-1)/(p-1)} = g^{q-1} = 1$$

puesto que g tiene orden $q-1$. Por lo tanto, el orden de g' es divisor de $p-1$. Supóngase que el orden de g', d, es un divisor propio de $p-1$. Entonces:

$$1 = (g')^d = g^{(q-1)d/(p-1)} = g^{(q-1)/\frac{(p-1)}{d}}$$

Pero g tiene orden exactamente $q-1$ y, por lo tanto, $g^{(q-1)/\frac{(p-1)}{d}}$ no puede ser 1. Así, queda probado que g' tiene orden $p-1$.

En consecuencia, para calcular el logaritmo discreto de las constantes de $\mathbb{F}_p^*[x]$, esto es, los elementos de $\mathbb{F}_p^*$, en base g, basta con calcular el logaritmo discreto de las mismas en base g' en $\mathbb{F}_p^*$. Nótese que esta tarea es sencilla puesto que p es pequeño y puede construirse fácilmente la tabla de logaritmos discretos o emplearse alguno de los métodos estudiados con anterioridad.

Una vez hechas estas consideraciones, se van a analizar las etapas del algoritmo. En primer lugar, se toma B como el conjunto de todos los polinomios mónicos irreducibles sobre $\mathbb{F}_p$ de grado menor o igual que k, donde $k \leq m$. Se tiene que $p = \mathbb{F}_p \leq \#B \leq \mathbb{F}_q = p^m$. Ha de comentarse que el tamaño de B crece rápidamente a medida que m crece y teniendo en cuenta que debemos encontrar al menos $\#B$ congruencias para resolver un sistema con el mismo número de incógnitas, sería conveniente que el cardinal de B no fuese muy grande. Por otro lado, si $\#B$ es muy pequeño, resultaría muy costoso encontrar simplemente una congruencia de las necesarias. Por lo tanto, m debe tener un tamaño moderado.

Se pasa entonces a la primera etapa del algoritmo. Para ello, se toma un entero t tal que $1 \leq t \leq q-2$ y se calcula:

$$c(x) \equiv g(x)^t \pmod{f(x)}$$

Se quiere ver que $c(x)$ puede escribirse:

$$c(x) \equiv c_0 \prod_{p(x)\in B} p(x)^{\alpha_p} \pmod{f(x)}$$

Si, efectivamente, se ha encontrado una igualdad de ese tipo, tomando logaritmos discretos a ambos lados de la misma se obtiene que:

$$ind(c(x)) - ind(c_0) \equiv \sum_{p(x)\in B} \alpha_p ind(p(x)) \pmod{q-1}$$

Nótese que $ind(c(x)) = t$ y que ya se había calculado $ind(c_0)$ ya que c_0 es una constante en $\mathbb{F}_p^*[x]$

Como ya se ha comentado, una vez obtenidas $\#B$ equivalencias de este tipo, se está en disposición de ejecutar el paso 2 del algoritmo para determinar el índice de todos los elementos de $\#B$. Tras ello, se pasa a la tercera y última etapa del Index-Calculus. En dicha etapa, ha de encontrarse, de la misma manera que el paso 1, una relación de la forma

$$a(x)g(x)^t \equiv a_0 \prod_{p(x)\in B} p(x)^{\alpha_p} \pmod{f(x)}$$

Como ya se explicó, se tiene así que

$$ind(a(x)) + t \equiv ind(a_0) + \sum_{p(x)\in B} \alpha_p ind(p(x)) \pmod{q-1}$$

y, por el isomorfismo,

$$ind(a) \equiv ind(a_0) + \sum_{p(x)\in B} \alpha_p ind(p(x)) - t \pmod{q-1}$$

Claramente, la complejidad del algoritmo reside en la factorización que se lleva a cabo en los pasos 1 y 3. De ahí la importancia de disponer de métodos eficaces de factorización de polinomios. En el apartado siguiente, se estudian algunos de esos métodos.

5.3.1. Factorización de polinomios en cuerpos finitos

Para empezar, conviene hacer algunas consideraciones:

- Sean $f(x)$ y $g(x)$ polinomios tales que $g(x)$ divide a $f(x)$. Se llamará *exponente* de $g(x)$ al mayor entero e tal que $g^e(x)$ divide a $f(x)$. Si e es mayor que 1, se dice que $g(x)$ es un *factor repetido* de $f(x)$.
- Todo polinomio mónico de un cuerpo finito puede ser factorizado de forma única como producto de polinomios irreducibles. Se trata de la *factorización canónica* de un polinomio y es la factorización que se quiere calcular. Es decir, si $f(x)$ es un polinomio de $\mathbb{F}_q[x]$ de grado n, se van a buscar polinomios irreducibles p_i y enteros e_i tales que:

$$f(x) = \prod_{i=0}^{m} p_i^{e_i}(x)$$

- Si en la expresión anterior los polinomios p_i no son irreducibles, se dice que $\prod_{i=0}^{m} p_i(x)^{e_i}$ es una *factorización parcial* de $f(x)$ (suponiendo no sea trivial).

En lo que sigue, salvo que se especifique otra cosa, se considera que $f(x)$ es un polinomio mónico de $\mathbb{F}_q[x]$ de grado n.

Se introducirán dos tipos de factorizaciones parciales y algoritmos para calcularlas. Asimismo, se presentarán dos algoritmos para factorizar polinomios que disponen de ciertas propiedades. Combinando ambas cosas será posible factorizar cualquier polinomio sobre cualquier cuerpo finito.

Factorización libre de cuadrados

En primer lugar, es necesario introducir la siguiente definición:

Definición 5.15. *Se dice que un polinomio $f(x)$ es libre de cuadrados si no existe ningún polinomio $g(x)$ tal que $g^2(x)$ divida a $f(x)$. Si $f(x)$ es un polinomio no libre de cuadrados, entonces existe $g(x)$ tal que $g^2(x)$ divide a $f(x)$ y $h(x) = f(x)/g^2(x)$ es libre de cuadrados. El polinomio $h(x)$ se denomina parte libre de cuadrados de $f(x)$.*

Dado $f(x) = \sum_{i=0}^{n} a_i x^i$, se define la derivada de $f(x)$, $f'(x)$, como $f'(x) = \sum_{i=1}^{n} i a_i x^{i-1}$.

Es fácil comprobar que la derivada de un polinomio satisface:

- $(f(x)^e)' = e f(x)^{e-1} f'(x)$
- $(fg)'(x) = f'(x)g(x) + f(x)g'(x)$

Si un polinomio $f(x)$ no es libre de cuadrados, entonces $f(x)$ y su derivada, $f'(x)$, tienen factores comunes, tal y como se pone de manifiesto en el siguiente lema:

Lema 5.16. *Sea $f(x)$ un polinomio no libre de cuadrados. Sea $g(x)$ un factor repetido de $f(x)$ con exponente e. Entonces $g(x)$ es un factor de $f'(x)$ con exponente $e' = e - 1$.*

Demostración. Sea $h(x) = f(x)/g(x)^e$. Entonces $f(x) = h(x)g(x)^e$. Utilizando las propiedades de la derivada:

$$f'(x) = h'(x)g(x)^e + h(x)eg(x)^{e-1} = g(x)^{e-1}(h'(x)g(x) + eh(x)g(x))$$

Nótese que $e - 1 > 0$ ya que $e > 1$ por ser $g(x)$ factor repetido. □

Corolario 5.17. *Para todo polinomio $f(x)$, el polinomio $f(x)/mcd(f(x), f'(x))$ es libre de cuadrados.*

Demostración. Sean $q_0(x), q_1(x), ..., q_k(x)$ polinomios irreducibles tales que existen $e_0, e_1, ..., e_k$ todos mayores que 1 de manera que:

$$h_1(x) = \frac{f(x)}{\prod_{i=0}^{k} q_i^{e_i}(x)}$$

sea libre de cuadrados. Entonces, por el lema 5.16, cada $q_i(x)$ es factor de $f'(x)$ con exponente $e_i - 1$, es decir:

$$h_2(x) = \frac{f'(x)}{\prod_{i=0}^{k} q_i^{e_i-1}(x)}$$

es libre de cuadrados. Entonces,

$$mcd(f(x), f'(x)) = \big(\prod_{i=0}^{k} q_i^{e_i-1}(x)\big) mcd(h_1(x), h_1(x))$$

y, por tanto,

$$\frac{f(x)}{mcd(f(x), f'(x))} = \frac{\big(\prod_{i=0}^{k} q_i(x)\big) h_1(x)}{mcd(h_1(x), h_2(x))}$$

Como cada $q_i(x)$ es irreducible, el producto de los $q_i(x)$ es libre de cuadrados. Por otro lado, como h_1 no tiene factores repetidos, $h_1(x)/mcd(h_1(x), h_2(x))$ también es libre de cuadrados. Además, por la manera en que se ha definido $h_1(x)$, $\prod_{i=0}^{k} q_i(x)$ y $h_1(x)/mcd(h_1(x), h_2(x))$ son coprimos y en consecuencia su producto tampoco tiene factores repetidos. □

Observación 5.18. *Sea $f(x) \in \mathbb{F}_q[x]$ con $q = p^n$. Entonces, $char(\mathbb{F}_q) = p$. Si $f(x)$ es tal que existe $g(x)$ con $f(x) = g(x)^p$, se tiene que $f'(x) = pg(x)^{p-1} = 0$. Por tanto, el resultado del lema anterior resulta trivial ya que $mcd(f(x), f'(x)) = f(x)$ y, en consecuencia, $f(x)/mcd(f(x), f'(x)) = 1$ que, obviamente, no tiene factores repetidos. En este caso, se podría encontrar un factor no trivial de $f(x)$ calculando* $\gcd(g(x), g'(x))$.

Los resultados presentados permiten calcular una factorización parcial cuyos factores son libres de cuadrados. El siguiente algoritmo realiza este proceso:

Algoritmo 5.19. LIBRE DE CUADRADOS

Input: $f(x) \in \mathbb{F}_q(x)$
Output: $f_1(x), f_2(x), ..., f_k(x)$ con $f(x) = \prod_{i=0}^{k} f_i(x)$

1. $i := 1$
2. **Mientras** $f(x) \neq 1$ **hacer**:
   a) **Si** $mcd(f(x), f'(x)) \neq f(x)$ hacer:
      1) $f_i(x) := f(x)/mcd(f(x), f'(x))$
      2) $f(x) := f(x)/f_i(x)$
      3) $i := i + 1$
   b) **si no, hacer**:
      1) Tomar k como la mayor potencia de p que divide a todas las potencias de x en $f(x)$, esto es, $k = \text{máx}\{j : x^{p^j} \mid x^h, \ \forall\ x^h \ monomio\ de\ f(x)\}$
      2) $g(x) := f(x)^{1/p^k}$
      3) $f_i(x) := g(x)/mcd(g(x), g'(x))$
      4) $f(x) := f(x)/f_i(x)$
      5) $i := i + 1$
3. **Devolver**: $\{f_1(x), ..., f_{i-1}(x)\}$

Si $mcd(f(x), f'(x)) \neq f(x)$, entonces el polinomio $f(x)$ no es potencia de orden p de ningún otro polinomio y, por el Corolario 5.17, cada $f_i(x)$ obtenido en este caso no tiene factores repetidos.

Si $mcd(f(x), f'(x)) = f(x)$, a partir de la observación 5.18, se calcula un polinomio $g(x)$ que no es potencia de orden p de ningún otro. A continuación se halla un factor libre de cuadrados de $f(x)$ tomando uno de los factores con esta propiedad de $g(x)$.

Por tanto, en cada iteración i se obtiene un factor $f_i(x)$ del polinomio original. Además, el valor de $f(x)$ se actualiza por el cociente de $f(x)/f_i(x)$. Este proceso se realiza hasta que el polinomio almacenado en $f(x)$ sea irreducible o producto de irreducibles. En cualquiera de los dos casos, se puede comprobar que el valor de $f(x)$ tras actualizarse es exactamente 1 y, por consiguiente, el algoritmo termina.

Además, cuando el proceso acaba, el valor de $f(x)$ está actualizado por $f_O(x)/\prod_{i=1}^{i-1} f_i(x)$, donde $f_O(x)$ es el polinomio original. Como el criterio de parada es $f(x) = 1$, es obvio que el producto de los factores obtenidos es, efectivamente, el polinomio original.

Nótese que los polinomios $f_i(x)$ obtenidos no tienen porqué ser irreducibles ni diferentes. Sólo se asegura que son libres de cuadrados.

Ejemplo 5.20. Se va a calcular una factorización libre de cuadrados de $f(x) = x^4 + 2x^3 - 4x^2 - 2x + 3$ en $\mathbb{F}_5[x]$.

1. i=1: $f'(x) = 4x^3 + 6x^2 - 8x - 2$, $mcd(f(x), f'(x)) = x - 1 \neq f(x)$

$$f_1(x) = \frac{f(x)}{mcd(f(x), f'(x))} = \frac{f(x)}{x-1} = x^3 + 3x^2 - x - 3$$

$$f(x) = \frac{f(x)}{f_1(x)} = x - 1$$

2. i=2: $f'(x) = 1$ y $mcd(f(x), f'(x)) = 1 \neq f(x)$

$$f_2(x) = \frac{f(x)}{1} = x - 1$$

$$f(x) = \frac{f(x)}{f_2(x)} = 1$$

y el algoritmo termina.

Se obtiene la siguiente factorización libre de cuadrados:

$$f(x) = (x^3 - 3x^2 - x + 3)(x - 1)$$

Esta factorización no coincide con la factorización canónica de $f(x)$:

$$f(x) = (x-1)^2(x+1)(x+3)$$

Factorización distinto grado

Se introduce el siguiente lema, que será la base del algoritmo presentado para obtener la factorización parcial deseada:

Lema 5.21 (Lema del producto de Gauss)**.** *Dado $n \in \mathbb{N}$, el producto de todos los polinomios mónicos e irreducibles de $\mathbb{F}_q[x]$ de grado d tal que $d \mid n$ es el polinomio $x^{q^n} - x$.*

El algoritmo descrito a continuación factoriza un polinomio libre de cuadrados $f(x)$ como producto de polinomios cuyos factores irreducibles son todos del mismo grado, esto es, encuentra polinomios $f_1(x), f_2(x), ..., f_k(x)$ tales que:

$$f(x) = \prod_{i=1}^{k} f_i(x)$$

donde cada $f_i(x)$ verifica:

$$f_i(x) = \prod_{j=1}^{m_i} q_j^{e_j^i}(x)$$

siendo todos todos los $q_i(x)$ polinomios irreducibles de grado i.

Algoritmo 5.22. DISTINTO GRADO

Input $f(x) \in \mathbb{F}_q[x]$ libre de cuadrados de grado n
Output $f_1(x), ..., f_k(x)$ tales que $f(x) = \prod_{i=1}^{k} f_i(x)$ y todos los factores irreducibles de $f_i(x)$ tienen grado i

1. $i := 1$
2. **Mientras** $deg(f(x)) > 0$ **hacer**:
   a) $f_i(x) := mcd(f(x), x^{q^i} - x)$
   b) $f(x) := f(x)/f_i(x)$
   c) $i := i + 1$
3. **Devolver**: $\{f_1(x), f_2(x), ..., f_{i-1}(x)\}$

En cada iteración i, $\gcd(f(x), x^{q^i} - x)$ es el producto de todos los factores irreducibles de $f(x)$ de grado divisor de i por el Lema del producto de Gauss y porque $f(x)$ es libre de cuadrados. Como todos los factores de grado menor que i son eliminados de $f(x)$ en los pasos anteriores, el $mcd(f(x), x^{q^i} - x)$ es el producto de todos los divisores de $f(x)$ de grado precisamente i. Si el divisor irreducible de mayor grado de $f(x)$ tiene grado k, entonces en la iteración k, $f(x)/f_k(x) = 1$. Por tanto, en este paso $deg(f(x)) = 0$ y el algoritmo termina.

Nótese que si todos los factores irreducibles de $f(x)$ tienen distinto grado, entonces la factorización obtenida por el algoritmo anterior es la factorización canónica de $f(x)$.

Ejemplo 5.23. *Sea $f(x) = x^5+2x^4-x-2$ un polinomio libre de cuadrados de $\mathbb{F}_3[x]$. Aplicando el algoritmo que se acaba de describir se obtiene una factorización en polinomios cuyos factores son todos del mismo grado.*

1. $i = 1$
2. *Como $deg(f(x)) = 3 > 0$,*
   a) $f_1(x) = mcd(f(x), x^3 - x) = x^3 + 2x^2 - x - 2$
   b) $f(x) = f(x)/f_1(x) = x^2 + 1$
   c) $i = 2$
3. *$deg(f(x)) = 1 > 0$, entonces:*
   a) $f_2(x) = mcd(f(x), x^6 - x) = x^2 + 1$
   b) $f(x) = f(x)/f_2(x) = 1$
   c) $i = 3$
4. *$deg(f(x)) = 0$ y, en consecuencia, el algoritmo termina.*

Por tanto, se obtiene la factorización $f(x) = (x^2 + 1)(x^3 + 2x^2 - x - 2)$. Nótese que dicha factorización no es la canónica.

Algoritmo de Berlekamp

El algoritmo de Berlekamp es un algoritmo para factorizar polinomios libres de cuadrados. Para obtener la factorización canónica de cualquier polinomio basta con utilizar el algoritmo 5.19 primero y después aplicar Berlekamp a cada factor libre de cuadrados obtenido.

Se presentan a continuación algunos resultados necesarios para comprender el funcionamiento del algoritmo.

Lema 5.24. *Todo polinomio $g(x) \in \mathbb{F}_q[x]$ satisface:*

$$g^q(x) - g(x) = \prod_{s \in \mathbb{F}_q} (g(x) - s)$$

Demostración. Por el lema del producto de Gauss (lema 5.21), $x^q - x = \prod_{s \in \mathbb{F}_q}(x - s)$. Tomando $x = g(x)$, se obtiene el resultado. □

Teorema 5.25. *Sean $f(x)$ y $g(x)$ polinomios mónicos de $\mathbb{F}_q[x]$ tales que $g(x)^q \equiv g(x) \pmod{f(x)}$. Entonces:*

$$f(x) = \prod_{s \in \mathbb{F}_q} mcd(f(x), g(x) - s)$$

Demostración. Por hipótesis, $g^q(x) \equiv g(x) \pmod{f(x)}$, esto es, $f(x) \mid g^q(x) - g(x)$ y por el lema 5.24, $f(x) \mid \prod_{s \in \mathbb{F}_q}(g(x) - s)$. Por tanto,

$$f(x) \mid mcd(f(x), \prod_{s \in \mathbb{F}_q} (g(x) - s)) = \prod_{s \in \mathbb{F}_q} mcd(f(x), (g(x) - s))$$

Por otro lado, cada término $mcd(f(x), g(x) - s)$ del producto divide a $f(x)$. Además, $g(x) - s_i$ y $g(x) - s_j$ son coprimos si $i \neq j$ y, por tanto, también lo son $mcd(f(x), g(x) - s_i)$ y $mcd(f(x), g(x) - s_j)$. Entonces,

$$\prod_{s \in \mathbb{F}_q} mcd(f(x), (g(x) - s)) \mid f(x)$$

En consecuencia, ambos polinomios son iguales. □

Obsérvese que si $g(x)$ tiene grado mayor o igual que uno, este teorema proporciona una factorización no trivial de $f(x)$. El algoritmo de Berlekamp es básicamente un algoritmo para hallar un polinomio $g(x)$ que satisfaga dicha condición.

El conjunto de todos los factores irreducibles de $f(x)$ puede ser considerado como un subconjunto de $R = \mathbb{F}_q[x]/(f(x))$. Si consideramos el anillo cociente como un álgebra sobre F_q, el conjunto de polinomios $g(x) \in \mathbb{F}_q[x]$ que satisfacen $g^q(x) \equiv g(x) \pmod{f(x)}$ forman una subálgebra de R. Dicha subálgebra se conoce como la *subálgebra de Belekamp de R* y se denota por $\mathcal{B}$. La estrategia del algoritmo de Berlekamp es calcular una base de $\mathcal{B}$ para generar polinomios de dicha subálgebra que por el teorema 5.25 proporcionarán fácilmente una factorización no trivial de $f(x)$. Para calcular dicha base se necesita la siguiente:

Definición 5.26. *Sea $f(x)$ un polinomio de grado n. Se definen n polinomios*

$$Q_i(x) = \sum_{j=0}^{n-1} q_{i,j} x^j \text{ para } i = 0, 1, ..., n-1$$

donde los coeficientes $q_{i,j}$ se toman de manera que $x^{iq} \equiv q_{i,n-1}x^{n-1} + ... + q_{i,0} = Q_i(x) \pmod{f(x)}$. La matriz de Berlekamp de $f(x)$ es la matriz cuadrada de dimensión n:

$$\mathcal{Q} = \begin{bmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,n-1} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ q_{n-1,0} & q_{n-1,1} & \cdots & q_{n-1,n-1} \end{bmatrix}$$

Los resultados presentados a continuación establecen la relación existente entre los polinomios de la subálgebra de Berlekamp y algunos vectores propios del endomorfismo $\mathcal{Q}$.

Lema 5.27. *Sea $g(x) = \sum_{i=0}^{n-1} g_i x^i$ un polinomio de $\mathbb{F}_q[x]$ de grado menor que n. Sea $\widetilde{g}$ el vector de coeficientes $(g_0, g_1, ..., g_{n-1})$. Se tiene que:*

$$g^q(x) \equiv \widetilde{g}\mathcal{Q} \pmod{f(x)}$$

Demostración.

$$g^q(x) = g(x^q) = \sum_{i=0}^{n-1} g_i x^{iq} \equiv \sum_{i=0}^{n-1} g_i Q_i(x) \pmod{f(x)}$$

Como

$$\sum_{i=0}^{n-1} g_i Q_i(x) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} g_i q_{i,j} x^j \right) = \sum_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} g_i q_{i,j} \right) x^j = \sum_{j=0}^{n-1} (\widetilde{g}\mathcal{Q})_j x^j$$

entonces,

$$g^q(x) \equiv \widetilde{g}\mathcal{Q} \pmod{f(x)}$$

□

Corolario 5.28. *Sea $g(x) = \sum_{i=0}^{n-1} g_i x^i$ un polinomio de $\mathbb{F}_q(x)$ de grado menor que n. Se tiene que $g(x)$ es un elemento de la subálgebra de Berlekamp, $\mathcal{B}$, si y sólo si el vector $\widetilde{g} = (g_0, g_1, ...g_{n-1})$ es un vector propio de $\mathcal{Q}$ asociado al valor propio 1,es decir, pertenece al núcleo de $\mathcal{Q} - Id$, donde Id es la matriz identidad.*

Demostración. Por el lema anterior, $g^q(x) \equiv \widetilde{g}\mathcal{Q} \pmod{f(x)}$. Además, $g(x)$ pertenece a $\mathcal{B}$ si y sólo si $g^q(x) \equiv g(x) \pmod{f(x)}$. Por tanto, que $g(x)$ sea un elemento de $\mathcal{B}$ es equivalente a decir que

$$\widetilde{g}\mathcal{Q} = g$$

Esto es,

$$\widetilde{g}(\mathcal{Q} - Id) = 0$$

como se quería demostrar. □

Por consiguiente, es necesario que 1 sea valor propio del endomorfismo $\mathcal{Q}$ para poder factorizar $f(x)$. Si 1 no es valor propio, entonces la subálgebra de Berlekamp es trivial y, en consecuencia, $f(x)$ es irreducible. Este hecho se puede generalizar por el siguiente resultado:

Lema 5.29. *Sea $f(x)$ un polinomio libre de cuadrados de $\mathbb{F}_q[x]$. El número de factores irreducibles de $f(x)$ es igual a la dimensión del núcleo de $\mathcal{Q} - I$.*

Demostración. Sea

$$f(x) = \prod_{i=1}^{m} p_i(x)$$

la factorización canónica de $f(x)$. Sea $g(x)$ un polinomio tal que $g^q(x) \equiv g(x) \pmod{f(x)}$, es decir, $f(x) \mid g^q(x) - g(x)$. Por el lema 5.24,

$$\prod_{i=1}^{m} p_i(x) \mid \prod_{s \in \mathbb{F}_q} (g(x) - s) \tag{5.4}$$

Los $p_i(x)$ son irreducibles y coprimos, también son coprimos los polinomios $(g(x) - s_j)$. Por tanto, la condición es cierta si y sólo si cada $p_i(x)$ divide a $(g(x) - s_j)$ para algún $s_j \in \mathbb{F}_q$. Esto es equivalente a que $g(x) \equiv s_j \pmod{p_i(x)}$. Por el teorema chino de los restos, existe un único $g(x)$ verificando que $g(x) \equiv s_j \pmod{p_i(x)}$ para $i = 0, 1, ..., m$. Cómo hay q^m posibilidades de establecer dichas congruencias, se puede afirmar que hay q^m polinomios $g(x)$ en $\mathcal{B}$. Se deduce el resultado del corolario 5.28. □

Se está ya en condiciones de presentar el algoritmo de Berlekamp.

Algoritmo 5.30. BERLEKAMP

Input $f(x) \in \mathbb{F}_q[x]$ libre de cuadrados de grado n
Output Factorización canónica de $f(x)$

1. Calcular la matriz de Berlekamp de $f(x)$, $\mathcal{Q}$.
2. Calcular una base B de $\mathcal{B}$, esto es, una base de $ker(\mathcal{Q} - I)$.
3. $i := 0$, $m := |B|$
4. $F = \{f(x)\}$
5. **Mientras** $i < m$ **hacer**:
   a) Elegir $\widetilde{g} \in B$ ($\widetilde{g}$ define un polinomio $g(x)$)
   b) $B := B \setminus \{\widetilde{g}\}$
   c) $F' := \{\}$
   d) **Para** todo $s \in \mathbb{F}_q$ **hacer**:
      1) $h(x) := mcd(f(x), g(x) - s)$
      2) **Si** $deg(h(x)) > 0$, $F' := F' \cup \{h(x)\}$
   e) $C = F \setminus F'$; $D = F' \setminus F$. Si $|D| > |C|$, $F = (F \setminus C) \cup D$
   f) $i := |F|$

En los tres primeros pasos del algoritmo, se calcula la matriz de Berlekamp asociada a $f(x)$, $\mathcal{Q}$, y una base, B, de $ker(\mathcal{Q} - I)$. Por el Corolario 5.28, los polinomios correspondientes a los vectores de B forman también una base de $\mathcal{B}$.

La variable i almacena el número de factores calculados en cada paso. Por el Lema 5.29, como $f(x)$ es libre de cuadrados, el número de factores irreducibles de $f(x)$ es $m = |B|$. Por este motivo, el algoritmo acaba cuando ha encontrado m factores, momento en el cual dichos factores constituyen la factorización canónica.

En cada iteración, se calcula una factorización parcial de $f(x)$ utilizando un cierto $g(x)$ de $\mathcal{B}$ a partir del teorema 5.25. Tanto el producto de los factores almacenados en $F = \{f_1(x), f_2(x), ..., f_k(x)\}$ como el de los almacenados en $F' = \{h_1(x), h_2(x), ..., h_l(x)\}$ es $f(x)$. Luego el producto de los elementos de C es igual al producto de los elementos de D. En efecto, si $F \cap F' = \{f_{s_1}(x), f_{s_2}(x), ..., f_{s_r}(x)\} = \{h_{t_1}(x), h_{t_2}(x), ..., h_{t_r}(x)\}$, entonces

$$\prod_{f_i(x) \in C} f_i(x) = \frac{f(x)}{f_{s_1}(x)...f_{s_r}(x)} = \frac{f(x)}{h_{t_1}(x)...f_{t_r}(x)} = \prod_{h_j(x) \in D} h_j(x)$$

Así, si $|D| > |C|$, se actualiza F como $F = (F \setminus C) \cup D$. Como el producto de los elementos de C y D es el mismo, el producto de los elementos en F sigue siendo $f(x)$. Además, como $|D| > |C|$, la factorización que se almacena en F tras la actualización tiene más factores que la almacenada anteriormente y, en consecuencia, se acerca más a la factorización canónica.

Ejemplo 5.31. *Se desea calcular la factorización canónica de $f(x) = x^3 - 2x^2 - x + 2$, que es libre de cuadrados, en $\mathbb{F}_5[x]$ aplicando el algoritmo de Berlekamp.*

1. $F = \{f(x)\}$
2. *Matriz de Berlekamp:* $\mathcal{Q} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ *ya que* $x^0 \equiv 1 \pmod{x^3 - 2x^2 - x + 2}$, $x^5 \equiv x \pmod{x^3 - 2x^2 - x + 2}$ *y* $x^{10} = x^2 \pmod{x^3 - 2x^2 - x + 2}$.
3. *Base de* $ker(\mathcal{Q} - I)$: $\{(1,0,0),(0,1,0),(0,0,1)\}$. *Entonces,* $m = 3$.
4. $\widetilde{g} = (0,1,0)$ *y, en consecuencia,* $g(x) = x$.
5. $mcd(f(x), x) = 1$; $mcd(f(x), x-1) = x-1$; $mcd(f(x), x-2) = x-2$; $mcd(f(x), x-3) = 1$; $mcd(f(x), x-4) = x-4$. *Por tanto,* $F' = \{x-1, x-2, x-4\}$.
6. $C = \{f(x)\}$, $D = \{x-1, x-2, x-4\}$. *Como* $|D| > |C|$, $F = F'$.
7. $i = |F| = 3 = m$ *y el algoritmo termina.*

Así, $x^3 - 2x^2 - x + 2 = (x-1)(x-2)(x-4)$.

Algoritmo de Cantor-Zassenhaus

El algoritmo de Cantor-Zassenhaus es una algoritmo para factorizar polinomios cuyos factores irreducibles son todos del mismo grado. Sin embargo, se puede utilizar para hallar la factorización canónica de un polinomio $f(x)$ cualquiera. Para ello, basta aplicar el algoritmo libre de cuadrados (algoritmo 5.19) a $f(x)$ y después el algoritmo distinto grado (algoritmo 5.22) a cada uno de los factores obtenidos. Después, sólo resta utilizar Cantor-Zassenhaus en cada uno de los factores hallados con el último algoritmo.

Los detalles del algoritmo de Cantor-Zassenhaus dependen del cuerpo $\mathbb{F}_q$ sobre el que se trabaje. Es preciso diferenciar los casos en que q es una potencia de primo impar de aquellos en los que es una potencia de 2. En lo que sigue, se considerará el primero de los casos y, tras presentar el algoritmo, se introducirán las modificaciones necesarias para que sea válido si q es par.

En lo que sigue, si $f(x) = \prod_{i=1}^{m} p_i(x)$ donde los $p_i(x)$ son primos entre sí, se denota por φ el isomorfismo:

$$\begin{array}{rccc} \varphi: & \dfrac{\mathbb{F}_q[x]}{(f(x))} & \longrightarrow & \dfrac{\mathbb{F}_q[x]}{(p_1(x))} \oplus ... \oplus \dfrac{\mathbb{F}_q[x]}{(p_m(x))} \\ & g(x) & \longmapsto & (g_1(x), ..., g_m(x)) \end{array}$$

definido en el teorema 3.8 (teorema chino de los restos para polinomios).

Teorema 5.32. *Sea* $f(x) = \prod_{i=1}^{m} p_1(x)p_2(x)...p_m(x)$ *un polinomio cuyos factores sean primos entre sí dos a dos. Sea* $g(x)$ *un polinomio de* $\dfrac{\mathbb{F}_q[x]}{(f(x))}$ *tal que:*

$$g(x) \neq 0, \pm 1 \quad y \quad \varphi(g(x)) = (g_1, g_2, ..., g_m) \text{ con } g_i \in \{0, 1, -1\} \text{ para } i = 1, ...m$$

Sean $A_0 = \{i \mid g_i = 0\}$, $A_1 = \{i \mid g_i = 1\}$ *y* $A_{-1} = \{i \mid g_i = -1\}$. *Entonces, dos de estos conjuntos son no vacíos y los polinomios asociados a dichos conjuntos de la siguiente manera:*

$$mcd(f(x), g(x)) = \prod_{i \in A_0} p_i(x)$$

$$mcd(f(x), g(x) - 1) = \prod_{i \in A_1} p_i(x)$$

$$mcd(f(x), g(x) + 1) = \prod_{i \in A_{-1}} p_i(x)$$

son factores no triviales de $f(x)$.

Demostración. Necesariamente dos de los conjuntos definidos son no vacíos. Si $g_i = 0$ para todo i, esto es, sólo A_0 fuese no vacío, por el teorema chino de los restos, $g(x) = 0$. De igual manera, si A_1 fuese el único conjunto no vacío, $g(x) = 1$ y si lo fuese A_{-1}, se tendría que $g(x) = -1$.

Se supone, sin pérdida de generalidad, que A_0 y A_1 son no vacíos. Para todo $i \in A_0$, $g(x) \equiv 0 \pmod{p_i(x)}$ y, por tanto, $p_i(x) \mid g(x)$. Además, si $i \notin A_0$, $g(x) \not\equiv 0 \pmod{p_i(x)}$ y, entonces, $p_i(x) \nmid g(x)$. La definición de máximo común divisor justifica que $mcd(f(x), g(x)) = \prod_{i \in A_0} p_i(x)$. Por otra parte, como A_1 es no vacío, no todos los factores de $f(x)$ pertenecen a A_0. Entonces, $\prod_{i \in A_0} p_i(x) \neq f(x)$.

De igual modo, se demuestra que $\prod_{i \in A_1} p_i(x)$ es un factor no trivial de $f(x)$, teniendo en cuenta que $g(x) \equiv 1 \pmod{p_i}$ y, por tanto, $g(x) - 1 \equiv 0 \pmod{p_i}$ para todo i en A_1. □

Todo polinomio $g(x) \in \dfrac{\mathbb{F}_q}{(f(x))}$ satisfaciendo las condiciones del teorema anterior proporciona una factorización no trivial de $f(x)$ que no ha de ser necesariamente la factorización canónica. Sin embargo, este resultado puede aplicarse de manera recursiva a cada factor obtenido hasta alcanzar la factorización deseada.

Sólo falta determinar cómo obtener un polinomio $g(x)$ que verifique las condiciones del teorema anterior (teorema 5.32). Los resultados presentados hasta ahora son válidos para cualquier polinomio con factores primos entre sí dos a dos. A partir de este momento, se considera que $f(x) = \prod_{i=1}^{m} p_1(x)p_2(x)...p_m(x)$ es un polinomio cuyos factores $p_i(x)$ son primos entre sí dos a dos y, además, tienen igual grado d. Sea $h(x)$ un polinomio cualquiera de $\dfrac{\mathbb{F}_q[x]}{(f(x))}$ y sea $k = (q^d - 1)/2$.

$$\varphi(h(x)^k) = (b_1^k(x), b_2^k(x), ..., b_m^k(x))$$

Como cada $p_i(x)$ es un polinomio irreducible de grado d, cada uno de los sumandos de $\bigoplus_{i=1}^{m} \mathbb{F}_q[x]/ < p_i(x) >$ es isomorfo a $\mathbb{F}_{q^d}[x]$. Entonces, $h_i^k(x) = h_i^{(q^d-1)/2}(x) = \pm 1$ si $h_i \neq 0$ para todo $i = 1, ..., m$. Si $h_i(x) = 0$, $h_i^k(x) = 0$. Si, además, $h^k(x) \neq 0, \pm 1$, dicho polinomio satisface las condiciones del teorema 5.32.

Se han presentado ya todos los resultados necesarios para justificar el funcionamiento del algortimo.

Algoritmo 5.33. CANTOR-ZASSENHAUS

Input $f(x) \in \mathbb{F}_q[x]$ con todos sus factores irreducibles del mismo grado, d
Output $f(x) = \prod_{i=1}^{3} f_i(x)$

1. Tomar $h(x) \in \mathbb{F}_q[x]/ < f(x) >$ tal que $h(x) \neq 0, \pm 1$
2. Calcular $g(x) = h(x)^{(q^d-1)/2}$
3. $f_1(x) := mcd(f(x), g(x))$
4. $f_2(x) := mcd(f(x), g(x) - 1)$
5. $f_3(x) := mcd(f(x), g(x) + 1)$
6. **Devolver** $f_1(x), f_2(x), f_3(x)$

Es claro que el algoritmo consiste únicamente en aplicar el teorema 5.32. Como se ha comentado con anterioridad, la factorización obtenida no tiene porqué ser la canónica pero esto se resuelve aplicando el algoritmo de manera recursiva a cada factor obtenido. La recursividad es posible porque cada factor es producto de polinomios irreducibles con el mismo grado. Además, como el grado de los factores obtenidos va disminuyendo, el proceso termina.

Se considera ahora el caso en que q es potencia de 2. En este caso, $k = (q-1)/2$ no es un entero por lo que el proceso descrito con anterioridad no es válido. No se podrá aplicar el teorema 5.32 sino una versión adaptada. Para factorizar en $\mathbb{F}_q[x]$ cuando q es par se distinguen los casos en que $q \equiv 1 \pmod 3$ y $q \equiv 2 \pmod 3$.

Si $q \equiv 1 \pmod 3$, entonces $(q^d - 1)/3$ es un entero. En tal caso, dado $h(x) \in \mathbb{F}_q[x]$, para todo $h_i(x) \neq 0$ se tiene que $h_i^k(x) = h_i^{(q^d-1)/3}(x) \in \{1, \rho, \rho^2\}$, donde ρ y ρ^2 son raíces cúbicas de la unidad en $\mathbb{F}_q$. Sea $g(x) = h^k(x)$. Se definen los conjuntos $A_0 = \{i \mid g_i = 0\}$, $A_1 = \{i \mid g_i = 1\}$,$A_\rho = \{i \mid g_i = \rho\}$ y $A_{\rho^2} = \{i \mid g_i = \rho^2\}$. Suponiendo que al menos dos de ellos son no vacíos y razonando de la misma manera que en el teorema 5.32, se tiene que al menos dos de los siguientes polinomios son factores no triviales de $f(x)$:

$$mcd(f(x), g(x)) = \prod_{i \in A} p_i(x)$$

$$mcd(f(x), g(x) - 1) = \prod_{i \in B} p_i(x)$$

$$mcd(f(x), g(x) - \rho) = \prod_{i \in D} p_i(x)$$

$$mcd(f(x), g(x) - \rho^2) = \prod_{i \in E} p_i(x)$$

Con estas modificaciones, el algoritmo de Cantor-Zassenhaus funciona.

Si, por el contrario, $q \equiv 2 \pmod 3$, no es seguro que $q^d \equiv 1 \pmod 3$ y las raíces cúbicas utilizadas en el caso anterior no han de existir necesariamente. Sin embargo, se puede trabajar en la extensión $\mathbb{F}_q(\rho)[x]$ y proceder de la misma manera que en el caso anterior para factorizar $f(x)$. A continuación, basta con combinar los factores que son conjugados para obtener la factorización en $\mathbb{F}_q[x]$.

Ejemplo 5.34. Dado $f(x) = x^3 - x^2 + x - 1$ en $\mathbb{F}_5[x]$, es posible hallar su factorización canónica utilizando el algoritmo de Cantor-Zassenhaus:

1. $h(x) = x - 1$
2. $g(x) = (x-1)^{\frac{5-1}{2}} = (x-1)^2 = x^2 - 2x + 1$
3. $f_1(x) = mcd(f(x), x^2 - 2x + 1) = x - 1$
4. $f_2(x) = mcd(f(x), x^2 - 2x) = x - 2$
5. $f_3(x) = mcd(f(x), x^2 - 2x + 2) = x - 3$

Por tanto, $f(x) = (x-1)(x-2)(x-3)$.

Los algoritmos de Berlekamp y Cantor-Zassenhaus que aquí se han presentado no son los únicos algoritmos de factorización existentes ni los más rápidos. En cambio, son los más utilizados por su sencillez y eficacia.

Bibliografía

[1] L. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to Cryptography*, Proceedings of the 20th Annual Symposium on Foundations of Computer Science, 55-60 (1979)

[2] S. Bai and R. P. Brent, *On the efficiency of Pollard's rho method for discrete logarithms*, The Australasian Theory Symposium (CATS2008), Conferences in Research and Practice in Information Technology 77, 125-131 (2008)

[3] M. Blum y S. Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, SIAM Journal on Computing 13(4), 850-864 (1984)

[4] B. den Boer,*Diffie–Hellman is as strong as discrete log for certain primes*,Advances in Cryptology - CRYPTO '88, LNCS 403,Springer, 530-539 (1988)

[5] R.P. Brent, *An Improved Monte Carlo Factorization Algorithm* BIT 20(2), 176–184 (1980)

[6] P. Caballero Gil, *Introducción a la Criptografía*, Ra-Ma (2002)

[7] A.A. Ciss, A.Y. Cheikh y D. Sow, *A Factoring and Discrete Logarithm based Cryptosystem*, International Journal of Contemporary Mathematical Sciences 8(11), 511 - 517 (2013)

[8] W.Diffie y M.E. Hellman, *New directions in Cryptography*, IEEE Transactions on Information Theory 22, 644-654 (1976)

[9] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory 31, 469-472 (1985)

[10] B. Harris, *Probability Distribution Related to Random Mappings*, Annals of Mathematical Statistics 31, 1045–1062 (1960)

[11] A. Kerckhoffs, *La Cryptographie militaire*, Journal des sciences militaires IX,5-23,161-193 (1883)

[12] N. Koblitz, *A course in Number Theory and Cryptography*, Springer-Verlag (1994)

[13] J. Massey y J.K. Omura, *U.S. Patent 4,567,600* (1982)

[14] U.M. Maurer,*Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms*, Advances in Cryptology - CRYPTO '94, LCNS 839, 271-281 (1994)

[15] L. Maurits, *Public key Cryptography using discrete logarithms in finite fields: Algorithms, efficient implementation and attacks*, Honours Thesis, Universidad de Adelaida (2006)

[16] C. Munuera y J. Tena, *Codificación de la informacion*, Valladolid: Secretariado de Publicaciones e Intercambio Científico, Universidad de Valladolid, (1997)

[17] NIST, *DSS (Announcing the Standard for Digital Signature Standard)*, FIPS Publication 186 (1994)

[18] A.M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptology: EUROCRYPT 84,LNCS 209,Springer, 224-314 (1984)

[19] A.M. Odlyzko, *Discrete logarithms: the past and the future*, Designs, Codes, and Cryptography 19, 129-145 (2000)

[20] R.W.K. Odoni, V.Varadharajan y P.W. Sanders, *Public key distribution in matrix rings*, Electronic Letters 20, 386-387 (1984)

[21] R.W.K. Odoni, y V.Varadharajan, *Security of public key distribution in matrix rings*, Electronic Letters 22, 46-47 (1985)

[22] S. Pohlig y M. Hellman, *An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance*, IEEE Transactions on Information Theory 24, 106–110 (1978)

[23] J. M. Pollard, *Monte Carlo methods for index computation (mod p)*, Mathematics of Computation 32(143), 918–924 (1978)

[24] K. Sakurai y H. Shizuya,*Relationships among the computational powers of breaking discrete log cryptosystems*, Eurocrypt'95, LNCS 921, 341-355(1995)

[25] D. Shanks, *Class number, a theory of factorization and generation*, Proceedings of Symposia in Pure Mathematics 20, 415—440 (1971)

[26] C.E. Shannon, *A mathematical theory of Communication*, The Bell System Technical Journal, 27,379-423, 623-656 (1948)

[27] D.R. Stinson, *Cryptography: theory and practice*, Discrete Mathematics and its Applications,Chapman and Hall (2005)

[28] E. Teske, *On random walks for Pollard's rho method*, Mathematics of Computation 70(234), 809–825 (2001)

[29] E. Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, Algorithmic Number Theory Symposium (ANTS IV), LNCS 1423, 541-553 (1998)

[30] Y. Tsiounis y M. Yung, *On the security of ElGamal based encryption*, Public Key Cryptography, LNCS 1431, 117-134 (1998)