Criptografía básica y algunas aplicaciones

Francisco Barranco Blázquez
Supervisor: Carlos Galindo

Octubre 2022

Universidad Jaume I
Departamento de Matemáticas
Castellón, España

Índice

1. Introducción

Desde la antiguedad fue importante para los seres humanos comunicarse de modo secreto. La ciencia que estudia esta comunicación se denomina criptología y tiene dos ramas. La criptografía que es el estudio de como enviar un mensaje entre un emisor y un receptor que no pueda ser entendido por un oponente y el criptoanalisis que estudia que puede hacer el oponente para descifrar el mensaje. Hasta la aparición de los ordenadores la criptología se usaba principalmente con fines militares. En este trabajo nos centraremos sobre todo en la criptografía. En la actualidad tiene un importante rol en casi todas las facetas de la economía. Por ejemplo, mucho dinero se mueve en el entorno de las criptodivisas, así que con el aumento del uso de estas criptomonedas, se puede observar como se está produciendo un aumento paralelo de las medidas de seguridad relacionadas. Una de estas medidas de seguridad es la necesidad de la criptografía en la seguridad de la red.

La seguridad de la red se ha convertido en una parte importante del sistema de comunicación moderno. La necesidad de la seguridad de la red surgió para mantener la confidencialidad e integridad de la información compartida entre usuarios y para protegerla del acceso no autorizado de terceros.

La criptografía es una forma de asegurar la información y la comunicación importantes mediante unas claves adecuadas. Estas claves sólo están disponibles para el propietario legítimo de la información. Existen diversos sistemas de encriptación para lograr la seguridad mientras se comunica en una red pública. La palabra criptografía deriva de una palabra griega que significa "escritura secreta". La criptografía puede entenderse como el proceso mediante el que un remitente envía un mensaje que originalmente existe en texto plano. Antes de la transmisión del mensaje a través de la red, se encripta y se convierte en texto cifrado. Cuando este mensaje es recibido por el destinatario, se descifra de nuevo en texto plano.

Hay muchas técnicas diferentes que se utilizan en la criptografía para encriptar información que utilizan un cierto conjunto de reglas, conceptos y resultados matemáticos y se implementan en algoritmos muy útiles en la actualidad. Entre otras aplicaciones, estos algoritmos se utilizan para las firmas electrónicas de los contratos electrónicos, la verificación para proteger la privacidad de los datos, la navegación web en Internet, la protección de las transacciones confidenciales, como las transacciones con tarjetas de débito y crédito, etc. La generación de claves criptográficas es clave en los procesos anteriores.

El objetivo de este trabajo es por un lado aportar al lector los conceptos matemáticos básicos que usa la criptografía para asegurar la información. Y por otro, introducirle en los métodos criptográficos más conocidos y alguna de sus aplicaciones. Algunos conceptos de teoría de números y matemáticas discretas son resumidos brevemente para luego dar paso a la explicación de ciertos criptosistemas que se han usado a lo largo de la historia y otros que son usados en la actualidad. Como se menciona, el texto solo aportará un conocimiento básico sobre el tema a tratar, para tener una lectura y un conocimiento más detallado sobre criptografía recomendamos acudir a los textos de D. R. Stinson [53], y de N. Koblitz [26].

El texto se estructura de la siguiente manera. En el **capítulo** 2 se presentan los conceptos de teoría de números que serán necesarios para obtener la base matemática, en el **capítulo** 3 se exponen algunos criptosistemas sencillos de clave privada que se han usado a lo largo de la historia y dos de los más populares criptosistemas de clave privada como son el DES y el AES. En el **capítulo** 4 se introducen los conceptos de matemática discreta que serán necesarios para entender el **capítulo** 5, en el cual se describen los criptosistemas de clave pública y los tests de primalidad necesarios para dichos criptosistemas. Finalmente el **capítulo** 6 muestra algunas aplicaciones más comunes en las que se utiliza la criptografía explicando de manera detallada el funcionamiento de cada aplicación.

2. Teoría de Números

El objetivo a alcanzar con este primer capítulo es el de recordar los conceptos y la notación utilizada en teoría de números elemental, teoría que será importante a la hora de avanzar en la lectura de los capitulos siguientes. Como se verá en capítulos futuros un tema al que prestaremos especial atención en criptografía será el de realizar estimaciones del número de operaciones de bits necesarios para realizar tareas por ordenador. Y por ello entraremos en mayor detalle en el tema de las estimaciones de tiempo. Para una lectura más detallada sobre teoría de números proponemos acudir al libro de N. Koblitz [26] publicado en 1994 que nos sirve de guía para el siguiente capítulo.

2.1. Estimaciones de Tiempo para Aritmética

Definición 2.1:(Números en diferentes bases) *Un entero no negativo n escrito en base b es una notación para n de la forma* $(d_{k-1}d_{k-2}\dots d_1d_0)_b$. *donde las d son b−dígitos, es decir, símbolos para los enteros entre* 0 *y* $b-1$*; esta notación significa que:*

$$n = d_{k-1}b^{k-1} + d_{k-2}b^{k-2} + \cdots + d_1b + d_0.$$

Si el primer dígito d_{k-1} *no es cero, llamamos a n un número en base b de k dígitos.*

Cualquier número entre b^{k-1} y b^k es un número de k dígitos en base b. Cuando la base sea claramente reconocible por el contexto podremos omitir el uso del parentesis y el subindice $(\dots)_b$. Dado que a veces es útil trabajar en bases distintas de la 10, hay que acostumbrarse a hacer aritmética en una base arbitraria y a convertir de una base a otra.

Las fracciones también pueden expandirse en cualquier base. Cuando $b > 10$ se acostumbra a utilizar letras para los dígitos más allá del 9, aunque también se pueden utilizar letras para todos los dígitos. Cuando $b = 26$, utilizamos las letras $A - Z$ para los dígitos $0 - 25$.

Las fracciones se expanden en cualquier base, representadas en la forma $(d_{k-1}d_{k-2}\dots d_1d_0.d_{-1}d_{-2}\dots)_b$.

Lemma 2.2: *Para convertir un número decimal n a la base b, obtenemos* d_0 *dividiendo n entre b y tomando el resto. Luego sustituimos n por el cociente y repetimos el proceso para obtener* d_1*, y así sucesivamente. Después de ocuparnos de la parte entera, la parte fraccionaria se convierte a la base b multiplicando por b, tomando la parte entera del resultado como* d_{-1}*, y repitiendo con la nueva parte fraccionaria.*

Lemma 2.3: *El número de dígitos en base b de un número entero n viene dado por:*

$$d_b(n) = [\frac{\log(n)}{\log(b)}] + 1, \tag{1}$$

donde [] *se usa para referirse a la función de mayor parte entera, y* log *denota el logaritmo natural.*

2.1.1. Estimación del tiempo (operaciones bit)

En primer lugar, expondremos las consideraciones que utilizamos:

1. Toda tarea complicada puede descomponerse en operaciones de bits.

2. La cantidad de tiempo utilizada por un ordenador para realizar una tarea es esencialmente proporcional al número de operaciones de bits.

3. Cuando hablamos de estimar el tiempo que se tarda en realizar algo, nos referimos a encontrar una estimación del número de operaciones de bits necesarias.

4. En estas estimaciones, no tendremos en cuenta el tiempo requerido para los pasos lógicos distintos de las operaciones de bits.

5. Definimos la estimación de tiempo para una tarea aritmética como un límite superior para el número de operaciones de bits.

6. Las tareas aritméticas pueden ampliarse a fracciones sin aumentar el número de operaciones de bits.

Proposición 2.4: *Sumar dos enteros n y m requiere como máximo* $1 + \text{máx}(\log_2 n, \log_2 m)$ *operaciones de bits.*

Demostración: Si uno tiene menos bits que el otro, rellenamos con ceros a la izquierda, para que tengan la misma longitud. Entonces, cada operación realizada (sumar los dos bits, contabilizar el acarreo) es lo que contamos como una operación de bits.

La misma estimación es válida para la resta.

Proposición 2.5: *Multiplicar dos enteros n y m requiere menos de* $(1 + \log_2 m)(1 + \log_2 n)$ *operaciones de bits.*

Demostración: Usamos el procedimiento conocido para multiplicar enteros de k y l bits n y m. Obtenemos a lo sumo l filas, por cada bit en m, y cada una desplazada una posición a la izquierda (lo que no añade operaciones de bits). Por lo tanto, tenemos $l - 1$ sumas, cada una de las cuales toma k operaciones de bits, por lo que requerimos a lo sumo $(l - 1)k$ operaciones de bits, concretamente, requerimos menos de lk operaciones de bits.

La misma estimación es valida para la división.

Proposición 2.6: *Calcular $n!$ requiere como máximo* $n(n-2)(1+\log_2 n)^2$ *operaciones de bits.*

Demostración: Calculamos los $n - 2$ productos necesarios para obtener $n!$. Como peor estimación del número de bits de cada uno de los términos, tomamos el número de dígitos de $n!$, y utilizamos que el número de dígitos del

producto es como máximo la suma de los dígitos de cada factor. A partir de aquí, como n tiene $1+\log_2 n$ bits, entonces $n!$ tiene como máximo $n(1+\log_2 n)$ bits.

Como tenemos $n-2$ multiplicaciones, cada una un número de a lo sumo $1+\log_2 n$ bits por un número con a lo sumo $n(1+\log_2 n)$ bits, obtenemos el límite deseado.

Proposición 2.7: *Multiplicar dos polinomios de grados n_1, n_2 con $n_2 \leq n_1$ cuyos coeficientes son enteros positivos $\leq m$ requiere como máximo $(n_1+n_2+1)[(1+n_2)(1+\log_2 m)^2+n_2(1+\log_2(n_2m^2))]$ operaciones de bits.*

Demostración: Para calcular cada coeficiente necesitamos como máximo n_2+1 multiplicaciones y n_2 sumas. Las multiplicaciones están limitadas por m, y los números que se suman son como máximo m^2, y como tenemos hasta n_2 números, tomamos n_2m^2 como límite del tamaño de los números que se suman.

Si suponemos que $m \geq 16$ y $m \geq \sqrt{n_2}$, podemos simplificar la cota a $4n_1^2(\log_2 m)^2$ operaciones de bits.

Proposición 2.8: *Se necesitan como máximo $2(m-1)m(1+\log_2 n)^2$ operaciones de bits para calcular $\binom{n}{m}$.*

Demostración: Como $\binom{n}{m} = \binom{n}{n-m}$ asumimos que $m \leq n/2$. Entonces calculamos $n \ldots (n-m+1)/2 \ldots m)$, que consiste en $m-1$ multiplicaciones y $m-1$ divisiones. Un límite para los productos es n^m y n, así que con un argumento similar al utilizado para el factorial, tenemos el límite deseado.

2.1.2. Notación Big-O

Definición 2.9: *Sean f, g funciones positivas tales que $f, g : \mathbb{N} \to \mathbb{N}$. Decimos que $f = \mathcal{O}(g)$ si $f < Cg$ cuando $n \to \infty$, y C es una constante.*

Podemos extender la definición a múltiples variables, utilizando que cuando todas las variables son mayores que algún número, $f < Cg$.

Escribir $f = \mathcal{O}(1)$ es lo mismo que f esté acotado.

Ejemplo 1:

1. Si f es un polinomio de grado d, entonces $f = \mathcal{O}(nd)$.
2. $\log(n) = O(n^\epsilon)$ para cualquier número positivo ϵ.
3. $d_b(n) = \mathcal{O}(\log(n))$.

4. Las operaciones de bits necesarias para calcular nm son $\mathcal{O}(\log(n)\log(m))$.

5. Las operaciones de bits necesarias para calcular $n!$ es $\mathcal{O}((n\log(n))^2)$.

6. Las operaciones de bits necesarias para calcular el producto de dos polinomios como se ha mencionado anteriormente es $\mathcal{O}(n_1 n_2((\log(m))^2 + \log(\text{mín}(n_1, n_2)))$.

7. Las operaciones de bits necesarias para calcular $\binom{n}{m}$ es $\mathcal{O}(m^2\log^2(n))$.

Proposición 2.10: *El número de operaciones de bits necesarias para transformar un entero binario a su representación en base* 10 *es* $\mathcal{O}(log^2(n))$.

Demostración: Dividimos n entre $10 = (1010)_2$. Tomamos el resto como el dígito de las unidades, y sustituimos n por el cociente, y repetimos. Repetimos esta operación un total de $d_{10}(n) = \mathcal{O}(\log(n))$ veces, cada una de ellas tomando $\mathcal{O}(log(n))$ divisiones.

Las operaciones de bits necesarias para transformar a una base arbitraria b también son $\mathcal{O}(\log^2(n))$. El mayor tiempo requerido para encontrar cada dígito se compensa por el hecho de que hay menos dígitos que encontrar.

Definición 2.11: *Decimos que un algoritmo que involucra enteros* $n_1, \ldots,$ n_r *de* $k_1, \ldots, k_r$ *bits es un algoritmo de tiempo polinómico si existen enteros* $d_1 \ldots d_r$ *tales que el número de operaciones de bits necesarias para realizar el algoritmo es* $\mathcal{O}(k_1^{d_1} \ldots k_r^{d_r})$.

2.2. Divisibilidad y algoritmo euclídeo

2.2.1. Divisores y divisibilidad

Definición 2.12:. *Sean* a, b *números enteros. Decimos que* a *divide a* b *(o que* b *es divisible por* a*), y escribimos* $a|b$ *si existe un número entero* k *tal que* $b = ak$*. En ese caso, llamamos* a *un divisor de* b.

Decimos que a es un *divisor propio* de b si a es un divisor positivo de b, pero no es igual a b.

Decimos que a es un *divisor no trivial* de b si a es un divisor positivo no igual a 1 ni a b.

Un *número primo* es un número entero p mayor que uno y sin divisores no triviales.

Un *número compuesto* es un número entero con al menos un divisor no trivial.

Si p es un número primo y α es un entero no negativo, escribimos $p^\alpha || b$ para significar que p^α es la mayor potencia de p que divide a b, y decimos que p^α divide exactamente a b.

Lemma 2.13:

1. *Si $a|b$ y c es un número entero cualquiera, entonces $a|bc$.*

2. *Si $a|b$ y $b|c$, entonces $a|c$.*

3. *Si $a|b$ y $a|c$, entonces $a|b \pm c$.*

Teorema 2.14: (Teorema fundamental de la aritmética). *Cualquier número natural n puede escribirse de forma única como un producto de números primos (excepto por el orden de los factores).*

Corolario 2.15: *Si $n = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ entonces n tiene $(\alpha + 1) \dots (\alpha_r + 1)$ divisores diferentes.*

Lemma 2.16:

1. *Si un número primo p divide a ab, entonces o $p|a$ o $p|b$.*

2. *Si $m|a$ y $n|a$, y si m y n no tienen divisores mayores que 1 en común, entonces $mn|a$.*

Definición 2.17: *Decimos que d es el máximo común divisor de a y b, y lo denotamos por $d = \gcd(a, b)$, si d es el mayor número entero d que divide a y b.*

Decimos que m es el mínimo común múltiplo de a y b, y lo denotamos por $m = \mathrm{lcm}(a, b)$, si m es el menor número entero positivo que dividen a y b.

2.2.2. El algoritmo Euclídeo

Teorema 2.18: (Euclídeo). *Sean a, b números enteros, y sean q, r el cociente y el resto de la división de a por b. Entonces:*

$$\gcd(a, b) = \gcd(b, r). \tag{2}$$

Proposición 2.19: (Algoritmo Euclídeo). *Sean $a > b$ números enteros. Para encontrar $\gcd(a, b)$, se dividide b entre a para obtener el cociente q_1 y el resto r_1. A continuación, divide r_1 entre b para obtener r_2, y repite la división*

de r_i entre r_{i+1} hasta que el siguiente resto sea exactamente cero. Ese resto final no nulo es precisamente $\gcd(a, b)$.

Proposición 2.20:. *El tiempo que se tarda en encontrar* $\gcd(a, b)$ *es* $\mathcal{O}(\log^3(a))$.

Demostración: Es fácil comprobar que $r_{j+2} < \frac{1}{2} r_j$. Como cada dos pasos cortan el tamaño al menos a la mitad, y el resto nunca baja de 1, entonces tenemos a lo sumo $2[\log_2(a)]$ divisiones, que es $\mathcal{O}(\log(a))$, y cada división implica números no mayores que a, por lo que cada una tarda $\mathcal{O}(\log^2(a))$, para un total de $\mathcal{O}(\log^3(a))$.

Podemos reducir la estimación anterior a $\mathcal{O}(\log^2(a))$, teniendo en cuenta el tamaño decreciente de los números en las sucesivas divisiones.

Proposición 2.21: *Sea $d = \gcd(a, b)$, con $a > b$. Entonces existen enteros u y v tales que $d = ua + bv$, y se pueden encontrar en $\mathcal{O}(\log^3(a))$.*

Definición 2.22: *Decimos que dos enteros a, b son relativamente primos si* $\gcd(a, b) = 1$.

Corolario 2.23: *Si $a > b$ son enteros relativamente primos, entonces 1 puede escribirse como una combinación lineal entera de a y b en tiempo polinómico.*

Definición 2.24: *Sea n un número entero positivo. La función phi de Euler $\varphi(n)$ es el número de enteros no negativos menores que n que son primos con n.*

Lemma 2.25:

1. $\varphi(1) = 1$.
2. $\varphi(p) = p - 1$.
3. $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$.

2.3. Congruencias

2.3.1. Propiedades básicas

Definición 2.26: *Decimos que a es congruente con b módulo m y escribimos $a \equiv b$ módulo m si la diferencia $a - b$ es divisible por m. En este caso, m se llama módulo de la congruencia.*

Lemma 2.27: *Se cumplen las siguientes propiedades:*

1. *La congruencia módulo m para m fijo es una relación de equivalencia.*

2. *Cada clase de equivalencia con respecto a la congruencia módulo m tiene exactamente un representante entre* 0 *y* $m-1$.

3. *Si* $a \equiv b \mod m$ *y* $c \equiv d \mod m$, *entonces* $a \pm c \equiv b \pm d \mod m$.

4. *Si* $a \equiv b \mod m$, *entonces* $a \equiv b \mod d$ *para cualquier divisor* $d|m$.

5. *Si* $a \equiv b \mod m$ *y* $a \equiv b \mod n$, *con* m *y* n *relativamente primos, entonces* $a \equiv b \mod mn$.

Proposición 2.28: *Los elementos del anillo de congruencias* $\mathbb{Z}/m\mathbb{Z}$ *que tienen inversos multiplicativos son aquellos relativamente primos a* m.

Además, podemos encontrar la inversa de cada clave en $\mathcal{O}(\log^3(m))$ *operaciones de bits.*

Demostración: Si $d = \gcd(a, m)$ fuera mayor que 1 y $ab \equiv 1 \mod m$, entonces d dividiría $ab-1$, y luego dividiría 1. Por tanto, $d = gcd(a, m) = 1$, y tenemos que $ua + vm = 1$. Eligiendo $b = u$, tenemos que m divide $1 - ua = 1 - ab$, como se desea.

Corolario 2.29: *Si* p *es un número primo, entonces cada clase de residuo no nulo tiene una inversa multiplicativa, que puede encontrarse en* $\mathcal{O}(\log^3(p))$ *operaciones de bits.*

Corolario 2.30: *La ecuación* $ax \equiv b \mod m$ *tiene solución si y sólo si* $\gcd(a, m)$ *divide a* b. *Una solución* x_0 *puede encontrarse en* $\mathcal{O}(\log^3(m))$ *operaciones de bits, y todas las soluciones son de la forma* $x = x_0 + nm/\gcd(a, m)$ *para todos los enteros* n.

Corolario 2.31: *Si* $a \equiv b \mod m$ *y* $c \equiv d \mod m$, *y* $\gcd(c, m) = 1$, *entonces* $ac^{-1} \equiv bd^{-1} \mod m$.

Teorema 2.32: (Pequeño teorema de Fermat). *Sea* p *un primo. Cualquier número entero* a *satisface* $a^p \equiv a \ mod\ p$, *y cualquier número entero* a *no divisible por* p *satisface* $a^{p-1} \equiv 1 \ mod\ p$.

Demostración: Supongamos que p no divide a a. Entonces $0a, 1a, \ldots, (p-1)a$ son un conjunto completo de residuos módulo p, ya que si no lo fueran, eso significaría que $ia \equiv ja$ para algún i, j, pero esto significaría que p divide $i - j$, y esto significaría que $i = j$.

Por lo tanto, $a, 2a, \ldots, (p-1)a$ es simplemente un reordenamiento de $1, 2, \ldots, p-1$ cuando se considera módulo p. De aquí tenemos que $a^{p-1}(p-1)! \equiv (p-1)! \mod p$, lo que significa que p divide a $a^{p-1} - 1$, ya que p no divide a $(p-1)!$. Entonces $a^{p-1} \equiv 1 \mod p$, y multiplicando por a

obtenemos el resultado deseado cuando p no divide a a. En el caso de que p divida a a, ambos lados son simplemente cero.

Corolario 2.33: *Si a no es divisible por p y si $n \equiv m \mod p-1$, entonces $a^n \equiv a^m \mod p$.*

Demostración: Tomamos $n > m$. Como $p-1$ divide a $n-m$, tenemos $n = m + c(p-1)$ para algún entero positivo c. Entonces multiplicando la congruencia $a^{p-1} \equiv 1 \mod m$ por sí misma c veces y por $a^m \equiv a^m \mod p$ se obtiene el resultado deseado.

Proposición 2.34: (Teorema del resto chino). *Supongamos que m_i y m_j son coprimos, entonces existe una solución x del sistema:*

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ x &\equiv a_2 \mod m_2 \\ &\cdots \\ x &\equiv a_r \mod m_r. \end{aligned}$$

Además, dos soluciones cualesquiera son congruentes entre sí módulo $m_1 \cdots m_r$.

Demostración: La unicidad de que dos soluciones cualesquiera son congruentes entre sí módulo $m_1 \cdots m_r$ es fácil de ver, ya que la diferencia entre dos soluciones cualesquiera será cero módulo m_i, y por tanto, para el producto de todas ellas.

Ahora consideramos M_i como el producto de todos los m_i excepto el i-ésimo. Como $\gcd(m_i, M_i) = 1$, existe N_i, inverso de M_i. Ahora consideramos $x = \sum a_i M_i N_i$. Claramente, $x \equiv a_i M_i N_i \equiv a_i \mod m_i$, como se desea.

Corolario 2.35: *La función phi de Euler es multiplicativa, es decir, $\varphi(mn) = \varphi(m)\varphi(n)$ cuando $\gcd(m,n) = 1$.*

Demostración: Para cada j en el intervalo entre 0 y $mn-1$, definimos j_1 y j_2 como los menores residuos no negativos módulo m y n. Del teorema anterior, para cada par, hay una y sólo una j en el rango para el que $j \equiv j_1 \mod m$ y $j \equiv j_2 \mod n$. Por tanto, las j que tenemos que contar están en correspondencia 1 a 1 con los pares j_1, j_2 tales que j_1 y j_2 son menores que m y n, y coprimos a ellos también. La cardinalidad de esos pares es $\varphi(m)\varphi(n)$.

Proposición 2.36: *Supongamos que se sabe que n es el producto de dos primos distintos p y q. Entonces, se puede calcular $\varphi(n)$ a partir de p, y q en $\mathcal{O}(\log(n))$ operaciones de bits, y calcular p y q a partir de n y $\varphi(n)$ en $\mathcal{O}(\log^3(n))$ operaciones de bits.*

Demostración: Podemos suponer que n es par, ya que en este caso tendríamos que $\varphi(n) = n/2 - 1$. Por la multiplicatividad de φ, tenemos que $\varphi(n) = n + 1 - (p + q)$, utilizando una sustracción y una suma.

Ahora, si no conocemos p, q, sabemos que $pq = n$ y $p + q = n + 1 - \varphi(n)$. Podemos sustituir esta última expresión por $2b$. Entonces, p, q son las raíces de $x^2 - 2bx + n$, es decir, $p, q = b \pm \sqrt{b^2 - n}$, que tiene como paso más lento la raíz cuadrada.

Proposición 2.37: *Si* $\gcd(a, m) = 1$*, entonces* $a\varphi(m) \equiv 1 \mod m$.

Demostración: Consideramos primero el caso en el que m es una potencia prima, p^α, y procedemos por inducción sobre α. El caso $n = 1$ está cubierto por el pequeño teorema de Fermat, y para $\alpha \geq 2$, como $a^{p^{\alpha-1}-p^{\alpha-2}} = 1 + p^{\alpha-1}b$ para algún b, por el supuesto de inducción. Elevando ambos lados a la potencia p-ésima vemos que $a^{p^\alpha - p^{\alpha-1}}$ es congruente con 1 módulo p^α.

El resultado en general puede verse fácilmente utilizando la multiplicatividad de φ.

Corolario 2.38: *Si* $\gcd(a, m) = 1$ *y* n' *es el menor residuo no negativo de* n *modulo* $\varphi(m)$*, entonces* $a^n \equiv a^{n'} \mod m$.

2.3.2. Exponenciación modular por el método del cuadrado repetido

Un cálculo básico en aritmética modular es encontrar $b^n \mod m$, cuando tanto m como n son muy grandes. Supondremos que $b < m$ y que cada operación se reduce inmediatamente módulo m y describiremos un algoritmo eficiente.

Usamos a para denotar el producto parcial. Cuando hayamos terminado, tendremos a igual al menor residuo no negativo de $b^n \mod m$. Empezamos con $a = 1$, y escribimos $n_0 \dots n_{k-1}$ los dígitos binarios de n. Si $n_0 = 1$ cambiamos a por b, luego elevamos b al cuadrado y establecemos $b_1 = b^2 \mod m$. Si $n_1 = 1$, multiplicamos a por b_1, de lo contrario mantenemos a sin cambios. A continuación, elevamos al cuadrado b_1 y fijamos $b_2 = b_1^2 \mod m$. Si $n_j = 1$, entonces incluimos b_j en el producto por a. Después del paso $k - 1$ tendremos el deseado $a \equiv b^n \mod m$.

Proposición 2.39: *El tiempo necesario para obtener* $b^n \mod m$ *es* $\mathcal{O}((\log(n))(\log^2(m)))$.

Proposición 2.40: $\sum_{d|n} \varphi(d) = n$.

Demostración: Sea $f(n)$ el lado izquierdo de la igualdad, tenemos que demostrar que $f(n) = n$.

Primero mostraremos que f es multiplicativa. Para demostrarlo, observamos que cualquier divisor $d|mn$ puede escribirse de forma única en la forma d_1d_2, donde $d_1|m$ y $d_2|n$, y d_1 y d_2 son coprimos. Entonces:

$$f(mn) = \sum_{d_1|m}\sum_{d_2|n}\varphi(d_1)\varphi(d_2) = (\sum_{d_1|m}\varphi(d_1))(\sum_{d_2|n}\varphi(d_2)) = f(m)f(n). \quad (3)$$

Ahora es suficiente con demostrar que $f(p^\alpha) = p^\alpha$, pero los diviores de p^α son p^j, encontes:

$$f(p^\alpha) = \sum_{j=0}^{\alpha}\varphi(p^j) = 1 + \sum_{j=0}^{\alpha}(p^j - p^{j-1}) = p^\alpha. \quad (4)$$

2.4. Algunas formas de factorizar

Proposición 2.41: *Para cualquier número entero $b \neq 1$ y cualquier número entero positivo n, $b^n - 1$ es divisible por $b - 1$ con cociente $b^{n-1} + b^{n-2} + \cdots + b^2 + b + 1$.*

Demostración: A partir de la identidad de los polinomios:

$$x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \cdots + x^2 + x + 1),$$

Remplazamos x por b. Si escribimos $b^n - 1$ en base b, tenemos n dígitos $b-1$. En cambio, $b^{n-1}+b^{n-2}+\cdots+b^2+b+1$ consta de n dígitos 1, que multiplicados por $n - 1$ nos dan la igualdad deseada.

Corolario 2.42: *Para cualquier entero b y cualquier entero positivo m y n, tenemos:*

$$b^{mn} - 1 = (b^m - 1)(b^{m(n-1)} + b^{m(n-2)} + \cdots + b^{2m} + b^m + 1) \quad (5)$$

Demostración: Solo tenemos que sustituir b por b^m en la proposición anterior.

Proposición 2.43: *Sean b y m coprimos. Si $b^a \equiv 1 \mod m$ y $b^c \equiv 1 \mod m$, y $d = \gcd(a, c)$, entonces $b^d \equiv 1 \mod m$.*

Demostración: Podemos escribir $d = ua + vc$, donde claramente o bien u o bien v es positivo, y el otro es negativo o cero. Supongamos que $u > 0$ sin pérdida de generalidad. Elevar $b^a \equiv 1 \mod m$ a la potencia u-ésima

y $b^c \equiv 1 \mod m$ a la $(-v)$-ésima potencia, y dividir las congruencias resultantes, obteniendo $b^{au+cv} \equiv 1 \mod m$.

Proposición 2.44: *Si p es un primo que divide a $b^n - 1$, entonces o bien p divide a $b^d - 1$ para algún divisor propio d de n, o bien $p \equiv 1 \mod n$. Si $p > 2$ y n es impar, entonces en el segundo caso tenemos que $p \equiv 1 \mod 2n$.*

Demostración: Como $b^n \equiv 1 \mod p$, por el teorema pequeño de Fermat, tenemos que $b^{p-1} \equiv 1 \mod p$. Por la proposición anterior, tenemos que $b^d \equiv 1 \mod p$, donde $d = \gcd(n, p-1)$. Si $d < n$, entonces p divide a $b^d - 1$ para un divisor propio d de n. Por otro lado, si $d = n$, como d divide a $p-1$, tenemos $p \equiv 1 \mod n$. Finalmente, si p y n son ambos impares y n divide a $p-1$, obviamente $2n$ divide a $p-1$.

3. Criptografía de Clave Privada o Clásica

La criptografía simétrica, conocida también como criptografía de clave secreta, consiste en utilizar un único secreto compartido para compartir datos cifrados entre las partes. Los cifrados de esta categoría se llaman simétricos porque se utiliza la misma clave para cifrar y descifrar los datos. En términos sencillos, el remitente encripta los datos utilizando una contraseña, y el destinatario debe conocer esa contraseña para acceder a los datos.

El cifrado simétrico es un proceso bidireccional. Con un bloque de texto plano y una clave determinada, los cifrados simétricos siempre producirán el mismo texto cifrado. Del mismo modo, el uso de esa misma clave en ese bloque de texto cifrado siempre producirá el texto plano original. El cifrado simétrico es útil para proteger datos entre partes con una clave compartida establecida y también se utiliza con frecuencia para almacenar datos confidenciales.

En este capítulo se presenta una serie de criptosistemas simples de clave privada que se llevan utilizando desde hace varias décadas, que ayudarán al lector a entender los conceptos de encriptado y desencriptado con algunos ejemplos sencillos al estilo de D. R. Stinson [53].

3.1. Algunos criptosistemas sencillos

Introducimos la descripción formal de un criptosistema, que puede pensarse como reglas para cifrar y descifrar un mensaje, con el fin de que éste sea desconocido para alguien que lo intercepte.

Definición 3.1: *Un criptosistema es una tupla $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, con las siguientes condiciones:*

1. *El conjunto $\mathcal{P}$ es un conjunto finito de posibles textos planos.*
2. *El conjunto $\mathcal{C}$ es un conjunto finito de posibles textos cifrados.*
3. *El espacio de claves $\mathcal{K}$ es un conjunto finito de claves posibles.*
4. *Para cada $K \in \mathcal{K}$, existe una regla de cifrado $e_k \in \mathcal{E}$ y una regla de descifrado correspondiente $d_k \in \mathcal{D}$, donde cada $e_K : \mathcal{P} \to \mathcal{C}$ y $d_K : \mathcal{C} \to \mathcal{P}$ son funciones tales que $d_K(e_K(x)) = x$ para cada elemento de texto plano $x \in \mathcal{P}$.*

Podemos ver que la propiedad principal es la última, que dice que si se cifra un texto plano y luego se descifra con las funciones asociadas, entonces se obtiene exactamente el texto plano original.

El uso de un criptosistema es sencillo: las dos personas que lo utilizan eligen primero una clave aleatoria y la comunican a través de un canal seguro. Más adelante, cuando uno quiere comunicar un mensaje, cifra el texto plano y envía el texto cifrado obtenido a el otro, que descifrará el texto cifrado para obtener el texto plano original.

Es evidente que cada función de cifrado es una función inyectiva, de lo contrario, el descifrado no podría realizarse de forma inequívoca.

Si los conjuntos, los textos planos y los textos cifrados son idénticos, entonces cada función de cifrado es simplemente una permutación.

Para que un criptosistema sea de utilidad práctica, debe satisfacer ciertas propiedades informales:

1. Cada función de cifrado y cada función de descifrado deben ser eficientemente computables.
2. Nadie debe ser capaz de determinar la clave o el texto plano al ver el texto cifrado.

La segunda idea, la seguridad, es que nadie debe ser capaz de realizar un criptoanálisis, es decir, de intentar calcular la clave dada una cadena de texto cifrado.

A partir de ahora, nos centraremos en criptosistemas sobre el alfabeto, y entonces consideraremos que $\mathcal{P}$ y $\mathcal{C}$ son $\mathbb{Z}_{26}$ o $\mathbb{Z}_{26}^m$.

Además, utilizaremos letras minúsculas para el texto plano y mayúsculas para el texto cifrado, con el fin de mejorar la legibilidad.

3.1.1. Cifrado por desplazamiento

Criptosistema 1 (cifrado por desplazamiento). *Sea* $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. *Para cada* $K \in \mathcal{K}$, *definimos:*

$$e_K(x) = x + K \quad \text{mód} \quad 26; \quad d_K(y) = y - K \quad \text{mód} \quad 26.$$

Como curiosidad, se sabe que para la clave particular $K = 3$, el criptosistema se llama a menudo el cifrado César debido a su uso reportado por Julio César.

Este criptosistema es realmente sencillo: a cada letra del alfabeto le corresponde un número módulo 26, al que añadimos la clave módulo 26 y obtenemos la letra asociada. Ciframos todo el mensaje aplicando esto a cada una de las letras. Para descifrar, hacemos lo mismo, pero tomamos la diferencia en lugar de sumar.

Ejemplo 1: Tomamos la clave $K = 11$, y el texto plano:

$$wewillmeetatmidnight.$$

Primero convertimos el texto plano en sus números asociados en $\mathbb{Z}_{26}$:

$$22, 4, 22, 8, 11, 11, 12, 4, 4, 19, 0, 19, 12, 8, 3, 13, 8, 6, 7, 19.$$

Ahora añadimos la clave a cada valor en $\mathbb{Z}_{26}$:

$$7, 15, 7, 19, 22, 22, 23, 15, 15, 4, 11, 4, 23, 19, 14, 24, 19, 17, 18, 4.$$

Por último, convertimos estos números en letras y obtenemos el cibertexto:

$$HPHTWWXPPELEXTOYTRSE.$$

Sin embargo, este criptosistema no es seguro ya que puede ser criptoanalizado fácilmente mediante la búsqueda exhaustiva de claves, es decir, buscando con todas las claves posibles. Como sólo hay 26 claves, podemos probar todas las reglas de descifrado posibles hasta obtener un texto plano significativo. Además, se puede demostrar que sólo se necesita una media de 13 intentos hasta que encontramos la clave y la regla de descifrado adecuadas. Lo vemos en el siguiente ejemplo:

Ejemplo 2: Nos dan el texto cifrado:

$$JBCRCLQRWCRVNBJENBWRWN.$$

Probamos sucesivamente las claves de descifrado, donde obtenemos lo siguiente:

$$jbcrclqrwcrvnbjenbwrwn$$

$$iabqbkpqvbqumaidmavqvm$$

$$hzapajopuaptlzhclzupul$$

$$gyzozinotzoskygbkytotk$$

$$fxynyhmnsynrjxfajxsnsj$$

$$ewxmxglmrxmqiweziwrmri$$

$$dvwlwfklqwlphvdyhvqlqh$$

$$cuvkvejkpvkogucxgupkpg$$

$$btujudijoujnftbwftojof$$

$$astitchintimesavesnine.$$

De aquí podemos deducir que el texto plano es la frase *a stitch in time saves nine*, y entonces podemos parar. La clave es $K = 9$ [53].

De aquí podemos deducir que el espacio de claves debe ser muy grande para mejorar la inviabilidad de una búsqueda exhaustiva de claves. Sin embargo, un espacio de claves grande no es suficiente para garantizar la seguridad.

3.1.2. El cifrado afín

Criptosistema 2 (Cifrado afín). *Sea* $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$*, y sea*

$$\mathcal{K} = \{(a, b) \in \mathbb{Z}_{26} : \gcd(a, 26) = 1\}.$$

Para $K = (a, b) \in \mathcal{K}$*, definimos:*

$$e_K(x) = ax + b \quad \text{mód} \quad 26; \quad d_K(y) = a^{-1}(y - b) \quad \text{mód} \quad 26.$$

Podemos ver claramente que si $a = 1$, tenemos el cifrado por desplazamiento.

La condición de que a y 26 sean coprimos es necesaria, ya que si a no fuera coprimo de 26, entonces la función de cifrado no sería necesariamente inyectiva, y no podríamos garantizar la existencia de a^{-1} para la función de descifrado.

Se puede demostrar que el número de claves posibles cuando se utiliza un alfabeto de m letras es $m\varphi(m)$, donde φ es la función *phi de Euler*. En particular, para $m = 26$ tenemos 312 claves posibles.

Ejemplo 3: Supongamos que $K = (7, 3)$. Tenemos el texto cifrado AXG. Primero traducimos las letras a residuos módulo 26, que son 0, 24 y 6. A continuación, el inverso multiplicativo de 7 es 15 en $\mathbb{Z}_{26}$, por lo que tenemos que $d_K(y) = 15(y - 3) = 15y - 19$. Aplicando esto a los números anteriores, obtenemos 7, 14 y 19, que se traducen en el texto plano *hot* [53].

Nótese que hemos mejorado el tamaño del conjunto de claves, ya que $m\varphi(m) \geq m$. Sin embargo, este número también es menor del que necesitaríamos.

3.1.3. Cifrado por sustitución

Criptosistema 3 (Cifrado por sustitución). *Sean $\mathcal{P}$ y $\mathcal{C}$ las 26 letras del alfabeto. $\mathcal{K}$ consiste en todas las permutaciones posibles de las letras. Para cada permutación $\pi \in \mathcal{K}$, definimos*

$$e_\pi(x) = \pi(x); \quad d_\pi(y) = i^{-1}(y).$$

Nótese que podemos utilizar este sistema para los números en $\mathbb{Z}_{26}$ en lugar de utilizarlo para las letras.

De la misma manera que el cifrado afín incluía el cifrado de desplazamiento, podemos ver el cifrado afín (y luego el cifrado de desplazamiento) como un caso particular del cifrado por sustitución, utilizando sólo $m\varphi(m)$ permutaciones de las 26! posibles.

Presentamos el siguiente ejemplo de uso del cifrado por sustitución.

Ejemplo 4: Sabemos que nuestra clave π es la permutación dada por:

a	b	c	d	e	f	g	h	i	j	k	l	m
X	N	Y	A	H	P	O	G	Z	Q	W	B	T
n	o	p	q	r	s	t	u	v	w	x	y	z
S	F	L	R	C	V	M	U	E	K	J	D	I

Ahora la función de descifrado es la permutación inversa, que se forma escribiendo primero las segundas líneas y luego ordenándolas por orden alfabético. Obtenemos:

A	B	C	D	E	F	G	H	I	J	K	L	M
d	l	r	y	v	o	h	e	z	x	w	p	t
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	g	f	j	q	n	m	u	s	k	a	c	i

Supongamos que tenemos el siguiente mensaje:

$$MGZVYZLGHCMHJMYXSSFMNHAHYCDLMHA.$$

Podemos descifrarlo en:

$$thisciphertextcannotbedecrypted.$$

Donde obtenemos la frase *this ciphertext cannot be decrypted* [53].

Observamos que el número de permutaciones posibles (es decir, el número de claves) es de 26!, un número muy grande. Por tanto, una búsqueda exhaustiva de claves es inviable incluso para un ordenador, debido al tamaño del número. Sin embargo, más adelante veremos que un cifrado de sustitución puede ser criptoanalizado por otros métodos.

3.1.4. Cifrado Vigenère

Criptosistema 4 (Cifrado de Vigenére). Sea $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}^m$. Para una clave $K = (k_1, \ldots, k_m)$, definimos

$$e_K(x_1, \ldots, x_m) = (x_1{+}k_1, \ldots, x_m{+}k_m); \quad d_K(y_1, \ldots, y_m) = (y_1{-}k_1, \ldots, y_m{-}k_m).$$

Ejemplo 5: Si tomamos la palabra CIPHER, cuyo equivalente numérico es $K = (2, 8, 15, 7, 4, 17)$ y codificamos la cadena:

$$thiscryptosystemisnotsecure.$$

Convertimos el texto plano en residuos módulo 26, los escribimos en grupos de seis y añadimos la palabra clave. Obtenemos lo siguiente:

$$VPXZGIAXIVWPUBTTMJPWIZITWZT.$$

Tenga en cuenta que el número de palabras clave posibles en este caso es de 26^m, por lo que incluso para valores pequeños de m, una búsqueda exhaustiva de claves requeriría mucho tiempo. Utilizando $m = 2$ ya tenemos

676 claves posibles, mientras que $m = 3$ significa 17576 claves posibles, lo suficientemente grande como para evitar la búsqueda exhaustiva de claves a mano.

En general, cuando consideramos una palabra clave de longitud m, un solo carácter alfabético puede ser codificado en m posibles caracteres alfabéticos, si la palabra clave tiene m caracteres distintos, lo que no es el mismo caso que los criptosistemas anteriores, que codificaban un solo carácter alfabético siempre en el mismo carácter alfabético. Este tipo de criptosistemas, como el cifrado de Vigenère, se denominan criptosistemas polialfabéticos, mientras que los otros, como el cifrado de sustitución, se denominan criptosistemas monoalfabéticos. Como veremos más adelante, un criptosistema polialfabético es menos vulnerable a algunos tipos de ataques.

3.1.5. Cifrado por permutación

Criptosistema 5 (Cifrado por Permutación). *Sea* $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^m$, *y sea* $\mathcal{K} = \sum_m$. *Para cada clave* π *(es decir, para cada permutación), definimos*

$$e_i(x_1, \ldots, x_m) = (x_{\pi(1)}, \ldots, x_{\pi(m)}); \quad d_\pi(y_1, \ldots, y_m) = (y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(m)}).$$

Al igual que con el cifrado por sustitución, es más conveniente utilizar caracteres alfabéticos que residuos. Obsérvese que los cifrados de sustitución y de permutación parecen similares, pero el cifrado de permutación permuta las letras utilizadas, y el cifrado de sustitución permuta la posición de las letras.

Ejemplo 6: Tenemos el siguiente texto cifrado:

$$EESLSHSALSESLSHBLEHSYEETHRAEOS.$$

Este texto cifrado ha sido codificado con la clave

x	1	2	3	4	5	6
$\pi(x)$	3	5	1	6	4	2

Primero obtenemos la permutación inversa:

x	1	2	3	4	5	6
$\pi^{-1}(x)$	3	6	1	5	2	4

Ahora dividimos el texto cifrado en grupos de seis letras:

$$EESLSH \quad | \quad SALSES \quad | \quad LSHBLE \quad | \quad HSYEET \quad | \quad HRAEOS$$

Reorganizamos cada grupo según la permutación $\pi - 1$, obteniendo lo siguiente:

$$shesel \quad | \quad lsseas \quad | \quad hellsb \quad | \quad ythese \quad | \quad ashore$$

Entonces, el texto plano es:

$$shesellsseashellsbytheseashore.$$

3.1.6. Cifrado Hill

Criptosistema 6 (Cifrado Hill). *Sea $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}^m$ y sea $\mathcal{K}$ el conjunto de matrices $m \times m$ invertibles sobre Z_{26}. Entonces, para una clave K, definimos*

$$e_K(x) = xK; \quad d_K(y) = yK^{-1}.$$

Ejemplo 7: Supongamos que hemos encriptado alguna palabra y que hemos obtenido $DELW$, y que sabemos:

$$K = \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}$$

En primer lugar, observamos que tenemos dos elementos que descifrar: $(3, 4)$, correspondiente a DE, y $(11, 22)$, correspondiente a LW. En primer lugar, obtenemos la inversa de K, es decir:

$$K^{-1} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}$$

A partir de aquí, podemos obtener:

$$\begin{pmatrix}3 & 4\end{pmatrix}\begin{pmatrix}7 & 18\\ 23 & 11\end{pmatrix} = \begin{pmatrix}9 & 20\end{pmatrix} \qquad \begin{pmatrix}11 & 22\end{pmatrix}\begin{pmatrix}7 & 18\\ 23 & 11\end{pmatrix} = \begin{pmatrix}11 & 24\end{pmatrix}$$

Es decir, hemos obtenido la palabra cuya representación numérica es $(9, 20, 11, 24)$, que corresponde a *july* [53].

En este criptosistema, cada letra se cifra utilizando una combinación de todas las letras del texto plano, a diferencia de los métodos anteriores.

Dada una permutación en $\sum_m$, podemos considerar su matriz de permutación asociada, y es fácil comprobar que el cifrado de Hill usando esta matriz es, de hecho, equivalente al cifrado de permutación usando la permutación. Es decir, el cifrado de Hill incluye el cifrado de permutación.

3.1.7. Cifrados de flujo

En los criptosistemas estudiados hasta ahora, los elementos sucesivos del texto plano se cifran utilizando la misma clave K, es decir

$$y = e_K(x_1)e_K(x_2)\dots.$$

Estos sistemas suelen denominarse cifrados en bloque.

Un enfoque alternativo consiste en utilizar cifradores de flujo, que generan un flujo de claves $z = z_1z_2\dots$, y lo utilizan para cifrar según la regla

$$y = e_{z_1}(x_1)e_{z_2}(x_2)\dots.$$

El tipo más sencillo de cifrado de flujo es aquel en el que el flujo de claves se construye a partir de la clave, independientemente de la cadena de texto plano, utilizando algún algoritmo específico. Este tipo puede definirse formalmente como sigue:

Definición 3.2: *Un cifrado de flujo síncrono es una tupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{E}, \mathcal{D})$ *junto con una función* g*, tal que se satisfacen las siguientes condiciones*

1. *El conjunto* $\mathcal{P}$ *es un conjunto finito de posibles textos planos.*
2. *El conjunto* $\mathcal{C}$ *es un conjunto finito de posibles textos cifrados.*
3. *El espacio de claves* $\mathcal{K}$ *es un conjunto finito de claves posibles.*
4. *El conjunto* $\mathcal{L}$ *es un conjunto finito llamado alfabeto del flujo de claves.*

5. *La función g es el generador del flujo de claves, que toma una clave K como entrada y genera una cadena infinita $z_1 z_2 \ldots$ llamada flujo de claves, donde $z_i \in \mathcal{L}$.*

6. *Para cada $z \in \mathcal{L}$, existe una regla de cifrado $e_z \in \mathcal{E}$ y una regla de descifrado correspondiente $d_z \in \mathcal{D}$, donde cada $e_z : \mathcal{P} \to \mathcal{C}$ y $d_z : \mathcal{C} \to \mathcal{P}$ son funciones tales que $d_z(e_z(x)) = x$ para cada elemento de texto plano $x \in \mathcal{P}$.*

Podemos pensar en un cifrado en bloque como un caso especial de un cifrado de flujo en el que el flujo de claves es constante.

Un cifrado de flujo es un cifrado de flujo periódico con período d si $z_{i+d} = z_i$.

El cifrado de Vigenére con una longitud de palabra clave m puede considerarse como un cifrado de flujo periódico con período m.

Tenemos otros métodos para generar flujos de claves sincrónicos. Trabajaremos con alfabetos binarios, pero esto se puede generalizar. Comenzamos con una m-tupla binaria $(k_1, \ldots, k_m)$ y definimos $z_i = k_i$ para $i \leq m$. Ahora generamos el flujo de claves utilizando una recurrencia lineal:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \qquad \text{mód} \quad 2.$$

Se dice que esta recurrencia tiene grado m ya que cada término depende de los m términos anteriores. Es lineal porque está definida por una función lineal de los términos anteriores. Además, podemos tomar $c_0 = 1$ sin pérdida de generalidad, ya que de lo contrario la recurrencia será de grado (como máximo) $m - 1$.

Si las constantes $c_0, \ldots, c_{m-1}$ se eligen de forma adecuada, entonces cualquier vector de inicialización de claves que no sea cero dará un flujo de claves periódico que tenga un periodo de $2^m - 1$, por lo que una clave corta puede dar un flujo de claves con un periodo muy largo. Esta es, sin duda, una propiedad deseable: más adelante veremos cómo se puede criptoanalizar el cifrado de Vigenère explotando el hecho de que el flujo de claves tiene un período corto.

Ejemplo 8: Si inicializamos el flujo de claves como $(1, 0, 0, 0)$, y tomamos la recurrencia:

$$z_{i+4} = z_i + z_{i+2} \qquad \text{mód} \quad 2.$$

Obtenemos un flujo de claves de período 15, que es:

$$100010011010111\ldots.$$

Cualquier otro vector de inicialización distinto de cero dará una permutación cíclica del mismo flujo de claves.

Otro aspecto atractivo de este método es que puede producirse eficientemente en hardware, utilizando un registro de desplazamiento de retroalimentación lineal, o $LFSR$. En un momento dado, el registro de desplazamiento contiene m elementos consecutivos del flujo de claves, digamos $z_i, \ldots, z_{i+m-1}$. Después de una unidad de tiempo, el registro de desplazamiento contiene $z_{i+1}, \ldots, z_{i+m}$ al cambiar los valores. Un cifrado de flujo no sincrónico es un cifrado de flujo en el que cada elemento del flujo de claves depende de los elementos anteriores del texto plano y/o del texto cifrado, así como de la clave K.

Criptosistema 7 (Cifrado de autoclave). *Sea $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathcal{L} = \mathbb{Z}_{26}$. Sea $z_1 = K$, y definamos $z_i = x_{i-1}$ para $i \geq 2$. Entonces, definimos*

$$e_z(x) = x + z \quad \text{mód} \quad 26; \quad d_z(y) = y - z \quad \text{mód} \quad 26.$$

Por supuesto, el cifrado Autokey es inseguro, ya que sólo hay 26 claves posibles.

Ejemplo 9: Supongamos que empezamos con la clave K, y tenemos el texto cifrado $ZVRQHDUJIM$, que corresponde a la cadena numérica:

$$(25, 21, 17, 16, 7, 3, 20, 9, 8, 12).$$

Entonces, calculamos:

$$\begin{aligned}
x_1 = d_8(25) = 25 - 8 \quad &\text{mód} \quad 26 = 17 \\
x_2 = d_{17}(21) = 21 - 17 \quad &\text{mód} \quad 26 = 4 \\
x_3 = d_4(17) = 17 - 4 \quad &\text{mód} \quad 26 = 13 \\
x_4 = d_{13}(16) = 16 - 13 \quad &\text{mód} \quad 26 = 3 \\
x_5 = d_3(7) = 7 - 3 \quad &\text{mód} \quad 26 = 4 \\
x_6 = d_4(3) = 3 - 4 \quad &\text{mód} \quad 26 = 25 \\
x_7 = d_{25}(20) = 20 - 25 \quad &\text{mód} \quad 26 = 21
\end{aligned}$$

$$x_8 = d_{21}(9) = 9 - 21 \quad \text{mód} \quad 26 = 14$$
$$x_9 = d_{14}(8) = 8 - 14 \quad \text{mód} \quad 26 = 20$$
$$x_{10} = d_{20}(18) = 18 - 20 \quad \text{mód} \quad 26 = 18$$

La cadena numérica obtenida es:

$$(17, 4, 13, 3, 4, 25, 21, 14, 20, 18).$$

A continuación, obtenemos el texto plano *rendezvous* [53].

3.2. Criptoanálisis de sistemas clásicos

A continuación, discutiremos algunas técnicas de criptoanálisis. El supuesto general que se suele hacer es que el adversario conoce el criptosistema utilizado.

En primer lugar, diferenciamos los modelos de ataque, que especifican la información de la que dispone el adversario.

- Ataque de sólo texto cifrado: el adversario posee una cadena de texto cifrado.
- Ataque de texto plano conocido: el adversario posee una cadena de texto plano y el correspondiente texto cifrado.
- Ataque de texto plano elegido: El adversario puede construir el texto cifrado para cada cadena de texto plano.
- Ataque de texto cifrado elegido: El adversario puede construir el texto plano para cada cadena de texto cifrado.

En cada caso, el objetivo es determinar la clave utilizada, ya que esto permitiría descifrar cualquier cadena de texto cifrado adicional cifrada con la misma clave.

Consideraremos que la cadena de texto plano es un texto inglés ordinario, sin puntuación ni espacios, ya que esto facilitaría el criptoanálisis.

En primer lugar, consideraremos un ataque de sólo texto cifrado. Muchas técnicas de criptoanálisis utilizan propiedades estadísticas de la lengua inglesa, a saber, las frecuencias relativas estimadas de las 26 letras del alfabeto. Se pueden dividir en cinco grupos, según sus probabilidades.

También es útil considerar las secuencias de dos o tres letras consecutivas, llamadas digramas y trigramas.

3.2.1. Criptoanálisis del cifrado afín

La técnica para descubrir la clave del cifrado afín es bastante sencilla: dado un texto cifrado, comparamos el análisis de frecuencia del texto cifrado con la frecuencia del alfabeto. A continuación, formulamos una hipótesis sobre las posibles codificaciones de las letras, y los caracteres más frecuentes del texto cifrado se corresponden con las letras más frecuentes del alfabeto. A continuación, expresamos esto numéricamente e intentamos resolver $K = (a, b)$. Recordemos que $\gcd(a, b) = 1$ es una condición necesaria. Hacemos conjeturas hasta obtener la clave, es decir, la que produce un texto plano aparentemente válido al descifrar el texto cifrado.

3.2.2. Criptoanálisis del cifrado por sustitución

La técnica utilizada para descubrir la clave y el texto plano de un texto cifrado con el cifrado por sustitución es similar a la utilizada en el cifrado afín: comparamos las frecuencias de las letras del texto cifrado y del alfabeto inglés, y hacemos conjeturas mientras intentamos obtener un texto plano que tenga sentido, obteniendo la clave en el proceso.

3.2.3. Criproanálisis del cifrado Vigenère

Comentamos sin detallar y muy brevemente algunas técnicas relevantes.

La primera es la prueba de *Kasiski*. Consiste en buscar en el texto cifrado pares de segmentos idénticos de una longitud mínima de tres, y registrar la distancia entre las posiciones iniciales de los dos segmentos. Si se obtienen varias distancias de este tipo $\delta_1, \delta_2, \ldots$, se conjetura que la longitud m de las palabras y clave en este apartado divide todas las distancias δ_i, y entonces m divide el máximo común divisor de las distancias δ_i.

Una técnica mejor usa el *índice de coincidencia* para obtener el valor m.

Definición 3.3: *Sea $x = x_1 \ldots x_n$ una cadena de n caracteres alfabéticos. El índice de coincidencia de x, denotado $I_C(x)$, se define como la probabilidad de que dos elementos aleatorios de x sean idénticos.*

Se espera que el índice de coincidencia esté entre 0,065 (donde diremos que la longitud de la palabra clave es correcta) y 0,038 (donde diremos que la longitud de la palabra clave no es correcta).

Una vez que hayamos determinado la longitud correcta de la palabra clave siguiendo el procedimiento mostrado en [53], dividimos en grupos de m

letras y tomamos la primera de cada grupo, la segunda de cada grupo, y así sucesivamente, hasta m.

3.2.4. Criptoanálisis del cifrado Hill

El cifrado Hill puede ser difícil de romper con un ataque de sólo texto cifrado, pero sucumbe fácilmente a un ataque de texto plano conocido. Primero determinamos el valor de m fácilmente probando $m = 2, 3, \ldots$. A continuación, para un determinado m, tomamos m pares distintos de texto plano y texto cifrado, los colocamos por filas en las matrices $m \times m$ X e Y, y luego calculamos K como $X^{-1}Y$. Si X no es invertible, probamos con otros conjuntos de pares de texto plano y texto cifrado.

3.2.5. Criptoanálisis del cifrado de flujo LFSR

Volvemos a utilizar el ataque de texto plano conocido, calculando $z_i = x_i + y_i \mod 2$ para cada $i \leq n$. Podemos escribir ell flujo de claves con la recurrencia lineal presentada en el apartado de los criptosistemas de flujo:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \qquad \text{mód} \quad 2.$$

Si $n \leq 2m$, entonces podemos escribir este sistema en forma de matriz, siendo Z la matriz de z's. Si esta matriz tiene una inversa, entonces m es el grado de la recurrencia utilizada para generar el flujo de claves y $c_0, \ldots, c_{m-1}$ se puede obtener multiplicando $z_{m+1}, \ldots, z_{2m}$ con Z^{-1}.

3.3. Algoritmos útiles de clave privada

Los criptosistemas anteriores son la base de criptosistemas más avanzados que se utilizan hoy en día. Veamos ahora los dos más importantes.

3.3.1. Algoritmo DES

El cifrado de datos (DES) es un estándar criptográfico que se propuso como algoritmo para intercambiar información de manera segura y secreta en 1970 y fue adoptado como estándar federal estadounidense por la Oficina Nacional de Estándares (NBS) en 1973. Fue desarrollado por IBM en la

década de 1970, pero más tarde fue adoptado por el gobierno de EE.UU. como estándar nacional.

El DES es un cifrado por bloques, lo que significa que durante el proceso de cifrado, el texto plano se divide en bloques de longitud fija y cada bloque se cifra al mismo tiempo. Básicamente, toma un texto plano de entrada de 64 bits y una clave de 64 bits (sólo se utilizan 56 bits para la conversión y el resto para la comprobación de la paridad) y produce un texto cifrado de 64 bits mediante el cifrado, que puede descifrarse de nuevo para obtener el mensaje utilizando la misma clave [37]. El algoritmo DES se muestra en la Fig. 1.

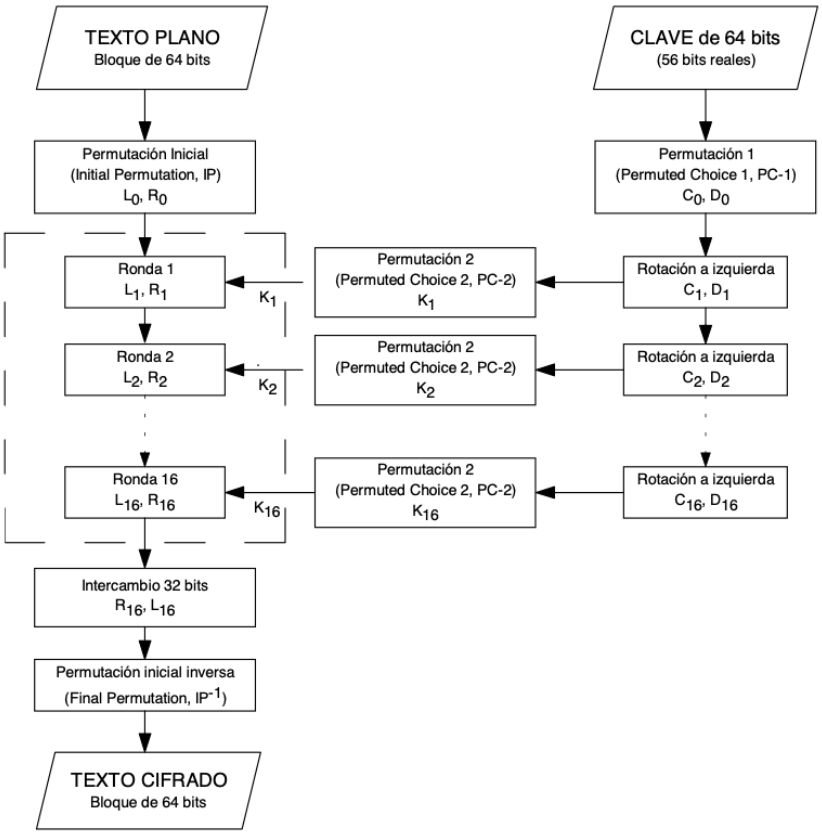

Figura 1: Esquema del algoritmo DES.

El algoritmo emplea tres tipos diferentes de operaciones: Permutaciones, rotaciones y sustituciones. Entre las transposiciones iniciales y finales, el al-

goritmo realiza 16 iteraciones de una función.

DES utiliza una clave de 56 bits. De hecho, la clave de 56 bits se divide en ocho bloques de 7 bits y se añade un octavo bit de paridad impar a cada bloque, es decir, se añade un "0" o un "1" al bloque para que haya un número impar de bits "1" en cada bloque de 8 bits [27]. Al utilizar los 8 bits de paridad para la detección rudimentaria de errores, una clave DES tiene en realidad 64 bits de longitud a efectos computacionales, aunque sólo tiene 56 bits de aleatoriedad o entropía.

El DES actúa entonces sobre bloques de 64 bits del texto plano, invocando 16 rondas de permutaciones, intercambios y sustituciones, como se muestra en la figura. La norma incluye tablas que describen todas las operaciones de selección, permutación y expansión que se mencionan a continuación; estos aspectos del algoritmo no son secretos [52]. Los pasos básicos del DES son:

1. El bloque de 64 bits que se va a cifrar se somete a una permutación inicial (IP), en la que cada bit se desplaza a una nueva posición de bit; por ejemplo, los bits 1, 2 y 3 se desplazan a la posición 58, 50 y 42, respectivamente.

2. La entrada permutada de 64 bits se divide en dos bloques de 32 bits, llamados izquierdo y derecho, respectivamente. Los valores iniciales de los bloques izquierdo y derecho se denominan L_0 y R_0.

3. A continuación, se realizan 16 rondas de operaciones sobre los bloques L y R. Durante cada iteración (donde n va de 1 a 16), se aplican las siguientes fórmulas:

$$L_n = R_{n-1},$$
$$R_n = L_{n-1} XOR(R_{n-1}, K_n).$$

En cualquier paso del proceso, el nuevo valor del bloque L se toma simplemente del valor del bloque R anterior. El nuevo bloque R se calcula tomando el bit-OR exclusivo (XOR) del bloque L anterior con los resultados de aplicar la función de cifrado DES, f, al bloque R anterior y a K_n. K_n es un valor de 48 bits derivado de la clave DES de 64 bits. En rondas sucesivas, ambas mitades se rotan hacia la izquierda en uno o dos bits (especificados para cada ronda), y luego se seleccionan 48 bits de clave de ronda mediante la elección permutada 2 (PC-2): 24 bits de la mitad izquierda y 24 de la derecha. Las

rotaciones tienen el efecto de que se utiliza un conjunto diferente de bits en cada clave redonda; cada bit se utiliza en aproximadamente 14 de las 16 claves redondas.

La función de cifrado, f, combina el valor del bloque R de 32 bits y la subclave de 48 bits de la siguiente manera. En primer lugar, los 32 bits del bloque R se expanden a 48 bits mediante una función de expansión (E); los 16 bits adicionales se encuentran repitiendo los bits en 16 posiciones predefinidas. A continuación, el bloque R expandido de 48 bits se cruza con la subclave de 48 bits [52]. El resultado es un valor de 48 bits que se divide en ocho bloques de 6 bits. Estos se introducen como entrada en 8 cajas de selección (S), denotadas $S_1, S_2, \ldots, S_8$.

Cada entrada de 6 bits produce una salida de 4 bits utilizando una tabla de búsqueda basada en las 64 entradas posibles; esto da como resultado una salida de 32 bits de la caja S. A continuación, los 32 bits se reordenan mediante una función de permutación (P), produciendo los resultados de la función de cifrado.

Los resultados de la última ronda del DES -es decir, L_{16} y R_{16}- se recombinan en un valor de 64 bits y se introducen en una permutación inicial inversa ($IP - 1$). En este paso, los bits se reorganizan en sus posiciones originales, de modo que los bits 58, 50 y 42, por ejemplo, se vuelven a colocar en las posiciones 1, 2 y 3, respectivamente. La salida de $IP - 1$ es el bloque de texto cifrado de 64 bits.

3.3.2. Algoritmo AES

El algoritmo Advanced Encryption Standard (AES) es uno de los algoritmos de cifrado por bloques que fue publicado por el National Institute of Standards and Technology (NIST) en el año 2000. El objetivo principal de este algoritmo era sustituir al algoritmo DES tras aparecer algunos aspectos vulnerables del mismo. El NIST invitó a los expertos que trabajan en encriptación y seguridad de datos de todo el mundo a presentar un innovador algoritmo de cifrado por bloques para cifrar y descifrar datos con una estructura potente y compleja.

Muchos grupos de todo el mundo presentaron sus algoritmos. El NIST aceptó cinco algoritmos para su evaluación. Tras aplicar varios criterios y parámetros de seguridad, seleccionaron uno de los cinco algoritmos de cifrado que propusieron dos criptógrafos belgas, Joan Daeman y Vincent Rijmen. El nombre original del algoritmo AES es el algoritmo Rijndel. Sin embargo, este

nombre no se ha convertido en un nombre popular para este algoritmo, sino que se reconoce como algoritmo Advanced Encryption Standard (AES) en todo el mundo [5].

AES es un cifrado iterativo en lugar de Feistel. Se basa en dos técnicas comunes para cifrar y descifrar datos conocidas como red de sustitución y permutación (SPN). La SPN es una serie de operaciones matemáticas que se llevan a cabo en los algoritmos de cifrado por bloques [35]. AES tiene la capacidad de tratar con 128 bits (16 bytes) como tamaño de bloque de texto plano fijo. Estos 16 bytes se representan en una matriz de 4x4 y AES opera sobre una matriz de bytes. Además, otra característica crucial de AES es el número de rondas. El número de rondas depende de la longitud de la clave. El algoritmo AES utiliza tres tamaños de clave diferentes para cifrar y descifrar datos (128, 192 o 256 bits). Los tamaños de las claves deciden el número de rondas, por ejemplo, AES utiliza 10 rondas para claves de 128 bits, 12 rondas para claves de 192 bits y 14 rondas para claves de 256 bits [41].

El cifrado es una técnica muy popular que desempeña un papel importante para proteger los datos de los intrusos. El algoritmo AES utiliza una estructura particular para cifrar los datos y proporcionar la mejor seguridad. Para ello, se basa en una serie de rondas y cada una de ellas consta de cuatro subprocesos. Cada ronda consta de los siguientes cuatro pasos para cifrar un bloque de 128 bits.

Transformación de bytes sustitutivos

La primera etapa de cada ronda comienza con la transformación SubBytes. Esta etapa depende de la caja S no lineal para sustituir un byte del estado por otro byte. De acuerdo con los principios de difusión y confusión de Shannon para el diseño de algoritmos criptográficos, tiene un papel importante para obtener mucha más seguridad [23]. Por ejemplo, en AES si tenemos hexa 53 en los datos de entrada (también denominados estados), tiene que ser reemplazado por hexa ED. ED se crea a partir de la intersección de 5 y 3. Para el resto de bytes del estado hay que realizar estas operaciones.

Transformación ShiftRows

El siguiente paso después de SubByte que se realiza sobre el estado es ShiftRow. La idea principal de este paso es desplazar los bytes del estado

cíclicamente hacia la izquierda en cada fila. En este proceso los bytes de la fila número cero permanecen y no realizan ninguna permutación. En la primera fila sólo se desplaza un byte circularmente hacia la izquierda. La segunda fila se desplaza dos bytes a la izquierda. La última fila se desplaza tres bytes a la izquierda [47]. El tamaño del nuevo estado no se cambia y sigue siendo el mismo tamaño original de 16 bytes, pero se cambia la posición de los bytes en el estado como se ilustra en la Fig. 2.

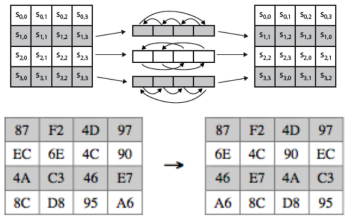

Figura 2: Ejemplo de Transformación ShiftRow.

Transformación MixColumns

Otro paso crucial que le ocurre al estado es MixColumn. La multiplicación se lleva a cabo fuera del estado. Cada byte de una fila en la transformación matricial se multiplica por cada valor (byte) de la columna del estado. En otras palabras, cada fila de la transformación matricial debe multiplicarse por cada columna del estado. Los resultados de estas multiplicaciones se utilizan con XOR para producir un nuevo cuatro bytes para el siguiente estado. En este paso el tamaño del estado no se cambia, es decir se mantuvo con el tamaño original 4x4 como se muestra en la Fig. 3.

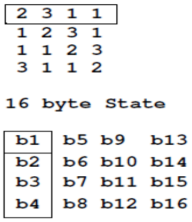

Figura 3: Matriz Multiplicación.

$$b1 = (b1 \cdot 2)XOR(b2 \cdot 3)XOR(b3 \cdot 1)XOR(b4 \cdot 1).$$

Y así sucesivamente hasta que se agoten todas las columnas del estado [5].

Transformación AddRoundKey

AddRoundKey es la etapa más importante del algoritmo AES. Tanto la clave como los datos de entrada se estructuran en una matriz de 4x4 bytes [28]. La Fig. 4 muestra cómo se distribuyen la clave de 128 bits y los datos de entrada en las matrices de bytes. AddRoundKey tiene la capacidad de proporcionar mucha más seguridad durante el cifrado de datos. Esta operación se basa en crear la relación entre la clave y el texto cifrado. El texto cifrado proviene de la etapa anterior. La salida de AddRoundKey se basa exactamente en la clave indicada por los usuarios [4]. Además, en la etapa también se utiliza la subclave y se combina con el estado. La clave principal se utiliza para derivar la subclave en cada ronda mediante el programa de claves de Rijndael. El tamaño de la subclave y del estado es el mismo. La subclave se añade combinando cada byte del estado con el byte correspondiente de la subclave utilizando XOR a nivel de bits [29].

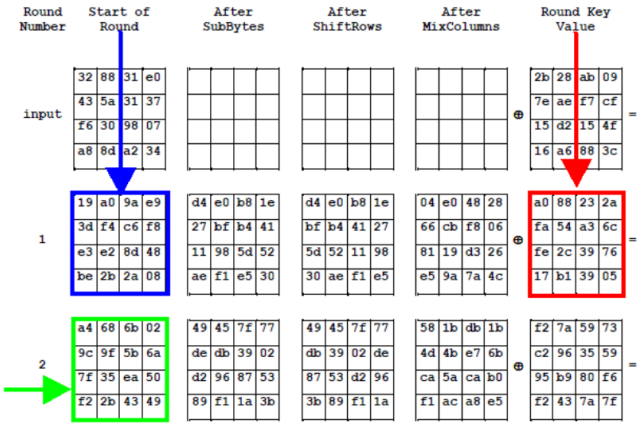

Figura 4: Trasnformación Add Round Key.

4. Cuerpos Finitos

Los cuerpos finitos, también conocidos como cuerpos de Galois, son la piedra angular para entender la mayor parte de los métodos criptográficos usados a día de hoy. Estos conceptos serán muy útiles para comprender la criptografía de clave pública que intoduciremos más adelante.

4.1. Anillos y cuerpos

Definición 4.1: *Un anillo* $(R, +, *)$ *es un conjunto* R*, junto con dos operaciones binarias, denotadas por* $+$ *y* $*$*, tal que verifiquen:*

- R *es un grupo abeliano con respecto a* $+$*.*
- R *es cerrado bajo* $*$*.*
- $*$ *es asociativo, es decir* $(a * b) * c = a * (b * c)$ *para todo* $a, b, c \in R$*;*
- *la ley distributiva se sostiene, es decir, que para todo* $a, b, c \in R$ $(a * b) + (a * c)$ *and* $(b + c) * a = (b * a) + (c * a)$*.*

Típicamente, usamos 0 para denotar el elemento identidad del grupo abeliano R con respecto a la adición, y $-a$ para denotar el inverso aditivo de $a \in R$.

Definición 4.2:

- *Un anillo se llama anillo con identidad si el anillo tiene una identidad multiplicativa (normalmente denotada* e *o* 1*).*
- *Un anillo se llama conmutativo si* $*$ *es conmutativo.*
- *Un anillo se llama dominio integral si es un anillo conmutativo con identidad* $e \neq 0$ *en el cual* $ab = 0$ *implica que* $a = 0$ *o* $b = 0$ *(es decir, no hay divisores nulos).*
- *Un anillo se llama anillo de división si los elementos no nulos forman un grupo bajo* $*$*.*
- *Un anillo de división conmutativo se llama cuerpo.*

Teorema 4.3: *Todo dominio integral finito es un cuerpo.*

Demostración: Sea R un dominio integral finito, y sean sus elementos $r_1, r_2, \ldots, r_n$. Consideremos un elemento fijo no nulo $r \in R$. Entonces los productos $rr_1, rr_2, \ldots, rr_n$ deben ser distintos, ya que $rr_i = rr_j$ implicaría $(r_i - r_j) = 0$, y desde que $r \neq 0$ debemos tener $r_i - r_j = 0$, es decir, $r_i = r_j$. Así, estos productos son precisamente los n elementos de R. Cada elemento de R es de la forma rr_i; en particular, la identidad $e = rr_i$ para algún $1 \leq i \leq n$. Como R es conmutativo, también tenemos $r_i r = e$, y por tanto r_i es el inverso multiplicativo de r. Así, los elementos no nulos de R forman un grupo conmutativo, y R es un cuerpo.

Teorema 4.4: *$\mathbb{Z}/(p)$, el anillo de las clases de residuos de los enteros módulo el ideal principal generado por un primo p, es un cuerpo.*

Demostración: Por el teorema 4.3, basta con demostrar que $\mathbb{Z}/(p)$ es un dominio integral. Ahora bien, $[a][b] = [ab] = [0]$ si y sólo si $ab = kp$ para algun $k \in \mathbb{Z}$. Como p es primo, p divide a ab si y sólo si p divide a uno de los factores. Por lo tanto, o bien $[a] = [0]$ o bien $[b] = [0]$, por lo que $\mathbb{Z}/(p)$ no contiene divisores cero.

Definición 4.5: *Una aplicación $\varphi : R \to S$, siendo R y S anillos, se llama homomorfismo de anillos si para cualquier $a, b \in R$ tenemos que:*

$$\varphi(a + b) = \varphi(a) + \varphi(b) \quad y \quad \varphi(ab) = \varphi(a)\varphi(b).$$

Un homomorfismo de anillos preserva tanto $+$ como $*$ e induce un homomorfismo del grupo aditivo de R en el de S. Conceptos como núcleo e imagen se definen de forma análoga al caso de los grupos.
Tenemos una versión anular del Primer Teorema del Isomorfismo:

Teorema 4.6: *Si φ es un homomorfismo de anillos desde un anillo R hacia un anillo S, entonces el anillo factorial $R/ker\varphi$ y el anillo S son isomorfos por la aplicación:*

$$r + ker\varphi \to \varphi(r).$$

Podemos utilizar aplicaciones para transferir una estructura de un sistema algebraico a un conjunto sin estructura. Dado un anillo R, un conjunto S y una aplicación biyectiva $\varphi : R \to S$, podemos utilizar φ para definir una estructura de anillo sobre S que convierta φ en un isomorfismo. En concreto, para $s_1 = \varphi(r_1)$ y $s_2 = \varphi(r_2)$, definimos

$$s_1 + s_2 \quad \text{como} \quad \varphi(r_1 + r_2), \quad \text{y} \quad s_1 s_2 \quad \text{como} \quad \varphi(r_1 r_2).$$

Esto se llama la estructura de anillo inducida por φ; cualquier propiedad extra de R es heredada por S.

Esta idea nos permite obtener una representación más conveniente para los cuerpos finitos $\mathbb{Z}/(p)$.

Definición 4.7: *Para un primo p, sea $\mathbb{F}_p$ el conjunto $\{0, 1, \ldots, p-1\}$ de enteros, y sea $\varphi : \mathbb{Z}/(p) \to \mathbb{F}_p$ la aplicación definida por $\varphi([a]) = a$ para $a = 0, 1, \ldots, p-1$. Entonces $\mathbb{F}_p$ dotado de la estructura de campo inducida por φ es un cuerpo finito, llamado cuerpo de Galois de orden p.*

A partir de lo anterior, el mapeo φ se convierte en un isomorfismo, por lo que $\varphi([a]+[b]) = \varphi([a]) + \varphi([b])$ y $\varphi([a][b]) = \varphi([a])\varphi([b])$. El cuerpo finito $\mathbb{F}_p$ tiene el elemento cero 0, el elemento identidad 1 y su estructura es la de $\mathbb{Z}/(p)$. Por tanto, calcular con elementos de $\mathbb{F}_p$ significa ahora aritmética ordinaria de enteros con reducción módulo p.

Teorema 4.8: *Un anillo $R \neq \{0\}$ de característica positiva con una identidad y sin divisores cero debe tener característica prima.*

Demostración: Como R contiene elementos distintos de cero, R tiene característica $n \geq 2$. Si n no fuera primo, podríamos escribir $n = km$ con $k, m \in \mathbb{Z}$, $1 < k$, $m < n$. Entonces $0 = ne = (km)e = (ke)(me)$, por lo que o bien $ke = 0$ o bien $me = 0$, ya que R no tiene divisores nulos. Por tanto, o bien $kr = (ke)r = 0$ para todo $r \in R$ o bien $mr = (me)r = 0$ para todo $r \in R$, contradiciendo la definición de n como característica.

Corolario 4.9: *Un cuerpo finito tiene característica prima.*

Demostración: A partir del Teorema 4.8, sólo tenemos que demostrar que un cuerpo finito F tiene una característica positiva. Consideremos los múltiplos $e, 2e, 3e, \ldots$ de la identidad. Dado que F sólo contiene un número finito de elementos, deben existir enteros k y m con $1 \leq k < m$ tales que $ke = me$, es decir, $(k-m)e = 0$, y por tanto $(k-m)f = (k-m)ef = 0f = 0$ para todo $f \in F$ por lo que F tiene característica positiva. Y por tanto prima.

4.2. Conceptos de teoría de cuerpos

Definición 4.10: *Sea F un cuerpo. Un subconjunto K que es a su vez un cuerpo bajo las operaciones de F se llama subcuerpo de F . El cuerpo F se llama extensión de cuerpo de K. Si $K \neq F$, K se llama subcuerpo propio de F.*

Definición 4.10: *Un cuerpo que no contiene subcuerpos propios se llama cuerpo primo.*

Por ejemplo, $\mathbb{F}_p$ es un cuerpo primo, ya que cualquier subcuerpo debe contener los elementos 0 y 1, y como es cerrado por adición debe contener todos los demás elementos, es decir, debe ser el cuerpo entero.

5. Criptografía de Clave Pública

La criptografía simétrica era muy adecuada para organizaciones como los gobiernos, el ejército y las grandes empresas financieras que se dedicaban a la comunicación clasificada.

Con la difusión de más redes informáticas inseguras en las últimas décadas, apareció una verdadera necesidad de utilizar la criptografía a mayor escala. La clave simétrica resultó no ser práctica si se utilizaba la misma clave en un periodo largo de tiempo debido a los problemas que planteaba la gestión de claves. Esto dio lugar a los criptosistemas de clave pública.

Las propiedades más importantes del esquema de cifrado de clave pública son:

- Se utilizan diferentes claves para el cifrado y el descifrado. Esta es una propiedad que diferencia a este esquema del de encriptación simétrica.
- Cada receptor posee una clave de descifrado única, generalmente denominada clave privada.
- El receptor debe publicar una clave de cifrado, denominada clave pública.
- En este esquema es necesario garantizar la autenticidad de la clave pública para evitar que el adversario se haga pasar por el receptor. Por lo general, este tipo de criptosistema implica a un tercero de confianza que certifica que una clave pública concreta pertenece únicamente a una persona o entidad específica.
- El algoritmo de cifrado es lo suficientemente complejo como para impedir que el atacante deduzca el texto plano a partir del texto cifrado y la clave de cifrado (pública).
- Aunque las claves privadas y públicas están relacionadas matemáticamente, no es posible calcular la clave privada a partir de la clave

pública. De hecho, la parte inteligente de cualquier criptosistema de clave pública consiste en diseñar una relación entre dos claves.

Con estas ideas básicas aparecieron algunos criptosistemas como el denominado *RSA* o los *criptosistemas logarítmicos*, que repasaremos durante este texto. Como veremos durante la lectura de estos será crucial la utilización de números primos grandes y, por eso acabaremos el capítulo mostrando algunos test de primalidad que serán utiles para verificar si estamos trabajando verdaderamente con números primos.

5.1. El criptosistema RSA

En 1978, Ron Rivest, Adi Shamir y Leonard Adleman introdujeron un algoritmo criptográfico que debía sustituir al algoritmo menos seguro del National Bureau of Standards (NBS). Cabe mencionar que RSA implementa un criptosistema de clave pública, así como firmas digitales. Además RSA está realmente motivado por los trabajos publicados por Diffie y Hellman [12] varios años antes, quienes describieron la idea de un algoritmo de este tipo, pero nunca lo desarrollaron realmente.

Introducido en el momento en que se esperaba que pronto surgiera la era del correo electrónico, RSA implementó dos ideas importantes: La criptografía de clave pública y las firmas digitales. Aunque éste no sólo es útil para el correo electrónico, sino para otras transacciones y transmisiones electrónicas, como las transferencias de fondos.

La seguridad del algoritmo RSA ha sido validada hasta ahora, ya que ningún intento genérico conocido de romperlo ha tenido éxito, sobre todo debido a la dificultad de factorizar números grandes $n = pq$, donde p y q son números primos grandes. Para entenderlo mejor haremos un pequeño resumen de la teoría de números que se aplica a este sistema.

5.1.1. Matemáticas del sistema

Hasta ahora, esperabamos que k_e y k_d fueran fáciles de calcular a través de la aritmética simple. Ahora debemos representar el mensaje numéricamente, para poder realizar estos algoritmos aritméticos sobre él. Ahora representaremos el mensaje M por un número entero entre 0 y $n-1$. Si el mensaje es demasiado largo, se puede dividir y cifrar por separado. Sean e, d, n enteros positivos, con (e, n) como clave de cifrado, (d, n) la clave de descifrado, sea $n = pq$.

Ahora encriptamos el mensaje elevándolo a la potencia e módulo n para obtener C, el mensaje cifrado. A continuación, desciframos C elevándolo a la potencia d, módulo n, para obtener de nuevo el mensaje, M. Formalmente, obtenemos estos algoritmos de cifrado y descifrado para k_e y k_d:

$$C \equiv k_e(M) \equiv M^e \qquad \text{mód} \quad n$$
$$M \equiv k_d(C) \equiv C^d \qquad \text{mód} \quad n.$$

Obsérvese que conservamos el mismo tamaño de información, ya que M y C son enteros entre 0 y $n-1$, y también debido a la congruencia modular. También hay que tener en cuenta la simplicidad del hecho de que las claves de cifrado/descifrado son pares de enteros, (e, n) y (d, n). Éstas son diferentes para cada usuario y, por lo general, deberían tener subíndices, pero aquí sólo consideraremos el caso general.

Ahora viene la cuestión de crear la clave de encriptación en sí. En primer lugar, elegimos "aleatoriamente" dos primos grandes p y q, los multiplicamos y producimos $n = pq$. Aunque n es público, no se revelará p y q ya que es esencialmente imposible factorizarlos a partir de n, y por lo tanto asegurará que d es prácticamente imposible de derivar a partir de e.

Ahora queremos obtener los e y d apropiados. Elegimos d para que sea un entero grande al azar, que debe ser coprimo a $(p-1)\cdot(q-1)$, lo que significa que la siguiente ecuación tiene que ser satisfecha:

$$\gcd(d, (p-1)\cdot(q-1)) = 1.$$

Querremos calcular e a partir de d, p y q, donde e es la inversa multiplicativa de d. Esto significa que necesitamos satisfacer:

$$e \cdot d = 1 \qquad \text{mód} \quad \varphi(n),$$

donde $\varphi(n)$ es la función de Euler.

Para n, obtenemos, por propiedades elementales de la función de Euler, que:

$$\varphi(n) = \varphi(p)\varphi(q) = (p-1)\cdot(q-1) = n - (p+q) + 1.$$

Por las leyes de la aritmética modular, la inversa multiplicativa de a módulo m existe si y sólo si a y m son coprimos. En efecto, como d y $\varphi(n)$ son coprimos, d tiene un inverso multiplicativo e en el anillo de los enteros módulo $\varphi(n)$.

Hasta aquí, podemos asegurar con seguridad lo siguiente:

$$k_d(k_e(M)) \equiv (k_e(M))^d \equiv (M^e)^d \qquad \text{mód} \quad n = M^{e \cdot d} \qquad \text{mód} \quad n$$

$$k_e(k_d(M)) \equiv (k_d(M))^e \equiv (M^d)^e \qquad \text{mód} \quad n = M^{e \cdot d} \qquad \text{mód} \quad n$$

Necesitaremos una importante identidad debida a Euler y Fermat: para cualquier entero M coprimo a n, tenemos

$$M^{\varphi(n)} \equiv 1 \qquad \text{mód} \quad n.$$

Y por lo tanto:

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \equiv (M^{\varphi(n)})^k M \equiv 1^k M \qquad \text{mód} \quad n = M$$

para un entero k.

Resulta que esto funciona para todo M, y de hecho vemos que $k_d(k_e(M)) = M$ y $k_e(k_e(M)) = M$ se mantienen para todo M, siendo $0 \leq M < n$. Por tanto, k_e y k_d son aplicaciones inversas.

5.1.2. Algoritmo

El algoritmo RSA se describe como sigue [45]:

En primer lugar se realiza un procedimiento para obtener las clave, el cual se desarrolla de la siguiente manera:

1. Se generan aleatoriamente dos primos p y q de longitud $k/2$ bits.
2. Se calcula la clave publica $n = pq$ y $\varphi(n) = (p-1)(q-1)$.
3. Se genera una clave de encriptación aleatoria e, donde $1 < e < \varphi(n)$, coprimo a $\varphi(n)$.
4. Se calcula el único entero d, para el cual $1 < d < \varphi(n)$, donde $e \cdot d \equiv 1 \mod \varphi(n)$.
5. Se devuelve la clave pública (e, n) y la clave privada d.

Una vez se ha llevado a cabo la generación de las claves, pasamos al proceso de encriptado, es decir el proceso de convertir el texto original en el texto encriptado.

Encriptación con las claves (e, n):

1. Representar el mensaje como un valor entero $M \in \{0, \ldots, n-1\}$.

2. Calcular $C = M^e \mod n$.

Y finalmente llegamos al proceso de desencriptación, es decir el procesdo de volver al obtener el texto original a partir del texto encriptado [45].

Desencriptación usando la clave d:

1. Calcular $M = C^d \mod n$.

Según los autores de RSA, el tiempo de cifrado por bloque no aumenta más rápido que el cubo del número de dígitos de n [45]. Ya que la exponenciación modular que se usa en el paso 2 de la encriptación y en el paso 1 de la desencriptación es un proceso muy eficiente computacionalmente siguiendo la explicación del apartado 2.3.2. El algoritmo se resume gráficamente en la Fig. 5.

5.1.3. Encontrar primos grandes

Encontrar n es el primer paso de todo el proceso. El número n se revelará con las claves de cifrado y descifrado, pero los números p y q, cuyo producto constituye n, no se mostrarán explícitamente. Hasta hace poco eran esencialmente imposibles de derivar de n, si elegimos, por ejemplo, los primos p y q de 100 dígitos, que harían un n de 200 dígitos. Sin embargo, hoy en día debemos utilizar números mucho más grandes.

Cada usuario tiene que elegir en privado sus propios dos números primos grandes p y q. Para ello, tenemos que generar, por ejemplo, números impares (pongamos de 100 dígitos) al azar hasta encontrar un primo. Tendremos que probar cada número, y según el teorema de los números primos, habrá aproximadamente $(\ln(10)^{100})/2 = 115$ números para probar.

Para comprobar la primalidad de un número b grande, podemos utilizar un algoritmo de Solovay y Strassen. En primer lugar, elegimos un número aleatorio a de una distribución uniforme en $1, \ldots, b-1$ y probamos si

$$(a, b) = 1 \quad y \quad J(a, b) = a^{(b-a)/2} \qquad \text{mód} \quad b.$$

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

El símbolo de Jacobi sólo está definido cuando a es un número entero y b es un número entero impar positivo. Además, $J(a, b)$ es 0 si $\gcd(a, b) \neq 1$ y ± 1 si $\gcd(a, b) = 1$. La ecuación anterior es siempre verdadera si b es primo, de lo

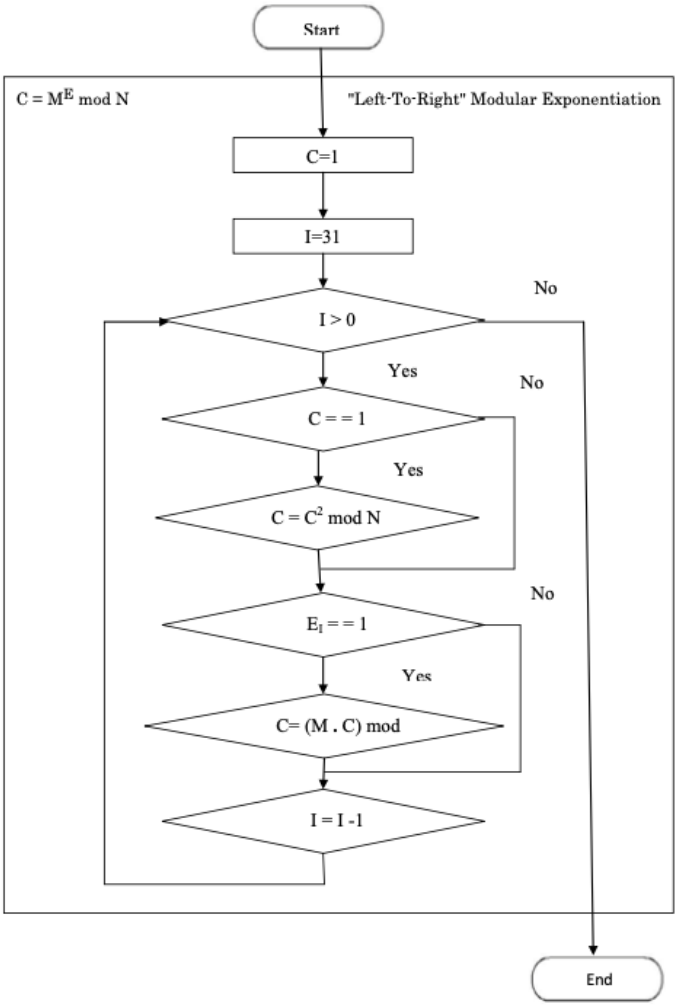

Figura 5: Algoritmo RSA

contrario (si b es compuesto), la ecuación tendrá una probabilidad de ser falsa de más del 50 %. Si es verdadera 100 veces para a's elegidas al azar, entonces b es casi seguro primo, con una probabilidad de ser compuesto de 1 cada 2^{100}. Si accidentalmente se utilizara un compuesto para p o q en el proceso, el receptor vería un error y se daría cuenta de que el descifrado no se hizo correctamente. Para protegerse de algoritmos de factorización sofisticados, p y q deben diferir en longitud por unos pocos dígitos, $\gcd(p-1, q-1)$ debe ser pequeño, y tanto $(p-1)$ como $(q-1)$ deben contener factores primos grandes.

También podríamos encontrar primos grandes tomando un número cuya factorización conozcamos, añadiéndole 1 y comprobando la primalidad. Si obtenemos un número que creemos que es primo, podríamos demostrar que lo es utilizando la factorización de $(p-1)$.

5.1.4. Encontrar d

Queremos encontrar un número d coprimo a $\varphi(n)$; cualquier número primo mayor que máx(p, q) está bien. Como el conjunto de primos $\mathbb{P}$ es grande, se asegura que un criptoanalista no pueda encontrar d mediante una búsqueda directa. De hecho, cualquier método para encontrar d que escoja d de un conjunto grande serviría.

5.1.5. Encontrar e a partir de d y $\varphi(n)$

Aquí, utilizamos una variación del algoritmo de Euclides para calcular el máximo común divisor de $\varphi(n)$ y d. Primero, calculamos una serie $x_0, x_1, x_2, \ldots$, donde $x_0 \equiv \varphi(n)$, $x_1 = d, \ldots$, $x_{i+1} \equiv x_{i-1} \pmod{x_i}$, hasta encontrar un $x_k = 0$. Entonces el $\gcd(x_0, x_1) = x_{k-1}$. Ahora, definimos los números a_i y b_i de manera que $x_i = a_i \cdot x_0 + b_i \cdot x_1$. Si $x_{k-1} = 1$, entonces b_{k-1} es el inverso multiplicativo de $x_1 \pmod{n}$, y es precisamente e. Dado que la dificultad de calcular la aritmética modular complicada contribuye en parte a la dificultad de descifrar RSA, tenemos que utilizar esto en nuestro beneficio. Por lo tanto, si $e < \log_2(n)$, encontramos otro e que no sea demasiado pequeño para que el mensaje cifrado sufra una reducción en módulo n lo que se suele llamar "*wrap-around*".

5.1.6. Seguridad del RSA

El algoritmo RSA es, en efecto, uno de los más potentes, pero ¿puede resistir a todo? Ciertamente, nada puede resistir la prueba del tiempo. De hecho, ninguna técnica de cifrado es perfectamente segura frente a un ataque de un criptoanalista. Métodos como la fuerza bruta son sencillos pero largos y pueden descifrar un mensaje, pero probablemente no todo un esquema de cifrado. También hay que tener en cuenta un enfoque probabilístico, es decir, que siempre existe la posibilidad de que alguien consiga "una clave entre un millón". Hasta ahora, no sabemos cómo demostrar si un esquema de cifrado es irrompible. Si no podemos demostrarlo, al menos veremos si alguien puede descifrar el código. Así es como se certificó esencialmente el estándar RSA. A pesar de los años de intentos, no se sabe que nadie haya podido descifrar el RSA. Esta resistencia a los ataques hace que RSA sea seguro en la práctica.

En esta sección veremos por qué descifrar RSA es al menos tan difícil como factorizar n. Hoy en día no existe ningún algoritmo que permita factorizar un número de 200 dígitos en un tiempo razonable.

Para demostrar que RSA es seguro, consideraremos cómo un criptoanalista puede intentar obtener la clave de descifrado a partir de la clave pública de cifrado, y no cómo un intruso puede intentar robar la clave de descifrado. Los autores de RSA ofrecen un ejemplo: el dispositivo de cifrado, estaría separado del resto del sistema. Generaría claves de cifrado y descifrado, pero no imprimiría la clave de descifrado, ni siquiera para su propietario. De hecho, borraría la clave de descifrado si detectara un intento de intrusión.

Factorizar n

Dado que conocer los factores de n revelaría $\varphi(n)$ y, por tanto, d, un criptoanalista rompería el código si factorizara n. Sin embargo, la factorización de números ha demostrado ser mucho más difícil que determinar la primalidad o la compositividad. No obstante, existen muchos algoritmos de factorización. Como el mostrado por a Knuth y Pollard [25]. Tambien Pollard [40] nos muestra un algoritmo no publicado, que factoriza n en aproximadamente

$$exp(\sqrt{ln(n) \cdot ln(ln(n))} = (ln(n))^{\frac{ln(n)}{ln(ln(n))}}$$

pasos. La siguiente tabla es la que presentaron los autores del RSA en 1978 [45]. Suponen que una operación del algoritmo de factorización de Schroeppel

tarda un microsegundo en calcularse, y presentan los siguientes datos para varias longitudes de n:

Digits	**Number of operations**	**Time**
50	1.4×10^{10}	3.9 hours
75	9.0×10^{12}	104 days
100	2.3×10^{15}	74 years
200	1.2×10^{23}	3.8×10^{9} years
300	1.5×10^{29}	4.9×10^{15} years
500	1.3×10^{39}	4.2×10^{25} years

Se recomienda que n tenga unos 200 dígitos. Sin embargo, la longitud de n puede variar, en función de la importancia de la velocidad de encriptación frente a la seguridad. En efecto, el RSA permite al usuario elegir la longitud de la clave y, por tanto, el nivel de seguridad, una flexibilidad que no se encontraba en muchos esquemas de cifrado anteriores a 1978.

5.2. Logaritmo discreto

El sistema RSA que se discutió en la sección anterior se basa en el hecho de que elegir dos primos grandes y multiplicarlos para obtener n es mucho más fácil que ir en la otra dirección, dado n, encontrar los dos primos grandes. Hay otros procesos fundamentales en la teoría de números que, aparentemente, también tienen esta propiedad de "un solo sentido". Uno de los más importantes es la elevación a una potencia en un gran cuerpo finito.

Cuando se trabaja con los números reales, la exponenciación, es decir, encontrar b^x con una precisión prescrita, no es significativamente más fácil que la operación inversa, encontrar el $\log_b x$. Pero ahora supongamos que tenemos un grupo finito, como $(\mathbb{Z}/n\mathbb{Z})^*$ o $\mathbb{F}_q^*$, con la operación grupal de multiplicación. Gracias al método de exponenciación modular se puede calcular b_x para x grandes con bastante rapidez. Pero, si nos dan un elemento y que sabemos que es de la forma b^x y suponemos que la "base" b es fija, ¿cómo podemos encontrar la potencia de b que da y, es decir, cómo podemos calcular $x = \log_b y$? Esta pregunta se denomina "*problema del logaritmo discreto*". La palabra "discreto" distingue la situación del grupo finito de la situación clásica o continua.

Definición 5.1: *Si G es un grupo finito, b es un elemento de G, e y es un elemento de G que es una potencia de b, entonces el logaritmo discreto*

de y a la base b es cualquier número entero x tal que $b^x = y$.

Ejemplo 5.2:. Si tomamos $G = \mathbb{F}_{19}^* = (\mathbb{Z}/19\mathbb{Z})^*$ y dejamos que b sea el generador 2, entonces el logaritmo discreto de 7 en base 2 es 6.

Veamos ahora algunos ejemplos de criptosistemas basados en la idea mostrada con el problema del logaritmo discreto.

5.2.1. El sistema de intercambio de claves Diffie-Hellman.

Dado que los criptosistemas de clave pública son relativamente lentos en comparación con los criptosistemas de clave privada, al menos en el estado actual de la tecnología y los conocimientos teóricos, a menudo es más realista utilizarlos en un papel limitado junto con un criptosistema de clave privada en el que se transmiten los mensajes reales. En particular, el proceso de acordar una clave para un criptosistema de clave privada puede realizarse de forma bastante eficiente utilizando un sistema de clave pública. La primera propuesta detallada para hacerlo, debida a W. Diffie y M. E. Hellman [12], se basaba en el problema del logaritmo discreto.

Suponemos que la clave para el criptosistema clásico es un gran número entero positivo. Por ejemplo, supongamos que queremos utilizar una transformación matricial afín de pares de dígrafos

$$C \equiv \begin{pmatrix} a & b \\ c & d \end{pmatrix} P + \begin{pmatrix} e \\ f \end{pmatrix} \qquad \text{mód} \quad N^2.$$

donde $0 < a, b, c, d, e, f < N^2$ y P es un vector de columnas que consiste en los equivalentes numéricos de dos dígrafos sucesivos de texto plano, es decir, en conjunto un bloque de cuatro letras, en un alfabeto de N letras. Una vez que tenemos un número entero seleccionado aleatoriamente k entre 0 y N^{12} podemos tomar a, b, c, d, e, f para que sean los seis dígitos de k escritos en la base N^2 como se explica en la seccion de *matrices de cifrado* en [26]. También debemos comprobar que $ad - bc$ es invertible módulo N^2, es decir, que no tiene ningún factor común con N; en caso contrario, deberemos elegir otro número entero aleatorio k.

Observamos que elegir un entero aleatorio en algún intervalo es equivalente a elegir un elemento aleatorio de un gran cuerpo finito de aproximadamente el mismo tamaño. Supongamos, por ejemplo, que queremos elegir un número aleatorio positivo $k < N^{12}$. Si nuestro cuerpo finito es un cuerpo primo de p elementos, simplemente dejamos que un elemento de $\mathbb{F}_p$, corresponda a un entero de 0 a $p - 1$ de la forma habitual; si el entero resultante

es mayor que N^{12}, lo reducimos módulo N^{12}. Si nuestro campo finito es $\mathbb{F}_{p^f}$, elegimos primero una base $\mathbb{F}_p$ de este campo, de modo que cada elemento corresponda a una f-tupla de elementos de $\mathbb{F}_p$; entonces tal f-tupla da un entero menor que p^f si consideramos las coordenadas como dígitos de un entero escrito en la base p. Esto da una correspondencia 1 a 1 entre $\mathbb{F}_{p^f}$, y $\mathbb{Z}/p^f\mathbb{Z} = (0, 1, 2, \ldots, p^f - 1)$. Pero estos dos conjuntos tienen una estructura muy diferente bajo la suma y la multiplicación. El primero es un cuerpo, es decir, todos los $p^f - 1$ elementos no nulos tienen inversos, mientras que el segundo es un anillo en el que p^{f-1} de los p^f elementos (los múltiplos de p) no tienen inversos.

Ahora describimos el método Diffie-Hellman para generar un elemento aleatorio de un gran cuerpo finito $\mathbb{F}_q$. Suponemos que q es de conocimiento público: todo el mundo sabe en qué campo finito estará nuestra clave. También suponemos que g es algún elemento fijo de $\mathbb{F}_q$, que tampoco se mantiene en secreto. Idealmente, g debería ser un generador de $\mathbb{F}_q$. El método descrito a continuación para generar una clave sólo conducirá a elementos de $\mathbb{F}_q$, que son potencias de g; por lo tanto, si realmente queremos que nuestro elemento aleatorio de $\mathbb{F}_q^*$, tenga la posibilidad de ser cualquier elemento, g debe ser un generador.

Supongamos que dos usuarios Alice y Bob quieren acordar una clave que utilizarán para cifrar los mensajes que se envíen posteriormente. Alice elige un número entero aleatorio a entre 1 y $q - 1$, que mantiene en secreto, y calcula $g^a \in \mathbb{F}_q$, que hace público. Bob hace lo mismo: elige un b aleatorio y hace público g^b. La clave secreta que utilizan es g^{ab}. Ambos usuarios pueden calcular esta clave. Por ejemplo, Alice conoce g^b y su propio secreto a. Sin embargo, un tercero sólo conoce g^a y g^b. Si se cumple la siguiente suposición para el grupo multiplicativo $\mathbb{F}_q^*$, un tercero no autorizado no podrá determinar la clave.

La hipótesis de Diffie-Hellman dice lo siguiente: Es computacionalmente inviable calcular g^{ab} conociendo sólo g^a y g^b.

La suposición de Diffie-Hellman es a priori tan fuerte como la suposición de que los logaritmos discretos no pueden ser calculados en el grupo. Es decir, si se pueden calcular logaritmos discretos, es obvio que la hipótesis de Diffie-Hellman falla. Algunas personas conjeturarían que la implicación inversa también es válida, pero eso sigue siendo una cuestión abierta. En otras palabras, nadie puede imaginar una forma de pasar de g^a y g^b a g^{ab} sin poder determinar primero a o b; pero es concebible que tal forma pueda existir.

Ejemplo 5.3: Supongamos que utilizamos un cifrado por turnos de unidades de mensajes de una sola letra en el alfabeto de 26 letras: $C \equiv P + B$ mód 26. Usamos B en lugar de h para denotar la clave de desplazamiento para no confundirla con la b del último párrafo. Para elegir B, tome el menor residuo no negativo módulo 26 de un elemento aleatorio de $\mathbb{F}_{53}$. Supongamos que Alice elige al azar $a = 29$ y busca la clave pública de Bob 2^b, que es, por ejemplo, $12 \in \mathbb{F}_{53}$. Entonces sabe que la clave de cifrado es $12^{29} = 21 \in \mathbb{F}_{53}$, es decir, $B = 21$. Al mismo tiempo, ella ha hecho público $2^{29} = 45$, por lo que Bob también puede encontrar la clave $B = 21$ elevando 45 a la potencia b (su exponente secreto es $b = 19$). Por supuesto, no hay seguridad en trabajar con un cuerpo tan pequeño; un extraño podría encontrar fácilmente el logaritmo discreto en base 2 de 12 o 45 módulo 53. Y, en cualquier caso, no hay seguridad en el uso de un cifrado por turnos de unidades de mensaje de una sola letra. Pero este ejemplo ilustra la mecánica del sistema de intercambio de claves Diffie-Hellman.

5.2.2. Criptositema ElGamal

En 1984 Taher ElGamal presentó un criptosistema que se basa en el problema del logaritmo discreto que se discutió en una sección anterior [13]. Se basa en la suposición de que el logaritmo discreto no se puede encontrar en un tiempo factible, mientras que la operación inversa de la potencia se puede calcular de forma eficiente.

El sistema de clave pública original propuesto por Diffie y Hellman requiere la interacción de ambas partes para calcular una clave privada común. Esto plantea problemas si el criptosistema se aplica a sistemas de comunicación en los que ambas partes no pueden interactuar en un tiempo razonable debido a retrasos en la transmisión o a la falta de disponibilidad de la parte receptora.

Por ello, ElGamal simplificó el algoritmo de intercambio de claves Diffie-Hellman introduciendo un exponente aleatorio k. Este exponente sustituye al exponente privado de la entidad receptora. Gracias a esta simplificación, el algoritmo puede utilizarse para cifrar en una dirección, sin necesidad de que la segunda parte participe activamente. El avance clave aquí es que el algoritmo puede utilizarse para encriptar mensajes electrónicos, que se transmiten por medio de servicios públicos de almacenamiento y reenvío.

En esta sección se presentará al lector el criptosistema ElGamal.

GENERACIÓN DE CLAVES

Como se ha comentado anteriormente, el requisito básico de un sistema criptográfico es al menos una clave para los algoritmos simétricos y dos claves para los asimétricos.

Los pasos de generación de claves son similares al esquema general explicado anteriormente. Con ElGamal, sólo el receptor necesita crear una clave por adelantado y publicarla. Siguiendo nuestro esquema de nombres de arriba, ahora seguiremos a Bob a través de su procedimiento de generación de claves.

Bob seguirá los siguientes pasos para generar su par de claves:

1. **Generación de primos y grupos:** Primero Bob necesita generar un primo grande p y el generador g de un grupo multiplicativo $\mathbb{Z}_p^*$ de los enteros módulo p.

2. **Selección de la clave privada:** Ahora Bob selecciona un entero b del grupo $\mathbb{Z}$ al azar y con la restricción $1 \leq b \leq p-2$. Este será el exponente privado.

3. **Ensamblaje de la clave pública:** A partir de esto podemos calcular la parte de la clave pública $g^b \mod p$. La clave pública de Bob en el criptosistema ElGamal es el triplete (p, g, g^b) y su clave privada es b.

4. **Publicación de la clave pública:** La clave pública tiene que ser publicada utilizando algún servidor de claves u otros medios, para que Alice pueda obtenerla.

PROCESO DE ENCRIPTADO

Para cifrar un mensaje M a Bob, Alice necesita primero obtener su triplete de claves públicas (p, g, g^b) de un servidor de claves o recibiéndola de él a través de un correo electrónico sin cifrar. No hay ningún problema de seguridad en esta transmisión, ya que la única parte secreta, b, se envía en g^b. Dado que la hipótesis central del criptosistema ElGamal dice que es inviable calcular el logaritmo discreto, esto es seguro.

Para el cifrado del mensaje de texto plano M, Alice tiene que seguir los siguientes pasos:

1. **Obtener la clave pública:** Como se ha descrito anteriormente, Alice tiene que adquirir la parte de la clave pública (p, g, g^b) de Bob de un servidor de claves oficial y de confianza.

2. **Preparar M para la codificación:** Escribe M como un conjunto de enteros $(m_1, m_2, \dots)$ en el rango de $\{1, \dots, p-1\}$. Estos enteros se codificarán uno a uno.

3. **Seleccionar el exponente aleatorio:** En este paso, Alice seleccionará un exponente aleatorio k que toma el lugar del exponente privado de la segunda parte en el intercambio de claves Diffie-Hellman. La aleatoriedad aquí es un factor crucial ya que la posibilidad de adivinar el k da una cantidad sensible de la información necesaria para descifrar el mensaje al atacante.

4. **Calcular la clave pública:** Para transmitir el exponente aleatorio k a Bob, Alice calcula $g^k \mod p$ y lo combina con el texto cifrado que se enviará a Bob.

5. **Cifrar el texto plano:** En este paso, Alice cifra el mensaje M, convirtiendolo en el texto cifrado C. Para ello, itera sobre el conjunto creado en el paso 2 y calcula para cada uno de los m_i:

$$c_i = m_i \cdot (g^b)^k.$$

   El texto cifrado C es el conjunto de todos los c_i con $0 < i \leq |M|$.

El mensaje cifrado resultante C se envía a Bob junto con la clave pública $g^k \mod p$ derivada del exponente privado aleatorio.

Incluso si un atacante escuchara esta transmisión, y en un segundo paso adquiriera también la parte de la clave pública g^b de Bob desde un servidor de claves, seguiría sin poder derivar $g^{b \cdot k}$, como sabemos gracias al problema del logaritmo discreto.

ElGamal aconseja utilizar un nuevo k aleatorio para cada uno de los bloques de mensajes individuales m_i. Esto mejora mucho la seguridad, ya que el conocimiento de un bloque de mensajes m_j no lleva al atacante al conocimiento de todos los demás m_i. La razón de esta capacidad es que si $c_1 = m_1 \cdot (g^b)^k \mod p$ y $c_2 = m_2 \cdot (g^b)^k \mod p$, a partir de conocer sólo m_1 se puede calcular la siguiente parte del mensaje m_2 mediante la siguiente fórmula:

$$\frac{m_1}{m_2} = \frac{c_1}{c_2}.$$

PROCESO DE DESENCRIPTADO

Después de recibir el mensaje encriptado C y la clave pública aleatoria g^k, Bob tiene que utilizar el algoritmo de encriptación para poder leer el texto plano M. Este algoritmo se puede dividir en unos pocos pasos simples:

1. **Calcular la clave compartida:** El criptosistema ElGamal ayudó a Alice a definir una clave secreta compartida sin la interacción de Bob. Este secreto compartido es la combinación del exponente privado b de Bob y el exponente aleatorio k elegido por Alice. La clave compartida se define mediante la siguiente ecuación:

$$(g^k)^{p-1-b} = (g^k)^{-b} = g^{-bk}.$$

2. **Desencriptado:** Para cada una de las partes del texto cifrado c_i, Bob calcula ahora el texto plano utilizando

$$m_i = (g^k)^{-b} \cdot c_i \qquad \text{mód} \quad p.$$

Después de combinar todos los m_i de vuelta a M puede leer el mensaje enviado por Alice.

5.3. Tests de primalidad

Hay muchas situaciones en las que se quiere saber si un número grande n es primo. Por ejemplo, en el criptosistema de clave pública RSA y en varios criptosistemas basados en el problema del logaritmo discreto en cuerpos finitos, necesitamos encontrar un primo grande "al azar". Una interpretación de un modo de hacerlo es elegir un número entero impar grande n_0 utilizando un generador de dígitos aleatorios y luego probar la primalidad de n_0, $n_0+2, \ldots$ hasta obtener el primer primo que es $\geq n_0$. Un segundo tipo de uso de las pruebas de primalidad es determinar si un número entero de un tipo muy especial es un primo.

Una prueba de primalidad es un criterio para que un número n no sea primo. Si n "pasa" un test de primalidad, es posible que sea primo. Si pasa muchos tests de primalidad, es muy probable que sea primo. Por otro lado, si n no pasa ningún test de primalidad, entonces es definitivamente compuesto. Pero esto nos deja con un problema muy difícil: encontrar los factores primos de n. En general, se tarda mucho más tiempo en factorizar un número grande una vez que se sabe que es compuesto que en encontrar un número primo del mismo orden de magnitud.

Por ello en esta sección presentaremos brevemente algunos test de primalidad usados típicamente y comentaremos detalladamente el test propuesto por Miller y Rabin. En la sección 5.1.3 se habló ya del algortitmo de Solovay-Strassen.

5.3.1. Test de Miller-Rabin para primalidad

Dado un número entero compuesto $n > 1$, este algoritmo demuestra con alta probabilidad muy rápidamente que n no es primo. Por otro lado, si n pasa la prueba, es simplemente probable que sea primo. El algoritmo consiste en repetir un simple paso, la prueba de Miller-Rabin, varias veces con diferentes inicializaciones aleatorias. La probabilidad de que un número compuesto no sea reconocido como tal por el algoritmo, puede hacerse arbitrariamente pequeña repitiendo el paso principal varias veces. El algoritmo fue propuesto por primera vez por M. Artjuhov [2]. Posteriormente, M. Rabin propuso la versión probabilística [44]. Bajo el supuesto de la Hipótesis de Riemann Generalizada (HGR) se puede demostrar que n es primo aplicando la prueba con suficiente frecuencia. Esto conduce al algoritmo condicional de G. Miller [34]. Bajo el supuesto de la GRH se ejecuta en tiempo polinómico.

La clave de este test se encuentra en el siguiente teorema:

Teorema 5.3: *Sea $n > 9$ un número entero compuesto positivo impar. Escribimos $n - 1 = 2^k m$ para algún exponente $k \geq 1$ y algún entero impar m. Sea*

$$B = \{x \in (\mathbb{Z}/n\mathbb{Z})^* : x^m = 1 \quad o \quad x^{m2^i} = -1 \quad \text{para} \quad \text{algún} \quad 0 \leq i \leq k\}.$$

Entonces tenemos que

$$\frac{\#B}{\varphi(n)} \leq \frac{1}{4}$$

donde $\varphi(n)$ es la función de Euler

Demostración: Sea l la mayor potencia de 2 tal que 2^l tiene la propiedad de dividir $p-1$ para cada primo p divisor de n. Entonces el conjunto B está contenido en

$$B' = \{x \in (\mathbb{Z}/n\mathbb{Z})^* : x^{m2^{l-1}} = \pm 1\}.$$

En efecto, es evidente que cualquier $x \in (\mathbb{Z}/n\mathbb{Z})^*$ que satisfaga $x^m = 1$ está contenido en B'. Por otra parte, si $x^{m2^i} = -1$ para algunos $0 \leq i < k$, tenemos $x^{m2^i} \equiv -1 \mod p$ para todo primo p que divide a n. Se deduce que para todo p, la potencia exacta de 2 que divide el orden de x módulo p, es igual a 2^{i+1}. En particular, 2^{i+1} divide a $p-1$ para todo divisor primo p de n. Por tanto, tenemos que $l \geq i+1$. Así que podemos escribir que $x^{m2^{l-1}} = (-1)^{2^{l-i-1}}$, que es -1 o $+1$ ependiendo de si $l = i+1$ o $l > i+1$. Se deduce que $B \subset B$‘.

Por el teorema del resto chino, el número de elementos $x \in (\mathbb{Z}/n\mathbb{Z})^*$ para los que tenemos $x^{m2^{l-1}} = 1$, es igual al producto variando en p del número de soluciones de la ecuación $X^{m2^{l-1}} = 1$ módulo p^{a_p}. Aquí p recorre los divisores primos de n y p^{a_p} es la potencia exacta de p que divide a n. Como cada uno de los grupos $(\mathbb{Z}/p^{a_p}\mathbb{Z})^*$ es cíclico, el número de soluciones módulo p^{a_p} viene dado por

$$((p-1)p^{a_p-1}, m2^{l-1}) = \gcd(p-1, m)2^{l-1}.$$

La última igualdad se deduce del hecho de que p no divide a m. Por lo tanto, tenemos

$$\#\{x \in (\mathbb{Z}/n\mathbb{Z})^* : x^{m2^{l-1}} = 1\} = \prod_{p|n} \gcd(p-1, m)2^{l-1}.$$

Del mismo modo, el número de soluciones de la ecuación $X^{m2^l} = 1$ módulo p^{a_p} es igual a $\gcd(p-1, m)2^l$, que es el doble del número de soluciones de

$X^{m2^{l-1}} = 1$ módulo p^{a_p}. Se deduce que el número de soluciones de la ecuación $X^{m2^{l-1}} = -1$ módulo p^{a_p} es también igual a $\gcd(p-1,m)2^{l-1}$. Por lo tanto, tenemos

$$\#B' = 2\prod_{p|n} \gcd(p-1,m)2^{l-1},$$

y por tanto

$$\frac{\#B'}{\varphi(n)} = 2\prod_{p|n} \frac{\gcd(p-1,m)2^{l-1}}{(p-1)p^{a_p-1}}.$$

Supongamos ahora que la propiedad $\#B/\varphi(n)$ es superior a $1/4$. Queremos derivar una contradicción. Como tenemos $B \subset B'$, la desigualdad anterior implica que

$$\frac{1}{4} < 2\prod_{p|n} \frac{\gcd(p-1,m)2^{l-1}}{(p-1)p^{a_p-1}}.$$

De esta desigualdad sacamos varias conclusiones. En primer lugar, observamos que $\gcd(p-1,m)2^{l-1}$ divide a $(p-1)/2$, de modo que el lado derecho es como máximo 2^{1-t}, donde t es el número de primos distintos que dividen a n. Se deduce que $t \leq 2$.

Supongamos que $t = 2$, de modo que n tiene precisamente dos divisores primos distintos. Si uno de ellos, digamos p, tiene la propiedad de que p^2 divide a n de modo que $a_p \geq 2$, entonces el lado derecho es como máximo $2^{1-2}/3 = 1/6$. La contradicción es, se deduce que todos los exponentes a_p son iguales a 1, de modo que $n = pq$ para dos primos distintos p y q. La desigualdad anterior se convierte ahora en

$$\frac{p-1}{\gcd(p-1,m)2^l} \cdot \frac{q-1}{\gcd(q-1,m)2^l} < 2.$$

Como los factores del lado izquierdo de esta desigualdad son enteros positivos, ambos son iguales a 1. Esto implica que $p-1 = \gcd(p-1,m)2^l$ y $q-1 = \gcd(q-1,m)2^l$. Se deduce que la potencia exacta de 2 que divide a $p-1$ así como la potencia exacta de 2 que divide a $q-1$ son iguales a 2^l y que las partes impares de $p-1$ y $q-1$ dividen a m. Considerando la relación $pq = 1 + 2^k m$ módulo de la parte impar de $p-1$, vemos que la parte impar de $p-1$ divide a la parte impar de $q-1$. Por simetría, las partes impares

de $p-1$ y $q-1$ son por tanto iguales. Esto implica que $p-1 = q-1$ y contradice el hecho de que $p \neq q$. Por tanto, tenemos $t = 1$ y, por tanto, $n = p^a$ para algún primo impar p y exponente $a \geq 2$. La desigualdad dice ahora que $p^{a-1} < 4$, por lo que $p = 3$ y $a = 2$, contradiciendo la hipótesis de que $n > 9$. Esto demuestra el teorema.

Según el teorema anterior, la probabilidad de que un número compuesto n pase una sola prueba de Miller-Rabin es, como máximo, del 25 %. Por lo tanto, la probabilidad de que n pase $\log(n)$ de tales pruebas es menor que $1/n$. La probabilidad de que un gran número compuesto n supere $(\log(n))^2$ pruebas es astronómicamente pequeña: menos de $n^{-\log(n)}$. Dado que para la mayoría de los números compuestos n la probabilidad de que este supere una prueba de Miller-Rabin es mucho menor que $1/4$, en la práctica uno ya está convencido de la primalidad de n, cuando n supera con éxito un número de pruebas de Miller-Rabin. Esto es suficiente para la mayoría de las aplicaciones comerciales.

Bajo la hipótesis conocida como la Hipótesis de Riemann Generalizada (HGR) para carácteres de Dirichlet, la prueba de Miller-Rabin puede transformarse en una prueba de primalidad determinista en tiempo polinómico como puede encontrarse en [34] aplicando el siguiente teorema.

Teorema 5.4: (GRH). *Sea n un número entero compuesto positivo e impar. Sea $n-1 = 2^k m$ para algún exponente $k \geq 1$ y algún entero impar m. Si para todos los enteros x entre 1 y $2(\log(n))^2$ se tiene*

$$x^m \equiv 1 \quad \text{mód} \quad n \quad o \quad x^{2^i m} \equiv 1 \quad \text{mód} \quad n \quad \text{para} \quad 0 \leq i < k$$

entonces n es un número primo.

Está claro cómo aplicar el teorema anterior y obtener una prueba que demuestre que n es primo bajo la condición de GRH: dado un número entero impar $n > 1$, simplemente probamos la condición del teorema para todo $a \in \mathbb{Z}$ que satisfaga $1 < a < 2(\log(n))^2$. Si n pasa todas estas pruebas y la GRH se cumple, entonces n es primo. Cada prueba implica una exponenciación en el anillo $\mathbb{Z}/n\mathbb{Z}$. Dado que el exponente es menor que n, esto se puede hacer utilizando sólo $O((\log(n)^{1+\mu})$ operaciones elementales para un número entero impar μ. Por tanto, se trata de una prueba de primalidad en tiempo polinómico.

5.3.2. Test AKS

En el verano de 2002, los tres informáticos indios M. Agrawal, N. Kayal y N. Saxena presentaron una prueba de primalidad determinista en tiempo polinómico. El algoritmo AKS depende en gran medida del teorema pequeño de Fermat y sus implicaciones.

La siguiente proposición es un caso especial del teorema de Euler y es necesario, pero insuficiente para determinar la primalidad. La prueba falla para los números pseudoprimos y los números de Carmichael.

Proposición 5.5: *Sea $a \in \mathbb{Z}$, $n \in \mathbb{Z}$, $n \geq 2$ y $\gcd(a, n) = 1$. Entonces n es primo si y solo si se cumple la ecuación:*

$$(x + a)^n \equiv x^n + a \qquad \text{mód} \quad n.$$

Desgraciadamente, esta prueba falla para una clase específica de números, conocidos como pseudoprimos, que incluyen los números de Carmichael.

Un problema adicional de este algoritmo es que tiene una complejidad de $\mathcal{O}(n)$. Esto es inaceptablemente alto para una prueba de primalidad que no es correcta para todos los números. Para reducir esta complejidad, los autores del algoritmo AKS formularon la siguiente modificación. Para un valor pequeño de r adecuadamente elegido, entonces para todos los valores de a y r:

$$(x + a)^n \equiv x^n + a \qquad \text{mód} \quad x^r - 1, n.$$

El número de valores de a que deben probarse y el valor de r están limitados por un polinomio en $\log n$, lo que da como resultado una prueba de primalidad que es determinista y puede ejecutarse en tiempo polinomial. El algoritmo AKS, como se muestra a continuación, es relativamente fácil de seguir y bastante sencillo de implementar. Para $n > 1$ con $n \in \mathbb{N}$ tenemos que los pasos a seguir para implementar el algoritmo son:

1. Si $n = a^b$ para $a \in \mathbb{N}$ y $b > 1$, n es compuesto.
2. Encontrar el r más pequeño tal que $\mathcal{O}_r(n) > 4(\log(n))^2$.
3. Si $1 < \gcd(a, n) < n$ para todo $a \leq r$, n es compuesto.
4. Si $n \leq r$, n es primo.

5. Para $a = 1$ hasta $[2(\varphi(r))^{\frac{1}{2}} \log(n)]$ hacer:

   Si $((x + a)^n \neq x^n + a \ (\text{mód } x^r - 1, n)$

   n es compuesto

6. n es primo

Este algoritmo se ejecuta en un tiempo de $\mathcal{O}(\log(n))^{10,5}$. Por lo tanto, a medida que n crece, el aumento del tiempo de ejecución será proporcional a $(\log(n))^{10,5}$. Se trata de una función de tiempo polinómico, que aunque no es tan rápida como las pruebas probabilísticas que se utilizan hoy en día, tiene la ventaja de ser totalmente determinista.

Recorrer el algoritmo es la mejor manera de entender qué se comprueba en cada fase. El paso inicial consiste en comprobar si n es una potencia perfecta, lo que permitiría determinar inmediatamente su composición. En caso de no ser una potencia propia de un número entero en el segundo paso entra en juego el teorema pequeño de Fermat, en este paso se trata de determinar el menor primo r que no divide a n. El equipo de AKS elige este punto para encontrar r porque los siguientes pasos están limitados por el valor de r. El valor de r se elige de tal manera que es el menor valor mayor que $4(\log(n))^2$ que todavía cumple el cambio propuesto por los autores a la proposición 5.5. El tercer paso determina el máximo común divisor entre n y todos los valores menores o iguales a r. Esto se hace para asegurarse de que todos los valores r y menores son relativamente primos a n. El siguiente paso simplemente establece que si el valor encontrado para r es mayor que n, entonces n es primo. El quinto paso es el que lo engloba todo y, como hemos visto, es el que más tiempo consume. Todas las operaciones de este paso se realizan en el anillo polinómico $(\mathbb{Z}/n)[x]/(x^r - 1)$. La exponenciación del polinomio se lleva a cabo en el lado izquierdo en el anillo y el lado derecho simplemente se reduce para permanecer en el anillo. Estos dos valores se comparan para determinar si son equivalentes. Esta relación tiene que ser probada para todos los valores de a desde 1 hasta el límite establecido en el algoritmo. Si la relación se cumple para todos los valores de a, el número es primo. Los cálculos del algoritmo están dominados por este paso; este paso determina el orden del algoritmo.

5.3.3. APR test

En este apartado se presentará el test llamado APR-test, llamado así por los autores de este Adleman, Pomerance y Rumely, el cual se presentó en 1980. Posteriormente fue simplificado y mejorado por Cohen y Lenstra. Puede utilizarse para demostrar la primalidad de números con miles de bits en un tiempo razonable.

Pero antes mostraremos algunas definiciones básicas y propiedades de las sumas de Gauss y Jacobi, que serán necesarias para el desarrollo del test.

SUMAS DE GAUSS Y JACOBI

Recordemos que un carácter χ módulo q es un homomorfismo de grupo de $(\mathbb{Z}/q\mathbb{Z})^*$ a $\mathbb{C}^*$, donde $\mathbb{C}$ son los números complejos. Si q es primo entonces el carácter χ puede definirse eligiendo el valor $\chi(g)$ para algún generador g de $(\mathbb{Z}/q\mathbb{Z})^*$.

Ejemplo 5.6: Si $q = 5$, entonces $g = 2$ es un generador de $(\mathbb{Z}/5\mathbb{Z})^*$, y todos los caracteres pueden definirse eligiendo valor para $\chi(g)$. Pero χ debe ser un homomorfismo, por lo que su imagen tiene que ser un subgrupo multiplicativo de orden cuatro en C^*. Sólo hay cuatro posibilidades de este tipo:

1. $\chi_1(g) = 1$ (carácter trivial),
2. $\chi_2(g) = -1$,
3. $\chi_3(g) = i$,
4. $\chi_4(g) = -i$.

Definición 5.7: *Sea χ un carácter módulo q. La suma de Gauss $\tau(\chi)$ está definida por:*

$$\tau(\chi) = \sum_{x \in (\mathbb{Z}/q\mathbb{Z})^*} \chi(x)\zeta_q^x,$$

donde $\zeta_q = e^{\frac{2\pi i}{q}}$.

Definición 5.8: *Sea χ_1, χ_2 dos carácteres módulo q. La suma de Jacobi $j(\chi_1, \chi_2)$ está definida por:*

$$j(\chi_1, \chi_2) = \sum_{x \in (\mathbb{Z}/q\mathbb{Z})^*} \chi_1(x)\chi_2(1-x).$$

Existe una conexión nada trivial para ambos objetos. La cual nos permite implementar la prueba de primalidad de manera muy efectiva.

Proposición 5.9: *Sean χ_1, χ_2 carácteres módulo q de tal manera que $\chi_1\chi_2 \neq \chi_0$. Entonces:*

$$j(\chi_1, \chi_2) = \frac{\tau(\chi_1)\tau(\chi_2)}{\tau(\chi_1\chi_2)}.$$

Está claro que si χ es un carácter módulo el número primo q, entonces sus valores pertenecen a algún grupo $\langle\zeta_n\rangle$, donde $n|q-1$. Pero significa que $\tau(\chi) \in \mathbb{Z}[\zeta_n, \zeta_q]$ y $j(\chi_1, \chi_2) \in \mathbb{Z}[\zeta_n]$. El segundo anillo es más sencillo y tiene menor coste de operaciones aritméticas. A continuación se mostrará cómo utilizarlo para implementar eficazmente la prueba de primalidad.

TEST DE PRIMALIDAD

Ahora intentaremos resumir la teoría y mostrar cómo se puede utilizar en la construcción de un algoritmo de prueba de primalidad. Se hará en dos pasos. El primer paso describe un algoritmo poco práctico basado en las sumas de Gauss. El segundo utiliza propiedades particulares de las sumas de Jacobi para trasladar los cálculos a un anillo más pequeño, donde las sumas de Gauss se sustituyen por sumas de Jacobi [11].

Suponemos que N ya ha pasado la prueba de Rabin-Miller y que es muy improbable que N sea compuesto. Nuestro objetivo es la demostración de la primalidad de N. En esta sección fijamos números primos p, q tales que $p^k|q-1$ y $p^{k+1} \nmid q-1$. Sea χ un carácter módulo q de orden $n = p^k$ en el grupo de caracteres.

TEST BÁSICO

La idea fundamental es demostrar una generalización del teorema pequeño de Fermat. Nos permite verificar muchas congruencias que se satisfacen con los números primos y que, en conjunto, implican la primalidad del número probado.

Proposición 5.10: *Sea G un grupo de Galois extensión de $\mathbb{Q}(\zeta_n)/\mathbb{Q}$, y denotese $\mathbb{Z}[G]$ su anillo de grupo, sea $\beta \in \mathbb{Z}[G]$. Entonces si N es primo, existe un $\eta(\chi) \in \langle\zeta_n\rangle$ tal que*

$$\tau(\chi)^{\beta(N-\theta_n)} \equiv \eta(\chi)^{-\beta N} \qquad \text{mód} \quad N.$$

donde $\eta(\chi) = \chi(N)$.

Nótese que $\mathbb{Z}[G]$ actúa no sólo sobre $\mathbb{Z}[\zeta_n]$ sino también sobre $\mathbb{Z}[\zeta_n, \zeta_q]$. Pero no importa porque la acción sobre ζ_q es trivial. En la versión final de la prueba, la congruencia de la proposición 5.10 en $\mathbb{Z}[\zeta_n, \zeta_q]$ se transformará en condición equivalente en $\mathbb{Z}[\zeta_n]$. Así que los resultados de esta sección son importantes sólo desde el punto de vista teórico y es innecesario dar ejemplos de operaciones en $\mathbb{Z}[\zeta_n, \zeta_q]$. Para presentar el resultado principal de esta sección, primero tenemos que definir la llamada condición $\mathcal{L}_p$.

Definición 5.11: *La condición* $\mathcal{L}_p$ *se cumple si para todos los divisores primos* r *de* N *y todos los enteros positivos* a *podemos encontrar* $l_p(r, a)$ *tal que*

$$r^{p-1} \equiv N^{(p-1)l_p(r,a)} \qquad \text{mód} \quad p^a.$$

Ahora ya podemos pasar al teorema que nos permitirá realizar la prueba de primalidad.

Teorema 5.12: *Sea* t *un número entero par. Definamos*

$$e(t) = 2 \prod_{q \quad prime,(q-1)|t} q^{\nu_q(t)+1}.$$

Supongamos que $(N, te(t)) = 1$ *y* $e(t) > N^{1/2}$. *Para cada par de primos* (p, q) *tal que* $(q-1)|t$ *y* $p^k||(q-1)$, *sea* $\chi_{p,q}$ *un carácter módulo* q *de orden* p^k, *si* g_q *es un generador módulo* q, *entonces podemos tomar* $\chi_{p,q}(g_q) = \zeta_{p^k}$.

Si se satisfacen las siguientes condiciones:

1. *Todo* $\chi_{p,q}$ *satisface la proposición 5.10 para algún* $\beta_{p,q} \notin \mathcal{P}$,
2. *la condición* $\mathcal{L}_p$ *es verdadera para todos los primos* $p|t$,
3. *para todo* $0 \leq i < t$ *y* $r = N^i \mod e(t)$ *si* $r \neq 1$, *entonces* $\nmid N$,

entonces N *es primo.*

Este teorema se interpreta como sigue. Si la congruencia de la proposición 5.10 es falsa para algunos $\chi_{p,q}$, entonces tenemos que N no es primo. Pero si es verdadera para todos los caracteres definidos entonces obtenemos alguna información extra sobre los posibles divisores de N. Esto permite demostrar la primalidad de N o da su factor no trivial. Así que el único problema es verificar la condición $\mathcal{L}_p$. La siguiente proposición ofrece un método práctico para comprobarla.

Proposición 5.13: *Supongamos que χ es un carácter módulo q de orden p^k que satisface la proposición 5.10 para algún $\beta \notin \mathcal{P}$. Si una de las siguientes condiciones es cierta, entonces se satisface $\mathcal{L}_p$.*

1. $p \geq 3$,
2. $p = 2$, $k = 1$ y $N \equiv 1 \mod 4$,
3. $p = 2$, $k \geq 2$ y $q^{\frac{N-1}{2}} \equiv -1 \mod N$.

SUMAS DE JACOBI

La prueba basada en las sumas de Gauss es asintóticamente rápida, pero está lejos de ser práctica. La razón principal de esta situación es el cálculo de $\tau(\chi)^{\beta(N-\theta N)}$. Hay que trabajar en $\mathbb{Z}[\zeta_n, \zeta_q]$ y esto es muy lento en la práctica.

Una de las posibles formas de hacer la prueba práctica, es sustituir la congruencia de la proposición 5.10 por alguna condición que dependa sólo de la suma de Jacobi que se encuentra en un anillo más pequeño $\mathbb{Z}[\zeta_n]$. Afortunadamente, es posible y las siguientes tres proposiciones dan una descripción completa de esta construcción.

Primero presentamos un resultado muy bonito que da una condición equivalente para todos los primos impares p prácticamente considerados.

Proposición 5.14: *Sea $3 \leq p < 6 \cdot 10^9$ y $p \neq 1093, 3511$. Si denotamos por E el conjunto de todos los enteros $1 \leq x < p^k$ coprimos a p, entonces la proposición 5.10 es equivalente a la congruencia:*

$$j(\chi, \chi)^{\alpha} \equiv \eta(\chi)^{-cN} \mod N,$$

donde

$$\alpha = \sum_{x \in E} [\frac{Nx}{p^k}] \sigma_x^{-1}$$

y $c = 2(2^{(p-1)p^{k-1}} - 1)/p^k$.

Tenga en cuenta que la restricción de p en la proposición anterior es totalmente irrelevante en la práctica. Incluso si queremos probar la primalidad de los números que tienen 10^9 dígitos decimales, nunca necesitaríamos primos mayores que 1093. Esto significa que el problema práctico de probar la proposicion 5.10 para $p \geq 3$ está resuelto. Las dos siguientes proposiciones describen el caso p = 2.

Proposición 5.15: *Sea χ un carácter módulo q de orden 2^k con $k \geq 3$. Denotemos por E el conjunto de todos los enteros $1 \leq x < 2^k$ que son congruentes a 1 o 3 módulo 8. Fijar $\delta_N = 0$ para N congruente a 1 o 3 módulo 8, $\delta_N = 1$ si N es congruente a 5 o 7 módulo 8. La proposición 5.10 puede sustituirse por*

$$(j(\chi,\chi)j(\chi,\chi^2))^{\alpha} j(\chi^{2^{k-3}},\chi^{3\cdot 2^{k-3}})^{2\delta_N} \equiv (-1)^{\delta_N}\eta(\chi)^{-cN} \qquad \text{mód} \quad N$$

donde

$$\alpha = \sum_{x\in E} [\frac{Nx}{2^k}]\sigma_x^{-1},$$

y $c = 3(3^{2^{k-2}} - 1)/2^k$.

Proposición 5.16: *Para $p = 2$, $k = 1$, y $\beta = 1$, la proposición 5.10 es equivalente a la congruencia*

$$(-q)^{\frac{N-1}{2}} \equiv \eta(\chi) \qquad \text{mód} \quad N.$$

.

Para $p = 2$, $k = 2$, y $\beta = 1$ la proposición 5.10 es equivalente a la congrencia

$$j(\chi,\chi)^{\frac{N-1}{2}} q^{\frac{N-1}{4}} \equiv \eta(\chi)^{-1} \qquad \text{mód} \quad N$$

si $N \equiv 1$ mod 4, y a la congruencia

$$j(\chi,\chi)^{\frac{N+1}{2}} q^{\frac{N-3}{4}} \equiv -\eta(\chi) \qquad \text{mód} \quad N$$

si $N \equiv 3$ mod 4.

5.3.4. Test de primalidad de la curva elíptica

La prueba de primalidad de la curva elíptica, propuesta por A. O. L. Atkin en 1988, es una de las pruebas de primalidad más usadas en la práctica [36]. Para explicar su principio, primero consideramos una versión de grupo multiplicativo de la prueba.

Teorema 5.17: *Sea $n > 1$ un número natural y supongamos que existe un elemento $a \in \mathbb{Z}/n\mathbb{Z}$ y un exponente $s > 0$ que satisface*

$$a^s = 1;$$

$$a^{s/q} - 1 \in (\mathbb{Z}/n\mathbb{Z})^* \quad \text{para todo divisor primo } q \text{ de } s,$$

entonces cualquier primo que divide a n *es congruente con* $1 \mod s$. *En particular, si* $s > \sqrt{n}$, *entonces* n *es primo.*

Es importante mencionar que s tiene que ser grande. De hecho, es necesario que $s > n$ para concluir que n es primo. Si n es grande, calcular un divisor s de $n-1$ con estas propiedades suele llevar mucho tiempo. Por lo tanto, sólo en raras ocasiones se demuestra que un número grande n es primo mediante la aplicación directa de este teorema.

Si, por casualidad, el número s pasa alguna prueba de primalidad probabilística y uno está seguro de que s es primo, entonces se puede reducir el problema de probar la primalidad de n a probar la primalidad de s, que es como máximo tan grande como $n/2$ y normalmente bastante más pequeña. En efecto, se elige un $x \in (\mathbb{Z}/n\mathbb{Z})$ al azar y se calcula un $a = x^r$. Es casi seguro que tenemos $a^s \equiv 1 \mod n$ y $a - 1 \in (\mathbb{Z}/n\mathbb{Z})$. Como $s > n$, el teorema anterior implica entonces que n es primo siempre que s sea primo. Sin embargo, la probabilidad de que esto ocurra es muy baja.

Las curvas elípticas proporcionan una salida a esta situación. El punto principal es que para el primo n hay muchas curvas elípticas E sobre $(\mathbb{Z}/n\mathbb{Z})$ y los órdenes de los grupos $E(\mathbb{Z}/n\mathbb{Z})$ están distribuidos de forma bastante uniforme en el intervalo $(n+1-2n^{\frac{1}{2}}, n+1+2n^{\frac{1}{2}})$. S. Goldwasser y J. Kilian [20] propusieron una prueba de primalidad basada en el principio del Teorema 5.17 y en un algoritmo determinista de tiempo polinómico para determinar el número de puntos de una curva elíptica sobre un cuerpo finito [46]. El tiempo de ejecución de su algoritmo probabilístico es de tiempo polinomial si se asume una determinada hipótesis no demostrada sobre la distribución de los números primos en intervalos cortos. Algunos años más tarde, L. Adleman y M.D. Huang [1] eliminaron la suposición, proponiendo una prueba probabilística que involucra variedades abelianas de dimensión 2. Ambas pruebas tienen un valor más teórico que práctico. Desde un punto de vista teórico, estos algoritmos han sido sustituidos por el algoritmos más deterministas de tiempo polinómico. Sin embargo, la idea clave conduce a un potente algoritmo práctico.

El resultado principal es el siguiente análogo elíptico del Teorema 5.17.

Teorema 5.18: *Sea* $n > 1$ *un número natural y sea* E *una curva elíptica sobre* $(\mathbb{Z}/n\mathbb{Z})$. *Supongamos que existe un punto* $P \in E(\mathbb{Z}/n\mathbb{Z})$ *y un entero* $s > 0$ *para el que*

$$sP = 0 \in E;$$

$$(s/q)P \neq 0 \in E(\mathbb{Z}/n\mathbb{Z}) \quad para \quad cualquier \quad divisor \quad p \quad de \quad n.$$

Entonces todo primo p que divide a n satisface $\#E(\mathbb{Z}/n\mathbb{Z}) \equiv 0 \mod s$. *En particular, si* $s > (n^{\frac{1}{4}} + 1)^2$, *entonces n es primo.*

El algoritmo reduce el problema de demostrar la primalidad de n, al problema de demostrar que un número menor es primo.

6. Aplicaciones de la Criptografía

En el pasado, la criptografía sólo se aplicaba para asegurar la información. Los sellos de cera, las firmas manuales y algunos otros tipos de métodos de seguridad se utilizaban generalmente para garantizar la fiabilidad y la precisión del transmisor. Con la llegada de las transmisiones digitales, la seguridad de transmisión se convierte en un aspecto más esencial, y por tanto los mecanismos de criptografía comenzaron a superar sus límites para mantener el nivel de seguridad al máximo. A continuación se exponen algunas de las aplicaciones más comunes en criptografía.

6.1. Funciones hash

El término función hash se ha utilizado en informática desde hace bastante tiempo y se refiere a una función que comprime una cadena de entrada arbitraria a una cadena de longitud fija. Sin embargo, si satisface algunos requisitos adicionales, puede utilizarse para aplicaciones criptográficas y entonces se conoce como función hash criptográficas. Las funciones hash criptográfica son una de las herramientas más importantes en el campo de la criptografía y se utilizan para lograr una serie de objetivos de seguridad como la autenticidad, las firmas digitales, la generación de pseudonúmeros, la esteganografía digital, el sellado de tiempo digital, etc. Gauravram [19] sugiere que el uso de funciones hash criptográficas en varias aplicaciones de procesamiento de información para lograr varios objetivos de seguridad está mucho más extendido que la aplicación de cifrados de bloque y cifrados de flujo.

Rompay [42] ha dado la siguiente definición formal de las funciones hash.

Definición 6.1: *Una función hash es una función $h : D \to R$, donde el dominio $D = \{0,1\}^*$ pasa a $R = \{0,1\}^n$ para algún $n \geq 1$.*

En otras palabras, una función hash transforma una cadena de longitud arbitraria $(*)$ en una cadena de longitud fija.

Las funciones hash criptográficas son, en general, de dos tipos: las funciones hash con clave, que utilizan una clave secreta, y las funciones hash sin clave, que no utilizan una clave secreta. Las funciones hash con clave se denominan código de autenticación de mensajes. En general, el término funciones hash se refiere a las funciones hash sin clave y nosotros nos concentraremos únicamente en las funciones hash sin clave. Las funciones hash sin clave o simplemente hash pueden clasificarse en **OWHF** (funciones hash de una vía), **CRHF** (funciones hash resistentes a las colisiones) y **UOWHF** (funciones hash universales de una vía) dependiendo de las propiedades adicionales que satisfagan.

6.1.1. Tipos de funciones hash

Funciones hash de una vía (OWHF)

La **OWHF** definida en Merkle [33] es una función hash H que satisface los siguientes requisitos:

1. H puede aplicarse a un bloque de datos de cualquier longitud. En la práctica, "cualquier longitud" puede estar en realidad limitada por alguna constante enorme, mayor que cualquier mensaje que queramos "hashear".

2. H produce una salida de longitud fija.

3. Dados H y x, cualquier entrada, es fácil obtener por ordenador el mensaje $H(x)$.

4. Dados H y $H(x)$, es computacionalmente inviable encontrar x.

5. Dados H y $H(x)$, es computacionalmente inviable encontrar x y x' de manera que $H(x) = H(x')$

Los tres primeros requisitos son imprescindibles para las aplicaciones prácticas de una función hash a la autenticación de mensajes y las firmas

digitales. El cuarto requisito, también conocido como resistencia a la preimagen o propiedad unidireccional, establece que es fácil generar un código de mensaje dado un mensaje, pero es muy difícil generar un mensaje dado un código. El quinto requisito, también conocido como propiedad de resistencia a la segunda imagen, garantiza que no se pueda encontrar un mensaje alternativo con el mismo código que un mensaje dado.

Funciones Hash Resistentes a Colisiones (CRHF)

Una de las primeras definiciones de funciones hash resistentes a colisiones fue dada por Merkle [32]. Basándose en la misma, **CRHF** puede definirse como una función hash, que satisface todos los requisitos de **OWHF** es decir, del 1 al 5 del caso anterior, y además satisface la siguiente propiedad de resistencia a colisiones:

Dado H, es computacionalmente inviable encontrar un par (x, y) tal que $H(x) = H(y)$.

Funciones Hash Universales de una Vía (UOWHF)

Mani Naor y Moti Yung [39] presentaron la idea de las funciones hash universales de una Vía y utilizando las mismas, presentaron un esquema de firma digital que no estaba basado en funciones de trampilla, que como recordamos son funciones que es fácil de calcular en una dirección, pero difícil de calcular en la dirección opuesta (encontrar su inversa) sin información especial, llamada "trampilla". Más bien, Mani Naor y Moti Yung [39] utilizaron funciones 1-1 One Way (trampilla) para construir **UOWHF** y, a su vez, implementar un esquema de firma digital.

Dejemos que U contenga un número finito de funciones hash con la misma probabilidad de ser utilizadas. Dejemos que un algoritmo probabilístico de tiempo polinómico A, donde A es un adversario de colisión, opere en dos fases. Recordemos que una colisión es cuando las cosas no salen según lo planeado, y dos entradas separadas en realidad dan como resultado el mismo valor hash. Inicialmente, A recibe la entrada k y da como salida un valor x conocido como valor inicial, luego se elige una función hash H de la familia U. A entonces recibe H y debe dar como salida un y tal que $H(x) = H(y)$. En otras palabras, después de obtener una función hash se intenta encontrar una colisión con el valor inicial. U se llamará familia de funciones hash universales de una vía A, la probabilidad de que A tenga éxito en tiempo polinómico es

despreciable.

6.1.2. Servicios de seguridad de las funciones hash criptográficas. Lograr la integridad y la autenticidad

La verificación de la integridad y la autenticidad de la información es una necesidad primordial en los sistemas y redes informáticos. La primera se refiere a la capacidad de proteger la información y la segunda sirve como prueba de que se trata de la persona real y no un intruso. En particular, dos partes que se comunican a través de un canal inseguro necesitan un método por el cual la información enviada por una parte pueda ser validada como auténtica por la otra [3].

La integridad y autenticación de mensajes puede implementarse de múltiples maneras. Se pueden utilizar mecanismos basados en la encriptación simétrica, pero tienen sus propios inconvenientes. Tsudik [55] ha destacado inconvenientes como la velocidad, el factor de coste, la optimización del tamaño de los datos, etc. Estos métodos combinan las funciones de confidencialidad y autenticación. La confidencialidad es la característica que permite que el mensaje mantenga oculto su contenido de tal forma que si una tercera persona ve el mensaje no pueda interpretar su contenido.. Sin embargo, hay situaciones en las que no es necesario cifrar todo el mensaje.

Para estas aplicaciones, mantener el mensaje en secreto no es la preocupación, pero autenticarlo sí es importante. Por ejemplo, en **SNMP** (Simple Network Management Protocol), suele ser importante que un sistema gestionado autentifique los comandos **SNMP** entrantes, pero no es necesario ocultar el tráfico **SNMP**.

Para implementar la autenticación e integridad de los mensajes, las técnicas alternativas son las funciones **MAC** o hash. Las **MAC** pueden construirse a partir de cifrados en bloque como el **DES**. Sin embargo, recientemente ha surgido un gran interés por la idea de construir **MAC**s a partir de Funciones hash criptográficas [3]. Además de utilizar las funciones hash para implementar **MAC**, las funciones hash pueden ser utilizadas para lograr los objetivos de autenticación e integridad de los mensajes sin el uso de cifrado simétrico. Tsudiac [55] ha detallado un protocolo basado en la misma idea. Rompay [42] también ha detallado las formas de asegurar la autenticación utilizando sólo funciones hash así como utilizando funciones hash con encriptación. El uso de las funciones hash para la autenticación de mensajes y para asegurar

la integridad de los mismos ha aumentado porque la mayoría de las funciones hash son más rápidas que los cifrados en bloque en la implementación de software y estas implementaciones de software están disponibles de forma fácil y gratuita [3].

Implementación de firmas digitales eficientes

La firma digital es un objetivo de seguridad de un criptosistema que pretende alcanzar el objetivo de autenticidad y un servicio de seguridad o propiedad de no repudio [19], que proporciona la seguridad contra la negación por parte de terceros no autorizados que participan en la comunicación. Las funciones **MAC** y hash por sí solas no implementan el objetivo de seguridad de las firmas digitales. Fueron Diffie y Hellman [12] quienes se dieron cuenta por primera vez de la necesidad de una firma electrónica dependiente del mensaje para evitar disputas entre emisor y receptor. El criptosistema RSA [45] fue el primer sistema criptográfico de clave pública con capacidad de firma digital, de la cual hablaremos más adelante. James Ellis, Clifford Cocks y Malcolm Willaimson del **GCHQ** (Government Communication Head Quarters), Cheltenham, Gran Bretaña, quizás inventaron la idea de la clave pública en 1972. Los tres británicos tuvieron que sentarse a ver cómo sus descubrimientos eran redescubiertos por Diffie, Hellman, Merkle, Rivest, Shamir y Adleman durante los tres años siguientes debido a las políticas del **GCHQ** de que todo el trabajo es alto secreto y no puede ser compartido con nadie [50].

Las funciones hash se utilizan para optimizar los esquemas de firma digital. Sin el uso de hash, la firma será del mismo tamaño que el mensaje. El concepto fundamental aquí es que en lugar de generar la firma para todo el mensaje que va a ser autenticado, el remitente del mensaje sólo firma el compendio del mensaje utilizando un algoritmo de generación de firmas. A continuación, el emisor transmite el mensaje y la firma al receptor previsto. El receptor verifica la firma del remitente calculando el compendio del mensaje utilizando la misma función hash que el remitente y comparándolo con el resultado del algoritmo de verificación de la firma. Es obvio que este enfoque ahorra mucha sobrecarga computacional en la firma y verificación de los mensajes en ausencia de funciones hash [19].

Autenticación de usuarios de sistemas informáticos

Las funciones hash pueden utilizarse para autenticar a los usuarios en

el momento de iniciar la sesión. Las contraseñas se almacenan en forma de compendio de mensajes para evitar el acceso a las mismas incluso a los administradores de bases de datos. Cada vez que el usuario intenta iniciar una sesión e introduce la contraseña, el compendio de mensajes de la contraseña introducida se calcula y se compara con el compendio almacenado en la base de datos. Si coincide, el inicio de sesión se realiza con éxito, de lo contrario el usuario no se autentifica.

Las funciones hash también se pueden utilizar para indexar datos en tablas hash, para la toma de huellas dactilares, para detectar datos duplicados o identificar archivos de forma única, y como sumas de comprobación para detectar la corrupción accidental de datos y también para generar números aleatorios.

Teniendo en cuenta esta amplia gama de aplicaciones, no es correcto decir que las funciones hash pertenecen a una sub-rama criptográfica en particular. Estas herramientas criptográficas merecen un estatus aparte. Se utilizan en casi todos los lugares de la criptología en los que se requiere un procesamiento eficaz de la información. Para una lectura más detallada de las funciones hash se puede acudir a [51].

6.2. Firma digital para encriptación asimétrica

Con la revolución de las tecnologías de almacenamiento y comunicación de datos digitales, la información digital puede almacenarse, copiarse, modificarse y transportarse fácilmente. Estas propiedades deseables son muy útiles, pero también presentan algunos problemas de seguridad; por ello, lo digital se considera poco fiable en ámbitos en los que la privacidad, la autenticación y la integridad de los datos son de gran interés, a menos que se le apliquen algunos procedimientos de seguridad.

La solución a todos estos problemas de seguridad es la firma digital. Cuando firmamos un documento digitalmente, enviamos la firma como un documento separado. En el caso de la firma digital, el destinatario recibe el mensaje y la firma. El destinatario necesita aplicar una técnica de verificación a la combinación del mensaje y la firma para comprobar la autenticidad. La firma digital garantiza la privacidad de los datos y evita el acceso no autorizado.

6.2.1. Factores clave de la firma digital

La privacidad, la autenticación, la integridad y el no repudio son cuatro factores clave para lograr la seguridad de la información[45]. La privacidad, también llamada confidencialidad, garantiza la protección de la información frente a personas no autorizadas.

PRIVACIDAD

La privacidad garantiza la seguridad de los datos frente al acceso no autorizado y la manipulación de los mismos. Significa que una transacción entre usuarios no puede ser vista ni interferida por terceros.

AUTENTICACIÓN

La autenticación sirve como prueba de que usted es la persona real y no un intruso. La autenticación es fundamental para que haya confianza entre las partes. La autenticación también es necesaria cuando los usuarios se comunican a través de la red y se conectan a ella para evitar la alteración de los datos [45].

INTEGRIDAD

La integridad se refiere a la capacidad de proteger la información, los datos o las transmisiones de alteraciones no autorizadas, incontroladas o accidentales. También podemos utilizar el término integridad para referirnos al funcionamiento del sistema, la red y la aplicación. Al evitar cambios no autorizados o no deseados en los datos, se puede lograr la integridad de los mismos, lo que garantiza la coherencia interna y externa y otros atributos de los datos, como la exactitud, la exhaustividad, etc [45].

NO REPUDIO

El no repudio proporciona la seguridad contra la negación por parte de terceros no autorizados que participan en la comunicación. De este modo, cuando el mensaje se envía al receptor, éste puede demostrar que el supuesto remitente envió realmente el mensaje [45].

6.2.2. Firma figital

La firma digital es un mecanismo de autenticación que permite al remitente de un mensaje adjuntar un código único que actúa como firma. Normalmente, la firma se forma tomando el hash del mensaje y cifrando el mensaje con la clave privada del remitente. La firma garantiza el origen y la integridad del mensaje. El estándar de firma digital es una norma del **NIST** que utiliza el algoritmo de hash seguro. El mensaje plano, la firma del mensaje y la clave pública del remitente se empaquetan juntos y se transforman en un mensaje firmado y cifrado utilizando la clave pública del destinatario. El destinatario desempaqueta el mensaje recibido, que es el mensaje firmado y cifrado, tras lo cual se utiliza la misma función de hash para calcular el resumen del mensaje recibido, que se compara con la firma descifrada. La Fig. 6 muestra un esquema del funcionamiento de la firma digital.

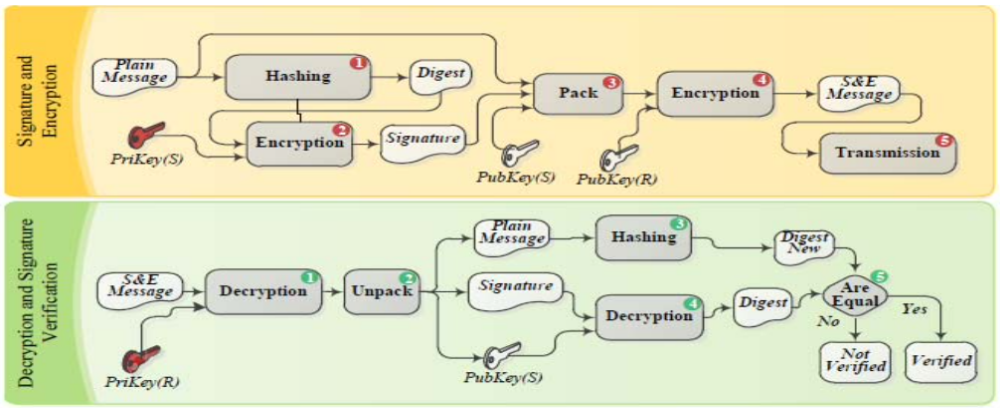

Figura 6: Esquema del funcionamiento de la firma digital.

La firma digital puede clasificarse en dos procesos:

1. Firma y Encriptado
2. Decriptado y Verificación

Los pasos que se siguen en estos procesos se describen a continuación [24]:

1. **Proceso 1: Firma y encriptación**
    a) **Hashing:** En este paso se calcula un pequeño resumen del mensaje, que es una representación única del mismo. Esta evaluación garantiza la integridad del mensaje. La firma digital se aplica a este pequeño resumen del mensaje. Esta evaluación genera un código único.
    b) **Encriptación:** En este paso, el resumen del mensaje se encripta utilizando la clave privada del remitente. Se utiliza para firmar el compendio del mensaje. El mensaje original puede recuperarse descifrando la firma del mensaje utilizando la correspondiente clave pública del remitente. Para obtener el no repudio, se realiza la firma.
    c) **Empaquetado:** El mensaje plano, la firma del mensaje y la clave pública del remitente se empaquetan juntos en una sola unidad empaquetada.
    d) **Encriptación:** La unidad empaquetada del mensaje, que contiene el mensaje sin formato y la firma del mensaje junto con la clave pública del remitente, se encripta utilizando la clave pública del destinatario para formar un mensaje firmado y encriptado.
2. **Proceso 2: Desencriptación y verificación**
    a) **Desencriptación:** En estos pasos, el mensaje recibido, que está firmado y cifrado, se descifra utilizando la clave privada del receptor para formar una unidad de mensaje empaquetada que contiene el mensaje sin formato, la firma y la clave pública del remitente.
    b) **Desembalaje:** El mensaje descifrado en el último paso se descompone en un mensaje en texto plano, la firma del mensaje y la clave pública del remitente.

c) **Hashing:** En este paso, el mensaje en texto plano que se obtiene después de descifrar y desempaquetar el mensaje recibido se introduce en la función hash utilizada por el remitente para calcular el resumen del mensaje.

d) **Desencriptación:** En este paso, la firma del mensaje recibido se descifra utilizando la clave pública recibida del remitente. Mediante el descifrado, se obtiene el compendio del mensaje antes de transmitirlo.

e) **Comparación:** Por último, en este paso, se llega al compendio del mensaje obtenido tras descifrar la firma del mensaje recibido y el compendio del mensaje calculado a partir del mensaje sin formato recibido por el destinatario.

Los datos o mensajes firmados y encriptados sólo pueden descifrarse utilizando la clave privada correcta del destinatario, lo que garantiza la privacidad. La verificación "hashing" y de la firma ofrece la integridad, la autenticidad y el no repudio.

Generalmente, los algoritmos de cifrado y descifrado que utilizan claves asimétricas son demasiado lentos para ser utilizados en mensajes largos, por lo que se genera una clave simétrica que se utiliza en las firmas digitales para cifrar la unidad empaquetada que contiene el mensaje sin formato, la firma del mensaje y la clave pública del remitente [24]. La clave simétrica se encripta utilizando la clave pública del destinatario para que sólo pueda ser descifrada por el receptor previsto, que puede utilizarla para descifrar la unidad empaquetada antes de desempaquetarla y extraer la semántica.

Para entender mejor la firma digital vemos ahora como se implementan utilizando dos algoritmos previamente vistos en este trabajo. Empezamos con el algoritmo RSA.

Supongamos que, Alice está enviando su firma (algún texto plano P) a Bob. Ella conoce la clave de cifrado de Bob $K_{E,B} = (n_B, e_B)$ y su propia clave de descifrado $K_{D,A} = (n_A, d_A)$. Lo que hace es enviar $f_B f_A^{-1}(P)$ si $n_A < n_B$, o bien $f_A^{-1} f_B(P)$ si $n_A > n_B$. Es decir, en el primer caso toma el menor residuo positivo de P^{d_A} módulo n_A ; entonces, con respecto a ese número módulo n_B, calcula $(p^{d_A} \mod n_A)_B^e \mod n_B$, que envía como unidad de mensaje de cifrado. En el caso $n_A > n_B$, primero calcula $P^{e_B} \mod n_B$ y luego, trabajando en módulo n_A, lo eleva a la potencia d_A-ésima. Claramente, Bob puede verificar la autenticidad del mensaje en el primer caso elevando a

la potencia d_B-ésima módulo n_B y luego a la potencia e_A-ésima módulo n_A; en el segundo caso, realiza estas dos operaciones en el orden inverso.

Veamos ahora como se genera la firma digital a partir de los criptosistemas basados en el logaritmo discreto.

Para poner en marcha el esquema (con el fin de poder firmar posteriormente los mensajes), Alice procede de la siguiente manera: (1) elige un primo q de unos 160 bits (para ello, utiliza un generador de números aleatorios y una prueba de primalidad); (2) a continuación, elige un segundo primo p que sea $\equiv 1$ (mód q) y tenga unos 512 bits; (3) elige un generador del único subgrupo cíclico de $\mathbb{F}_p^*$ de orden q (calculando $g_0^{(p-1)/q}$) (mód p) para un entero aleatorio g_0; si este número es $\neq 1$, será un generador); (4) toma un entero aleatorio x en el rango $0 < x < q$ como su clave secreta, y establece su clave pública igual a $y = g^x$ (mód p).

Supongamos ahora que Alice quiere firmar un mensaje. En primer lugar, aplica una función hash a su texto plano, obteniendo un número entero h en el rango $0 < h < q$. A continuación, elige un número entero aleatorio k en el mismo rango, calcula g^k (mód p), y establece r igual al residuo no negativo menor módulo q de este último número (es decir, g^k se calcula primero módulo p, y el resultado se reduce entonces módulo el primo menor q). Finalmente, Alice encuentra un número entero s tal que $sk \equiv h + xr$ (mód q). Su firma es entonces el par (r, s) de enteros módulo q.

Para verificar la firma, el receptor Bob calcula $u_1 = s^{-1}h$ ((mód q) y $u_2 = s^{-1}r$ (mód q). A continuación, calcula $g^{U_1}y^{U_2}$ (mód p). Si el resultado concuerda módulo q con r, está satisfecho.

6.3. Criptografía visual

Finalmente concluiremos este trabajo explicando un método alternativo de criptografía muy relacionado con la esteganografía.

La criptografía visual fue introducida por primera vez en 1994 por Noar y Shamir [38]. La criptografía visual es una técnica criptográfica que permite encriptar la información visual, por ejemplo, texto impreso, notas manuscritas o imágenes, de tal manera que el descifrado puede ser realizado por el sistema visual humano, sin la ayuda de ordenadores. El esquema de criptografía visual elimina el complejo problema del cálculo en el proceso de descifrado, y las imágenes secretas pueden restaurarse mediante una operación de apilamiento. Esta propiedad hace que la criptografía visual sea especialmente útil para el requisito de baja carga computacional.

6.3.1. Escquemas en blanco y negro

Compartición de un único secreto

Naor y Shamir [38] propusieron un esquema de codificación para dividir una imagen binaria en dos partes, $Share_1$ y $Share_2$. Si el píxel es blanco, se elige una de las dos filas de la Fig. 7 para generar $Share_1$ y $Share_2$. Del mismo modo, si el píxel es negro, se elige una de las dos filas siguientes de la tabla para generar $Share_1$ y $Share_2$. En este caso, cada píxel p se codifica en dos píxeles blancos y dos píxeles negros, cada uno de los cuales no da ninguna pista sobre el píxel p, ya sea blanco o negro. La imagen secreta sólo se muestra cuando se superponen ambas acciones.

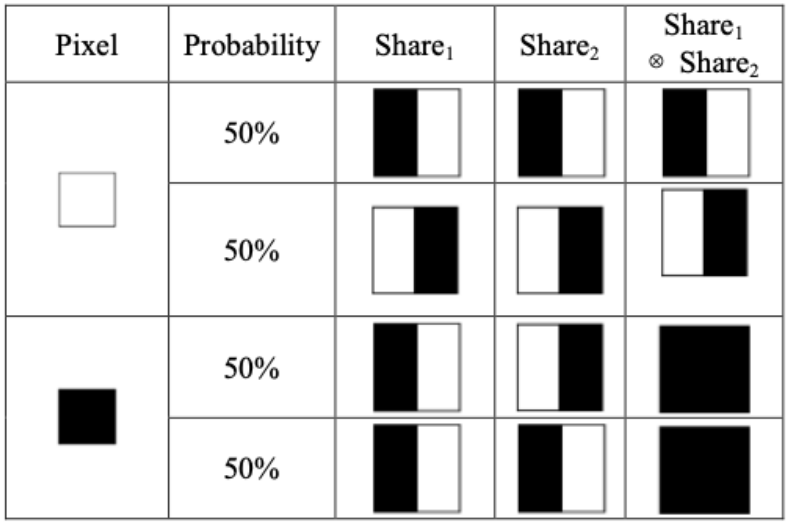

Figura 7: Esquema de Naor y shamir para codificar un píxel binario en dos acciones (*shares*).

Para ocultar una imagen binaria en dos partes significativas, Chin-Chen Chang et al [6] sugirieron esquemas de ocultación de imágenes en el dominio espacial. Estas dos acciones secretas se incrustan en dos imágenes de cobertura de nivel de gris. Para decodificar los mensajes ocultos, las imágenes incrustadas pueden superponerse. Liguo Fang [14] recomienda un esquema de tipo $(2, n)$ basado en la combinación de la expansión de píxeles y el con-

traste. Xiao-Qing y Tan [62] sugirieron esquemas de compartición de secretos visuales con umbral que mezclan operaciones **XOR** y **OR** con inversión y que se basan en un código de corrección de errores lineal binario.

La desventaja de los esquemas anteriores es que sólo se puede incrustar un conjunto de mensajes confidenciales, por lo que para compartir grandes cantidades de mensajes confidenciales hay que generar varias acciones.

Compartición de múltiples secretos

Wu y Chen [60] fueron los primeros investigadores en presentar los esquemas de criptografía visual para compartir dos imágenes secretas en dos acciones. Ocultaron dos imágenes binarias secretas en dos partes aleatorias, A y B, de modo que el primer secreto puede verse apilando las dos partes, denominadas $A \oplus B$, y el segundo secreto puede obtenerse girando primero A θ grados en sentido contrario a las agujas del reloj. Diseñaron el ángulo de rotación θ para que fuera de 90^o. Sin embargo, es fácil obtener que θ puede ser 180^o o 270^o. Para superar la restricción del ángulo del esquema de Wu y Chen [60], Hsu et al. [22] propusieron un esquema para ocultar dos imágenes secretas en dos imágenes compartidas rectangulares con ángulos de rotación arbitrarios. Wu y Chang [61] también refinaron la idea de Wu y Chen [60] codificando las imágenes compartidas como círculos, de modo que las restricciones de los ángulos de rotación ($\theta = 90^o, 180^o$ o 270^o) pueden eliminarse.

Shyu et al. [49] fueron los primeros investigadores en aconsejar la compartición de múltiples secretos en criptografía visual. Este esquema codifica un conjunto de $n \geq 2$ secretos en dos círculos compartidos. Los n secretos pueden obtenerse uno a uno apilando la primera acción y la segunda acción rotada con n ángulos de rotación diferentes. Para codificar formas ilimitadas de imagen y eliminar la limitación de que las transparencias sean circulares, Fang [16] ofreció un esquema de criptografía visual reversible. En este esquema se codifican dos imágenes secretas en dos acciones; una imagen secreta aparece con sólo apilar dos acciones y la otra imagen secreta aparece con apilar dos acciones después de invertir una de ellas. Feng et al. [17] desarrollaron un esquema visual de compartición de secretos para ocultar múltiples imágenes secretas en dos acciones. El esquema propuesto analiza los píxeles secretos y los bloques de acciones correspondientes para construir un gráfico de relación de apilamiento, en el que los vértices denotan los bloques de acciones y los bordes denotan dos bloques apilados en el ángulo de descifrado

deseado. Según este gráfico y el conjunto de patrones visuales predefinidos, se generan dos acciones.

Para proporcionar más aleatoriedad en la generación de las acciones, Ulutas et al. [56] aconsejaron un esquema de compartición de secretos basado en la rotación de las acciones. En este esquema las acciones son de forma rectangular y se crean de forma totalmente aleatoria. Apilando las dos acciones se reconstruye el primer secreto. Si se gira la primera acción 90^o en sentido contrario a las agujas del reloj y se apila con la segunda, se reconstruye el segundo secreto. Chen et al. [9] ofrecieron los esquemas de encriptación de imágenes múltiples mediante la rotación de cuadrículas aleatorias, sin ninguna expansión de píxeles ni rediseño del libro de códigos. Fang [15] ofrece un método de compartición de secretos visuales reversible sin expansión que no necesita definir la tabla de búsqueda. Para codificar cuatro secretos en dos acciones y recuperar las imágenes reconstruidas sin distorsiones, Fu et al. [18] pretenden un esquema de criptografía visual por rotación. La construcción del esquema de criptografía visual por rotación se basó en un conjunto de matrices correlativas y una permutación aleatoria, que puede utilizarse para codificar cuatro imágenes secretas en dos acciones. Weir et al. [59] sugirieron compartir múltiples secretos utilizando criptografía visual. Se genera una clave maestra para todos los secretos; en consecuencia, los secretos se comparten utilizando la clave maestra y se obtienen múltiples acciones.

Todos los esquemas anteriores sólo pueden utilizarse para compartir imágenes secretas en blanco y negro, pero hoy en día muchas imágenes son a color. Para satisfacer esta demanda se han realizado investigaciones para compartir las imágenes en color.

6.4. Esquemas en color

Compartición de un único secreto

Hasta el año 1997 los esquemas de criptografía visual se aplicaban sólo a imágenes en blanco y negro. El primer esquema de criptografía visual en color fue desarrollado por Verheul y Van Tilborg [57]. Las imágenes secretas coloreadas pueden ser compartidas con el concepto de arcos para construir un esquema de criptografía visual coloreada. En el esquema de criptografía visual en color un píxel se transforma en m subpíxeles, y cada subpíxel se divide en c regiones de color. En cada subpíxel, hay exactamente una región de color coloreada, y todas las demás regiones de color son negras. El color

de un píxel depende de las interrelaciones entre los subpíxeles apilados. Para un esquema de criptografía visual en color con c colores, la expansión de píxeles m es $c \times 3$. Yang y Laih [63] mejoraron la expansión de píxeles a $c \times 2$ de Verheul y Van Tilborg [57]. Pero en ambos esquemas la compartición generada no tenía sentido.

Para compartir una imagen secreta en color y también para generar la cuota significativa para transmitir la imagen secreta en color, Chang y Tsai [7] anticiparon un esquema de criptografía visual en color. Para una imagen de color secreta se seleccionan dos imágenes de color significativas como imágenes de cobertura que tienen el mismo tamaño que la imagen de color secreta. Entonces, de acuerdo con una tabla de índices de color predefinida, la imagen secreta en color se ocultará en dos imágenes de camuflaje. Una de las desventajas de este esquema es que se requiere espacio adicional para acumular la Tabla de Índices de Color. En este esquema también el número de subpíxeles es proporcional al número de colores en la imagen secreta como en los esquemas de Verheul y Van Tilborg [57] y el de Yang y Laih [63]. Cuanto más colores haya en la imagen secreta, mayor será el tamaño de las acciones. Para superar esta limitación, Chang et al. [8] desarrollaron un esquema de compartición de imágenes secretas en color basado en la criptografía visual modificada. Este esquema proporciona una forma más eficiente de ocultar una imagen gris en diferentes acciones. En este esquema el tamaño de las acciones es fijo; no varía cuando el número de colores que aparecen en la imagen secreta es diferente. El esquema no requiere ninguna tabla de índice de color predefinida. Aunque la expansión de píxeles es fija en [8], este esquema no es adecuado para la imagen secreta de color verdadero. Para compartir la imagen de color verdadero, Lukac y Plataniotis [31] introdujeron un esquema basado en el nivel de bits operando directamente en los planos de bits de la imagen secreta.

Para ocultar una imagen secreta de color en múltiples imágenes de color se desea que las imágenes de camuflaje generadas contengan menos ruido. Para ello, Youmaran et al. [64] inventaron un criptografía visual mejorada para ocultar una imagen de color en múltiples imágenes de cobertura de color. Este esquema proporciona una mejora en la relación señal/ruido de las imágenes de camuflaje, produciendo imágenes con una calidad similar a las originales. Para reducir la expansión de los píxeles en el esquema de criptografía visual en color, Shyu [48] aconsejó un esquema de compartición de secretos visuales en color más eficiente con una expansión de píxeles de $|\log_2(c * m)|$, donde m es la expansión de píxeles del esquema binario explo-

tado. Considerando la transmisión de imágenes en color a través de canales con limitaciones de ancho de banda, Heidarinejad et al. [21] inventaron un esquema de criptografía visual rentable. La solución ofrece una reconstrucción perfecta al tiempo que produce acciones con un tamaño menor que el de la imagen de entrada utilizando la máxima distancia separable. Este esquema proporciona una expansión de píxeles inferior a uno. Para mejorar la velocidad de codificación Zhang et al. [65] presentaron una codificación multi-pixel que puede codificar un número variable de píxeles para cada ejecución. Liu et al. [30] desarrollaron un esquema de criptografía visual en color bajo el modelo de criptografía visual de Naor y Shamir sin expansión de píxeles. En este esquema el aumento del número de colores de la imagen secreta recuperada no aumenta la expansión de píxeles. Qiao et al. [43] sugirieron un esquema de criptografía visual para imágenes en color basado en la técnica de medios tonos. Un esquema de intercambio de imágenes secretas para imágenes secretas de color verdadero ideado por Tsai et al. [54]. En el esquema propuesto, mediante la combinación de redes neuronales y la variante de compartir el secreto visual, la calidad de la imagen secreta reconstruida y las imágenes de camuflaje son visualmente iguales a las correspondientes imágenes originales.

Compartición de múltiples secretos

Chen et al. [10] anticipó una criptografía visual multi-secretos que se extiende desde la compartición visual de secretos tradicional. El libro de códigos de la compartición visual de secretos tradicional se implementa para generar imágenes compartidas macro bloque por macro bloque de tal forma que las múltiples imágenes secretas se convierten en sólo dos imágenes compartidas y se decodifican todos los secretos uno por uno apilando dos de las imágenes compartidas de forma desplazada. Este esquema se puede utilizar para múltiples imágenes secretas binarias, grises y de color con una expansión de píxeles de 4.

Wang et al. [58] proporcionaron una construcción general para esquemas de criptografía visual extendida utilizando el algoritmo de extensión de la matriz. Se sugirió un método de construcción general para imágenes secretas simples o múltiples y binarias, en escala de grises y en color, utilizando la extensión de la matriz con acciones significativas. Utilizando el algoritmo de extensión matricial, cualquier esquema de criptografía visual existente con acciones de aspecto aleatorio puede modificarse fácilmente para utilizar acciones significativas.

7. Agradecimientos

En primer lugar, me gustaría dar las gracias a mi director de tesis, el profesor del departamento de matemáticas Carlos Galindo Pastor[1]. El correo del profesor Galindo siempre estaba disponible cuando me encontraba con algún problema o tenía alguna duda sobre mi investigación o redacción. Permitió que este documento fuera mi propio trabajo, pero al mismo tiempo me guió por el camino correcto siempre que necesité ayuda.

Por último, debo expresar mi más sincero agradecimiento a mi familia por haberme brindado un apoyo infalible y un estímulo continuo durante mis años de estudiante y, como última instancia, con este enriquecedor estudio e investigación del proceso de esta tesis de máster. Su realización probablemente se habría vuelto ardua sin ellos. Gracias.

Referencias

[1] L. M. Adleman y M. A. Huang. *Primality testing and abelian varieties over finite fields*. Springer, 2006.

[2] M. Artjuhov. "Certain criteria for primality of numbers connected with the little Fermat theorem". En: *Acta Arith.* 12.355-364 (1966), pág. 67.

[3] M. Bellare, R. Canetti y H. Krawczyk. "Keying hash functions for message authentication". En: *Annual International Cryptology Conference*. Springer. 1996, págs. 1-15.

[4] J. Benvenuto. "Galois field in cryptography". En: *University of Washington* 1.1 (2012), págs. 1-11.

[5] A. Berent. "Advanced Encryption Standard by Example". En: *Document available at URL http://www. networkdls. com/Articles/AESby Example. pdf (April 1 2007) Accessed: June* (2013).

[6] C. Chang, J. Chuang y P. Lin. "Sharing a secret two-tone image in two gray-level images". En: *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*. Vol. 2. IEEE. 2005, págs. 300-304.

[1] `http://www3.uji.es/~galindo/`

[7] C. Chang, C. Tsai y T. Chen. "A new scheme for sharing secret color images in computer network". En: *Proceedings Seventh International Conference on Parallel and Distributed Systems (Cat. No. PR00568)*. IEEE. 2000, págs. 21-27.

[8] C. Chang y T. Yu. "Sharing a secret gray image in multiple images". En: *First International Symposium on Cyber Worlds, 2002. Proceedings.* IEEE. 2002, págs. 230-237.

[9] T. Chen, K. Tsao y K. Wei. "Multiple-image encryption by rotating random grids". En: *2008 Eighth International Conference on Intelligent Systems Design and Applications*. Vol. 3. IEEE. 2008, págs. 252-256.

[10] T. Chen, K. Tsao y C. Wu. "Multi-secrets visual secret sharing". En: *2008 14th Asia-Pacific Conference on Communications*. IEEE. 2008, págs. 1-5.

[11] A. Chmielowiec. "Primality proving with Gauss and Jacobi sums". En: *J. Telec. and Inf. Tech.* 4 (2004), págs. 69-75.

[12] W. Diffie y M. E. Hellman. "New directions in cryptography". En: *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*. 2022, págs. 365-390.

[13] ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". En: *IEEE Trans. Inf. Th.* 31.4 (1985), págs. 469-472.

[14] L. Fang y B. Yu. "Research on pixel expansion of (2, n) visual threshold scheme". En: *2006 First International Symposium on Pervasive Computing and Applications*. IEEE. 2006, págs. 856-860.

[15] W. Fang. "Non-expansion visual secret sharing in reversible style". En: *Int. J. Comp. Sci. and Net. Sec.* 9.2 (2009), págs. 204-208.

[16] W. P. Fang. "Visual Cryptography in reversible style". En: *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP 2007)*. Vol. 1. IEEE. 2007, págs. 519-524.

[17] J. Feng, H. Wu, C. Tsai, Y. Chang e Y. Chu. "Visual secret sharing for multiple secrets". En: *Pattern Recognit.* 41.12 (2008), págs. 3572-3581.

[18] Z. Fu y B. Yu. "Research on rotation visual cryptography scheme". En: *2009 International Symposium on Information Engineering and Electronic Commerce*. IEEE. 2009, págs. 533-536.

[19] P. Gauravaram. "Cryptographic hash functions: cryptanalysis, design and applications". Tesis doct. Queensland University of Technology, 2007.

[20] S. Goldwasser y J. Kilian. "Almost all primes can be quickly certified". En: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. 1986, págs. 316-329.

[21] M. Heidarinejad, A. A. Yazdi y K. N. Plataniotis. "Algebraic visual cryptography scheme for color images". En: *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2008, págs. 1761-1764.

[22] H. Hsu, T. Chen e Y. Lin. "The ringed shadow image technology of visual cryptography by applying diverse rotating angles to hide the secret sharing". En: *IEEE International Conference on Networking, Sensing and Control, 2004*. Vol. 2. IEEE. 2004, págs. 996-1001.

[23] R. Jain, R. Jejurkar, S. Chopade, S. Vaidya y M. Sanap. "AES algorithm using 512 bit key implementation for secure communication". En: *Int. J. Innov. Res. Comput. Commun. Eng* 2.3 (2014), págs. 3516-3522.

[24] R. Kaur y A. Kaur. "Digital signature". En: *2012 International Conference on Computing Sciences*. IEEE. 2012, págs. 295-301.

[25] D. Knuth. *Seminumerical algorithms. The Art of Computer Programming*. 1969.

[26] N. Koblitz. *A course in number theory and cryptography*. Vol. 114. Springer Science & Business Media, 1994.

[27] A. G. Konheim. *Cryptography, a primer*. John Wiley & Sons, Inc., 1981.

[28] U. Kretzschmar. "Aes128–ac implementation for encryption and decryption". En: *TI-White Paper* (2009).

[29] H. Lee, K. Lee e Y. Shin. "Aes implementation and performance evaluation on 8-bit microcontrollers". En: *arXiv:0911.0482* (2009).

[30] F. Liu, C. K. Wu y X. J. Lin. "Colour visual cryptography schemes". En: *IET Inf. Sec.* 2.4 (2008), págs. 151-165.

[31] R. Lukac y K. N. Plataniotis. "Bit-level based secret sharing for image encryption". En: *Pattern Recognit.* 38.5 (2005), págs. 767-772.

[32] R. C. Merkle. "One way hash functions and DES". En: *Conference on the Theory and Application of Cryptology*. Springer. 1989, págs. 428-446.

[33] R. C. Merkle. *Secrecy, authentication, and public key systems*. Stanford University, 1979.

[34] G. L. Miller. "Riemann's Hypothesis and Tests for Primality". En: *J. Comp. Sys. Sci.* 13 (1976), págs. 300-317.

[35] A. A. Mohamed y A. H. Madian. "A Modified Rijndael Algorithm and it's Implementation using FPGA". En: *2010 17th IEEE International Conference on Electronics, Circuits and Systems*. IEEE. 2010, págs. 335-338.

[36] F. Morain. "Primality proving using elliptic curves: an update". En: *International Algorithmic Number Theory Symposium*. Springer. 1998, págs. 111-127.

[37] A. Nadeem y M. Y. Javed. "A performance comparison of data encryption algorithms". En: *2005 International Conference on Information and Communication Technologies*. IEEE. 2005, págs. 84-89.

[38] M. Naor y A. Shamir. "Visual cryptography". En: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1994, págs. 1-12.

[39] M. Naor y M. Yung. "Universal one-way hash functions and their cryptographic applications". En: *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*. 1989, págs. 33-43.

[40] J. M. Pollard. "Theorems on factorization and primality testing". En: *Math. Proc. Cambridge Phil. Soc.* Vol. 76. 3. Cambridge University Press. 1974, págs. 521-528.

[41] N. Pramstaller, F. K. Gurkaynak, S. Haene, H. Kaeslin, N. Felber y W. Fichtner. "Towards an AES crypto-chip resistant to differential power analysis". En: *Proceedings of the 30th European Solid-State Circuits Conference*. IEEE. 2004, págs. 307-310.

[42] B Preneel, J Vandewalle y B. Van Rompay. *Analysis and desing of cryptographic hash functions, MAC algorithms and block ciphers*. 2004.

[43] W. Qiao, H. Yin y H. Liang. "A kind of visual cryptography scheme for color images based on halftone technique". En: *2009 International Conference on Measuring Technology and Mechatronics Automation*. Vol. 1. IEEE. 2009, págs. 393-395.

[44] M. O. Rabin. "Probabilistic algorithm for testing primality". En: *J. Num. Th.* 12.1 (1980), págs. 128-138.

[45] R. L. Rivest, A. Shamir y L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". En: *Comm. ACM* 21.2 (1978), págs. 120-126.

[46] R. Schoof. "Elliptic curves over finite fields and the computation of square roots mod p". En: *Math. Comp.* 44.170 (1985), págs. 483-494.

[47] N. Selmane, S. Guilley y J. Danger. "Practical setup time violation attacks on AES". En: *2008 Seventh European Dependable Computing Conference*. IEEE. 2008, págs. 91-96.

[48] S. J. Shyu. "Efficient visual secret sharing scheme for color images". En: *Pattern Recognit.* 39.5 (2006), págs. 866-880.

[49] S. J. Shyu, S. Huang, Y. Lee, R. Wang y K. Chen. "Sharing multiple secrets in visual cryptography". En: *Pattern Recognit.* 40.12 (2007), págs. 3633-3651.

[50] S. Singh. *The code book: the science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2011.

[51] R. Sobti y G. Geetha. "Cryptographic hash functions: a review". En: *Int. J. Comp. Sci. Iss. (IJCSI)* 9.2 (2012), pág. 461.

[52] W. Stallings. *Cryptography and network security, 4/E*. Pearson Education India, 2005.

[53] D. R. Stinson. *Cryptography: theory and practice*. Chapman y Hall/CRC, 2005.

[54] D.U Tsai, G. Horng, T. Chen e Y. Huang. "A novel secret image sharing scheme for true-color images with size constraint". En: *Inf. Sci.* 179.19 (2009), págs. 3247-3254.

[55] G. Tsudik. "Message authentication with one-way hash functions". En: *ACM SIGCOMM Comp. Commun. Rev.* 22.5 (1992), págs. 29-38.

[56] M. Ulutas, R. Yazici, V. V. Nabiyev y G. Ulutas. "(2, 2)-Secret Sharing scheme with improved share randomness". En: *2008 23rd International Symposium on Computer and Information Sciences*. IEEE. 2008, págs. 1-5.

[57] E. R. Verheul y H. CA. Van Tilborg. "Constructions and properties of k out of n visual secret sharing schemes". En: *Des., Codes and Crypt.* 11.2 (1997), págs. 179-196.

[58] D. Wang y X. Yi F.and Li. "On general construction for extended visual cryptography schemes". En: *Pattern Recognit.* 42.11 (2009), 3071-3082.

[59] J. Weir y W. Yan. "Sharing multiple secrets using visual cryptography". En: *2009 IEEE International Symposium on Circuits and Systems*. IEEE. 2009, págs. 509-512.

[60] C. Wu y L.H. Chen. "A study on visual cryptography". Tesis doct. Master Thesis, Institute of Computer e Information Science, National Chiao., 1998.

[61] H. Wu y C. Chang. "Sharing visual multi-secrets using circle shares". En: *Comp. Stand. & Interfaces.* 28.1 (2005), págs. 123-135.

[62] T. Xiao-Qing. "Two kinds of ideal contrast visual cryptography schemes". En: *2009 International Conference on Signal Processing Systems*. IEEE. 2009, págs. 450-453.

[63] C.g Yang y C. Laih. "New colored visual secret sharing schemes". En: *Des., Codes and Crypt.* 20.3 (2000), págs. 325-336.

[64] R. Youmaran, A. Adler y A. Miri. "An improved visual cryptography scheme for secret hiding". En: *23rd Biennial Symposium on Communications, 2006*. IEEE. 2006, págs. 340-343.

[65] H. Zhang, X. Wang, W. Cao e Y. Huang. "Visual Cryptography for general access structure by Multi-pixel Encoding with Variable Block Size". En: *2008 International Symposium on Knowledge Acquisition and Modeling*. IEEE. 2008, págs. 340-344.