Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

CRIPTOGRAFÍA POSTCUÁNTICA

Autora: Marina Rodríguez Rodríguez
Tutor: Jose Ramón Brox López
Año 2024

ABSTRACT

The rapid advancement of quantum computing poses significant threats to current cryptographic systems, particularly those relying on classical public-key algorithms such as RSA. This constitutes a serious problem for modern digital communications, financial transactions, sensitive data, national security systems, healthcare records, intellectual property... Post-quantum cryptography (PQC) seeks to develop cryptographic protocols resistant to quantum attacks, ensuring data security in a quantum-enabled future. This paper presents an analysis of various PQC algorithms, including lattice-based, code-based, hash-based and multivariate polynomial. We analyze their theoretical foundations, security assumptions, and efficiency. Additionally, we provide a comparative study of these algorithms, highlighting their strengths and weaknesses in practical implementations. With this, we will provide a complete and detailed view of the current state of post-quantum cryptography.

Keywords: public key cryptosystem, digital signature, complexity, reduction, postquantum cryptography, multivariate cryptography, error-correcting codes, lattices.

RESUMEN

El rápido avance de la computación cuántica plantea amenazas significativas para los sistemas criptográficos actuales, particularmente aquellos que dependen de algoritmos de clave pública clásicos como RSA. Esto constituye un serio problema para las comunicaciones digitales modernas, las transacciones financieras, los datos sensibles, los sistemas de seguridad nacional, los registros de salud, la propiedad intelectual... La criptografía postcuántica busca desarrollar protocolos criptográficos resistentes a los ataques cuánticos. En este trabajo se presentan las distintas ramas en las que se está trabajando: sistemas criptográficos basados en retículos, códigos correctores de errores, funciones hash y polinomios multivariable. Analizamos sus fundamentos teóricos, su seguridad y su eficiencia. Además, proporcionamos un estudio comparativo de estos algoritmos, destacando sus fortalezas y debilidades en implementaciones prácticas. Con esto, daremos una visión completa y detallada del estado actual de la criptografía postcuántica.

Palabras clave: criptosistema de clave pública, firma digital, complejidad, reducción, criptografía postcuántica, criptografía multivariable, códigos correctores, retículos.

Índice general

Capítulo 1

Introducción

Con motivo de la inminente amenaza que los ordenadores cuánticos suponen a la criptografía moderna, una gran comunidad internacional ha surgido para tratar de que nuestra infraestructura de clave pública pueda mantenerse intacta utilizando nuevas primitivas resistentes a la computación cuántica. Esta es un área de investigación muy activa; cada día se hacen nuevos avances, se plantean nuevas propuestas o se demuestra que viejas propuestas no son tan seguras como parecían.

Un criptosistema resistente a ordenadores cuánticos debe cumplir dos requerimientos:

1. Debe ser eficiente con el hardware actual.
2. Debe ser resistente a adversarios con ordenadores tanto clásicos como cuánticos.

Los principales criptosistemas propuestos hasta la fecha son: basados en retículos, basados en ecuaciones multivariable, basados en funciones hash y basados en códigos correctores.

Algunas instituciones importantes, como el Instituto Europeo de Normas de Telecomunicaciones (ETSI) o el Instituto Nacional de Estándares y Tecnología (NIST) están intensificando el esfuerzo para desarrollar estas tecnologías. En concreto, el NIST tiene un papel muy importante en la estandarización de la criptografía postcuántica. En 2016 planteó un concurso ([12]) en el que se aceptaron propuestas para algoritmos de cifrado de clave pública, firma digital e intercambio de claves resistentes a la computación cuántica. Los criterios incluían requisitos de seguridad y rendimiento. Se estableció finales de 2017 como fecha límite de presentación de los algoritmos y después las propuestas que cumplieron ciertos criterios preliminares estuvieron sujetas a 5 años de escrutinio público en el que podían ser sometidas a todo tipo de ataques antes de ser estandarizadas.

En 2018 se presentaron las 64 propuestas que pasaron a la primera ronda. Estas propuestas fueron: 26 basadas en retículos, 19 basadas en códigos correctores de errores, 9 basadas en ecuaciones multivariable y 10 de otras ideas (hash, isogenias...). Un año después se anunciaron los 26 algoritmos que pasaban a la segunda ronda. 12 de ellos eran de retículos, 7 de códigos correctores de errores, 4 de ecuaciones multivariable y 3 de otros. En 2020, el NIST anunció los 7 candidatos que pasaban a la ronda final o tercera ronda: 4 para clave pública o intercambio de claves y 3 para firma digital. Además, se elegían 8 candidatos de reserva, 5 para clave pública o intercambio de claves y 3 para firma digital. De los 4 finalistas de clave pública, 3 son basados en retículos y 1 está basado en códigos. De los 3 finalistas de firma digital, 2 están basados en retículos y 1 en esquemas multivariable.

Finalmente, los algoritmos ganadores fueron CRYSTALS-KYBER para el establecimiento de claves y CRYSTALS-Dilithium para firma digital, ambos de retículos. Estos algoritmos, así como algunos otros, serán detallados en las secciones posteriores.

La organización de este trabajo es la siguiente: comenzaremos definiendo conceptos básicos de computación cuántica y criptografía moderna, detallando los principales criptosistemas que se utilizan en la actualidad. Se ha incluido una sección en la que nos familiarizaremos con la reducción de un problema a otro y con las clases de complejidad, conceptos esenciales en criptografía. Después explicaremos el algoritmo de Shor, cuya implementación en un ordenador cuántico es la que pone en peligro un gran número de criptosistemas que se usan hoy en día. Con todas estas bases asentadas, pasaremos a explicar los cuatro campos principales en los que se está trabajando en criptografía postcuántica: funciones hash, ecuaciones multivariable, códigos correctores y retículos. El orden elegido es siguiendo el estándar del NIST, de menos a más seguros, y también iremos de menos a más en profundidad, detallando mucho más la criptografía basada en códigos y retículos.

Cabe destacar que la criptografía postcuántica es un tema de máxima actualidad; durante los meses de la realización de este trabajo se ha publicado un artículo ([13]) que habría podido suponer un gran cambio en la criptografía, pues parecía comprometer la seguridad de la criptografía basada en retículos. Esto ha causado un gran revuelo en la comunidad, aunque al final se ha detectado un error aparentemente insalvable en el artículo. No obstante, con este tipo de cosas nos damos cuenta de que en cualquier momento puede aparecer un artículo que cambie completamente la visión y las ideas que se tienen sobre cuál es el mejor enfoque para la criptografía postcuántica. Si cada mes tuviéramos que actualizar el trabajo o añadir un epílogo, siempre tendríamos cosas nuevas que contar. Es un campo muy prometedor con numerosas ideas que explotar, y el objetivo de este trabajo es dar una visión amplia de la criptografía postcuántica, relacionando las diferentes ramas en las que se está trabajando, mostrando las fortalezas y debilidades de cada una de ellas.

Capítulo 2

Complejidad computacional

La teoría de la complejidad computacional se centra en clasificar los problemas computacionales según la cantidad de recursos necesarios para resolverlos. Estos recursos pueden incluir tiempo (cuánto tarda un algoritmo en completarse), espacio (cuánta memoria requiere) y otros factores como el número de procesadores. Toda este capítulo se basa en el capítulo 9 del libro [3].

Para hablar de complejidad computacional es indispensable utilizar la notación O grande.

Definición 2.0.1. *Sea f una función que se va a estimar, ya sea una función real o compleja, y sea g, la función de comparación, una función real. Definamos ambas funciones en algún subconjunto ilimitado de los números naturales, y supongamos que $g(n)$ es estrictamente positiva para todos los valores suficientemente grandes de n. Se dice que*

$$f(n) = O(g(n)) \text{ cuando } n \to \infty \tag{2.1}$$

y se lee "$f(n)$ ***es O grande de*** *$g(n)$" si el valor absoluto de $f(n)$ es, a lo sumo, un múltiplo constante positivo de $g(n)$ para todos los valores suficientemente grandes de n. Es decir, $f(n) = O(g(n))$ si existe un número real positivo M y un número natural n_0 tales que*

$$|f(n)| \leq Mg(n) \quad \text{para todo} \quad n \geq n_0. \tag{2.2}$$

Normalmente se escribe simplemente que

$$f(n) = O(g(n)). \tag{2.3}$$

Es decir, es una forma de describir cómo crece el tiempo de ejecución o el uso de memoria a medida que los datos de entrada aumentan.

2.1. Clases de complejidad

La criptografía moderna hasta ahora ha dependido de problemas matemáticos que sean virtualmente imposibles de resolver en un tiempo computacionalmente razonable con los medios y los algoritmos actuales en un ordenador clásico, como por ejemplo la factorización o el problema del logaritmo discreto. No existen algoritmos clásicos conocidos que produzcan la solución de estos problemas en tiempo razonable para datos de entrada lo suficientemente grandes.

En computación se entiende como tiempo de ejecución razonable el tiempo polinómico:

Definición 2.1.1. *Decimos que un algoritmo se ejecuta en* ***tiempo polinómico*** *si existe una constante k y una función polinómica $p(n)$ tal que el tiempo de ejecución del algoritmo en cualquier instancia de tamaño n es $O(p(n))$, donde $p(n) = n^k$ para algún $k \geq 0$.*

Pero un problema que solo se puede resolver en un tiempo superior al polinómico, por ejemplo, un algoritmo que tenga tiempo de ejecución exponencial, se considera un problema difícil, y por tanto es útil para la criptografía. Nótese que a la hora de seleccionar un algoritmo, no solo debemos considerar su tiempo computacional, sino también su complejidad espacial, es decir, cuánta memoria usa.

Para clasificar los problemas en función de su complejidad, se utilizan las clases P, NP, NP-completo y NP-hard. Veamos la definición de cada uno de ellos. Antes es necesario definir el siguiente concepto: una **máquina de Turing** es un modelo matemático abstracto de un dispositivo de cómputo que formaliza el concepto intuitivo de un algoritmo o procedimiento computacional. Fue propuesta por Alan Turing en 1936 para establecer un estándar teórico para la computabilidad y la complejidad de los problemas.

La clase de problemas que se pueden resolver en un tiempo polinómico por una máquina de Turing determinista se denominan **P**. Otra clase de complejidad importante es la clase **NP** (viene de polinómico no determinista). Esta es la clase de problemas en los cuales la solución se puede verificar[1] en un tiempo polinómico mediante una máquina de Turing determinista pero encontrar esa solución no necesariamente se podrá realizar en un tiempo razonable. En otras palabras, *pueden ser muy difíciles de resolver pero son fáciles de verificar.*

Los problemas **NP-completos** son los problemas tales que cualquier otro problema de NP se puede reducir a ellos en tiempo polinómico. Por lo tanto, si se puede resolver eficientemente cualquiera de los problemas de NP-completos, se puede resolver eficientemente cualquier problema de NP también. Los problemas que son igual o más difíciles que los NP-completos se denominan **NP-hard**. Son al menos tan difíciles como cualquier problema en NP, pero no necesariamente pertenecen a NP.

Hay un detalle técnico que hay que tener en cuenta: cuando hablamos de estas clases de problemas, pueden ser problemas de conteo o de decisión. Un problema de decisión es un tipo específico de problema computacional que tiene una respuesta binaria: sí o no. Es decir, para cada instancia del problema, la respuesta es "sí" si cierta propiedad se cumple o "no" si no se cumple. Un problema de conteo es un tipo de problema en el que el objetivo no es simplemente decidir si se cumple cierta propiedad como en los problemas de decisión, sino contar el número exacto de soluciones que satisfacen la propiedad.

Ejemplo de problema: factorización de números enteros.

Versión decisional: dado un número entero N, ¿existe un divisor no trivial de N (un divisor que no sea 1 ni N)? La entrada será un número entero N, y la salida será "sí" si existe un divisor no trivial de N o "no" en caso contrario.

Versión de conteo: dado un número entero N, ¿cuántos divisores no triviales tiene N? La entrada de nuevo será un número entero N, y la salida será un número entero que representa el total de divisores no triviales de N (divisores que no sean ni 1 ni N).

De forma rigurosa, cuando hablamos de problemas de conteo asociados a los problemas de decisión se debería utilizar #P y #NP, aunque en muchos textos se hace un pequeño abuso de lenguaje y se utiliza directamente P y NP, cosa que haremos también en este trabajo.

[1] Verificar significa, dada una potencial solución, ejecutar algún algoritmo de tiempo polinómico que confirme si esa es una solución real o no.

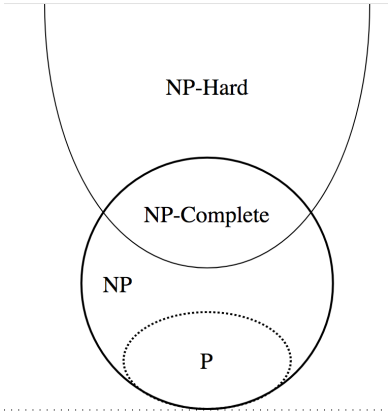

Figura 2.1: Las clases NP, P, NP-hard y el conjunto de problemas NP-completos.

Entonces, si pudiéramos resolver alguno de los NP-completos en tiempo polinómico, podríamos hacer lo mismo con todos los problemas NP, y por tanto NP sería igual a P. Este es uno de los problemas del milenio. Pero, aunque no se ha podido probar, el consenso en teoría de la computación en general y en criptografía en particular tiende a que $P \neq NP$.

El problema de la factorización que se usa en criptografía es, dado N tal que $N = pq$ con p y q primos, encontrar dichos primos. Este problema es NP pero se cree que no es NP-completo. No existe una prueba matemática de esta última afirmación, pero hay evidencias que nos indican que debe ser así. En primer lugar, todos los problemas NP-completos que se conocen pueden tener una solución, pero también pueden tener más de una, o ninguna. En cambio, el problema de la factorización tiene exactamente una solución siempre. Además, la estructura matemática de este problema permite que algoritmos como el GNFS mejoren considerablemente el tiempo de ejecución con respecto al algoritmo directo de prueba y error. El resto de problemas NP-completos conocidos no tienen esta estructura.

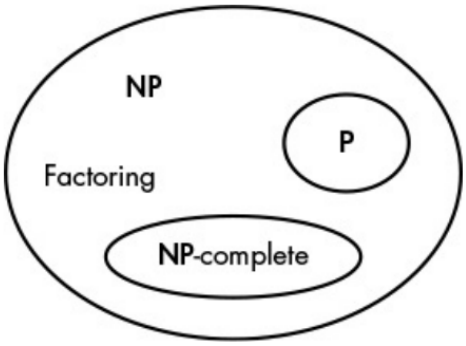

Figura 2.2: Las clases NP, P y el conjunto de problemas NP-completos. [3]

Por el mismo razonamiento, el DLP (problema del logaritmo discreto) tampoco se cree que sea NP-completo.

Un ordenador cuántico puede factorizar muy fácilmente usando el algoritmo de Shor, pero se cree que no podría resolver los problemas NP-completos. Es por eso que esta clase de problemas está siendo muy estudiada para la criptografía postcuántica. Por ejemplo, algunas variantes de problemas de retículos como el SVP y el LWE que veremos en las secciones posteriores son problemas NP-hard que ya han sido

instaurados con éxito en criptografía. Los problemas multivariable (que consisten en resolver sistemas de ecuaciones no lineales) también son problemas NP-completos y también los veremos en secciones posteriores.

En esta sección hemos mencionado la importancia que tienen los resultados sobre clases de complejidad para garantizar la seguridad en diferentes esquemas criptográficos. Pero también puede dar el caso contrario: que gracias a la criptografía, se obtenga un resultado matemático sobre clases de complejidad. Por ejemplo, esto se puede ver en [28], en el que usando criptografía, asumiendo un resultado que se cree que debe ser cierto, llega a demostrar que entonces $P = BPP$, siendo BPP (Bounded Probability Polynomial time), es decir, es la clase de problemas que se pueden resolver con algoritmos en tiempo polinómico, pero además usan aleatoriedad, y requerimos que sean correctos con una alta probabilidad. No entraré en los detalles de esto, pero muestra que los avances en criptografía también pueden ayudar a obtener resultados en teoría de la complejidad.

2.2. Reducción de un problema a otro

En primer lugar vamos a dar una noción de la idea de reducción, ya que es algo muy utilizado para determinar qué criptosistemas pueden ser resistentes y cuáles no. Una reducción es un procedimiento o un algoritmo que transforma un problema en otro. Es decir, diremos que el problema A es reducible al problema B si existe un algoritmo que nos permita convertir el problema A en el problema B. Esto es útil cuando queremos resolver A u obtener información sobre su dificultad, y ya sabemos resolver o tenemos más información sobre B.

Se puede emplear una reducción suficientemente eficiente de A en B para demostrar que B es, al menos, tan difícil como A, ya que si B fuera fácil (se puede resolver en tiempo polinómico), entonces también podríamos resolver A en tiempo polinómico usando la reducción. Por tanto, si podemos establecer esta reducción, si *B es fácil, entonces A es fácil*. También es muy útil pensar en el contrarrecíproco: si se puede establecer esta reducción, *si A no es fácil, entonces B no es fácil*, o en otras palabras, *si A es difícil, B es difícil*. Esta idea es clave, ya que la seguridad de todo criptosistema se basa en la dificultad de algún problema matemático. Por lo tanto, si podemos reducir un problema conocido que sea demostradamente difícil a romper un esquema criptográfico, entonces podemos deducir que romper el esquema es al menos tan difícil como resolver el problema difícil. Esto proporciona una garantía de seguridad basada en la dificultad de problemas bien estudiados.

Por lo tanto, si podemos reducir A en B de manera eficiente, entonces deducimos dos ideas clave:

1. Si B es fácil, entonces A es fácil.
2. Si A es difícil, entonces B es difícil.

Para determinar la complejidad de un algoritmo, es necesario especificar las condiciones bajo las cuales se calcula dicha complejidad. En la teoría de la complejidad computacional, es común considerar la complejidad del caso peor, que consiste en seleccionar la instancia del problema que requiere más tiempo de ejecución entre todas las posibles, y utilizar su costo como medida de complejidad. Sin embargo, también es posible analizar el caso medio, que consiste en un caso aleatorio o la media de todos los casos.

En el ámbito de la criptografía moderna, al evaluar la capacidad de un algoritmo para proporcionar seguridad en la encriptación, no tiene sentido basarse únicamente en la complejidad del caso peor. Al encriptar datos, las claves se eligen al azar, lo que significa que es mucho más probable encontrarse con el

caso medio que con el caso peor. Por lo tanto, en el contexto de la criptografía, es fundamental comprender la complejidad del caso medio y estudiar la reducción entre casos peores de un problema a casos promedios de otro, cosa que veremos en capítulos posteriores.

Capítulo 3

Fundamentos de la computación cuántica

Capítulo basado en [62]. Un ordenador cuántico utiliza las leyes de la física cuántica para su funcionamiento. Los ordenadores clásicos utilizan bits que pueden estar en uno de dos estados: 0 o 1. En contraste, los ordenadores cuánticos utilizan qubits, que pueden estar en una superposición de ambos estados, es decir, en una combinación lineal compleja de los estados $|0\rangle$ y $|1\rangle$,

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix}\alpha_0\\ \alpha_1\end{pmatrix} \tag{3.1}$$

donde los coeficientes están sujetos a la condición de normalización $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Se dice en este caso que el estado $|\psi\rangle$ es una superposición de los estados $|0\rangle$ y $|1\rangle$ con amplitudes α_0 y α_1.

Para conocer el estado de un bit, basta con mirarlo y ver si está en el estado $|0\rangle$ o en el estado $|1\rangle$; *medir* el estado en el que está el bit no lo altera. En cambio, si tenemos un qubit en el estado 3.1, no hay ningún modo mediante el cual podamos conocer cuál es el estado del qubit; es decir, no podemos conocer las amplitudes α_0 y α_1. Si intentamos medir ese estado, el estado del qubit cambia: cada vez que medimos un qubit, automáticamente se convierte en un bit. Una **puerta de medida** es un proceso que, aplicado a un qubit, nos da el resultado $|0\rangle$ o $|1\rangle$. Las amplitudes nos determinan la probabilidad de obtener un resultado u otro: $|\alpha_0|^2$ es la probabilidad de que, tras medir el qubit, se encuentre en el estado $|0\rangle$, y análogamente para $|\alpha_1|^2$. La idea esencial es la siguiente: *Las amplitudes no son la probabilidad de que esté en un estado u otro. Un qubit está en ambos estados a la vez. Las amplitudes son la probabilidad de que* ***al medirlo*** *nos dé un estado u otro.* De hecho, el poder de los ordenadores cuánticos se debe al hecho de que un sistema puede estar en muchos estados al mismo tiempo.

Si tenemos un sistema con n qubits, este sistema se describe mediante un espacio de Hilbert[1] de dimensión 2^n. Esto significa que el estado del sistema completo puede ser representado como una superposición (es decir, una combinación lineal) de 2^n estados básicos posibles:

[1]Los estados cuánticos son vectores en el espacio de Hilbert, dotado de un producto escalar, que permite calcular la probabilidad de transición entre estados. Para dos estados $|\psi\rangle$ y $|\phi\rangle$, el producto escalar se denota como $\langle\psi|\phi\rangle$ y satisface varias propiedades importantes, como la linealidad, la conmutatividad conjugada ($\langle\psi|\phi\rangle = \langle\phi|\psi\rangle^*$) y la positividad ($\langle\psi|\psi\rangle \geq 0$). Además, la norma de un vector de estado $|\psi\rangle$ se define como $\| |\psi\rangle \| = \sqrt{\langle\psi|\psi\rangle}$. Como ya hemos dicho, los estados cuánticos están normalizados, es decir, $\langle\psi|\psi\rangle = 1$.

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle. \tag{3.2}$$

Los coeficientes complejos α_i de nuevo satisfacen la condición de normalización:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1. \tag{3.3}$$

Cada $|i\rangle$ es uno de los estados base donde i toma valores de 0 a $2^n - 1$; estos estados conforman la base computacional o base clásica, que es ortonormal: está formada por los estados $|i\rangle$. Cada $|i\rangle$ es una cadena binaria de longitud n formada por los estados individuales $|0\rangle$ y $|1\rangle$, es decir, se pueden representar en notación binaria como $|b_1 b_2 \cdots b_n\rangle$, donde $b_i \in \{0, 1\}$; representa el producto tensorial de n estados básicos $|0\rangle$ o $|1\rangle$.

Ejemplo de 2 qubits

Supongamos dos qubits: el primero en el estado $|u\rangle = \beta_0|0\rangle + \beta_1|1\rangle$ y el segundo en el estado $|v\rangle = \gamma_0|0\rangle + \gamma_1|1\rangle$.

El estado conjunto de estos dos qubits se describe mediante el producto tensorial:

$$|u\rangle \otimes |v\rangle = (\beta_0|0\rangle + \beta_1|1\rangle) \otimes (\gamma_0|0\rangle + \gamma_1|1\rangle) = \beta_0\gamma_0|00\rangle + \beta_0\gamma_1|01\rangle + \beta_1\gamma_0|10\rangle + \beta_1\gamma_1|11\rangle \tag{3.4}$$

donde $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ son los estados de la base computacional para dos qubits.

Sin embargo, no todos los estados de dos qubits pueden escribirse como productos tensoriales de los estados de los qubits individuales. Un estado general de dos qubits es:

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle. \tag{3.5}$$

Para que este estado pueda ser expresado como un producto tensorial, debe cumplirse la condición:

$$\alpha_0\alpha_3 = (\beta_0\gamma_0)(\beta_1\gamma_1) = (\beta_0\gamma_1)(\beta_1\gamma_0) = \alpha_1\alpha_2. \tag{3.6}$$

Si esta condición no se cumple, el estado $|\psi\rangle$ es un **estado entrelazado**. El entrelazamiento cuántico ocurre cuando el estado del sistema no puede descomponerse en estados individuales de sus componentes; si los separamos y medimos solo uno de ellos, también alteraremos el estado del otro qubit que estaba entrelazado aunque no midamos este directamente. En los sistemas clásicos, un sistema compuesto siempre puede ser descrito como un producto de los estados de sus componentes individuales, y estos son completamente independientes. El estado total del sistema es simplemente la combinación de cada uno de los bits individuales.

Actualmente, existen ordenadores cuánticos a una escala pequeña, con un número reducido de qubits, ya que mantener el entrelazamiento entre un mayor número de qubits de forma estable se vuelve considerablemente más complejo. Sin embargo, esto está cambiando a una velocidad vertiginosa. Además de

intentar aumentar el número de qubits para así tener más potencia de cálculo, la investigación también se está centrando en reducir el número de errores asociados al entrelazamiento. Para evitar estos errores asociados, se pueden implementar códigos correctores de errores en los ordenadores cuánticos, usando varios qubits físicos para almacenar la información de un único qubit teórico (a este conjunto se le llama qubit lógico). Los investigadores han estimado que las técnicas de corrección de errores actuales requerirán más de 1,000 qubits físicos por cada qubit lógico. Esto significa que una máquina útil necesitaría millones de qubits físicos. No obstante, se está trabajando en un esquema alternativo de corrección de errores llamado quantum low-density parity check (qLDPC), que podría reducir la tasa de error significativamente. El enfoque qLDPC requiere que cada qubit esté conectado directamente a al menos seis otros qubits ([10]). En 2024 han empezado a aparecer ordenadores cuánticos con corrección de errores, es decir, con qubits lógicos. El récord está actualmente en 48 qubits lógicos ([8]), conseguido utilizando 280 qubits físicos (esa tasa de la que hablábamos de 6 qubits físicos para cada qubit lógico). A medida que la tecnología avanza y los ordenadores cuánticos se hacen más y más potentes, cuando se desarrolle un ordenador cuántico a una escala suficientemente grande, debido al algoritmo de Shor, la criptografía moderna dejará de ser segura. De hecho, el algoritmo de Shor requiere aproximadamente $2n + 2$ qubits para factorizar un número de n bits ([26]). Por lo tanto, para desencriptar una clave RSA de 1024 bits, se necesitarían alrededor de 2050 qubits lógicos. A la velocidad a la que se están desarrollando los ordenadores cuánticos, se hace cada vez más urgente encontrar algoritmos resistentes a estos ordenadores en los que podamos basar la criptografía.

Un detalle esencial de los ordenadores cuánticos es que podemos transformar un estado en otro que nos "interese" más. Esto es precisamente lo que va a hacer el algoritmo de Shor. Supongamos que tenemos un estado cuántico que es superposición de varios estados.

$$|\psi\rangle = \alpha_0|\text{parte que nos interesa}\rangle + \alpha_1|\text{parte que no nos interesa}\rangle \tag{3.7}$$

Queremos obtener una información sobre la parte que nos interesa. Lo que se hace es transformar el estado 3.7, modificando sus amplitudes de forma que α_0 sea mucho mayor y α_1 mucho menor. Esto se consigue mediante la interferencia: cuando tenemos **interferencia constructiva**, las amplitudes de los estados se suman, y cuando tenemos **interferencia destructiva**, las amplitudes de los estados se cancelan entre sí. Esto se hace mediante la transformada de Fourier cuántica (QFT), que aprovecha la interferencia haciendo que sea constructiva en la parte que nos interesa y destructiva en la parte que no nos interesa. Con esto, al medir $|\psi\rangle$, obtendremos esa parte que nos interesa con mayor probabilidad, consiguiendo así la información que queremos. Se verá con más detalle en las secciones posteriores, mostrando que nos permite obtener una información que sirva para romper criptosistemas como RSA y Diffie-Hellman.

3.1. Algoritmo de Shor

En 1994, Peter Shor, profesor de Matemáticas del MIT, publicó un artículo ([56]) en el que había ideado un algoritmo cuántico de factorización de números grandes de forma mucho más eficiente que los algoritmos existentes para ordenadores clásicos. Esto cambió por completo el paradigma de la criptografía hasta el momento: un gran número criptografías de clave pública, tales como RSA, llegarían a ser obsoletas si el algoritmo de Shor es implementado alguna vez en una computadora cuántica práctica.

3.1.1. Funcionamiento del algoritmo de Shor

Este algoritmo fue publicado por Shor, aunque para la bibliografía también he utilizado otras fuentes en las que la explicación me resultaba más clara[2]. Como todos los algoritmos de computación cuántica, el algoritmo de Shor es probabilístico: da la respuesta correcta con una probabilidad de al menos $1 - \frac{1}{2^k}$

[2]En particular me he basado en el artículo de Wikipedia en inglés y en un vídeo divulgativo https://youtu.be/lvTqbM5Dq4Q especialmente para la parte de la subrutina cuántica.

donde k es el número de factores primos distintos del número, por lo que si $N = pq$, entonces $k = 2$ y la probabilidad es de al menos 3/4 ([45], Teorema 5.3). La probabilidad de fallo puede ser disminuida repitiendo el algoritmo.

Queremos encontrar un factor no trivial de N, siendo N un número compuesto. Una observación básica es que, si N es par, 2 es trivialmente un factor. Supondremos que N es impar. También tenemos que comprobar si N es una potencia de un número primo. Para potencias de un primo, existen algoritmos de factorización clásicos eficientes, por lo tanto, para el resto del algoritmo de Shor vamos a asumir también que N no es una potencia de un primo.

Elegimos un número entero a al azar con $2 \leq a < N$. Podemos encontrar un posible divisor no trivial de N realizando el $mcd(a, N)$ mediante el algoritmo de Euclides. Si esto produce un factor no trivial (es decir, $mcd(a, N) \neq 1$), el algoritmo está terminado, y el otro factor no trivial es $\frac{N}{mcd(a,N)}$. Si no obtenemos un factor no trivial, eso significa que N y nuestro a elegido son coprimos. Aquí, el algoritmo ejecuta la subrutina cuántica explicada en la sección 3.1.3, que devolverá el orden r de a, es decir,

$$a^r \equiv 1 \bmod N. \tag{3.8}$$

De (3.8) obtenemos que $N|a^r - 1$. Podemos reescribir esta condición como:

$$N \mid (a^{r/2} - 1)(a^{r/2} + 1). \tag{3.9}$$

Nótese que esto no implica que N divide a uno de los dos factores, y de hecho esto en la explicación se tendrán en cuenta las dos alternativas.

Además, dado que hemos separado la expresión de esta manera utilizando la diferencia de cuadrados, el algoritmo no funciona para r impares (porque $a^{r/2}$ debe ser un número entero), lo que significa que el algoritmo tendría que reiniciarse con un nuevo a. Supondremos entonces que r es par.

Como ya hemos dicho, de 3.9 podemos encontrarnos con tres alternativas:

1. $N \mid a^{r/2} - 1$.
2. $N \mid a^{r/2} + 1$.
3. Que N divida al producto de ambos pero no divida a ninguno de los dos.

EL algoritmo solo va a funcionar si estamos en el caso 3, ahora veremos por qué. Calculamos $d = mcd(N, a^{r/2} - 1)$.

- No puede darse el primer caso nunca, ya que esto implicaría que $a^{r/2} \equiv 1 \bmod N$, lo que implicaría contradictoriamente que $\frac{r}{2}$ sería el orden de a, pero hemos dicho que el orden de a es r.
- Si $d = 1$, entonces el caso 2 se cumple, entonces no podemos encontrar un factor no trivial de N. La razón de esto se debe a que en este caso, $a^{r/2} \equiv -1 \bmod N$. Esto no proporciona información útil para la factorización de N, ya que el objetivo es encontrar divisores de N que sean distintos de 1 y N. De nuevo habría que elegir un nuevo a y repetir el algoritmo.
- En caso contrario ($d \neq 1$), entonces se cumple el caso 3. Entonces d es un factor no trivial de N, siendo el otro $\frac{N}{d}$ y el algoritmo ha terminado.

Puede parecer que estos dos problemas mencionados dificultan mucho este algoritmo, pero como hemos dicho antes, para cualquier a elegido aleatoriamente la probabilidad de que ninguno de estos problemas aparezca es de al menos $3/4$. Es decir, con 4 elecciones aleatorias de a, tenemos más de 99 % de probabilidad de encontrar un a adecuado. Por lo tanto, este algoritmo factoriza con éxito después de pocas ejecuciones.

3.1.2. Ejemplo del algoritmo para $N = 15$

Vamos a ilustrar el algoritmo de factorización de Shor con un ejemplo usando números pequeños. Queremos factorizar $N = 15$.

1. En primer lugar, verificamos si N es par o una potencia de un número primo. Como es impar y no es una potencia de un número primo (no es de la forma p^k donde p es primo y k es un entero mayor que 1), continuamos.
2. Elegimos un número a al azar tal que $2 \leq a < N$. Supongamos que elegimos $a = 7$.
3. Calculamos el $\text{mcd}(a, N)$ usando el algoritmo de Euclides: $\text{mcd}(7, 15) = 1$, es decir, 7 y 15 son coprimos.
4. Ahora debemos encontrar el orden r de a, es decir, el menor entero r tal que $a^r \equiv 1 \bmod N$. Este paso se realiza utilizando la subrutina cuántica en el algoritmo de Shor, que detallaremos en la siguiente subsección. Para este ejemplo, supongamos que encontramos que el orden r de 7 mod 15 es 4. Es decir, $7^4 \equiv 1 \bmod 15$.
5. Ahora debemos comprobar si r es par. En nuestro caso, sí que lo es.
6. Calculamos $a^{r/2} \bmod N$: $a^{r/2} = 7^{4/2} = 7^2 = 49$, y $49 \bmod 15 = 4$. Entonces, $7^2 \equiv 4 \bmod 15$.
7. Ahora debemos comprobar las condiciones. Tenemos que $15 \mid (4 - 1)(4 + 1) = 3 \cdot 5$.
8. Por último, calculamos $\text{mcd}(N, a^{r/2} - 1) = \text{mcd}(15, 3) = 3$. Dado que $3 \neq 1$, hemos encontrado un factor no trivial de 15.

Resultado: $d = 3$ es un factor de 15. El otro factor es $N/d = 15/3 = 5$. Y por tanto hemos factorizado 15 en $3 \cdot 5$.

3.1.3. Subrutina cuántica

El objetivo de la subrutina cuántica del algoritmo de Shor es encontrar el orden r que cumpla 3.8. Este es el paso clave que una computadora clásica, al menos por ahora, no puede realizar en un tiempo razonable, pero una computadora cuántica sí. Se crea una superposición de todos los posibles valores de x en un registro cuántico y se inicializa así:

$$\frac{1}{\sqrt{Q}} \sum_{x=0}^{Q-1} |x\rangle \otimes |0\rangle \tag{3.10}$$

donde Q es un valor suficientemente grande, típicamente cercano a N^2. Se define la función $f(x) = a^x \bmod N$ y se aplica de la siguiente forma:

$$\frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\otimes|f(x)\rangle=\frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\otimes|a^x \bmod N\rangle. \tag{3.11}$$

Para entenderlo un poco mejor, vamos a verlo de la siguiente forma: lo que tenemos en 3.11 es una superposición de los $|x\rangle\otimes|a^x \bmod N\rangle$, y $a^x \bmod N$ no es otra cosa que el residuo o resto (que llamaremos b_x) de dividir a^x entre N. Es decir, lo que tenemos es una superposición de los $|x\rangle\otimes|b_x\rangle$.

$$\frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\otimes|b_x\rangle. \tag{3.12}$$

Buscamos r, y r será justamente el menor x que cumpla que su residuo sea =1 (pues el que tiene residuo 1 es el que cumple 3.8, o en otras palabras $a^r = m_r N + 1$). Por tanto, necesitamos efectuar una medida sobre la superposición que nos dé el estado $|x\rangle\otimes|1\rangle$, y ese x será el r. Sin embargo, efectuar una medida sobre esto nos va a dar uno de los estados pero no necesariamente el que nos interesa. La probabilidad de que al medir obtengamos uno de esos estados viene determinado, como ya sabemos, por las amplitudes que los acompañan. Por eso, nos interesa hacer que la amplitud del estado que nos interesa aumente y las demás disminuyan, aumentando las probabilidades de que al medir obtengamos $|r\rangle\otimes|1\rangle$.

Sabemos que $a^x = m_{x,0}N + b_x$. Aquí entra en juego la siguiente propiedad matemática:

$$a^x = m_{x,0}N + b_x \tag{3.13}$$
$$a^{x+r} = m_{x,1}N + b_x \tag{3.14}$$
$$a^{x+2r} = m_{x,2}N + b_x \tag{3.15}$$
$$\vdots \tag{3.16}$$
$$a^{x+tr} = m_{x,t}N + b_x \tag{3.17}$$
$$\vdots \tag{3.18}$$

Vemos que para un x, si sumamos múltiplos de r, el residuo siempre es el mismo. Y esta es la clave: la función $f(x) = a^x \bmod N$ es periódica con periodo r, donde r es el orden de a. Por lo tanto, en la superposición, vamos a tener muchos estados cuyo parte asociada al residuo va a ser igual, y cuya parte asociada a x va a estar separada una de la siguiente por r. Es decir,

$$|x\rangle\otimes|b_x\rangle, |x+r\rangle\otimes|b_x\rangle, |x+2r\rangle\otimes|b_x\rangle, ..., |x+tr\rangle\otimes|b_x\rangle... \tag{3.19}$$

Y esto así para cada x. Si tenemos una superposición y al hacer una medida obtenemos un valor que puede haber venido de varios estados de la superposición (es decir, obtenemos por ejemplo $|b_x\rangle$), el estado final será una superposición de todos estados asociados a ese $|b_x\rangle$, es decir, obtendremos una superposición de 3.19. Dado que esto se cumple para todo x, siempre obtendremos una superposición de estados cuyo primer ket del producto tensorial van a estar separados entre sí por r, es decir, el período es r, o en otras palabras, esos estados asociados a $|b_x\rangle$ aparecerán con una frecuencia $1/r$. Para trabajar con frecuencia, hay una herramienta matemática idónea: la transformada de Fourier. Eso es exactamente lo que se hace: aplicar la transformada de Fourier cuántica (QFT) sobre una superposición de todos esos estados $|x\rangle, |x+r\rangle, |x+2r\rangle, ..., |x+tr\rangle$.... Entonces aplicamos la transformada de Fourier cuántica (QFT) a 3.12:

$$\mathrm{QFT}\left(\frac{1}{\sqrt{Q}}\sum_{x=0}^{Q-1}|x\rangle\otimes|b_x\rangle\right) \tag{3.20}$$

Al aplicar la QFT, se transforma en una superposición de estados cuánticos que reflejan la periodicidad de la función $f(x)$. Matemáticamente, la QFT de una función periódica con período r produce picos en los múltiplos de $\frac{Q}{r}$. Entonces, la salida tras aplicar la QFT va a ser de la forma:

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left| c + k\frac{N}{r} \right\rangle \tag{3.21}$$

Después de la medición, se obtiene un valor c que es un múltiplo aproximado de $\frac{N}{r}$. La fracción $\frac{N}{r}$ representa la frecuencia de los picos en la distribución de la QFT. Ahora usamos el algoritmo de fracciones continuas, que toma como entradas c y N. La salida va a ser una fracción $\frac{k}{r}$ que aproxima $\frac{c}{N}$. Al ejecutar el algoritmo varias veces obtendremos $\frac{k_1}{r}$, $\frac{k_2}{r}$, $\frac{k_3}{r}$... Para suficientes valores de $\frac{k_i}{r}$ vamos a poder obtener el valor de r.

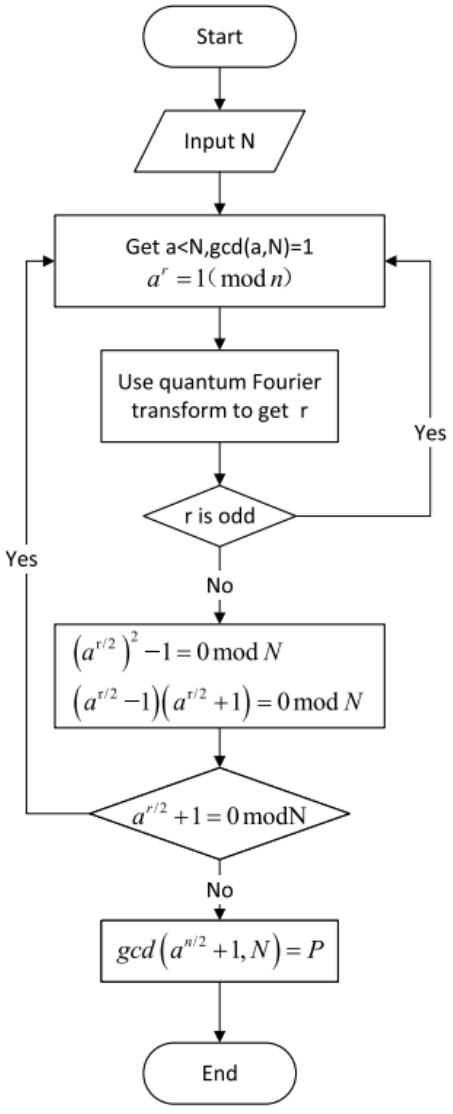

Figura 3.1: Esquema del algoritmo de Shor.([19])

Capítulo 4

Criptografía moderna

La criptografía consiste en la protección de la información y la comunicación empleando algoritmos matemáticos complejos. En este capítulo veremos brevemente cuáles son esos algoritmos y los conceptos básicos para entender cómo funciona la criptografía en la actualidad.

4.1. Conceptos básicos

El concepto más básico de la criptografía es el **cifrado**, que es el proceso de convertir información legible, que llamaremos texto plano, en un formato ilegible, que llamaremos texto cifrado, utilizando un algoritmo y una clave. El **descifrado** es el proceso inverso al cifrado, en el que el texto cifrado se convierte nuevamente en el texto plano utilizando la clave correspondiente.

Un algoritmo criptográfico nos permite realizar dichas operaciones de cifrado y descifrado. Hay dos tipos:

- Algoritmos simétricos: Utilizan la misma clave tanto para el cifrado como para el descifrado. En general rápidos y eficientes, su seguridad se basa en la longitud de la clave. Ejemplos: AES, DES.
- Asimétricos: Utilizan dos claves, una pública y otra privada. La clave pública se utiliza para el cifrado y la clave privada para el descifrado. Estos algoritmos son en general más lentos que los simétricos pero facilitan el intercambio seguro de claves. Ejemplos: RSA, ECC.

Hemos destacado la rapidez y eficiencia de los algoritmos simétricos, pero que ambas partes puedan acordar la clave puede ser un problema si el canal por el que se transmite la información no es seguro. Es por esto que, en la práctica, lo que se utiliza es un criptosistema **híbrido**, que va a recoger las ventajas de los algoritmos simétricos (su eficiencia) y de los asimétricos (su seguridad). Un criptosistema híbrido tiene dos partes: primero se utiliza la criptografía de clave pública para intercambiar una clave simétrica que llamaremos clave de sesión, y una vez que las dos partes la conocen, a partir de ese momento se comunicarán cifrando con esa clave que han intercambiado; al ser esta una clave simétrica, el cifrado va a ser más rápido. Este intercambio de la clave de sesión se consigue mediante un **KEM (Key Encapsulation Mechanism)**, que es un sistema criptográfico utilizado para encapsular una clave secreta y enviarla (cifrada) de forma segura a través de un canal no seguro. Más en detalle, un esquema híbrido funciona de la siguiente manera:

1. Se genera una clave simétrica, la clave de sesión.

2. Se encapsula esta clave utilizando un algoritmo KEM: se usa la clave pública del receptor para cifrar la clave de sesión. Esto produce un mensaje encapsulado que contiene la clave de sesión cifrada.

3. Por otro lado, el mensaje se cifra usando la clave de sesión.

4. Se envían las dos partes al receptor: la clave de sesión encapsulada (paso 2) y el mensaje cifrado (paso 3).

5. El receptor utiliza su clave privada para desencapsular y así recuperar la clave de sesión simétrica (el algoritmo KEM asegura que solo el receptor con la clave privada correcta puede desencapsular la clave).

6. Una vez que el receptor ha recuperado la clave de sesión, utiliza esta clave para descifrar los datos cifrados.

Para construir un criptosistema seguro, es fundamental utilizar una **función one-way** o función unidireccional. Estas funciones tienen la propiedad de ser fáciles de calcular en una dirección, pero extremadamente difíciles de revertir. En criptografía se necesitan funciones unidireccionales con una característica especial, llamada una *trapdoor*, que es un secreto que permite invertir la función de manera eficiente. Sin embargo, sin el conocimiento de la *trapdoor*, la función sigue siendo difícil de invertir. Por eso, el estudio de la complejidad es una parte esencial de la criptografía; necesitamos encontrar una función que al revertirla tenga una complejidad suficiente para que nuestro criptosistema pueda ser efectivo.

A su vez, es también fundamental utilizar generadores de números aleatorios (RNGs) que sean robustos y que cumplan con ciertos requisitos específicos para generar la clave y así garantizar la seguridad de los sistemas criptográficos.

Es importante poder cifrar y descifrar si queremos enviar un mensaje confidencial. Pero igual de importante es verificar la integridad del mensaje recibido y la identidad de nuestro interlocutor. Las **firmas digitales** se utilizan justamente para eso: para verificar la autenticidad e integridad de un mensaje. Normalmente utilizan algoritmos de clave pública para generar una firma que puede ser verificada por cualquier persona que tenga la clave pública correspondiente.

El **cifrado homomórfico** es una forma de cifrado que permite realizar cálculos en datos cifrados sin tener que descifrarlos primero. Si desciframos el texto cifrado sobre el que hemos aplicado dichas operaciones, obtendremos el mismo resultado que si las operaciones se hubieran realizado sobre los datos no cifrados. Este tipo de cifrados, desde los más antiguos hasta los más acuales, se basan en retículos.

Otro concepto importante es el **oráculo**, que es una herramienta teórica que facilita el análisis y la prueba de seguridad de algoritmos y sistemas criptográficos. Un oráculo puede entenderse como una "caja negra" que puede resolver instantáneamente un problema específico o una clase de problemas. El funcionamiento interno es desconocido o irrelevante para el análisis; solo se tiene en cuenta qué entradas acepta y qué salidas produce. El propósito es simplificar el análisis centrándose únicamente en la relación entre las entradas y las salidas sin preocuparse por cómo se producen esas salidas internamente. Por lo tanto, esta herramienta nos proporciona una manera de modelar el comportamiento idealizado de funciones y la capacidad de los atacantes, lo que permite evaluar la robustez de los esquemas criptográficos bajo suposiciones específicas.

Por último, un último aspecto muy importante es la diferencia entre algoritmos deterministas y probabilísticos. Un algoritmo **determinista** es aquel que, dado un mismo conjunto de entradas, produce la misma salida siempre. No utiliza elementos aleatorios en su proceso. Puede ser menos seguro en algunos

contextos criptográficos porque no incorpora aleatoriedad. Por el contrario, un algoritmo **probabilístico** sí que utiliza elementos aleatorios en su proceso, lo que significa que la misma entrada puede producir diferentes salidas en diferentes ejecuciones (y de hecho, casi siempre será así). Al añadir esta aleatoriedad se suele incrementar la seguridad.

4.2. Tipos de ataques a criptosistemas

El objetivo de los ataques es comprometer la seguridad del sistema de cifrado, revelando información sobre el mismo que permita al atacante deducir información sobre el texto plano o incluso sobre la clave privada. Según la metodología utilizada, los ataques a los criptosistemas se clasifican de la siguiente manera:

Ciphertext Only Attacks (COA): En este método, el atacante tiene acceso a un conjunto de textos cifrados. No tiene acceso al texto plano correspondiente. Se dice que COA tiene éxito cuando el texto en claro correspondiente se puede determinar a partir de un conjunto dado de texto cifrado. Con este ataque a veces es posible determinar la clave de cifrado.

Known Plaintext Attack (KPA): En este método, el atacante conoce el texto plano de algunas partes del texto cifrado. La tarea consiste en descifrar el resto del texto cifrado utilizando esta información. Esto se puede hacer determinando la clave o mediante algún otro método.

Chosen Plaintext Attack (CPA): En un ataque CPA, un atacante puede cifrar todos los textos planos de su elección, lo que le permite deducir alguna información sobre sobre el texto plano o sobre la clave privada. Este tipo de ataques se vuelven extremadamente importantes en el contexto de la criptografía de clave pública, donde la clave de cifrado es pública y los atacantes pueden cifrar cualquier texto plano que elijan. En algunos casos, solo una pequeña parte del texto plano necesita ser elegida por el atacante: tales ataques se conocen como **ataques de inyección de texto plano**. Se pueden distinguir dos formas de CPA:

- **Ataque de texto plano elegido por lotes**, donde el atacante elige todos los textos planos antes de que sean cifrados.
- **Ataque de texto plano elegido adaptativo**, donde el atacante va eligiendo los sucesivos textos planos basándose en la información de los cifrados anteriores.

Chosen Ciphertext Attack (CCA): En un ataque CCA, un atacante tiene acceso a un oráculo que le permite descifrar cualquier texto cifrado de su elección (excepto el que el atacante intenta revelar). Es decir, puede, a partir de cualquier texto cifrado de su elección, recibir el texto plano correspondiente, con lo que puede obtener información del criptosistema.

Broadcast: Este ataque tiene como objetivo recuperar un único mensaje enviado a varios destinatarios. Aquí, el criptanalista conoce solo varios textos cifrados, pero sabe que provienen del mismo mensaje. Dado que el mismo mensaje se cifra con varias claves públicas, el criptoanalista utiliza esta información para recuperar el mensaje.

Ataque de reenvío de mensaje: Este tipo de ataque se da si el mismo mensaje se cifra y se envía dos o más veces al mismo destinatario con vectores de error aleatorios diferentes. Se basa en el hecho de que, haciendo esto, es posible realizar ciertas inferencias sobre los errores y, por lo tanto, sobre el mensaje

original.

Ataque de mensaje relacionado: En un ataque de mensajes relacionados, el atacante obtiene varios cifrados de tal manera que existe una relación conocida entre los correspondientes textos planos, por lo que puede obtener información sobre ellos a partir de los textos cifrados. Este ataque es una generalización del ataque de reenvío de mensajes.

Maleabilidad: La maleabilidad es una propiedad de algunos algoritmos criptográficos. Un algoritmo de cifrado es maleable si es posible que un adversario transforme un texto cifrado en otro texto cifrado que se descifra en un texto plano relacionado con el original. Es decir, dado $\mathbf{m}$, es posible generar otro texto cifrado que se descifra como $f(\mathbf{m})$, para una función conocida f, sin necesidad de conocer $\mathbf{m}$. Por tanto, se puede utilizar esta característica que tienen algunos criptosistemas para atacarlos.

4.3. RSA

Bibliografía utilizada: [31]. RSA es un algoritmo de clave pública que nos permite cifrar y descifrar. La longitud de la clave puede ser variable (a mayor longitud, mayor seguridad pero menor eficiencia). El bloque del texto plano debe ser menor que la longitud de la clave y el bloque del texto cifrado será de la longitud de la clave. Se suele utilizar para cifrar una clave secreta, y luego esa clave secreta se usa para cifrar el mensaje usando otros algoritmos de clave secreta más eficientes (como DES y IDEA), es decir, como la parte asimétrica de un criptosistema híbrido.

El algoritmo funciona de la siguiente manera: generamos una clave pública y una privada. Para ello, escogemos 2 primos p y q grandes, que mantendremos en secreto, los multiplicamos y obtenemos n. Para generar la clave pública, elegimos un número entero positivo e menor que $\phi(n)$ que sea coprimo con $\phi(n)$, siendo ϕ la función de Euler. Como conocemos p y q, podemos obtener $\phi(n)$: $\phi(n) = (p-1)(q-1)$. La clave pública será $\langle e, n\rangle$. Para generar la clave privada, encontramos el número d que satisfaga $e \cdot d = 1 \mod \phi(n)$, es decir, que $d \cdot e - 1$ es dividido exactamente por $\phi(n) = (p-1) \cdot (q-1)$. Esto suele calcularse mediante el algoritmo de Euclides extendido. $\langle d, n\rangle$ será la clave privada.

Para cifrar un mensaje m ($< n$), usamos la clave pública y el texto cifrado se obtiene haciendo $c = m^e \mod n$. Solo las personas que tengan la clave privada podrán descifrar el texto cifrado haciendo $m = c^d \mod n$. Además, solo los que sepan la clave privada pueden firmar el mensaje m con la firma $s = m^d \mod n$ usando la clave privada. Cualquiera puede verificar una firma usando la clave pública comprobando $m = s^e \mod n$.

El algoritmo funciona porque hemos elegido un d y un e tales que $de = 1 \mod \phi(n)$, y por tanto para cualquier x, se tiene $x^{de} = x^{ed} = x \mod n$. Una encriptación RSA consiste en elevar x a la e. Si a ese valor lo elevamos a la d (realizar una desencriptación RSA), volvemos a obtener x. En el caso de generación de firmas, elevamos x a la d para generarla y elevamos ese valor a la e para verificarla, con lo que volveremos a obtener x.

4.3.1. Algunos detalles técnicos sobre RSA

Lo primero que tenemos que hacer si queremos implementar RSA es encontrar dos números primos grandes. Sabemos que hay infinitos primos, pero a medida que consideramos números cada vez más grandes, la densidad de los números primos disminuye. Esto significa que la proporción de números primos

con respecto al total de números naturales tiende hacia cero a medida que consideramos intervalos más amplios. La función contadora de primos $\pi(n)$ nos indica de manera asintótica el número de primos menores que n cuando $n \to \infty$.

$$\pi(n) \sim \frac{n}{\ln(n) + B}. \tag{4.1}$$

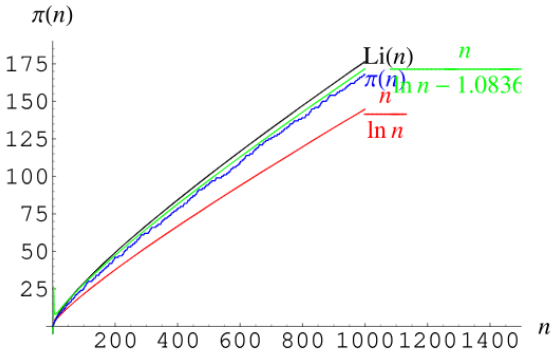

Figura 4.1: Representación de la función contadora de primos $\pi(n)$.

Por tanto, la probabilidad de que al elegir un número cualquiera entre 1 y n sea primo es aproximadamente $\frac{1}{\ln(n)}$, y por tanto para números grandes (que son los que nos interesan en criptografía), la probabilidad es pequeña. Y para ver si es primo o no, debemos comprobarlo. Un método directo sería dividir n entre todos los números menores o iguales que $\sqrt{n}$ y comprobar si alguna división es entera. Sin embargo, con números grandes esto lleva demasiado tiempo. A nivel teórico, existe un algoritmo determinista de tiempo polinómico (AKS, [1]) que nos permite comprobar si un número es primo o no. Sin embargo, en la práctica se usa otro algoritmo probabilístico más rápido, el algoritmo de Miller y Rabin. Con él podremos determinar si un número es probablemente primo (no nos lo asegura).

Usaremos el Teorema de Fermat: *si p es primo y $0 < a < p$, entonces* $a^{p-1} = 1 \bmod p$. Por tanto, si con nuestro n elegimos un $a < n$ cualquiera, si no se cumple $a^{n-1} = 1 \bmod n$, definitivamente el número n no es primo y podemos descartarlo. Si se cumple puede que no sea primo, así que esto a priori tampoco parece una buena idea. Sin embargo, la probabilidad de que un número n generado al azar cumpla $a^{n-1} = 1 \bmod n$ sin ser primo (podemos llamar a esto falso positivo) es muy pequeña, por lo que podemos correr el riesgo. Una forma rápida de reducir el riesgo es probar para varios valores de a.

Sin embargo, hay un problema: existen números que son falsos positivos para todo a, los números de Carmichael. Para solventar este problema, se selecciona un número impar $n > 2$ que deseamos verificar si es primo. Se descompone $n - 1$ como $2^b c$, donde b es el mayor entero posible y c es impar. Se elige un número aleatorio a tal que $2 \leq a \leq n - 2$. Se calcula $a^{n-1} \bmod n$ realizando $a^c \bmod n$, comprobando cada vez que elevamos al cuadrado si el resultado es igual a $+1$; si lo es, comprobar si el número que ha sido elevado al cuadrado es $+1$ o -1. Si no lo es, n no es primo. Si $a^{n-1} \bmod n$ es $+1$ o -1, entonces n pasa el test de primalidad para ese a. De lo contrario, reemplazamos el resultado por su cuadrado y comprobamos si es $+1$ o -1 (tendremos que hacer esto como máximo $b-1$ veces). Si es $+1$, n no es primo (porque el resultado anterior es una raíz cuadrada de 1 diferente de $+1$ y -1). Si es -1, entonces n pasa el test de primalidad para ese a. Si al cabo de esas $b - 1$ veces no lo hemos conseguido, entonces n no es

primo, ya que $a^{(n-1)/2}$ no es $+1$ o -1.

Otro de los detalles técnicos que debemos considerar al implementar RSA es encontrar los valores d y e. Como ya hemos visto, para e elegimos cualquier número que sea coprimo con $(p-1)(q-1)$, y solo falta encontrar d que cumpla $ed = 1 \mod \phi(n)$, lo cual se realiza fácilmente con el algoritmo de Euclides. Y para ver que e y $(p-1)(q-1)$ son coprimos, bastaría con probar hasta encontrar un e que lo cumpla. Sin embargo, lo que se suele hacer es elegir primero e y luego seleccionar primos p y q tales que que e y $(p-1)(q-1)$ sean coprimos. Esto nos interesa porque al poder elegir e, podemos elegir valores suficientemente pequeños tal que permitan hacer las operaciones de cifrar y verificar firmas de forma mucho más rápida y eficiente pero suficientemente grandes para que no sea inseguro.

Para finalizar con esta sección, es importante mencionar el GNFS (General number field sieve). GNFS es el mejor algoritmo de factorización clásico conocido, y lo hace en un tiempo subexponencial. Sin embargo, la factorización sigue siendo un problema que sin ordenadores cuánticos no sabemos resolver en un tiempo polinómico. Como ya hemos visto, con ordenadores cuánticos sí que sabemos resolver en tiempo polinómico gracias al algoritmo de Shor.

4.4. Diffie-Hellman

El intercambio de claves Diffie-Hellman es un criptosistema de clave pública que permite a dos partes acordar de manera segura una clave secreta compartida a través de un canal de comunicación inseguro.

El funcionamiento es el siguiente: Se eligen dos números primos grandes p y g. p se utiliza como un número primo grande y g tal que $g^x \mod p$ son distintos para diferentes valores de x. A continuación cada una de las dos partes, llamémoslas Alice y Bob, elige una clave privada. Estas claves privadas son números aleatorios que llamaremos a y b (a será la de Alice y b la de Bob). Cada parte calcula su clave pública. Para Alice, la clave pública A es $g^a \mod p$ y para Bob, la clave pública B es $g^b \mod p$. Alice y Bob intercambian sus claves públicas A y B, y solo ellos dos podrán calcular la clave compartida K. Esto se hace utilizando la clave pública del otro y su propia clave privada. Para Alice, $K = B^a \mod p$. Para Bob, $K = A^b \mod p$. Nótese que la clave compartida es igual en ambos lados: $(g^a)^b \mod p = (g^b)^a \mod p$.

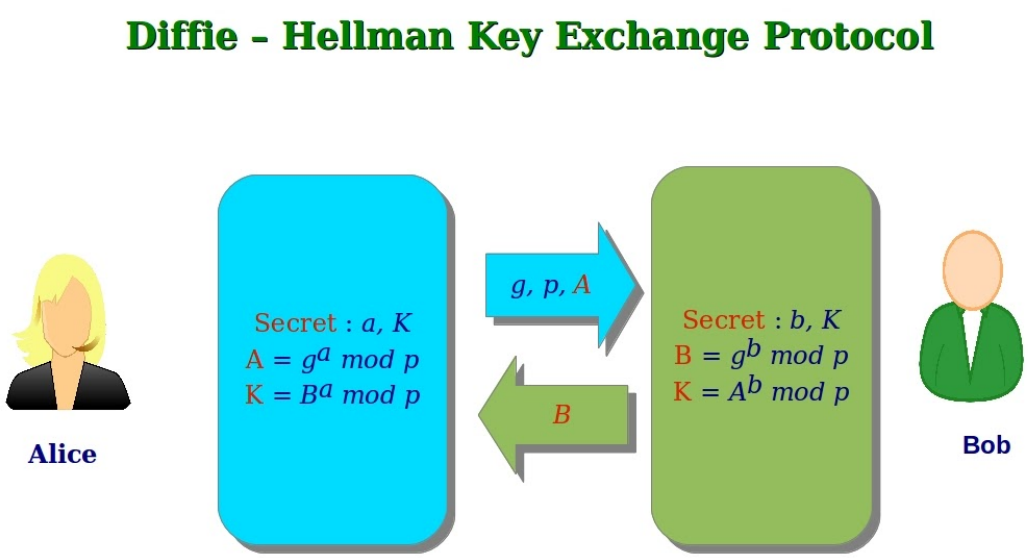

Figura 4.2: Esquema Diffie-Hellman [42].

La magia del protocolo Diffie-Hellman radica en la dificultad del problema del logaritmo discreto: sin ordenadores cuánticos, no se puede calcular K sin conocer a o b en un tiempo razonable. De esta manera, Alice y Bob pueden compartir información pública (A y B) a través de un canal no seguro y, sin embargo, calcular la misma clave secreta (K) sin que nadie más pueda.

Una debilidad de este criptosistema es que, aunque permite acordar una clave secreta K, no hay autenticación; Alice no tiene forma de asegurarse de que Bob es realmente Bob (y viceversa). Para solucionar este problema, Elgamal en su artículo [23] nos proporcionó un criptosistema para el cifrado y la firma digital.

4.5. Curvas elípticas

Tras plantear el intercambio de claves de Diffie-Hellman, surgió la idea de aplicarlo sobre curvas elípticas en lugar de sobre el grupo multiplicativo de un cuerpo finito debido a las ventajas específicas que las curvas elípticas ofrecen. Estas ventajas incluyen una mayor seguridad con claves más cortas y la eficiencia computacional, ya que las operaciones aritméticas en curvas elípticas, como la suma de puntos y la multiplicación escalar, son más eficientes en comparación con las operaciones de factorización de números grandes o las exponenciaciones modulares utilizadas en otros sistemas.

Definición 4.5.1. *Una **curva elíptica** es un conjunto de puntos dados en un cuerpo F que satisfacen la siguiente ecuación:*

$$y^2 + a_1xy + a^2y = x^3 + a_3x^2 + a_4x + a_5 \tag{4.2}$$

con $a_1, ..., a_5 \in F$. Esta ecuación se puede simplificar cuando trabajamos en un cuerpo que no sea de característica 2 o 3; en ese caso la curva se puede escribir como

$$y^2 = x^3 + ax + b \tag{4.3}$$

*donde $a, b \in F$. Esto se denomina la **forma normal de Weierstrass**, que es el tipo de curvas elípticas usadas en criptografía.*

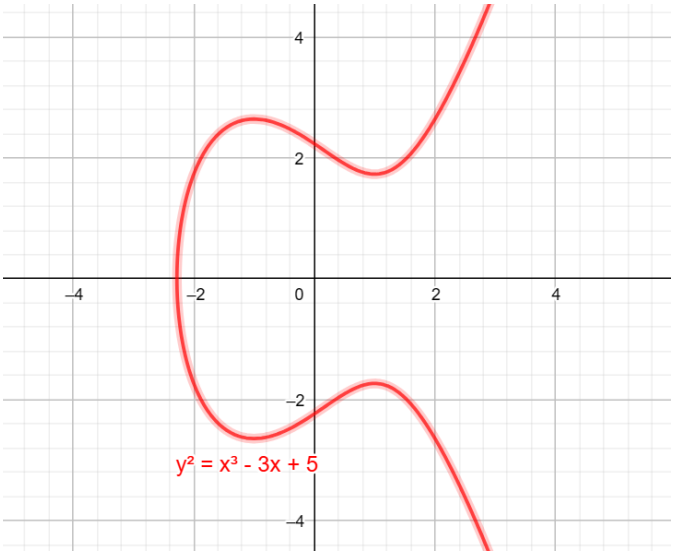

Figura 4.3: Ejemplo de curva elíptica en forma de Weierstrass.

Vemos que por la propia forma de la ecuación de la forma de Weierstrass, la curva elíptica es simétrica respecto al eje x, por lo que para un x_1, si (x_1, y_1) pertenece a la curva, entonces $(x_1, -y_1)$ también pertenece.

Para construir criptosistemas sobre curvas elípticas es importante definir la suma de puntos sobre estas curvas: Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ dos puntos de la curva elíptica. El punto suma $R = P + Q = (x_3, y_3)$ es la intersección entre la recta que pasa por los puntos P y Q y la curva elíptica:

$$\left\{ \begin{array}{c} y = \dfrac{(y_2 - y_1)x + y_1 x_2 - y_2 x_1}{x_2 x_1} \\ y^2 = x^3 + ax + b \end{array} \right\}. \tag{4.4}$$

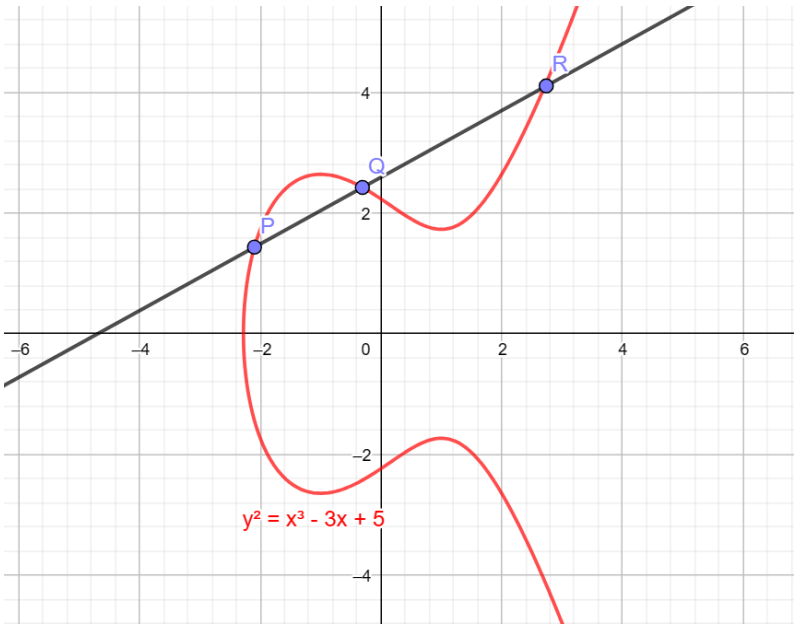

Figura 4.4: Suma de dos puntos de una curva elíptica.

Los puntos de una curva elíptica forman un grupo cíclico o producto de dos cíclicos con la suma + tal y como la hemos definido ([59]), y a partir de estos grupos podemos construir sistemas criptográficos.

4.5.1. Diffie-Hellman sobre curvas elípticas (ECDH)

El intercambio de claves de Diffie-Hellman en curvas elípticas (ECDH) es un protocolo criptográfico de clave pública que se basa en la dificultad del problema del logaritmo discreto en curvas elípticas (ECDLP). En ECDH, cada parte genera un par de claves, una pública y otra privada, al igual que en DH. La clave privada es un número entero aleatorio, mientras que la clave pública es un punto en una curva elíptica generado al multiplicar la clave privada por un punto base predefinido en la curva elíptica. Luego, cada parte intercambia su clave pública con la otra parte. Utilizando sus claves privadas y las claves públicas de la otra parte, ambas partes calculan un punto común en la curva elíptica que solo ellos conocen. Veamos cómo funciona detalladamente.

Alice y Bob acuerdan una curva elíptica $\mathbb{E}$ sobre un cuerpo $\mathbb{F}_q$ y un punto base $P \in \mathbb{E}/F_q$. Alice genera una clave secreta aleatoria k_A y computa $P_A = k_A P$. Bob genera una clave secreta aleatoria k_B y computa $P_B = k_B P$. Alice y Bob intercambian P_A y P_B (claves públicas). Ambos computan $P_{AB} = k_A P_B = k_B P_A$ (cada uno usando su clave secreta y la clave pública del otro), y por tanto P_{AB} es la clave compartida que solo ellos pueden saber. k_A y k_B son valores aleatorios $\in \{1, \ldots, n-1\}$ con n el orden del grupo generado por G.

La diferencia entre DH y ECDH es principalmente el grupo con el que se eligen las claves secretas. DH usa el grupo multiplicativo de enteros módulo un primo p, y ECDH usa el grupo aditivo de puntos de una curva elíptica.

4.6. Impacto del algoritmo de Shor en la criptografía

Hemos visto que los algoritmos clásicos conocidos no pueden factorizar en tiempo $O((logN)^k)$ para ningún k (como hemos dicho, el mejor algoritmo clásico conocido hasta la fecha es de tiempo subexponencial). Sin embargo, con un ordenador cuántico, el algoritmo de Shor puede romper RSA en tiempo polinómico. Y lo mismo pasa con muchas otras criptografías de clave pública. Por esta razón la computación cuántica es una amenaza para muchos algoritmos actuales y se está trabajando en criptografía resistente a los ordenadores cuánticos.

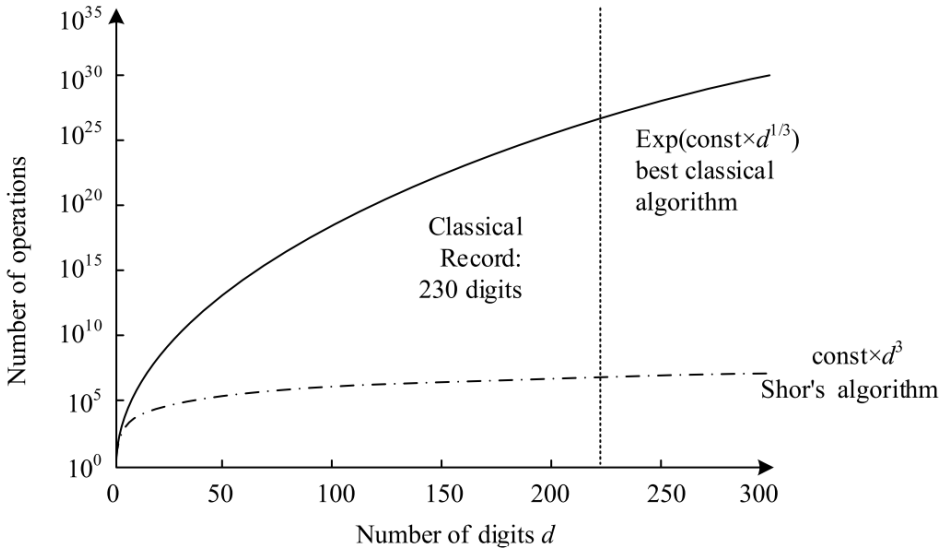

Figura 4.5: Con el algoritmo de Shor, la factorización pasa de ser un problema de tiempo exponencial a ser un problema de tiempo polinómico. ([19])

Criptosistema	**Tipo**	**Uso**	**Impacto de un ordenador cúantico**
AES	Clave simétrica	Cifrado	Se necesitan claves más largas
SHA-2, SHA-3	Función hash	-	Se necesita un output más largo
RSA	Clave pública	Firmas, establecimiento de claves	Deja de ser seguro
ECDH	Clave pública	Firmas, intercambio de claves	Deja de ser seguro

Cuadro 4.1: Impacto de los ordenadores cuánticos en algunos de los algoritmos criptográficos más comunes. Tabla basada en el documento del NIST [12].

4.6.1. Uso del algoritmo de Shor para romper Diffie-Hellman

Sección basada en [14]. Además de romper RSA, el algoritmo de Shor también se puede aplicar para romper Diffie-Hellman; es un algoritmo cuántico que permite calcular eficientemente el logaritmo discreto, es decir, obtener el valor de a en la ecuación $A = g^a \bmod p$ dado g, A y p.

Para ello, se prepara un registro cuántico en un estado superpuesto de todos los posibles valores de a. Este registro puede representarse como una superposición de estados $|k\rangle$ para $k = 0, 1, 2, \ldots, N-1$ donde

N es un número grande relacionado con p. Posteriormente, se define una función cuántica que calcula $f(k) = g^k \bmod p$. Aplicamos esta función a todos los valores de k en paralelo debido a la naturaleza de la superposición cuántica. Esto se logra utilizando puertas cuánticas para realizar la exponenciación modular de manera eficiente.

El paso clave es, de nuevo, aplicar la transformada de Fourier cuántica al registro cuántico que contiene los resultados de la evaluación de $f(k)$. Esto transforma la superposición cuántica $\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k, f(k)\rangle$ en otra superposición que hace más evidente la periodicidad de $f(k)$. Esta transformación redistribuye los coeficientes de la superposición de manera que los estados cuánticos correspondientes a múltiplos del período tienen amplitudes significativamente más grandes, de manera que al hacer una medición, como las amplitudes asociadas a esos estados múltiplos del periodo r de la función $f(k)$ son mayores, habrá más probabilidades de que el estado de superposición colapse en los estados múltiplos de r, proporcionando información sobre el período de la función $f(k)$.

Una vez conocido el período r, podemos utilizarlo para calcular el valor de a. Dado que $g^r \equiv 1 \bmod p$, podemos resolver la ecuación $A = g^a \bmod p$ mediante métodos algebraicos utilizando r para encontrar a.

4.7. Otros algoritmos

Hemos visto como el algoritmo de Shor supone una amenaza para la criptografía de clave pública, en particular para RSA y Diffie-Hellman. Sin embargo, hay otro algoritmo muy conocido que utiliza las ventajas de la computación cuántica para atacar a la criptografía, en este caso a la criptografía simétrica. Este es el algoritmo de Grover. No rompe completamente los esquemas simétricos, pero reduce el tiempo necesario para ataques de fuerza bruta, concretamente produce una mejora cuadrática, por lo que necesitaremos duplicar el tamaño de las claves.

Capítulo 5

Criptografía basada en funciones hash

Capítulo basado en [39] y [22]. Las funciones hash criptográficas son herramientas fundamentales en la criptografía moderna, utilizadas para garantizar la integridad y autenticidad de los datos. Son funciones rápidas y eficientes, y juegan un papel crucial en diversas aplicaciones criptográficas en la actualidad. Su resistencia inherente a los ataques cuánticos, junto con su eficiencia y versatilidad, las hace indispensables en el diseño de nuevos esquemas criptográficos que protegerán la información en la era postcuántica.

Una función hash es una función matemática $H : D \to R$, donde el dominio es $D = \{0,1\}^*$ y la imagen es $R = \{0,1\}^n$ para algún $n \geq 1$. Esta función lleva un valor de entrada numérico m de longitud arbitraria en un valor numérico (texto cifrado) condensado de salida h de longitud fija, denominado hash. Es decir: $h = H(m)$.

Una función hash criptográfica debe tener las siguientes propiedades que garantizan su efectividad y seguridad:

- **Función unidireccional**: esta propiedad requiere que para una función hash H, dado cualquier valor hash h, sea computacionalmente inviable encontrar una entrada m tal que $H(m) = h$. En otras palabras, debe ser fácil de calcular para cada entrada, pero extremadamente difícil de invertir dado la imagen de una entrada aleatoria.

- **Resistencia a colisiones dirigidas**: esta propiedad requiere que, dada una función hash H y cualquier entrada m, sea computacionalmente inviable encontrar otra entrada m' tal que $m' \neq m$ y $H(m) = H(m')$.

- **Resistencia a colisiones**: esta propiedad requiere que, dada una función hash H, sea computacionalmente inviable encontrar dos entradas m y m' tales que $m \neq m'$ y $H(m) = H(m')$. Debido al tamaño fijo de los valores hash en comparación con el tamaño mucho mayor y arbitrario de las entradas, se espera que existan colisiones en las funciones hash (en otras palabras, como el dominio es mayor que la imagen, H no es inyectiva). Sin embargo, estas colisiones deben ser computacionalmente muy difíciles de encontrar.

- **Determinística**: esta propiedad requiere que una función hash H deba mapear consistentemente una entrada dada m a un valor hash h. Es decir, al aplicar a una misma entrada siempre se obtiene la misma salida.

- **Efecto avalancha**: esta propiedad requiere que un cambio en solo un bit de los datos de entrada resulte en un gran cambio en la salida.
- **Velocidad del hash**: una propiedad ideal de una función hash criptográfica es su capacidad para operar a una velocidad razonable. En muchas situaciones, un algoritmo de hashing debe calcular valores hash de manera bastante rápida.

Las funciones hash criptográficas se clasifican en dos grandes categorías:

1. Funciones hash sin clave. Son las funciones hash tal y como la hemos definido hasta ahora, que toman una entrada de longitud arbitraria y producen una salida de longitud fija sin requerir una clave secreta. Si no se especifica, cuando se habla de funciones hash se suele hablar de este tipo.
2. Funciones hash con clave. Estas tienen dos entradas, un mensaje de entrada y una clave secreta.

Las funciones hash sin clave también son conocidas como *código de detección de manipulaciones* (MDC). Esto se debe a que la propiedad del efecto avalancha permite detectar manipulaciones en un mensaje, ya que si se altera un solo bit, al aplicar la función el hash obtenido es completamente distinto, lo que nos permite detectar rápidamente que ha habido una manipulación en nuestro mensaje original. Es decir, los MDC permiten garantizar la integridad (asegurar que cualquier alteración, ya sea accidental o maliciosa) en el mensaje será detectada.

Las funciones hash con clave son conocidas como *código de autenticación de mensajes* (MAC). Es la misma idea que los MDC: permiten garantizar la integridad, pero además al requerir la clave secreta sirve para autenticidad (garantiza que el mensaje proviene de una fuente legítima que conoce la clave secreta).

Además de integridad y autenticación, las funciones hash también sirven para las firmas digitales, ya que se puede usar una función hash para resumir un mensaje antes de aplicar una firma digital. Esto reduce el tamaño del mensaje a firmar y proporciona una forma eficiente de verificar la integridad del mensaje.

Las funciones hash podrían volverse vulnerables frente a los ordenadores cuánticos, ya que se basan en problemas computacionales difíciles de resolver para los ordenadores clásicos, pero puede que no lo sean para ordenadores cuánticos. Por ejemplo, pueden basarse en la factorización y en búsquedas en bases de datos no estructuradas, lo cual ya sabemos que puede suponer un problema: los algoritmos de factorización son vulnerables frente a un ordenador cuántico que utilice el algoritmo de Shor, y la búsqueda en bases de datos no estructuradas sufre una mejora cuadrática debido al algoritmo de Grover.

Por tanto, se está trabajando en el desarrollo de funciones hash que se construyan basándose en problemas matemáticos resistentes a ordenadores cuánticos, como los de retículos. La definición exacta de retículo se verá en el capítulo 8, pero vamos a dar una idea intuitiva: un retículo es un conjunto de puntos en el espacio que podemos obtener a partir de combinaciones lineales enteras de un conjunto de vectores de una base. Estos puntos se organizan en un patrón regular y repetitivo. La idea de utilizar retículos para su uso en criptografía fue desarrollada por Ajtai: demostró cómo ciertos problemas de retículos pueden ser utilizados para crear sistemas criptográficos seguros. Estos problemas los detallaremos de nuevo en la sección 8, pero por ahora nos quedamos con esta idea: algunos problemas matemáticos de retículos parece que no se pueden resolver en tiempo polinómico por computadoras clásicas o cuánticas. Por tanto, estos problemas son idóneos para su aplicación en criptografía. Reducir problemas de caso peor de retículos a problemas promedio puede aplicarse para diseñar funciones hash ([36]). Esto asegura que si un adversario puede encontrar una colisión o una preimagen para una instancia promedio de la función hash, también podría hacerlo para cualquier instancia del problema subyacente, lo cual es extremadamente difícil. En otras palabras: *resolver una colisión (encontrar dos entradas diferentes con el mismo hash) o*

una preimagen (encontrar una entrada dada una salida) es tan difícil como resolver estos problemas de retículos. Esta técnica es la que se utiliza en SWIFFT:

5.0.1. SWIFFT

En el artículo [36] se presenta SWIFFT, una colección de funciones hash criptográficas que utilizan la Transformada Rápida de Fourier (FFT) para proporcionar una forma eficiente y segura de generar hashes. SWIFFT está diseñado para ser resistente a ataques de ordenadores cuánticos.

Funcionamiento de SWIFFT

1. Los datos de entrada se dividen en bloques y se representan como polinomios cuyos coeficientes pertenecen a un anillo específico, que a menudo es un anillo finito. Un ejemplo común es el anillo de los enteros módulo q, denotado como $\mathbb{Z}/q\mathbb{Z}$.
2. Cada bloque se transforma utilizando la FFT. La utilización de FFT permite que SWIFFT sea muy rápido y adecuado para grandes volúmenes de datos.
3. Los datos transformados se convolucionan con coeficientes específicos. Esta operación mezcla los datos de manera que cualquier cambio en la entrada original resulta en grandes cambios en la salida.
4. Se vuelve al dominio original mediante una Transformada Inversa de Fourier (IFFT).
5. Los datos resultantes se reducen modularmente para producir el hash final.

La seguridad de SWIFFT se basa en la dificultad de resolver problemas de retículos asociados con las transformaciones y convoluciones realizadas. La estructura matemática utilizada en SWIFFT está diseñada para que encontrar colisiones o preimágenes sea equivalente a resolver problemas difíciles en retículos.

Capítulo 6

Criptografía basada en ecuaciones multivariable

Bibliografía principal utilizada para este capítulo: [46]. La criptografía de ecuaciones multivariable se basa en la dificultad de resolver sistemas de ecuaciones polinómicas multivariable sobre cuerpos finitos. Este enfoque se considera una alternativa prometedora en el contexto de la criptografía postcuántica debido a que los sistemas de ecuaciones multivariable son difíciles de resolver incluso para ordenadores cuánticos. En función del tipo de construcción de estos criptosistemas, podemos dividirlos en dos tipos:

- Construcción estándar (bipolar).
- Construcción mixta.

La construcción estándar fue con la que se empezó a trabajar en criptografía de ecuaciones multivariable. Son los más sencillos o los más intuitivos a priori. Sin embargo, la necesidad de mejorar la seguridad dio lugar a las construcciones mixtas. Estas parten de construcciones estándar, añadiendo algunas modificaciones para hacerlas más robustas. Estas modificaciones pueden incluir mezclar esquemas estándar para evitar ataques conocidos, aplicar algunas transformaciones sobre las ecuaciones para así aumentar su complejidad o incorporar estructuras algebraicas adicionales sobre ellas. En definitiva, podemos pensar en la construcción estándar como la idea básica del criptosistema y la construcción mixta como una mejora sobre esas construcciones estándar que darán como resultado criptosistemas más seguros.

También existen dos métodos para obtener los esquemas de autentificación de clave pública, cada uno basado en un problema distinto:

- El esquema IP, basado en el problema de isomorfismo de polinomios.
- El esquema MQ, basado en el problema de resolver un sistema de polinomios cuadráticos multivariable.

6.1. Construcción estándar o bipolar

La idea básica consiste en elegir un sistema $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}^m$ de m polinomios multivariable cuadráticos en n variables que puede ser invertido fácilmente (se denomina *aplicación central*). Elegimos dos aplicaciones lineales invertibles $\mathcal{S} : \mathbb{F}^m \to \mathbb{F}^m$ y $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$. La función de estas dos aplicaciones es ocultar

la estructura de la aplicación central $\mathcal{F}$ en la clave pública. La clave pública del criptosistema será $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} : \mathbb{F}^n \to \mathbb{F}^m$, que se espera que sea muy difícil de invertir. Las aplicaciones $\mathcal{S}$ y $\mathcal{T}$ son introducen para que el nuevo sistema de ecuaciones sea indestinguible de un sistema aleatorio. La clave privada consiste en $(\mathcal{S}, \mathcal{F}, \mathcal{T})$, y por tanto con esto podemos invertir la clave pública.

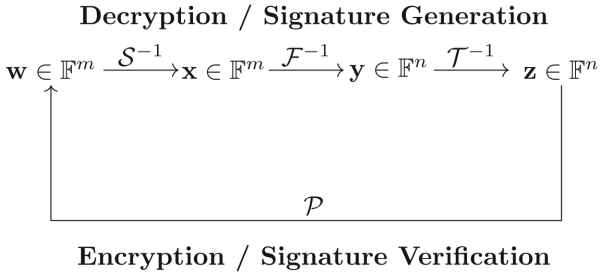

Figura 6.1: Esquema de la construcción bipolar. [46]

6.1.1. Esquemas de cifrado ($m \geq n$)

Para **cifrar** un mensaje $\mathbf{z} \in \mathbb{F}^n$, hacemos $\mathbf{w} = \mathcal{P}(\mathbf{z})$. El texto cifrado de $\mathbf{z}$ es $\mathbf{w} \in \mathbb{F}^m$.

Para **descifrar** el texto cifrado $\mathbf{w} \in \mathbb{F}^m$, aplicamos sucesivamente $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}), \mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ y $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. El vector $\mathbf{z} \in \mathbb{F}^n$ es el texto plano de $\mathbf{w}$. Como $m \geq n$, la preimagen de $\mathbf{x}$ en $\mathcal{F}$ (y en consecuencia el texto plano) es único. Esta unicidad se debe al hecho de que $m \geq n$. En sistemas donde m (la dimensión de la imagen de $\mathcal{P}$) es mayor o igual que n (la dimensión del espacio original de $\mathbf{z}$), la aplicación $\mathcal{P}$ no puede colapsar varios valores diferentes de $\mathbf{z}$ en un mismo $\mathbf{w}$. Esto significa que para cada $\mathbf{w} \in \mathbb{F}_m$, existe una única secuencia $(\mathbf{z}, \mathbf{x}, \mathbf{y})$ tal que $\mathcal{P}(\mathbf{z}) = \mathbf{w}$, ya que $\mathcal{S}$, $\mathcal{F}$ y $\mathcal{T}$ son aplicaciones invertibles, y su composición en $\mathcal{P}$ preserva la unicidad en la recuperación de $\mathbf{z}$.

6.1.2. Esquemas de firma ($m \leq n$)

Generación de firmas

Para firmar un documento d, usamos una función hash $\mathcal{H} : \{0,1\}^* \longrightarrow \mathbb{F}^m$ para obtener el valor $\mathbf{w} = H(d) \in \mathbb{F}^m$. Calculamos $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w})$, $\mathbf{y} = \mathcal{F}^{-1}(\mathbf{x})$ and $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. La firma de d es $\mathbf{z} \in \mathbb{F}^n$. Aquí, $\mathcal{F}^{-1}(\mathbf{x})$ significa encontrar una de las preimágenes de $\mathbf{x}$ bajo la aplicación central $\mathcal{F}$. Como $n \geq m$, sabemos que esa pre-imagen existe, y por tanto cada mensaje tiene una firma.

Verificación de firmas

Para verificar la autenticidad de un documento, se hace **w'** $= \mathcal{P}(\mathbf{z})$ y el valor $\mathbf{w} = \mathcal{H}(d)$ del documento. Si **w'** $= \mathbf{w}$ se cumple, la firma es aceptada; de lo contrario es rechazada.

6.2. Sistemas mixtos

Para construir un esquema multivariable de tipo mixto, consideramos la aplicación cuadrática $\mathcal{F} : \mathbb{F}^{m+n} \longrightarrow \mathbb{F}^m$ de la forma

$$\mathcal{F}(y_1, \ldots, y_n, x_1, \ldots, x_m) = (h_1, \ldots, h_m). \tag{6.1}$$

$\mathcal{F}$ debe cumplir 2 condiciones:

1. Para cada $(y'_1, \ldots, y'_n) \in \mathbb{F}^n$ fijo, la aplicación $\mathcal{F}(y'_1, \ldots, y'_n, x_1, \ldots, x_m) : \mathbb{F}^m \longrightarrow \mathbb{F}^m$ es lineal.
2. Para cada $(x'_1, \ldots, x'_m) \in \mathbb{F}^m$ fijo, la aplicación $\mathcal{F}(y_1, \ldots, y_n, x'_1, \ldots, x'_m) : \mathbb{F}^n \longrightarrow \mathbb{F}^m$ es un sistema eficientemente invertible de ecuaciones cuadráticas multivariable.

La clave pública de este sistema mixto se define como $\mathcal{P} = \mathcal{L} \circ \mathcal{F} \circ (\mathcal{S} \times \mathcal{T})$ donde $\mathcal{S} : \mathbb{F}^n \longrightarrow \mathbb{F}^n$ y $\mathcal{T} : \mathbb{F}^m \longrightarrow \mathbb{F}^m$ son invertibles, afines y $\mathcal{L} : \mathbb{F}^m \longrightarrow \mathbb{F}^m$ es una aplicación invertible y lineal. Por tanto, $\mathcal{P} : \mathbb{F}^{m+n} \longrightarrow \mathbb{F}^m$ es una aplicación cuadrática. Pero para cualquier $(z'_1, \ldots, z'_n) \in \mathbb{F}^n$ el sistema $\mathcal{P}(z'_1, \ldots, z'_n, w_1, \ldots, w_m)$ se convierte en una aplicación lineal desde $\mathbb{F}^m$ a sí mismo debido a la condición 1. Esta es la idea que explicábamos al principio: la composición de estas aplicaciones es una modificación de las ecuaciones, aumentando la seguridad del criptosistema.

La clave privada consiste en las aplicaciones $\mathcal{L}, \mathcal{F}, \mathcal{S}, \mathcal{T}$. En general $\mathcal{L}$ no es necesaria para descifrar o firmar pero aún así debe mantenerse en secreto.

6.2.1. Esquemas de cifrado ($m \geq n$)

Para **cifrar** un mensaje $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}^n$, se resuelve el sistema lineal

$$\mathcal{P}(z_1, \ldots, z_n, w_1, \ldots, w_m) = (0, \ldots, 0) \tag{6.2}$$

para $w_1, \ldots, w_m$. El vector $\mathbf{w} = (w_1, \ldots, w_m) \in \mathbb{F}^m$ es el texto cifrado del mensaje $\mathbf{z}$.

Para **descifrar** un texto cifrado $\mathbf{w} \in \mathbb{F}^m$, hacemos $\mathbf{x} = (x_1, \ldots, x_m) = \mathcal{T}(\mathbf{w})$. Luego se resuelve

$$\mathcal{F}(y_1, \ldots, y_n, x_1, \ldots, x_m) = (0, \ldots, 0) \tag{6.3}$$

para $\mathbf{y} = y_1, \ldots, y_n$. Debido a la condición 2 de $\mathcal{F}$, 6.3 es invertible. Por último, se calcula el texto plano $\mathbf{z} = \mathcal{S}^{-1}(\mathbf{y})$. Dado que tenemos $m \geq n$, el texto plano es único.

6.2.2. Esquemas de firma ($m \leq n$)

Generación de firma

Para generar una firma para un documento d, se utiliza una función de hash $H : \{0,1\}^* \to \mathbb{F}^m$ para calcular un valor hash $\mathbf{w} = H(d) = (w_1, \ldots, w_m) \in \mathbb{F}^m$. Luego se calcula $\mathbf{x} = (x_1, \ldots, x_m) = \mathcal{T}(\mathbf{w})$ y se resuelve

$$\mathcal{F}(y_1, \ldots, y_n, x_1, \ldots, x_m) = (0, \ldots, 0) \tag{6.4}$$

para $\mathbf{y} = y_1, \ldots, y_n$. De nuevo, (6.4) es un sistema de ecuaciones cuadráticas multivariadas eficientemente invertible (por la condición 2). La firma del mensaje d es $\mathbf{z} = \mathcal{S}^{-1}(\mathbf{y}) \in \mathbb{F}^n$. Dado que tenemos $n \geq m$, cada mensaje tiene una firma.

Verificación de firma

Para verificar la autenticidad de una firma $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}^n$, se calcula el valor hash $\mathbf{w} = H(d) = (w_1, \ldots, w_m)$ y se evalúa

$$\mathcal{P}(z_1, \ldots, z_n, w_1, \ldots, w_m). \tag{6.5}$$

Si el resultado es $(0, \ldots, 0) \in \mathbb{F}^m$, la firma es aceptada; de lo contrario, es rechazada.

6.3. Problemas en los que se basa la criptografía multivariable

La seguridad de los esquemas bipolares y mixtos se fundamenta en 2 problemas matemáticos: el problema MQ y el problema IP.

6.3.1. Problema MQ

Definición 6.3.1. ***Problema de resolución de sistemas polinómicos****. Dado un sistema de m ecuaciones polinómicas no lineales $\mathcal{P} = (p^{(1)}, \ldots, p^{(m)})$ en las variables $x_1, \ldots, x_n$, encontrar valores $x'_1, \ldots, x'_n$ tales que satisfagan el sistema, es decir,*

$$p^{(1)}(x'_1, \ldots, x'_n) = \ldots = p^{(m)}(x'_1, \ldots, x'_n) = 0.$$

Resolver sistemas de polinomios multivariable es NP-completo. Para demostrar esto, lo que se hace es reducir el problema el problema 3-SAT, que ahora definiremos, al problema MQ, por lo que MQ es al menos tan difícil como 3-SAT y se sabe que 3-SAT es NP-completo. Pero como además comprobar una solución del problema MQ se hace entiempo polinómico, está en NP y por tanto es exactamente NP-completo.

Definición 6.3.2. ***3-SAT (3-Satisfacibilidad)*** *es un problema computacional en la teoría de satisfacibilidad booleana. Consiste en determinar si una fórmula booleana dada, en forma normal conjuntiva (FNC), puede ser satisfecha asignando valores de verdad a sus variables de modo que cada cláusula en la fórmula contenga exactamente tres literales (variables booleanas o sus negaciones).*

Vamos a detallar un poco esta definición. Este problema parte de una **fórmula booleana**, es decir, una expresión construida con variables booleanas (que pueden tomar el valor verdadero o falso) combinadas mediante operaciones como AND ($\wedge$), OR ($\vee$), y NOT ($\neg$). Esta forma booleana va a estar en **forma normal conjuntiva (FNC)**; con esto nos referimos a que está compuesta por una serie de cláusulas, y cada cláusula es una disyunción (OR) de exactamente tres literales (variables o sus negaciones). El problema consiste en determinar si existe alguna manera de asignar valores de verdad a las variables de la fórmula para que cada una de las cláusulas se vuelva verdadera al menos una vez. Si es posible encontrar tal asignación, se dice que la fórmula es satisfacible; de lo contrario, es insatisfacible.

Como hemos dicho, 3-SAT es conocido por ser NP-completo. Esto lo convierte en un problema central en la teoría de la complejidad computacional y por ello es útil para determinar la complejidad de otro problema mediante reducciones, como es el caso del problema MQ.

La reducción (está sacada del teorema 10 de [24]) es:

Teorema 6.3.1. *El problema 3-SAT y un sistema de ecuaciones polinómicas sobre el cuerpo finito $\mathbb{F}_2$ son equivalentes.*

La idea de la demostración es la siguiente: utiliza la transformación de fórmulas booleanas en la forma FNC de problemas 3-SAT en polinomios sobre el cuerpo finito $\mathbb{F}_2$. Cada cláusula en la fórmula se convierte en un polinomio mediante operaciones de suma y multiplicación en $\mathbb{F}_2$. La clave de la demostración radica en que la fórmula booleana es satisfacible si y solo si ciertos polinomios derivados de ella no son cero en $\mathbb{F}_2$. Esto demuestra la equivalencia computacional entre resolver problemas 3-SAT y evaluar polinomios sobre $\mathbb{F}_2$.

Demostración. Sea $\phi(x_1, \ldots, x_n)$ una fórmula booleana en FNC, que es una conjunción de las cláusulas $C_1, \ldots, C_m$, donde cada cláusula contiene como máximo tres literales. Para cada C_i, construimos un polinomio Q_i sobre $\mathbb{F}_2$ aplicando las transformaciones:

$$\neg x = 1 \oplus x, \quad A \vee B = A \oplus B \oplus (A \cdot B)$$

para todas las variables booleanas x y todas las fórmulas booleanas A, B (donde $\oplus$ y $\cdot$ son respectivamente la suma y la multiplicación sobre $\mathbb{F}_2$), y expandiendo y recolectando términos. Por ejemplo, la cláusula $x_1 \vee x_2 \vee \neg x_3$ se transforma en:

$$(x_1 \oplus x_2 \oplus x_1 \cdot x_2) \oplus (x_3 \oplus 1) \oplus (x_1 \oplus x_2 \oplus x_1 \cdot x_2) \cdot (\neg x_3 \oplus 1) = x_1 \cdot x_2 \cdot x_3 \oplus x_1 \cdot x_3 \oplus x_2 \cdot x_3 \oplus x_3 \oplus 1 \tag{6.6}$$

Finalmente, sea $P_i = 1 \oplus Q_i$ (para $1 \leq i \leq m$), $S = \{P_i\}_{i=1}^m$. Claramente,

$$C_i(\mu_1, \ldots, \mu_n) = 1 \iff P_i(\mu_1, \ldots, \mu_n) = 0 \tag{6.7}$$

para $1 \leq i \leq m$) y $(\mu_1, \ldots, \mu_n) \in \mathbb{F}_2^n$.

Así, ϕ es satisfacible si y solo si $M[S] \neq \emptyset$. La longitud de cada Q_i está acotada por $O(\log n)$, ya que cada C_i contiene solo tres variables. □

Por razones de eficiencia, la mayoría de esquemas utilizan solo polinomios cuadráticos (grado 2), y el problema de resolver este caso concreto se llama el problema MQ (por Multivariate Quadratic). ¿Este problema sigue siendo NP-completo? Esta duda nos surge ya que, por ejemplo, si nos restringimos a polinomios lineales, el problema pasaría a ser P en vez de NP-completo. Del mismo modo, podría ser que al restringirnos a los cuadráticos, el problema dejara de ser NP-completo. Pero esto no ocurre por la siguiente razón: todo sistema de grado máximo d se puede pasar a otro sistema cuadrático con más variables. Esto se puede conseguir cogiendo cada monomio m de grado mayor que 2 y se parte en dos de menor grado, $m = m_1 \cdot m_2$; ahora se añaden las variables $z_1 = m_1$, $z_2 = m_2$ y se ponen estas ecuaciones dentro del sistema. Haciendo esto de manera recursiva se llega a un sistema cuadrático de manera eficiente. Por lo tanto, resolver sistemas generales se puede hacer tan rápido como resolver sistemas cuadráticos, y por tanto los cuadráticos también van a ser NP-completos.

6.3.2. Problema IP

Existen 3 versiones de este problema:

Definición 6.3.3. ***Problema IP1S (Isomorfismo de polinomios con 1 secreto)****: Dados dos sistemas multivariable no lineales $\mathcal{A}$ y $\mathcal{B}$ tales que $\mathcal{B} = \mathcal{A} \circ \mathcal{T}$ con $\mathcal{T}$ una aplicación afín o lineal, y encontrar una aplicación $\mathcal{T}'$ tal que $\mathcal{B} = \mathcal{A} \circ \mathcal{T}'$*

Definición 6.3.4. ***Problema IP2S (Isomorfismo de polinomios con 2 secretos)****: Dados dos sistemas multivariable no lineales $\mathcal{A}$ y $\mathcal{B}$ tales que $\mathcal{B} = \mathcal{S} \circ \mathcal{A} \circ \mathcal{T}$ con $\mathcal{T}$ y $\mathcal{S}$ aplicaciones afines o lineales, y encontrar unas aplicaciones $\mathcal{T}'$ y $\mathcal{S}'$ tales que $\mathcal{B} = \mathcal{S}' \circ \mathcal{A} \circ \mathcal{T}'$*

Definición 6.3.5. ***Problema EIP (Isomorfismo de polinomios extendido)****: Dada una clase especial $\mathcal{C}$ de sistemas multivariable no lineales y un sistema multivariable no lineal $\mathcal{P}$ que se puede escribir como $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ con $\mathcal{T}$ y $\mathcal{S}$ aplicaciones afines y $\mathcal{F} \in \mathcal{C}$, encontrar una descomposición de $\mathcal{P}$ de la forma $\mathcal{P} = \mathcal{S}' \circ \mathcal{F}' \circ \mathcal{T}'$ con $\mathcal{T}'$ y $\mathcal{S}'$ aplicaciones afines y $\mathcal{F}' \in \mathcal{C}$.*

El problema IP2S se usa para construir esquemas en los que la aplicación central es pública, como por ejemplo el Matsumoto-Imai que veremos posteriormente. Cuando la aplicación central forma parte de la clave privada, la seguridad del esquema se basa en el problema EIP. Un ejemplo es el esquema UOV que veremos también posteriormente.

6.4. Criptosistema de Matsumoto-Imai (MI)

Sea $\mathbb{F}$ un cuerpo finito de característica 2 con q elementos y sea $g(X) \in \mathbb{F}[X]$ un polinomio irreducible de $\mathbb{F}$ de grado n. Por lo tanto, el cuerpo $\mathbb{E} = \mathbb{F}[X]/g(X)$ es una extensión de grado n sobre $\mathbb{F}$. Sea $\phi : \mathbb{F}^n \to \mathbb{E}$ el isomorfismo estándar entre el espacio vectorial $\mathbb{F}^n$ y la extensión del cuerpo, $\mathbb{E}$, es decir,

$$\phi(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i X^{i-1} \tag{6.8}$$

Hay que hacer algunos cambios al esquema para garantizar la biyectividad de la aplicación central.

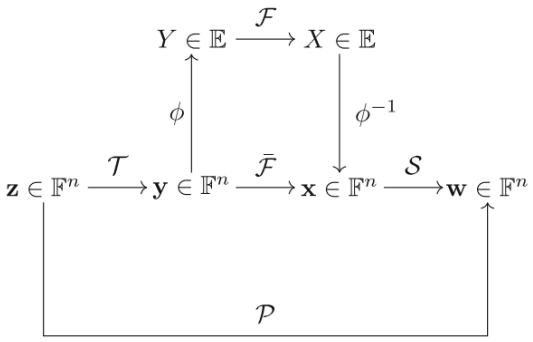

Figura 6.2: Esquema del criptosistema de Matsumoto-Imai. [46]

La aplicación central $\mathcal{F} : \mathbb{E} \to \mathbb{E}$ del esquema MI es biyectiva sobre la extensión del cuerpo $\mathbb{E}$, que se define como

$$\mathcal{F}(Y) = Y^{q^\theta+1}. \tag{6.9}$$

con $0 < \theta < n$ y $gcd(q^n - 1, q^\theta + 1) = 1$. Esta condición, junto a la condición de que el cuerpo sea de característica 2, garantiza que $\mathcal{F}$ sea biyectiva.

Para invertir la aplicación central $\mathcal{F}$, usamos el algoritmo extendido de Euclides para calcular un entero h con $h(q^\theta + 1) = 1 \bmod (q^n - 1)$. Por lo tanto, obtenemos

$$\mathcal{F}^{-1}(X) = X^h = Y^{h(q^\theta+1)} = Y^{k(q^n-1)+1} = Y. \tag{6.10}$$

La clave pública es $\mathcal{P} = \mathcal{S} \circ \overline{\mathcal{F}} \circ \mathcal{T} = \mathcal{S} \circ \phi^{-1} \circ \mathcal{F} \circ \phi \circ \mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$ con las dos aplicaciones lineales invertibles $\mathcal{S} : \mathbb{F}^n \to \mathbb{F}^n$ y $\mathcal{T} : \mathbb{F}^n \to \mathbb{F}^n$. La clave privada consiste en $\mathcal{S}$, h y $\mathcal{T}$. Sin embargo, también podemos asumir que h es público, ya que θ está en un rango muy pequeño. Con esto queremos decir que aunque h sea parte de la clave privada, su seguridad depende en gran medida de la elección de θ, que se asume tiene solo un número limitado de valores posibles dentro de un conjunto específico. Dado que θ

está restringido a un rango pequeño, la exposición de h (o la determinación de su valor por parte de un atacante) podría no comprometer significativamente la seguridad del criptosistema, siempre y cuando θ esté bien protegido y tenga un espacio de búsqueda limitado.

Este esquema puede usarse tanto para cifrado como para firmas digitales:

6.4.1. Esquema de cifrado

Cifrado

Para cifrar un texto plano $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}^n$, simplemente se calcula $\mathbf{w} = \mathcal{P}(\mathbf{z}) \in \mathbb{F}^n$.

Descifrado

Para descifrar un texto cifrado $\mathbf{w} \in \mathbb{F}^n$, se calcula recursivamente $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}) \in \mathbb{F}^n$, $X = \phi(\mathbf{x}) \in \mathbb{E}$, $Y = \mathcal{F}^{-1}(X) \in \mathbb{E}$, $\mathbf{y} = \phi^{-1}(Y) \in \mathbb{F}^n$ y $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. El texto plano correspondiente al texto cifrado $\mathbf{w}$ es $\mathbf{z} \in \mathbb{F}^n$.

6.4.2. Esquema de firma digital

Generación de firmas

Para generar una firma para un documento d, se usa una función hash $\mathbf{H} : \{\mathbf{0}, \mathbf{1}\}^* \to \mathbb{F}^{\mathbf{n}}$ para calcular el valor hash $\mathbf{w} = H(d) \in \mathbb{F}^n$. Después, se calcula $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{w}) \in \mathbb{F}^n$, $X = \phi(\mathbf{x}) \in \mathbb{E}$, $Y = F^{-1}(X) \in \mathbb{E}$, $\mathbf{y} = \phi^{-1}(Y) \in \mathbb{F}^n$ y $\mathbf{z} = \mathcal{T}^{-1}(\mathbf{y})$. La firma del documento d es $\mathbf{z} \in \mathbb{F}^n$.

Verificación de firmas

Para verificar si $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{F}^n$ es realmente una firma válida de d, se calcula $\mathbf{w} = H(d) \in \mathbb{F}^n$ y $\mathbf{w}' = \mathcal{P}(\mathbf{z}) \in \mathbb{F}^n$. Si $\mathbf{w}' = \mathbf{w}$ se cumple, la firma es aceptada, de lo contrario, es rechazada.

6.5. Seguridad y ataques estándar

Para la mayoría de los criptosistemas de clave pública multivariable, no existe una prueba de seguridad que reduzca la seguridad del esquema a la dificultad de un problema matemático bien conocido (por ejemplo, el problema MQ). Por tanto, la seguridad de estos esquemas se estima mediante la complejidad de los ataques relevantes contra los esquemas.

Los ataques estándar contra los esquemas de clave pública multivariable se pueden dividir en dos grupos:

- **Ataques directos:** El atacante intenta resolver directamente el sistema de ecuaciones polinómicas que define el criptosistema $\mathcal{P}(\mathbf{z}) = \mathbf{w}$, considerando el sistema como una instancia del problema MQ. Se pueden usar para atacar cualquier criptosistema de clave pública multivariable, pero han demostrado ser más eficientes contra sistemas con una gran estructura algebraica como HFE.

- **Ataques estructurales:** Los ataques estructurales intentan utilizar la estructura del mapa central de un criptosistema de clave pública multivariable para encontrar la composición $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$ de la clave pública. Los ejemplos más importantes de ataques estructurales contra esquemas multivariable son:

  - **Ataque de linealización de ecuaciones:** Se basa en explotar la estructura algebraica de las ecuaciones del sistema para transformarlas en un sistema de ecuaciones lineales que es más fácil de resolver. Es el ataque que se suele utilizar contra el criptosistema de Matsumoto-Imai (MI). De hecho, Patarin rompió el MI básico con un ataque de este tipo. Por ello se planteó otro criptosistema: el HFE, que consiste en agregar términos adicionales a la aplicación central de Matsumoto–Imai, asegurando que la aplicación central siga siendo eficientemente invertible y que la clave pública resultante siga siendo cuadrática. No obstante, el HFE básico tampoco se considera resistente a día de hoy, puesto que o bien se toman parámetros que lo hacen eficiente pero inseguro, o bien parámetros que lo hacen seguro pero ineficiente. Posteriormente se han propuesto algunas variantes de HFE, como la variante minus[1], la variante plus[2] o la variante vinegar[3], que sí que son resistentes. Por tanto, MI sigue siendo muy útil hoy en día, ya que se utiliza como base para construir esquemas más avanzados.
  - **Ataques diferenciales:** Los ataques diferenciales analizan cómo cambia la salida de la función de la clave pública cuando se realizan pequeños cambios en las entradas para obtener información sobre la aplicación central.
  - Por último, también hay que mencionar algunos ataques en los que los criptoanalistas utilizan propiedades de la aplicación central para obtener información sobre la clave privada.

[1] En la variante minus se eliminan algunas ecuaciones de la clave pública. Esto introduce una componente de "ruido" o "aleatoriedad" en el cifrado, dificultando los ataques criptográficos directos.
[2] La variante plus agrega ecuaciones cuadráticas elegidas al azar a la clave pública.
[3] Agrega perturbación a la aplicación central utilizando variables externas.

Capítulo 7

Criptografía basada en códigos correctores

Bibliografía utilizada para esta sección: [18], [60] y el curso [29].

Hasta ahora, hemos visto la importancia de construir algoritmos que nos permitan cifrar un mensaje y que éste no pueda ser descifrado fácilmente, de manera que solo el receptor que nos interese pueda descifrar nuestro mensaje. Sin embargo, es igual de importante que ese mensaje cifrado llegue a ese receptor sin haber sido modificado. Y esto es exactamente lo que hacen los *códigos correctores de errores*, que fueron introducidos para preservar la calidad de la información transmitida a través de un canal ruidoso, corrigiendo los errores en ese mensaje transmitido. El principio clave para lograr esta tarea es agregar redundancia. Los *códigos detectores de errores* permiten localizar errores en el mensaje pero sin corregirlos.

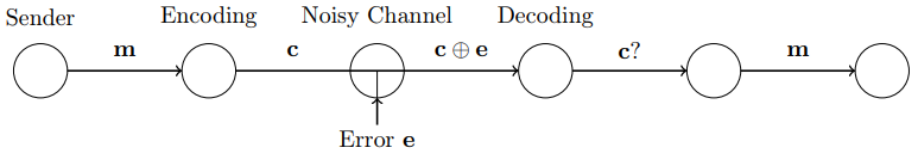

Figura 7.1: Un texto cifrado en el que se ha introducido un error muestra la necesidad de encontrar un mecanismo que pueda corregir y/o localizar ese error. [18]

7.1. Conceptos básicos sobre códigos correctores

Definición 7.1.1. *Un código de bloque C de longitud n es un subconjunto de A^n, donde A se define como el conjunto de alfabetos de C. Generalmente, A es un cuerpo finito. Un código de bloque q-ario de longitud n es un conjunto dado de secuencias de longitud n, de símbolos donde cada símbolo es elegido de un cuerpo finito $\mathbb{F}_q$. Un código de bloque de longitud n en el cual cada palabra de código es una repetición de un solo símbolo se llama un código de repetición de longitud n.*

Definición 7.1.2. *La distancia de Hamming*[1] *entre dos vectores de $\mathbb{F}_q^n$, denotada por $d_H(x, y)$, se define*

[1] La distancia de Hamming es una métrica en $\mathbb{F}_q^n$, ya que para todos los $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$, se cumplen las siguientes condiciones:

- **No negatividad:** $d_H(\mathbf{x}, \mathbf{y}) \geq 0$.

como el número de diferencias en los dígitos de x y y.

Definición 7.1.3. *La distancia mínima o distancia de Hamming de un código C se define como $d(C) :=$ mín$\{d(x, y)$ para todo $x, y \in C$, con $x \neq y\}$. Es decir, es la menor distancia de Hamming entre dos palabras distintas cualesquiera del código.*

La distancia de Hamming nos ayuda a corregir errores. Por ejemplo, si tenemos el código $C = \{(0,0,0),(1,1,1)\} \subset \mathbb{F}_2^3$. Si recibimos la palabra $\mathbf{y} = (0,1,1)$, podemos comprobar que $d(\mathbf{y},(0,0,0)) = 2$ y $d(\mathbf{y},(1,1,1)) = 1$. Como el mínimo de las distancias es la del vector $(1,1,1)$, suponemos que esa es la palabra que se envió (asumiendo una distribución de probabilidad de errores en la que es menos probable que ocurran 2 errores que que ocurra solo uno).

El número de errores que se pueden corregir depende de la distancia mínima:

Teorema 7.1.1. ***Capacidad correctora***
Un código con distancia mínima d puede detectar hasta $d-1$ errores y corregir $t = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Demostración:

Detección de errores

Consideremos dos palabras código $\mathbf{c_1}$ y $\mathbf{c_2}$ con distancia d entre ellas.

- Si una palabra código $\mathbf{c_1}$ es transmitida y hasta $d-1$ errores ocurren, la palabra recibida $\mathbf{r}$ tendrá una distancia de Hamming de al menos 1 y como máximo $d-1$ de $\mathbf{c_1}$.
- Si $\mathbf{c_2}$ es otra palabra código, la distancia de Hamming entre $\mathbf{r}$ y $\mathbf{c_2}$ será al menos $d-(d-1) = 1$. Esto significa que $\mathbf{r}$ no puede ser confundida con otra palabra código diferente a $\mathbf{c_1}$.

Por lo tanto, se puede detectar hasta $d-1$ errores, ya que cualquier palabra recibida que difiera de una palabra código en al menos 1 y hasta $d-1$ posiciones es detectable como incorrecta. Si hubiera d errores o más, la palabra resultante puede ser transformada en otra palabra código válida.

Corrección de errores

Consideremos la esfera de radio t alrededor de una palabra código $\mathbf{c}$. Esta esfera contiene todas las palabras que tienen una distancia de Hamming de t o menos de $\mathbf{c}$. Si se introduce un error de t bits o menos en $\mathbf{c}$, la palabra recibida $\mathbf{r}$ estará dentro de la esfera de radio t centrada en $\mathbf{c}$. No hay dos palabras código distintas $\mathbf{c_1}$ y $\mathbf{c_2}$ tal que sus esferas de radio t se superpongan. Si existieran dos palabras código $\mathbf{c_1}$ y $\mathbf{c_2}$ con una palabra $\mathbf{r}$ común en sus esferas de radio t, esto implicaría que la distancia entre $\mathbf{c_1}$ y $\mathbf{c_2}$ sería $\leq 2t < d$, lo cual es una contradicción porque la distancia mínima es d.

Así, una palabra recibida $\mathbf{r}$ que está dentro de la distancia t de una única palabra código puede ser corregida a esa palabra código. Por lo tanto, el código puede corregir hasta $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ errores. □

Decimos entonces que t es la capacidad correctora y que el código es t-corrector.

- $d_H(\mathbf{x},\mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$.
- **Simetría:** $d_H(\mathbf{x},\mathbf{y}) = d_H(\mathbf{y},\mathbf{x})$.
- **Desigualdad triangular:** $d_H(\mathbf{x},\mathbf{y}) \leq d_H(\mathbf{x},\mathbf{z}) + d_H(\mathbf{z},\mathbf{y})$.

Con esta distancia, $\mathbb{F}_q^n$ se convierte en un espacio métrico.

Definición 7.1.4. *El peso de Hamming, $wt(x)$, de la cadena de bits $x \in \mathbb{F}_q^n$ es el número de coordenadas no nulas en x.*

La relación entre la distancia y el peso es $wt(x) = d(x, 0)$.

7.2. Códigos lineales

Definición 7.2.1. ***Código lineal q-ario**: Sea C un subespacio de dimensión k de $\mathbb{F}_q^n$. Entonces decimos que C es un código q-ario $[n, k]$ de longitud n y dimensión k; y si $d(C) = d$, entonces C es un código q-ario $[n, k, d]$.*

Los códigos lineales C de dimensión k y longitud n sobre $\mathbb{F}_q$ son subespacios vectoriales de $\mathbb{F}_q^n$[2], y por lo tanto, la dimensión de estos subespacios es finita y además todos sus elementos pueden ser descritos en términos de una base.

Sea C[3] un código lineal $[n, k]$ sobre $\mathbb{F}_q$. Supongamos que tenemos un mensaje $\mathbf{m} = (m_1, \ldots, m_k) \in \mathbb{F}_q^k$, entonces hay q^k mensajes posibles. Cada palabra de código de C puede representar una pieza de información, por lo que C puede representar q^k piezas de información distintas.

Figura 7.2: Diagrama del proceso de codificación lineal

Toda transformación lineal se puede representar por una matriz.

Definición 7.2.2. ***Matriz generadora**: La matriz generadora de un código lineal q-ario $[n, k]$ C es una matriz de $k \times n$, cuyas filas forman una base del espacio vectorial C sobre $\mathbb{F}_q$.*

Como la base no es única, tampoco lo será la matriz generadora.

Por tanto, C se puede escribir como $C = \{\mathbf{m}G : \mathbf{m} \in \mathbb{F}_q^k\}$.

---

[2]

- 1) C es cerrado para la suma: $\forall \mathbf{c_1}, \mathbf{c_2} \in C \Rightarrow \mathbf{c_1} + \mathbf{c_2} \in C$
- 2) C es cerrado bajo la multiplicación por un escalar: $\forall \lambda \in \mathbb{F}_q, \forall \mathbf{c} \in C \Rightarrow \lambda \mathbf{c} \in C$
- 3) El vector cero siempre es una palabra de código: $(0, \ldots, 0) \in C$

[3] Nótese que C es isomorfo a $\mathbb{F}_q^k$. Esto se puede ver fácilmente: hemos dicho que C es un subespacio vectorial de $\mathbb{F}_q^n$ de dimensión k. Esto implica que existe una base $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ de C tal que cualquier código palabra $\mathbf{c} \in C$ puede escribirse como una combinación lineal de estos k vectores base. Definimos la aplicación $\varphi : \mathbb{F}_q^k \to C$ definida por

$$\varphi(a_1, a_2, \ldots, a_k) = a_1\mathbf{c}_1 + a_2\mathbf{c}_2 + \cdots + a_k\mathbf{c}_k \tag{7.1}$$

Esta aplicación es lineal y biyectiva:

- Inyectividad: Si $\varphi(a_1, a_2, \ldots, a_k) = \varphi(b_1, b_2, \ldots, b_k)$, entonces $a_1\mathbf{c}_1 + a_2\mathbf{c}_2 + \cdots + a_k\mathbf{c}_k = b_1\mathbf{c}_1 + b_2\mathbf{c}_2 + \cdots + b_k\mathbf{c}_k$. Como $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ es una base, esto implica que $(a_1, a_2, \ldots, a_k) = (b_1, b_2, \ldots, b_k)$.
- Sobreyectividad: Para cada $\mathbf{c} \in C$, hay un $(a_1, a_2, \ldots, a_k) \in \mathbb{F}_q^k$ tal que $\mathbf{c} = a_1\mathbf{c}_1 + a_2\mathbf{c}_2 + \cdots + a_k\mathbf{c}_k$.

7.2.1. Codificación

A la hora de codificar se consideran bloques de k símbolos de información, y se necesita una función que envíe vectores de longitud k a palabras del código (de manera inyectiva). Nos referimos a aplicar una función del tipo:

$$\text{Codificar} : \mathbb{F}_q^k \to \mathbb{F}_q^n, \tag{7.2}$$

es decir, modificamos los primeros k bits y adjuntamos $n-k$ bits de información adicionales al mensaje $\mathbf{m}$:

$$(m_1, m_2, \ldots, m_k) \to (\hat{m}_1, \hat{m}_2, \ldots, \hat{m}_k, \hat{m}_{k+1}, \ldots, \hat{m}_n) \tag{7.3}$$

Siendo $(\hat{m}_{k+1}, \ldots, \hat{m}_n)$ $n-k$ bits de redundancia.

Sea G una matriz generadora del código C, que es una matriz $k \times n$. Como hemos dicho antes, C contiene q^k palabras de código y por lo tanto C puede ser utilizado para comunicar q^k mensajes distintos. La codificación con un código lineal se puede realizar multiplicando el vector mensaje con la matriz generadora del código lineal, es decir, $\mathbf{m} \to \mathbf{m} \cdot G$. El vector resultante $\mathbf{m} \cdot G$ es de hecho una palabra de código de C, ya que es una combinación lineal de las filas de la matriz generadora.

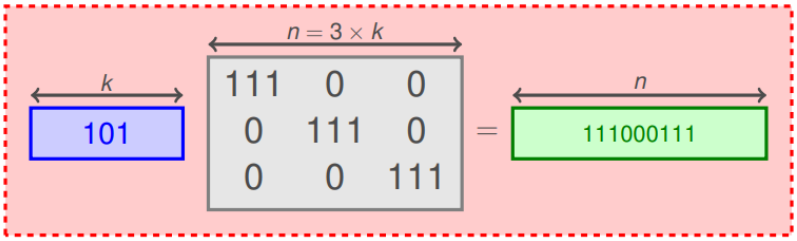

Figura 7.3: Ejemplo: código de repetición.

La idea importante es la siguiente: codificar es un problema sencillo, es una multiplicación matricial.

Definición 7.2.3. *La matriz $H_{(n-k)\times n}$ es una **matriz de comprobación de paridad** de C si y solo si C es el espacio nulo de H. Es decir:*

$$C = \{\mathbf{x} \in \mathbb{F}_q^n \,|\, \mathbf{x}H^T = \mathbf{0}\}. \tag{7.4}$$

Ejemplo: Código de Hamming [7,4]

Los códigos de Hamming ([25]) son códigos binarios, construidos sobre $\mathbb{F}_2$. La idea es añadir bits de paridad, es decir, bits que sean la suma de símbolos en otras posiciones. En este caso tenemos 3 bits de paridad r_1, r_2, r_3 para 4 bits de información m_1, m_2, m_3, m_4. Se calculan los bits de paridad de la siguiente manera:

$$r_1 = m_1 + m_2 + m_4 \bmod 2 \tag{7.5}$$
$$r_2 = m_1 + m_3 + m_4 \bmod 2 \tag{7.6}$$
$$r_3 = m_2 + m_3 + m_4 \bmod 2 \tag{7.7}$$

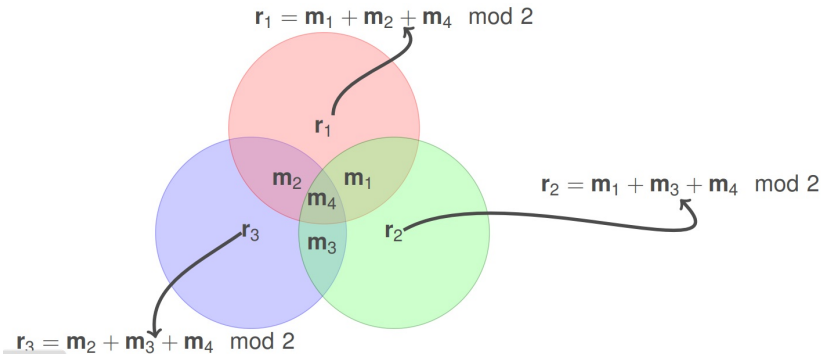

Figura 7.4: Código de Hamming [7,4].[29]

Para un código de Hamming $[7, 4]$, la distancia mínima d es 3. Por tanto, puede corregir $\lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{3-1}{2} \rfloor = 1$ errores. Por otro lado, puede detectar hasta $d - 1 = 3 - 1 = 2$ errores. Si solo uno de los bits de paridad es incorrecto en la palabra recibida, ese bit contiene el error; en otro caso, si más de un bit de paridad es incorrecto, se puede determinar qué bit de información contiene el error. Veamos un ejemplo:

Supongamos que recibimos la palabra $(m_1, m_2, m_3, m_4, r_1, r_2, r_3) = (1, 1, 1, 1, 0, 0, 1)$. Calculamos los bits de paridad.

$$r_1 = m_1 + m_2 + m_4 = 1 + 1 + 1 = 1 \tag{7.8}$$
$$r_2 = m_1 + m_3 + m_4 = 1 + 1 + 1 = 1 \tag{7.9}$$
$$r_3 = m_2 + m_3 + m_4 = 1 + 1 + 1 = 1 \tag{7.10}$$

Observamos que hay discrepancia en los bits r_1, r_2 y de esto podemos concluir (siempre que haya ocurrido un único error)[4] que se ha producido ese error en el bit de información m_1. La palabra corregida es $(0, 1, 1, 1, 0, 0, 1)$.

Vamos a calcular una matriz generadora G. Para codificar los vectores de información $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$ tenemos que añadir los vectores de paridad correspondientes (calculados con 7.5, 7.6 y 7.7) nos da una base del código, que escribimos por filas:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{7.11}$$

Y para obtener una matriz de paridad escribimos las ecuaciones 7.8, 7.9 y 7.10 de forma matricial:

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{7.12}$$

Proposición 7.2.1. *Sea C un código lineal q-ario $[n, k]$ con matriz generadora G. Entonces H es una matriz de comprobación de paridad para C si $G \cdot H^T = 0$.*

DEMOSTRACIÓN Usando la definición de matriz de comprobación de paridad, $\mathbf{c}H^T = \mathbf{0}$, para todo $\mathbf{c} \in C$. Y como cada palabra de código tiene la forma $\mathbf{c} = \mathbf{m}G$ con $\mathbf{m} \in \mathbb{F}_q^k$, se sustituye y $(\mathbf{m}G)H^T = 0$,

[4] Aquí de nuevo hay que asumir que el error tiene una distribución de probabilidad que hace más probable que ocurran menos errores a que ocurran más.

para todo $\mathbf{m} \in F_q^k$, de lo que se concluye que $GH^T = 0$. $\square$

La codificación es más simple cuando la matriz generadora se da en la llamada *forma estándar*, es decir, de la forma $[I_k|A_{k\times(n-k)}]$. Como $\mathbf{m}G = \mathbf{c}$ y el primer bloque de G es la identidad, las k primeras componentes de $\mathbf{c}$ van a ser $\mathbf{m}$. Esta forma se utiliza en el criptosistema HyMES, que veremos posteriormente. También podemos escribir una matriz de comprobación de paridad en esta forma: $[Y|I_{n-k}]$.

Figura 7.5: Visualización de los distintos bloques de G y H en forma estándar.

Podemos relacionar la matriz generadora y la matriz de comprobación de paridad cuando están en forma estándar de la siguiente manera:

Proposición 7.2.2. *$G = [I_k|A_{k\times(n-k)}]$ es matriz generadora de C $\iff$ $H = [-A^T_{(n-k)\times k}|I_{n-k}]$ es matriz comprobadora de paridad de C.*

Demostración
La implicación $\Rightarrow$:

Supongamos que $G = [I_k \mid A_{k\times(n-k)}]$ es una matriz generadora de C. Para probar que $H = [-A^T_{(n-k)\times k} \mid I_{n-k}]$ es una matriz comprobadora de paridad de C, necesitamos demostrar que cualquier vector $\mathbf{c}$ en C satisface la relación $\mathbf{c}H^T = \mathbf{0}$.

Tomemos un vector $\mathbf{c}$ en C. Por definición de G, $\mathbf{c}$ puede ser expresado como $\mathbf{c} = \mathbf{m}G$ para algún vector de mensaje $\mathbf{m}$ de longitud k. Entonces,

$$\mathbf{c} = \mathbf{m}[I_k \mid A_{k\times(n-k)}] = [\mathbf{m}I_k \mid \mathbf{m}A_{k\times(n-k)}]. \tag{7.13}$$

Por tanto,

$$\mathbf{c}H^T = [\mathbf{m}I_k \mid \mathbf{m}A_{k\times(n-k)}] \begin{bmatrix} -A_{k\times(n-k)} \\ I_{n-k} \end{bmatrix} = \mathbf{m}(-A) + \mathbf{m}A = \mathbf{m}(-A + A) = \mathbf{0}. \tag{7.14}$$

La otra implicación $\Leftarrow$:

Supongamos que $H = [-A^T_{(n-k)\times k} \mid I_{n-k}]$ es una matriz comprobadora de paridad de C. Para demostrar que $G = [I_k \mid A_{k\times(n-k)}]$ es una matriz generadora de C, necesitamos demostrar que cualquier vector $\mathbf{c}$ en C puede ser expresado como $\mathbf{c} = \mathbf{m}G$ para algún vector de mensaje $\mathbf{m}$ de longitud k.

Tomemos un vector $\mathbf{c}$ en C. Por definición de H, $\mathbf{c}$ satisface la relación $H\mathbf{c}^T = \mathbf{0}$. Entonces,

$$H\mathbf{c}^T = \mathbf{0} \tag{7.15}$$

$$\begin{bmatrix} -A^T & I_{n-k} \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \mathbf{0} \tag{7.16}$$

$$-A^T\mathbf{c}_1 + \mathbf{c}_2 = \mathbf{0} \tag{7.17}$$

Esto nos lleva a la ecuación

$$A^T \mathbf{c}_1 = \mathbf{c}_2. \tag{7.18}$$

Como $\mathbf{c} = [\mathbf{c}_1 \mid \mathbf{c}_2]$ y $\mathbf{c}_2 = A^T \mathbf{c}_1$, podemos escribir:

$$\mathbf{c} = [\mathbf{c}_1 \mid A^T \mathbf{c}_1]. \tag{7.19}$$

Definiendo $\mathbf{m} = \mathbf{c}_1$, tenemos:

$$\mathbf{c} = [\mathbf{m} \mid A^T \mathbf{m}] = \mathbf{m}[I_k \mid A]. \tag{7.20}$$

Así, cualquier vector $\mathbf{c}$ en C puede ser expresado como $\mathbf{c} = \mathbf{m}G$ para algún vector de mensaje $\mathbf{m}$ de longitud k. Esto demuestra que $G = [I_k \mid A_{k\times(n-k)}]$ es una matriz generadora de C. □

7.2.2. Decodificación

Supongamos que enviamos la palabra de código $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ a través de un canal y el vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ es recibido. Definimos el **vector de error** e como: $\mathbf{e} = \mathbf{y} - \mathbf{x} = (e_1, e_2, \ldots, e_n)$.

El decodificador debe decidir a partir de $\mathbf{y}$ qué palabra de código $\mathbf{x}$ fue transmitida, o equivalentemente cuál es el vector de error correspondiente $\mathbf{e}$.

Figura 7.6: Esquema básico de la decodificación. m es el mensaje, G es la matriz de codificación, e es el ruido.

Este proceso se llama decodificación con un código lineal. Por ejemplo, un método es el método de decodificación por síndrome.

Decodificación por síndrome

Primero establecemos algunas definiciones:

Definición 7.2.4. *Sea C un código lineal de longitud n sobre $\mathbb{F}_q$, y sea $\mathbf{u} \in \mathbb{F}_q^n$ cualquier vector de longitud n; definimos el **coset** de C determinado por $\mathbf{u}$ como el conjunto*

$$C + \mathbf{u} = \{\mathbf{v} + \mathbf{u} : \mathbf{v} \in C\} = \mathbf{u} + C. \tag{7.21}$$

Definición 7.2.5. *Un vector en un coset se llama **líder del coset** si tiene peso de Hamming mínimo. En caso de que hubiera varios con peso de Hamming mínimo, cualquiera de ellos puede ser elegido como el líder del coset.*

Sea C un código $[n, k]$ con matriz de verificación de paridad H. Por tanto, si $\mathbf{c} \in C$, entonces $H\mathbf{c}^T = 0$.

Definición 7.2.6. *El **síndrome de un vector** $\mathbf{x} \in \mathbb{F}_q^n$ es el vector $S(\mathbf{x}) = H\mathbf{x}^T \in \mathbb{F}_q^{n-k}$.*

Veamos algunas propiedades útiles. Sean $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$, tenemos que:

1. $S(\mathbf{u} + \mathbf{v}) = S(\mathbf{u}) + S(\mathbf{v})$.
2. $S(\mathbf{y}) = 0$ si y solo si $\mathbf{y} \in C$.
3. $S(\mathbf{u}) = S(\mathbf{v})$ si y solo si $\mathbf{u}$ y $\mathbf{v}$ están en el mismo coset de C.

Sea $\mathbf{y} = \mathbf{c} + \mathbf{e}$, donde $\mathbf{y}$ es la palabra recibida, $\mathbf{c}$ es la palabra codificada enviada y $\mathbf{e}$ es el vector de error de la palabra recibida.

$$H\mathbf{y}^T = H(\mathbf{c} + \mathbf{e})^T = H\mathbf{c}^T + H\mathbf{e}^T = H\mathbf{e}^T. \tag{7.22}$$

Decodificación por síndrome - Tabla de búsqueda
Una tabla que relaciona cada líder de coset con su síndrome se llama tabla de búsqueda de síndromes. Pasos para construir una tabla de búsqueda de síndromes:

1. Paso 1: Enumerar todos los cosets para el código y eligir de cada coset una palabra de peso mínimo como líder del coset $\mathbf{u}$.
2. Paso 2: Encontrar una matriz de verificación de paridad H para el código y, para cada líder de coset $\mathbf{u}$, calcular su síndrome $S(\mathbf{u}) = H\mathbf{u}^T$.

La decodificación por síndrome funciona de la siguiente manera: se calcula el síndrome del vector recibido (7.22) y toma el líder de coset cuyo síndrome coincide con el del vector recibido.

1. Paso 1: Para el vector recibido $\mathbf{y}$, calcular el síndrome $S(\mathbf{y})$.
2. Paso 2: Encontrar el líder de coset $\mathbf{u}$ correspondiente al síndrome $S(\mathbf{y}) = S(\mathbf{u})$ en la tabla de búsqueda de síndromes.
3. Paso 3: Decodificar $\mathbf{y}$ como $\mathbf{v} = \mathbf{y} - \mathbf{u}$.

7.3. Uso de los códigos en teoría de la información

El uso más evidente de los códigos correctores en la teoría de la información es, teniendo cualquier criptosistema (por ejemplo, RSA), asegurarnos de que un mensaje cifrado ha llegado a su receptor sin haber sido modificado (ya sea por un agente externo malicioso o simplemente por ruido en el canal de comunicación), detectando y corrigiendo los errores en caso de haberlos.

Por lo tanto, son esenciales para preservar la calidad de la información que se transmite en cualquier comunicación. Sin embargo, también tienen otro uso fundamental: nos permiten construir criptosistemas. La idea fundamental es la siguiente: codificar es fácil (es una multiplicación matricial en el caso de códigos lineales), mientras que, en general, decodificar es muy difícil ([18]).

Sin embargo, para algunas familias de códigos y con determinadas distancias de código, existen algoritmos de decodificación eficientes si se tiene cierta información (*trapdoor*), por ejemplo, los códigos Reed-Solomon o los códigos Goppa. Esto hace que sean idóneos para su uso en criptografía, y por tanto, usando esta idea, se construyeron criptosistemas como los que detallamos a continuación:

7.4. Criptosistema de McEliece

Este criptosistema de clave pública es muy antiguo; se publicó en 1978 en [40], pero no obtuvo el reconocimiento que obtuvieron otros criptosistemas de esa época ya que se necesitan claves de gran tamaño. Sin embargo, cuando el algoritmo de Shor apareció, comprometiendo la seguridad de los criptosistemas basados en teoría de números, el valor de los criptosistemas basados en códigos ascendió, y en particular el del criptosistema de McEliece, que hasta ahora ha demostrado ser resistente siempre y cuando elijamos un código adecuado.

La idea de McEliece es la siguiente: Alice tiene un código C el cual puede decodificar eficientemente con hasta t errores. La clave privada va a ser precisamente ese algoritmo de decodificación eficiente. Por otra parte, la clave pública va a ser (G, t) siendo G la matriz generadora de C y t el número de errores que puede corregir ese algoritmo de la clave privada. Por razones de seguridad obvias, Alice no quiere que G revele ninguna información sobre cómo decodificar C. En ese caso, la situación perfecta corresponde a una matriz G que está distribuida uniformemente. Ahora, Bob quiere enviar un mensaje $\mathbf{m}$ a Alice. Para ello, lo cifra, haciendo $\mathbf{m}G + \mathbf{e} = \mathbf{y}$, siendo $\mathbf{e}$ un vector de error aleatorio de peso de Hamming t. Luego envía $\mathbf{y}$ a Alice. Ahora Alice recupera $\mathbf{m}$ a partir de $\mathbf{y}$ gracias a su algoritmo de decodificación.

Uno puede preguntarse por qué Bob ha elegido un vector de error aleatorio $\mathbf{e}$ de peso de Hamming t y no $\leq t$, ya que con peso $\leq t$ también funcionaría, pues Alice puede decodificar con hasta t errores. La razón es que decodificar es más difícil cuanto mayor sea el peso de Hamming del vector $\mathbf{e}$. Por lo tanto, nos interesa que tenga el peso de Hamming lo más grande posible, es decir, t, para que sea lo más seguro posible.

En realidad, al implementar el algoritmo hay algunos consideraciones técnicas que hay que tener en cuenta: en realidad la clave pública no es la matriz generadora G, sino un producto de la matriz generadora por otras 2 matrices ($A = S \cdot G \cdot P$)[5]. Este detalle es muy importante para garantizar la seguridad: al publicar el producto en lugar de la matriz generadora directamente podemos seguir realizando la decodificación pero al no mostrar públicamente G, no damos ninguna información sobre nuestro código y por tanto no es vulnerable a diferentes tipos de ataques. Veamos cómo funciona el algoritmo con más detalle:

Generación de claves

Primero, Alice genera las claves (la pública y la privada). Para ello:

(i) Selecciona un código Goppa binario $[n, k]$ (definiremos estos códigos en la sección posterior), con su matriz generadora G de tamaño $k \times n$, capaz de corregir t errores.

(ii) Luego, selecciona una matriz binaria no singular S de tamaño $k \times k$ y una matriz de permutación P de tamaño $n \times n$.

(iii) Calcula la matriz $A = S \cdot G \cdot P$.

(iv) Publica la clave pública: (A, t) y guarda la clave privada: (S, G, P).

[5] Esta idea ya la hemos visto aplicada de forma similar en el esquema bipolar de la criptografía multivariable; realizábamos una composición de la aplicación central con otras dos para no dar ninguna información sobre nuestra aplicación central y así intentar conseguir que la clave pública indistinguible de un sistema aleatorio.

Envío del mensaje cifrado

Para enviar un mensaje cifrado a Alice, Bob tiene que realizar lo siguiente:

(i) Tiene un mensaje $\mathbf{m}$ de longitud k y la clave pública de Alice: (A, t).

(ii) Genera un vector binario aleatorio de error $\mathbf{z}$ de n bits con peso de Hamming t.

(iii) Cifra el mensaje $\mathbf{c} = \mathbf{m} \cdot A + \mathbf{z}$ y se lo envía a Alice.

Descifrado del mensaje

Una vez que Alice ha recibido el texto cifrado $\mathbf{c}$, recupera el mensaje de la siguiente manera:

(i) Calcula P^{-1}.

(ii) Multiplica $\mathbf{c}$ por P^{-1} y calcula $\mathbf{c} \cdot P^{-1} = \mathbf{m} \cdot A \cdot A^{-1} \cdot S \cdot G + \mathbf{z} \cdot P^{-1} = \mathbf{m} \cdot S \cdot G + \mathbf{z} \cdot P^{-1}$ (vemos que tiene peso t por ser P una matriz de permutación, que no altera la cantidad de unos y ceros).

(iii) Finalmente, usa el algoritmo de decodificación (normalmente se utiliza el algoritmo de Patterson, que veremos en secciones posteriores) y con ello obtiene $\mathbf{m}$.

La seguridad del criptosistema de McEliece reside en la dificultad del *problema general de decodificación*: dado un código [n, k] C sobre $\mathbb{F}_q$, un entero t_0 y un vector $\mathbf{x} \in \mathbb{F}_q^n$, encontrar una palabra de código $\mathbf{c} \in C$ con $d(\mathbf{x}, \mathbf{c}) \leq t_0$. En general esto es un problema difícil (NP-hard), véase [5] y [4], pero para algunos tipos de códigos existen algoritmos de decodificación eficientes con complejidad polinómica. Un ejemplo de esto son los códigos de Reed-Solomon o los códigos Goppa, que son los que se utilizan en McEliece.

Un detalle muy importante que hay que tener claro de este algoritmo es que el código C no puede ser uno cualquiera; tiene que cumplir dos propiedades:

- Evidentemente tiene que tener un algoritmo de decodificación eficiente.
- Además tiene que ser indistinguible de un código aleatorio. Esto es necesario por lo que hemos dicho antes: no nos interesa que G revele ninguna información sobre cómo decodificar C, ya que G es público y entonces el algoritmo dejaría de ser seguro. Esto se intenta solucionar con la introducción de las matrices S y P en el algoritmo.

Hay varias familias de códigos que cumplen estas dos condiciones y por tanto son candidatas para ser usadas en criptografía. Sin embargo, muchos de ellos han demostrado ser inseguros:

- Códigos Reed-Solomon generalizados. En 1992 se publicó un artículo que demostró que existe un algoritmo que recupera el mensaje en tiempo polinómico. [58].
- Subcódigos de códigos Reed-Solomon generalizados. En [61], publicado en 2010, se demostró que tampoco eran seguros.
- Códigos Reed-Muller. También se demostró en [41] (2007) y [15] (2013) que estos códigos no eran resistentes.

Y como estos hay muchas más familias de códigos. Los únicos que, hasta el momento, siguen siendo resistentes, son los códigos Goppa binarios[6]. Veámoslos:

7.5. Códigos Goppa

Definición 7.5.1. *Sea* $g(z) = g_0 + g_1 z + g_2 z^2 + \ldots + g_t z^t$[7] $\in \mathbb{F}_{q^m}[z]$, *y sea* $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$ *tal que,* $g(\alpha_i) \neq 0$ *para todo* $\alpha_i \in L$. *Entonces, el código lineal definido por*

$$\{\mathbf{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_q^n : \sum_{i=1}^{n} \frac{c_i}{(z - \alpha_i)} \equiv 0 \ mod\ g(z)\} \tag{7.23}$$

se llama ***código Goppa*** *con parámetros* $g(z)$ *y* L *denotado por* $\Gamma(L, g(z))$.

Para intentar entender mejor esta definición, podemos pensar en los g_i y los α_i como números de m dígitos en base q, y tenemos que $\sum_{i=1}^{n} \frac{c_i}{(z-\alpha_i)}$ es múltiplo de $g(z)$.

Para cada i (con $1 \leq i \leq n$), $g(\alpha_i) \neq 0$ equivalente a $\gcd(z - \alpha_i, g(z)) = 1$, es decir, $(z - \alpha_i)$ no es un factor de $g(z)$, la fracción $\frac{1}{z-\alpha_i}$ se calcula en $\frac{\mathbb{F}_{q^m}[z]}{\langle g(z) \rangle}$ como:

Teorema 7.5.1. *El inverso multiplicativo de* $(z - \alpha_i)$ *existe en el anillo cociente* $\frac{\mathbb{F}_{q^m}[z]}{\langle g(z) \rangle}$; *el valor de* $(z - \alpha_i)^{-1}$ *en* $\frac{\mathbb{F}_{q^m}[z]}{\langle g(z) \rangle}$ *es* $-\left(\frac{g(z)-g(\alpha_i)}{z-\alpha_i}\right) g(\alpha_i)^{-1}$. *Por tanto,*

$$\mathbf{c} \in \Gamma(L, g) \iff \sum_i c_i \left(\frac{g(\alpha_i) - g(z)}{z - \alpha_i}\right) g(\alpha_i)^{-1} \equiv 0 \ mod\ g(z). \tag{7.24}$$

A partir de este teorema podemos derivar el siguiente corolario que nos permite caracterizar los códigos:

$$\mathbf{c} \in \Gamma(L, g) \iff \sum_i c_i \left(\frac{g(\alpha_i) - g(z)}{z - \alpha_i}\right) g(\alpha_i)^{-1} = 0 \text{ como un polinomio en } \frac{\mathbb{F}_{q^m}[z]}{\langle g(z) \rangle}. \tag{7.25}$$

La matriz de control de paridad sobre $\mathbb{F}_{q^m}$ para los códigos Goppa es:

$$H = \begin{bmatrix} g_t g(\alpha_1)^{-1} & g_t g(\alpha_2)^{-1} & \cdots & g_t g(\alpha_n)^{-1} \\ (g_t \alpha_1 + g_{t-1}) g(\alpha_1)^{-1} & (g_t \alpha_2 + g_{t-1}) g(\alpha_2)^{-1} & \cdots & (g_t \alpha_n + g_{t-1}) g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ (g_t \alpha_1^{t-1} + \cdots + g_1) g(\alpha_1)^{-1} & (g_t \alpha_2^{t-1} + \cdots + g_1) g(\alpha_2)^{-1} & \cdots & (g_t \alpha_n^{t-1} + \cdots + g_1) g(\alpha_n)^{-1} \end{bmatrix} \tag{7.26}$$

[6] Un detalle importante es el siguiente: esto no significa que los códigos Goppa sean más resistentes que los otros que hemos puesto como ejemplo. Haciendo una búsqueda rápida se puede ver que los códigos Goppa son menos estudiados que los Reed-Solomon o los Reed-Muller. Por ejemplo, a día 22 de mayo de 2024, buscando en Google Académico vemos que *Reed-Muller* aparece en alrededor de 20700 artículos, *Reed-Solomon* en alrededor de 69500 artículos, mientras que *Goppa* solo en 13600 artículos. También haciendo una búsqueda rápida en el archivo de ePrint de criptología (que es un repositorio en el que se publica un acceso rápido a investigaciones recientes en criptología que todavía no han pasado por un proceso de revisión), y aquí *Reed-Muller* aparece en alrededor de 40 artículos, *Reed-Solomon* en alrededor de 69 artículos, mientras que *Goppa* solo en 37 artículos. Es decir, los códigos Goppa han sido menos estudiados que los otros, por lo tanto es posible que en un tiempo dejen de ser seguros al igual que los otros.

[7] La distancia mínima d de un código de Goppa $\Gamma(L, g(x))$ de longitud n es $d \geq t + 1$, siendo t el grado del polinomio $g(z)$. No confundir con el t definido antes.

Esta matriz se puede descomponer en el producto de tres matrices de la siguiente forma:

$$H = \underbrace{\begin{bmatrix} g_t & 0 & \cdots & 0 \\ g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & \cdots & g_t \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \cdots & \alpha_n^{t-1} \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} g(\alpha_1)^{-1} & 0 & \cdots & 0 \\ 0 & g(\alpha_2)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g(\alpha_n)^{-1} \end{bmatrix}}_{Y} \tag{7.27}$$

Ahora, como tenemos que $\mathbf{c} \in \Gamma(L, g) \iff \mathbf{c}H^T = \mathbf{0}$, entonces $\mathbf{c}(CXY)^T = 0$, y por tanto $\mathbf{c}Y^T X^T C^T = 0$. Como la matriz C es invertible, necesariamente $\mathbf{c}Y^T X^T = 0$. Y por tanto, $\mathbf{c}(XY)^T = 0$, por lo que la matriz XY puede verse como la matriz de control de paridad para $\Gamma(L, g)$ sobre $\mathbb{F}_{q^m}$. [8]

$$XY = \begin{bmatrix} g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \alpha_2 g(\alpha_2)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} g(\alpha_1)^{-1} & \alpha_2^{t-1} g(\alpha_2)^{-1} & \cdots & \alpha_n^{t-1} g(\alpha_n)^{-1} \end{bmatrix}_{t \times n} \tag{7.28}$$

Esta matriz es $t \times n$. Pero podemos ver cada elemento de la matriz de $\mathbb{F}_{q^m}$ como vectores columna de longitud m sobre $\mathbb{F}_q$ por isomorfismo de espacio vectorial, y entonces tenemos una matriz de verificación de paridad para $\Gamma(L, g)$ sobre $\mathbb{F}_q$ que es una matriz $mt \times n$, con al menos t columnas linealmente independientes sobre $\mathbb{F}_q$. Por lo tanto, la distancia de Hamming del código Goppa es $d(\Gamma(L, g)) \geq t + 1$. Esto, por el teorema 7.1.1, nos indica que como mucho puede detectar t errores y corregir $\lfloor \frac{t}{2} \rfloor$ [9].

Un caso particular de los códigos Goppa son los **códigos Goppa binarios**, que resultan de aplicar $q = 2$ en la definición. Es decir:

Definición 7.5.2. *Sea $g(z) = g_0 + g_1 z + g_2 z^2 + \ldots + g_t z^t \in \mathbb{F}_{2^m}[z]$, y sea $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \subseteq \mathbb{F}_{2^m}$ tal que, $g(\alpha_i) \neq 0$ para todo $\alpha_i \in L$. Entonces, el código definido por*

$$\{\mathbf{c} = (c_1, c_2, \ldots, c_n) \in \{0, 1\}^n : \sum_{i=1}^{n} \frac{c_i}{(z - \alpha_i)} \equiv 0 \ mod\ g(z)\} \tag{7.29}$$

*se llama **código Goppa binario** con parámetros $g(z)$ y L, denotado por $\Gamma(L, g(z))$.*

7.5.1. Ejemplo de código Goppa binario

Sea $\mathbb{F}_{2^4}$ el cuerpo isomorfo a $\frac{\mathbb{F}_2[x]}{\langle x^4 + x + 1 \rangle}$. Sea α una raíz de $x^4 + x + 1$ en una extensión algebraicamente cerrada de $\mathbb{F}_2$, entonces, dado que el orden multiplicativo de α es 15, puede usarse para generar todos los

[8] La matriz Y es invertible al ser diagonal con elementos distintos de cero en la diagonal.
[9] De nuevo seguimos con esta notación en la que t es el grado del polinomio $g(z)$.

elementos de $\mathbb{F}_{2^4}^*$ es decir, $\mathbb{F}_{2^4} = \langle\alpha\rangle$. Por lo tanto, podemos $\mathbb{F}_{2^4}$ como:

$$\begin{aligned}
0 &= && = (0,0,0,0)^T,\\
1 &= && = (1,0,0,0)^T,\\
\alpha &= && = (0,1,0,0)^T,\\
\alpha^2 &= && = (0,0,1,0)^T,\\
\alpha^3 &= && = (0,0,0,1)^T,\\
\alpha^4 &= 1+\alpha && = (1,1,0,0)^T,\\
\alpha^5 &= \alpha+\alpha^2 && = (0,1,1,0)^T,\\
\alpha^6 &= \alpha^2+\alpha^3 && = (0,0,1,1)^T,\\
\alpha^7 &= 1+\alpha+\alpha^3 && = (1,1,0,1)^T,\\
\alpha^8 &= 1+\alpha^2 && = (1,0,1,0)^T,\\
\alpha^9 &= \alpha+\alpha^3 && = (0,1,0,1)^T,\\
\alpha^{10} &= 1+\alpha+\alpha^2 && = (1,1,1,0)^T,\\
\alpha^{11} &= \alpha+\alpha^2+\alpha^3 && = (0,1,1,1)^T,\\
\alpha^{12} &= 1+\alpha+\alpha^2+\alpha^3 && = (1,1,1,1)^T,\\
\alpha^{13} &= 1+\alpha^2+\alpha^3 && = (1,0,1,1)^T,\\
\alpha^{14} &= 1+\alpha^3 && = (1,0,0,1)^T.
\end{aligned} \tag{7.30}$$

Consideremos el código Goppa $\Gamma(L, g(z))$ definido por

$$g(z) = (z+\alpha)(z+\alpha^{14}) = z^2 + z(1+\alpha^3) + \alpha z + 1 = z^2 + z(1+\alpha+\alpha^3) + 1 = z^2 + \alpha^7 z + 1 \tag{7.31}$$

$$L = \{\alpha^i | 2 \leq i \leq 13\} \tag{7.32}$$

La matriz de comprobación de paridad H se calcula usando (7.28). Para ello necesitamos calcular $g(\alpha^2)^{-1} = (\alpha^4 + \alpha^9 + 1)^{-1} = (0,0,0,1)^T = \alpha^{12}$, y de manera similar el resto de entradas.

$$H = \begin{pmatrix} \alpha^9 & \alpha^{10} & \alpha^9 & \alpha^{14} & \alpha^6 & 0 & \alpha^{10} & \alpha^8 & \alpha^2 & \alpha^7 & \alpha^{14} & \alpha^6 \\ \alpha^{12} & \alpha^6 & \alpha^6 & \alpha & \alpha^{11} & 1 & \alpha^{14} & \alpha^8 & \alpha^{11} & \alpha^{14} & \alpha^{12} & \alpha \end{pmatrix} \tag{7.33}$$

Las entradas pueden ser observadas como vectores binarios de longitud 4 (escritos por columnas) de acuerdo con (7.30):

$$H = \begin{pmatrix} 0&1&0&1&0&0&1&1&0&1&1&0 \\ 1&1&1&0&0&0&1&0&0&1&0&0 \\ 0&1&0&0&1&0&1&1&1&0&0&1 \\ 1&0&1&1&1&0&0&0&0&1&1&1 \\ 1&0&0&0&0&1&1&1&0&1&1&0 \\ 1&0&0&1&1&0&0&0&1&0&1&1 \\ 1&1&1&0&1&0&0&1&1&0&1&0 \\ 1&1&1&0&1&0&1&0&1&1&1&0 \end{pmatrix} \tag{7.34}$$

Entonces, el espacio nulo de H produce la matriz generadora G de $\Gamma(L, g(z))$:

$$G = \begin{pmatrix} 1\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0 \\ 0\,1\,0\,0\,1\,1\,1\,1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,1\,1\,0\,0\,0\,1 \end{pmatrix} \tag{7.35}$$

7.5.2. Codificación con códigos Goppa

Sea $\Gamma(L, g(z))$ un código Goppa, donde $g(z)$ es algún polinomio primitivo con $deg(g(z)) = t$ y $|L| = n$. Sea $dim_{\mathbb{F}_q}(\Gamma(L, g(z))) = k$, y G sea una matriz generadora de tamaño $k \times n$ para el código Goppa respectivo; entonces, la codificación de un vector de mensaje $\mathbf{m}$ de longitud k sobre $\mathbb{F}_q$ es $\mathbf{m}G$.

7.5.3. Decodificación con códigos Goppa

Corrección de errores por síndrome de códigos Goppa

Supongamos que recibimos el vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ con r errores, donde $2r + 1 \leq d$ (para así tener el número máximo de corrección de errores, teorema 7.1.1). Sea $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$,

$$\mathbf{y} = (y_1, y_2, \ldots, y_n) = (c_1, c_2, \ldots, c_n) + (e_1, e_2, \ldots, e_n) = \mathbf{c} + \mathbf{e}$$

donde $e_i \neq 0$ en exactamente r posiciones (ya que tenemos r errores). Para corregir los errores necesitamos:

- Localizar las posiciones de error (digamos $B = \{i : 1 \leq i \leq n \text{ y } e_i \neq 0\}$);
- Encontrar los valores correspondientes de error (valores de e_i para $i \in B$).

Para conseguir esto, vamos a definir dos nuevos polinomios:

Definición 7.5.3.

- *Polinomio localizador de errores* $\sigma(z) := \prod_{i \in B}(z - \alpha_i)$ *(tiene grado* r*);*
- *Polinomio evaluador de errores* $w(z) := \sum_{i \in B} e_i \prod_{j \in B; j \neq i}(z - \alpha_j)$ *(tiene grado* $r - 1$*).*

Ya habíamos visto la definición de síndrome, pero aplicada a los códigos Goppa tiene esta forma:

$$S(\mathbf{y}) := \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} + \sum_{i \in B} \frac{e_i}{z - \alpha_i} = \sum_{i \in B} \frac{e_i}{z - \alpha_i} \bmod (g(z)). \tag{7.36}$$

Proposición 7.5.1. *Sea* $\mathbf{e}$ *el vector de error con peso de Hamming* r*. Entonces* $\sigma(z), w(z)$ *y* $S(\mathbf{y})$ *cumplen las siguientes propiedades:*

1. $\deg(\sigma(z)) = r$*;*
2. $\deg(w(z)) \leq r - 1$*;*
3. $\gcd(\sigma(z), w(z)) = 1$*;*
4. $e_k = \frac{w(\alpha_k)}{\sigma'(\alpha_k)}$*, donde* $k \in B$ *y* σ' *representa la derivada de* σ*;*
5. $\sigma(z)S(\mathbf{y}) \equiv w(z)$ *mod* $(g(z))$*.*

Algoritmo para corregir r errores en un código Goppa:

1. Calcular el síndrome
$$S(\mathbf{y}) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}. \tag{7.37}$$
2. Resolver la ecuación
$$\sigma(z)S(\mathbf{y}) \equiv w(z) \bmod (g(z)) \tag{7.38}$$
escribiendo
$$\sigma(z) = \sigma_0 + \sigma_1 z + \cdots + \sigma_{r-1} z^{r-1} + z^r, \tag{7.39}$$
$$w(z) = w_0 + w_1 z + \cdots + w_{r-1} z^{r-1}, \tag{7.40}$$
y resolver t ecuaciones y $2r$ incógnitas. Si el código es binario, tomar $w(z) = \sigma'(z)$.
3. Determinar el conjunto de posiciones de error $B = \{i : 1 \leq i \leq n \text{ y } \sigma(\alpha_i) = 0\}$.
4. Calcular los valores de error $e_i = \frac{w(\alpha_i)}{\sigma'(\alpha_i)}$ para todo $i \in B$. Con esto ya tenemos el vector de error $\mathbf{e} = (e_1, e_2, \ldots, e_n)$ definido por e_i para $i \in B$ y ceros en otros lugares (no hay error).

Algoritmo de Patterson para la corrección de errores

El algoritmo de Patterson decodifica solo códigos Goppa binarios, pero al fin y al cabo estos son muy utilizados debido a su buena capacidad de implementación, por lo que este algoritmo de decodificación es útil en criptografía.

Al igual que el algoritmo anterior, el algoritmo de Patterson calcula el síndrome $S(\mathbf{y})$ de un vector recibido y luego resuelve la ecuación clave
$$\sigma(z)S(\mathbf{y}) \equiv w(z) \bmod (g(z)). \tag{7.41}$$
Pero como el código es binario, $w(z) = \sigma'(z)$. El polinomio localizador de errores se puede dividir en potencias pares e impares de z de tal manera que $\sigma(z) = u^2(z) + zv^2(z)$, ya que el cuerpo tiene característica 2. Los pasos a seguir del algoritmo son los siguientes:

Entrada: El vector recibido $\mathbf{y}$ y el código Goppa $\Gamma(L, g)$.

1. Calcular el síndrome $S(\mathbf{y})$ como un elemento de $\frac{\mathbb{F}_{2^m}[z]}{\langle g(z) \rangle}$.
2. Calcular $T(z) = S(\mathbf{y})^{-1} \bmod g(z)$.
3. Calcular $P(z) = \sqrt{T(z) + z} \bmod g(z)$.
4. Calcular $u(z)$ y $v(z)$ con $u(z) = v(z)S(\mathbf{y}) \bmod g(z)$.
5. Calcular el polinomio localizador $\sigma(z) = u^2(z) + zv^2(z)$.
6. Encontrar las raíces de $\sigma(z)$.
7. Encontrar las posiciones de error, es decir, el vector de error $\mathbf{e}$.

Salida: El vector de error $\mathbf{e}$.

Recuperación del mensaje m después de obtener e (y por tanto c)

Una vez obtenido **e** con alguno de los algoritmos de corrección de errores anteriores, hacemos

$$\mathbf{c} = \mathbf{y} - \mathbf{e}. \tag{7.42}$$

A partir de **c** podemos recuperar directamente el mensaje original **m** resolviendo el sistema:

$$G^T \cdot \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \tag{7.43}$$

7.6. Criptosistema de Niederreiter

El criptosistema de Niederreiter es un esquema de cifrado basado en códigos de corrección de errores, específicamente en códigos Goppa. Fue propuesto por Harald Niederreiter en 1986 en [44] como una variante del criptosistema de McEliece.

La seguridad de este criptosistema se basa en la dificultad de resolver el problema de decodificación por síndrome. En este esquema, la clave pública es una matriz de verificación de paridad un poco modificada, y la clave privada incluye matrices de permutación y transformación, así como la matriz de verificación de paridad original del código Goppa.

El principal problema de McEliece era el tamaño de claves necesario. Niederreiter, por otro lado, es más ventajoso en términos de tamaño de las claves y eficiencia, convirtiéndolo en un candidato adecuado para aplicaciones de criptografía postcuántica.

Generación de claves

Alice genera las dos claves (la pública y la privada) de la siguiente forma:

1. Alice selecciona una matriz (aleatoria) de permutación P de tamaño $n \times n$ y una matriz no singular S de tamaño $(n-k) \times (n-k)$;
2. Escoge también una matriz de verificación de paridad H de tamaño $(n-k) \times n$ para un código Goppa $\Gamma(L, g)$ de dimensión $k = n - mt$, donde $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ y g es un polinomio Goppa de grado t sobre $\mathbb{F}_{q^m}$;
3. Publica su clave pública: La matriz $S \cdot H \cdot P$ (que es $(n-k) \times n$) y mantiene su clave privada: (P, S y H).

Envío del mensaje cifrado

Bob quiere enviar un mensaje a Alice, y para ello:

1. Bob tiene un mensaje **m** de longitud n y peso de Hamming t;
2. Calcula y envía el texto cifrado $\mathbf{c} = S \cdot H \cdot P \cdot \mathbf{m}^T$.

Descifrado del mensaje

Alice recibe el texto cifrado $\mathbf{c}$, y para recuperar el mensaje original:

1. Calcula $\mathbf{z}$ tal que $H\mathbf{z}^T = S^{-1}\mathbf{c}$;
2. Aplica el algoritmo de Patterson para la decodificación de códigos Goppa en el vector $\mathbf{z}$ para obtener la palabra código $\mathbf{z} - \mathbf{m} \cdot P^T$, el vector de error $\mathbf{m} \cdot P^T$ y así $\mathbf{m}$.

7.7. Comparación entre McEliece y Niederreiter

Estos dos esquemas de cifrado están basados en códigos de corrección de errores, concretamente en los códigos Goppa, y son candidatos para la criptografía postcuántica debido a su resistencia a los ataques de computadoras cuánticas. Ambos utilizan como problema difícil la dificultad de la decodificación de códigos lineales, aunque lo hacen de maneras ligeramente diferentes.

McEliece utiliza una matriz generadora de un código Goppa para formar la clave pública, mientras que Niederreiter utiliza una matriz de verificación de paridad en lugar de una matriz generadora. Además, en McEliece el mensaje se cifra como una palabra de código perturbada por un vector de error, mientras que en Niederreiter el mensaje se cifra como un síndrome asociado con un vector de error.

Ambos sistemas son iguales en términos de seguridad, pero Niederreiter tiende a ser más eficiente en términos de tamaño de clave y velocidad de cifrado/descifrado, lo que lo convierte en una opción atractiva en implementaciones prácticas de criptografía postcuántica.

7.8. El esquema de cifrado híbrido de McEliece (HyMES)

En 2008, Bhaskar Biswas and Nicolas Sendrier publicaron el artículo *McEliece Cryptosystem Implementation: Theory and Practice* ([7]) en el que modificaban el esquema de McEliece, conservando sus ventajas (su seguridad) y mejorando sus desventajas:

- Tamaño de la clave.
- Velocidad de cifrado.
- Tasa de información.

Esta propuesta combina las ideas de los esquemas de McEliece y de Niederreiter, por lo que llamamos a este sistema híbrido. Las ideas clave de este método son las siguientes:

1. La matriz generadora se escribe en forma estándar, es decir, escalonada. Esto nos permite reducir el tamaño de la clave al necesitarse menos entradas en la matriz.
2. Codifica la información en el error, aumentando la tasa de información.

El esquema es el siguiente:

7.8.1. Generación de Claves

- Generamos un soporte $L = (\alpha_1, \ldots, \alpha_n)$ de n elementos distintos de $\mathbb{F}_{2^m}$.
- Generamos un polinomio generador mónico e irreducible $g(z) \in \mathbb{F}_{2^m}[z]$ de grado t.
- La clave privada es el par (L, g) (es decir, el código Goppa $\Gamma(L, g)$ y su decodificador).
- La clave pública es una matriz $k \times (n-k)$ binaria R donde $G = (Id_k \mid R)$ es una matriz generadora de $\Gamma(L, g)$ en forma estándar.

7.8.2. Cifrado

Definimos una función inyectiva $\varphi : \{0,1\}^\ell \to W_{n,t}$, siendo $W_{n,t}$ un conjunto de palabras de longitud n y peso de Hamming t. Tanto φ como φ^{-1} deben ser fáciles de computar.

El texto plano está compuesto por dos partes $(\mathbf{x}, \mathbf{x}')$, con $\mathbf{x} \in \{0,1\}^k$ y $\mathbf{x}' \in \{0,1\}^\ell$. El texto cifrado está en $\{0,1\}^n$:

$$\{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^n \tag{7.44}$$

$$(\mathbf{x}, \mathbf{x}') \mapsto (\mathbf{x}, \mathbf{x}R) + \varphi(\mathbf{x}') \tag{7.45}$$

Es decir, lo que estamos haciendo es dividir el mensaje inicial en dos partes, y con la segunda construir el vector de error. Además aprovechamos que la matriz G al estar en forma estándar, una parte será la identidad, y por tanto podemos codificar de esa manera.

7.8.3. Descifrado

El texto cifrado tiene la forma $\mathbf{y} = \mathbf{x}G + \mathbf{e}$, con $\mathbf{e} = \varphi(\mathbf{x}')$ de peso de Hamming $\leq t$. Aplicar el decodificador de $\Gamma(L, g)$ sobre $\mathbf{y}$ proporcionará $\mathbf{x}$ y $\mathbf{x}' = \varphi^{-1}(\mathbf{e})$.

Observación: Tratamos de elegir ℓ aproximadamente $\log_2 \binom{n}{t}$ para una mejor tasa de información y seguridad.

7.9. Ataques contra los criptosistemas basados en códigos

Sección basada en [11].

7.9.1. Ataque *broadcast*

Recordemos que este tipo de ataque consistía en, conocidos varios textos cifrados del mismo mensaje, obtener información sobre este y así comprometer la seguridad del criptosistema. En un artículo publicado en 2011 [43], Niebuhr y Cayrel demostraron que con un pequeño número de textos cifrados, los esquemas Niederreiter y HyMES pueden ser quebrantados mediante este ataque. Sin embargo, también demuestran que esto no puede aplicarse contra el esquema de McEliece debido a que el vector de error añadido a cada cifrado es completamente aleatorio, mientras que en Niederreiter no se añadía ningún vector de

error, y en HyMES, a pesar de que sí que se añade error, ese error se construye a partir del propio mensaje.

Una forma de arreglar la vulnerabilidad de Niederreiter y HyMES a este ataque es transformar el mensaje $\mathbf{m}$ en un mensaje aleatorio y permitir encontrarlo desde el texto cifrado por el destinatario legítimo. En lugar de codificar $\mathbf{m}$ únicamente, codificamos $(\mathbf{m}||\mathbf{e_i})$ como el mensaje siendo $\mathbf{e_i}$ el vector de error.

7.9.2. Ataque partial known plaintext

Este tipo de ataque se puede realizar si se conoce parte del texto plano.

Ataque al esquema de McEliece

Tenemos un mensaje $\mathbf{m} = (\mathbf{m_l}||\mathbf{m_r})$ de k bits ($k = k_l + k_r$), en el que los k_l bits izquierdos no son conocidos para el atacante, pero los k_r bits del final sí que lo son, es decir, el atacante conoce $\mathbf{m_r}$. Entonces, la dificultad de recuperar el texto plano desconocido $\mathbf{m_l}$ en el esquema de McEliece con parámetros (n, k) es equivalente a la de recuperar el texto plano completo en el esquema de McEliece con parámetros (n, k_l), ya que:

$$\mathbf{c} = \mathbf{m}G \oplus \mathbf{e} \tag{7.46}$$
$$\mathbf{c} = \mathbf{m_l}G_l \oplus \mathbf{m_r}G_r \oplus \mathbf{e} \tag{7.47}$$
$$\mathbf{c} \oplus \mathbf{m_r}G_r = \mathbf{m_l}G_l \oplus \mathbf{e} \tag{7.48}$$
$$\mathbf{c}' = \mathbf{m_l}G_l \oplus \mathbf{e} \tag{7.49}$$

donde G_l y G_r son las k_l filas superiores y las k_r filas inferiores restantes en G, respectivamente.

Por lo tanto, esto no rompe el esquema como tal pero hace que los tamaños de claves sean mucho más pequeños, lo que hará que la seguridad sea mucho menor. En el esquema de HyMES se utiliza el mismo método para recuperar todo el mensaje con los mismos parámetros.

Ataque al esquema de Niederreiter

El problema se modela de la misma manera, conociendo una parte $\mathbf{e_r}$ del vector de error $\mathbf{e} = (\mathbf{e_l}||\mathbf{e_r})$, encontrar la otra parte $\mathbf{e_l}$.

Como $n = n_l + n_r$, usamos la misma idea, pero ahora en vez de descomponer la matriz generadora descomponemos la matriz de paridad: $H = (H_l||H_r)$ donde H_l y H_r son las n_l columnas izquierdas y las n_r columnas derechas restantes en H, respectivamente.

$$\mathbf{s} = H\mathbf{e}^T \tag{7.50}$$
$$\mathbf{s} = H_l\mathbf{e_l}^T \oplus H_r\mathbf{e_r}^T \tag{7.51}$$
$$\mathbf{s} \oplus H_r\mathbf{e_r}^T = H_l\mathbf{e_l}^T \tag{7.52}$$
$$\mathbf{s}' = H_l\mathbf{e_l}^T \tag{7.53}$$

De nuevo esto no rompe el esquema como tal pero hace que la seguridad sea mucho menor. Cuanto mayor sea la parte conocida del texto plano, más inseguro.

7.9.3. Ataque de reenvío de mensaje

Si el mismo mensaje se cifra y se envía dos o más veces con vectores de error aleatorios diferentes al mismo destinatario, se puede obtener cierta información sobre los errores y, por lo tanto, sobre el mensaje original.

Veamos cómo afecta este ataque al esquema de McEliece con parámetros $n = 2048, k = 1608, d = 81, t = 40$. Consideremos un mensaje $\mathbf{m}$. Enviamos $\mathbf{c_1} = \mathbf{m}G + \mathbf{e_1}$ y después $\mathbf{c_2} = \mathbf{m}G + \mathbf{e_2}$. Observamos que

$$\mathbf{c_1} - \mathbf{c_2} = (\mathbf{m}G + \mathbf{e_1}) - (\mathbf{m}G + \mathbf{e_2}) = \mathbf{e_1} - \mathbf{e_2}. \tag{7.54}$$

Por tanto, como $\mathbf{c_1} - \mathbf{c_2} = \mathbf{e_1} - \mathbf{e_2}$, un atacante que vea los textos cifrados puede restarlos y así conocer el vector $\mathbf{e_1} - \mathbf{e_2}$. Con este vector no conocemos explícitamente $\mathbf{e_1}$ y $\mathbf{e_2}$, pero sí tenemos ya bastante información sobre ellos. Es decir, el vector $\mathbf{e_1} - \mathbf{e_2}$ va a tener 1 en las posiciones de los bits en las que lo tengan $\mathbf{e_1}$ y $\mathbf{e_2}$. Puede darse el caso de que tengamos un bit 1 justamente en la misma posición en los dos vectores de error $\mathbf{e_1}$ y $\mathbf{e_2}$; si eso ocurriera veríamos un 0 en esa posición y no tendríamos ninguna información, ya que los dos errores se contrarrestan y sería como si no hubiera error. No obstante, la probabilidad de esto es muy pequeña:

$$P(\mathbf{e_1}(\delta) = \mathbf{e_2}(\delta) = 1) \leq \left(\frac{40}{2048}\right)^2 = 0{,}0004. \tag{7.55}$$

siendo $\mathbf{e_i}(\delta)$ el bit en la posición δ-ésima del vector $\mathbf{e_i}$.

Y si además no son solo dos cifrados del mismo mensaje, sino más, realizando diferencias de distintos pares de vectores de errores obtenemos cada vez más información sobre ellos hasta el punto de poder determinarlos. Por tanto, es crucial que cada mensaje se cifre con un vector de error único y aleatorio para evitar este tipo de ataque.

Ni HyMES ni Niederreiter están amenazados por este ataque, ya que se reenvía el mismo error.

7.9.4. Ataque de mensaje relacionado

Como vimos ya, en este tipo de ataque, si tenemos varios textos planos que tienen una relación conocida, un atacante puede obtener información sobre ellos a partir de sus respectivos textos cifrados. Este ataque es una generalización del ataque de reenvío de mensajes. Supongamos que se enviaron dos mensajes $\mathbf{m_1}$ y $\mathbf{m_2}$ al mismo destinatario, la condición de mensaje relacionado se verifica si existe una relación lineal entre $\mathbf{m_1}$ y $\mathbf{m_2}$; por ejemplo, $\delta\mathbf{m} = \mathbf{m_1} \oplus \mathbf{m_2}$.

Por tanto, $\mathbf{c_1} = \mathbf{m_1}G \oplus \mathbf{e_1}$ y $\mathbf{c_2} = \mathbf{m_2}G \oplus \mathbf{e_2}$. Y entonces:

$$\mathbf{c_1} \oplus \mathbf{c_2} = \mathbf{m_1}G \oplus \mathbf{m_2}G \oplus \mathbf{e_1} \oplus \mathbf{e_2} = (\delta\mathbf{m})G \oplus \mathbf{e_1} \oplus \mathbf{e_2}. \tag{7.56}$$

Obtenemos $\mathbf{c_1} \oplus \mathbf{c_2} \oplus (\delta\mathbf{m})G = \mathbf{e_1} \oplus \mathbf{e_2}$ y aplicamos el proceso de ataque de reenvío de mensajes usando $\mathbf{c_1} \oplus \mathbf{c_2} \oplus (\mathbf{m_1} \oplus \mathbf{m_2})G$ en lugar de $\mathbf{c_1} \oplus \mathbf{c_2}$.

Este ataque también funciona en el esquema HyMES. En este caso, dividimos los dos mensajes: $\mathbf{m_1} = \mathbf{m_1'}\|\mathbf{m_1''}$ y $\mathbf{m_2} = \mathbf{m_2'}\|\mathbf{m_2''}$. Si $\delta\mathbf{m} = \delta\mathbf{m'}\|\delta\mathbf{m''} = (\mathbf{m_1'} \oplus \mathbf{m_2'})\|(\mathbf{m_1''} \oplus \mathbf{m_2''})$ y $\mathbf{m_1''} \neq \mathbf{m_2''}$, entonces se

puede calcular:

$$\mathbf{c_1} \oplus \mathbf{c_2} = \mathbf{m'_1}G \oplus \mathbf{m'_2}G \oplus \varphi(\mathbf{m''_1}) \oplus \varphi(\mathbf{m''_2}) = (\delta\mathbf{m'})G \oplus \varphi(\mathbf{m''_1}) \oplus \varphi(\mathbf{m''_2}). \tag{7.57}$$

Obtenemos $\mathbf{c_1} \oplus \mathbf{c_2} \oplus (\delta m)G = \varphi(\mathbf{m''_1}) \oplus \varphi(\mathbf{m''_2})$ y aplicamos el proceso de ataque de reenvío de mensajes usando $\mathbf{c_1} \oplus \mathbf{c_2} \oplus (\mathbf{m_1} \oplus \mathbf{m_2})G$ en lugar de $\mathbf{c_1} \oplus \mathbf{c_2}$.

Para el esquema de Niederreiter, este ataque no es posible porque el cifrado es determinista.

7.9.5. Ataques de texto plano elegido (CPA)

En este tipo de ataque, el atacante puede elegir textos planos arbitrarios para ser cifrados y obtener los correspondientes textos cifrados. Con esto puede obtener alguna información adicional que reduzca la seguridad del esquema de cifrado, incluso podría revelar la clave privada del esquema.

Los algoritmos de cifrado de clave pública deterministas como los esquemas de Niederreiter y HyMES son vulnerables a ataques simples de tipo "diccionario", donde el atacante construye una tabla de mensajes probables y sus correspondientes textos cifrados. Sin embargo, algunos algoritmos probabilísticos como el esquema de McEliece también son vulnerables.

La complejidad de decodificar el texto cifrado disminuye exponencialmente con cada bit conocido. Por ejemplo, atacar un texto cifrado con McEliece usando parámetros (n, k) y conociendo k_l bits es equivalente a atacar un texto cifrado de McEliece cifrado usando parámetros $(n, k - k_l)$.

7.9.6. Ataques de texto cifrado elegido (CCA)

En un ataque de texto cifrado elegido, un atacante tiene acceso a un oráculo de descifrado que permite descifrar cualquier texto cifrado elegido (excepto el que el atacante intenta revelar). En general, el atacante debe elegir todos los textos cifrados por adelantado antes de consultar el oráculo, pero si puede hacer el algoritmo adaptativo (es decir, pudiendo elegir los diferentes textos cifrados en función de la información que va obteniendo) el algoritmo funciona así:

El ataque de texto cifrado elegido adaptativo: Dado que el atacante tiene acceso al oráculo, para el criptosistema de McEliece o HyMES, el texto cifrado $\mathbf{c} = \mathbf{m}G \oplus \mathbf{e}$, puede cambiar dos bits de $\mathbf{c}$ para obtener $\mathbf{c'} = \mathbf{m}G \oplus \mathbf{e'}$ y enviarlo al oráculo de descifrado. Este descifrado tendrá éxito si estos bits cambiados son ambos diferentes: un "1" en $\mathbf{e}$ se cambia a "0" y un "0" se cambia a "1", es decir, el peso de Hamming $\mathrm{wt}(\mathbf{e}) = \mathrm{wt}(\mathbf{e'}) = t$ y $\mathrm{wt}(\mathbf{e} \oplus \mathbf{e'}) = 2$. Cuando esto ocurre, la salida del oráculo será el texto plano $\mathbf{m}$. Esto ocurre con la probabilidad:

$$P_r = P(\mathrm{wt}(\mathbf{e}) = \mathrm{wt}(\mathbf{e'}) = t \text{ y } \mathrm{wt}(\mathbf{e} \oplus \mathbf{e'}) = 2) = \frac{\binom{t}{1}\binom{n-t}{1}}{\binom{n}{2}} = \frac{2t(n-t)}{n(n-1)}. \tag{7.58}$$

Para $n = 1024$ y $t = 50$, la probabilidad es $P_r = 0{,}09$ (esto significa que será necesario hacerlo alrededor de 11 veces), y para $n = 2048$ y $t = 81$, la probabilidad es $P_r = 0{,}08$ (esto significa intentarlo alrededor de 12 veces).

Para el esquema HyMES, la segunda parte $\mathbf{m}_2$ será $\varphi(\mathbf{m}_2) = \mathbf{e} = \mathbf{m}_1 G \oplus \mathbf{c}$, es decir, $\mathbf{m}_2 = \varphi^{-1}(\mathbf{e}) = \varphi^{-1}(\mathbf{m}_1 G \oplus \mathbf{c})$.

Para el esquema de Niederreiter, con la misma probabilidad, dado que $\mathbf{c} = H\mathbf{e}^T$, el atacante puede enviar al oráculo $\mathbf{c}' = \mathbf{c} \oplus H[i] \oplus H[j]$, donde $H[i]$ y $H[j]$ son las columnas correspondientes a los dos bits cambiados del mensaje $\mathbf{e}$, lo que devuelve $\mathbf{e}'$ y $\mathbf{e}$ es revelado cambiando estos bits nuevamente.

7.9.7. Maleabilidad

Como se muestra en [27], publicado en 2001, McEliece es maleable: Sea $\delta\mathbf{m}$ una cadena de bits en la que todos los bits son iguales a '1' y $G[i]$ la i-ésima fila de la matriz pública G. A partir de $\delta\mathbf{m}$, el atacante obtiene el texto cifrado:

$$\mathbf{c'} = \mathbf{c} \bigoplus_{i=1}^{k} G[i] = (\mathbf{m} \oplus \delta\mathbf{m})G \oplus \mathbf{e} = \mathbf{m'}G \oplus \mathbf{e} \quad \text{(en el esquema HyMES, } \mathbf{e} = \varphi(\mathbf{m}_2)\text{)}. \tag{7.59}$$

Por lo tanto, el oráculo devuelve $\mathbf{m'}$ y el atacante puede montar un ataque de mensaje relacionado entre $\mathbf{c}$ y $\mathbf{c'}$; $\mathbf{m'} = \mathbf{m} \oplus \delta\mathbf{m}$.

Para el criptosistema de Niederreiter, $\mathbf{c} = H\varphi(\mathbf{m})^T$, y, $\mathrm{wt}(\varphi(\mathbf{m})) = t$. El atacante puede tomar aleatoriamente la i-ésima columna de H (denotada $H[i]$) y enviarla al oráculo $\mathbf{c'} = \mathbf{c} \oplus H[i]$. El descifrado tendrá éxito si tenemos un '1' en el i-ésimo bit de $\varphi(\mathbf{m})$. Sea $\mathbf{e}_1$ un vector de longitud n con solo el i-ésimo bit no nulo. El oráculo devuelve el vector $\mathbf{e}$ de peso $t-1$ tal que

$$\mathbf{c'} = H\mathbf{e}^T = \mathbf{c} \oplus H[i] = H\varphi(\mathbf{m})^T \oplus H\mathbf{e}_1^T = H(\varphi(\mathbf{m}) \oplus \mathbf{e}_1)^T. \tag{7.60}$$

Entonces:

$$\varphi(\mathbf{m}) = \mathbf{e} \oplus \mathbf{e}_1, \quad \text{es decir,} \quad \mathbf{m} = \varphi^{-1}(\mathbf{e} \oplus \mathbf{e}_1). \tag{7.61}$$

Esto tiene éxito con la probabilidad:

$$P_r = P(\mathbf{e}[i] = 1) = \frac{\binom{t}{1}}{\binom{n}{1}} = \frac{t}{n}. \tag{7.62}$$

Para $n = 1024$ y $t = 50$, $P_r = 0,05$, y para $n = 2048$ y $t = 81$, $P_r = 0,04$.

7.9.8. Comparativa

En las siguientes tablas podemos qué criptosistemas son vulnerables a según qué tipo de ataques, en primer lugar para ataques de texto plano y en segundo lugar de texto cifrado. Las casillas marcadas con $\times$ significan que dicho criptosistema es vulnerable a ese ataque.

Texto plano						
Esquema	**Broadcast**	**KPA**	**Reenvío mensaje**	**Mensaje relacionado**	**CPA adaptativo**	**CPA**
McEliece		×	×	×	×	
Niederreiter	×	×			×	×
HyMES	×	×		×	×	×

Cuadro 7.1: Vulnerabilidad de los tres esquemas criptográficos a diferentes tipos de ataques de texto plano.

Texto cifrado			
Esquema	**CCA**	**CCA adaptativo**	**Maleabilidad**
McEliece	×	×	×
Niederreiter	×	×	×
HyMES	×	×	×

Cuadro 7.2: Vulnerabilidad de los tres esquemas criptográficos a diferentes tipos de ataques de texto cifrado.

Capítulo 8

Criptografía basada en retículos

Bibliografía fundamental: [35], [54] y [47]. La criptografía basada en retículos es un campo emergente en la criptografía que ha ganado una atención considerable debido a su potencial resistencia a los ataques de ordenadores cuánticos. Se basa en problemas matemáticos relacionados con los retículos, que son conjuntos discretos de puntos en el espacio euclidiano. Estos problemas son particularmente conocidos por su complejidad computacional, lo que los hace candidatos ideales para construir sistemas criptográficos seguros.

Notación: Todos los vectores escritos de la forma $(x_1, ..., x_n)$ denotan vectores columna. Usaremos los brackets cuadrados para denotar matrices y vectores fila.

Definición 8.0.1. *Dado un conjunto de n vectores linealmente independientes $B = \{b_1, ..., b_n\}$ en $\mathbb{R}^m$, un **retículo** asociado a B es el conjunto de todas las combinaciones lineales enteras de los vectores de B, es decir*

$$\mathcal{L}(b_1, ..., b_n) = \left\{ \sum_{i=1}^{n} x_i b_i : x_i \in \mathbb{Z}, 1 \leq i \leq n \right\}. \tag{8.1}$$

El conjunto $b_1, ..., b_n$ es una base B del retículo. Una base se puede representar mediante la matriz $B = [b_1, ..., b_n] \in \mathbb{R}^{m \times n}$ donde las columnas son los vectores de la base.

Usando la notación matricial, el retículo generado por una matriz $B \in \mathbb{R}^{m \times n}$ se puede definir como

$$\mathcal{L}(B) = \{Bx : x \in \mathbb{Z}^n\}. \tag{8.2}$$

Como sabemos, no hay una única base; cualquier retículo admite múltiples bases, y esto es de esencial importancia para la criptografía, ya que como veremos, en muchos casos es útil encontrar una base que sea de una determinada forma, como por ejemplo, casi ortogonal.

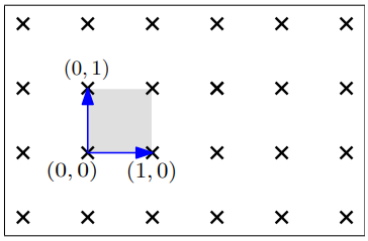

Figura 8.1: Una base de $\mathbb{Z}^2$.

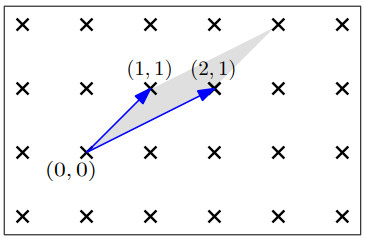

Figura 8.2: Otra base de $\mathbb{Z}^2$.

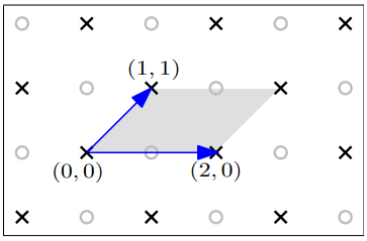

Figura 8.3: No es base de $\mathbb{Z}^2$.

Figura 8.4: Ejemplo de un retículo y de algunas bases.

El **rango del retículo** es n y su **dimensión** es m. Si $n = m$, el retículo es **de rango máximo**. A partir de ahora trabajaremos únicamente con este tipo de retículos.

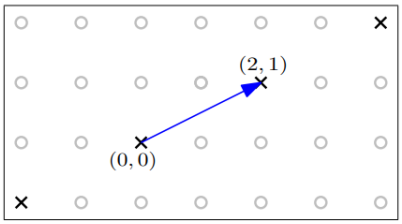

Figura 8.5: Retículo de rango no máximo.

Los retículos q-arios tienen una gran importancia en la criptografía.

Definición 8.0.2. *Retículos q-arios: Son retículos $\mathcal{L}$ que satisfacen $q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n$ para algún entero q (posiblemente primo).*

Dada una matriz $A \in \mathbb{Z}_q^{n\times m}$ para enteros q, m y n, podemos definir dos retículos q-arios m-dimensionales:

$$\Lambda_q(A) = \{y \in \mathbb{Z}^m : \mathbf{y} = A^T\mathbf{s} \bmod q \text{ para algún } \mathbf{s} \in \mathbb{Z}^n\}. \tag{8.3}$$

$$\Lambda_q^{\perp}(A) = \{\mathbf{y} \in \mathbb{Z}^m : A\mathbf{y} = \mathbf{0} \bmod q\}. \tag{8.4}$$

El primer retículo q-ario es generado por las filas de A; el segundo contiene todos los vectores que son ortogonales modulo q a las filas de A.

Ejemplo de un retículo no q-ario en $\mathbb{R}^2$

Consideremos los vectores en $\mathbb{R}^2$:

$$\mathbf{b}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad \mathbf{b}_2 = \begin{pmatrix} 1 \\ \sqrt{3} \end{pmatrix}. \tag{8.5}$$

Estos vectores generan un retículo Λ en $\mathbb{R}^2$ tal que:

$$\mathcal{L} = \{n_1\mathbf{b}_1 + n_2\mathbf{b}_2 \mid n_1, n_2 \in \mathbb{Z}\}. \tag{8.6}$$

Podemos escribir explícitamente:

$$\mathcal{L} = \left\{ \begin{pmatrix} 2n_1 + n_2 \\ n_2\sqrt{3} \end{pmatrix} \mid n_1, n_2 \in \mathbb{Z} \right\}. \tag{8.7}$$

8.1. Problemas que se usan en criptografía basada en retículos

La criptografía basada en retículos se fundamenta en varios problemas matemáticos que se creen intrínsecamente difíciles de resolver, tanto para computadoras clásicas como cuánticas. En esta sección analizaremos cuáles son esos problemas, y en las secciones posteriores veremos su utilidad a la hora de elaborar esquemas criptográficos.

$\lambda_1(\mathcal{L}(B))$ se refiere a la primera longitud mínima del retículo $\mathcal{L}(B)$. Es decir, $\lambda_1(\mathcal{L}(B))$ representa la longitud del vector más corto que se puede encontrar en el retículo.

$\lambda_n(\mathcal{L}(B))$ se refiere a la n-ésima longitud mínima del retículo $\mathcal{L}(B)$. Es decir, $\lambda_n(\mathcal{L}(B))$ representa la longitud del n-ésimo vector más corto que se puede encontrar en el retículo.

Shortest Vector Problem (SVP): Dada una base B del retículo, encontrar el vector no nulo más pequeño en $\mathcal{L}(\mathcal{B})$, es decir, encontrar un vector no nulo $\mathbf{v} \in \mathcal{L}(B)$ tal que $\|\mathbf{v}\|$[1]$= \lambda_1(\mathcal{L}(B))$.

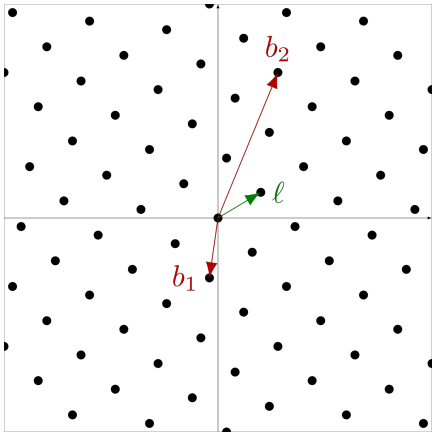

Figura 8.6: Retículo de 2 dimensiones con base $\{\mathbf{b}_1, \mathbf{b}_2\}$ y vector más corto ℓ. [55]

SVP es difícil de resolver en retículos de dimensiones elevadas. Una variante de este problema que se utiliza bastante es encontrar un conjunto de vectores del retículo pequeños y linealmente independientes:

Shortest Independent Vectors Problem (SIVP): Dada una base del retículo B del retículo n-dimensional $\mathcal{L}(\mathcal{B})$, encontrar n vectores linealmente independientes v_1,..., $v_n \in \mathcal{L}(\mathcal{B})$ tales que $max_{i\in[1,n]}\|v_i\| = \lambda_n(\mathcal{L}(\mathcal{B}))$.

Closest Vector Problem (CVP): Dada una base del retículo $\mathcal{L}(\mathcal{B})$ y un vector $\mathbf{t}$ que no está en el retículo, encontrar un vector $\mathbf{v}$ del retículo que sea el más cercano a $\mathbf{t}$, es decir, el que para todo $\mathbf{w} \in \mathcal{L}(\mathcal{B})$

[1]Obviamente vale cualquier norma, pero se suele utilizar la euclídea.

satisface $\|\mathbf{v} - \mathbf{t}\| \leq \|\mathbf{w} - \mathbf{t}\|$.

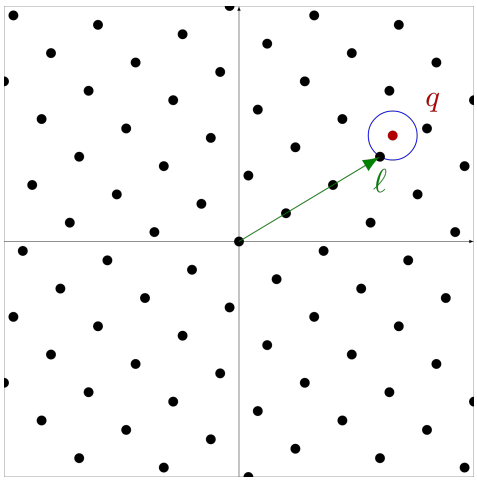

Figura 8.7: Retículo de 2 dimensiones y vector más cercano a q. [55]

Un caso especial de CVP es:

α-Bounded Distance Decoding Problem (BDDα): Dada una base del retículo $\mathcal{L}(\mathcal{B})$ y un vector $\mathbf{t} \in \mathbb{R}^n$ que satisface $dist(\mathbf{t}, B) \leq \alpha\lambda_1(\mathcal{L})$, encontrar un vector $\mathbf{v} \in \mathcal{L}$ que sea el más cercano a $\mathbf{t}$, es decir, el que para todo $\mathbf{w} \in \mathcal{L}(\mathcal{B})$ satisface $\|\mathbf{v} - \mathbf{t}\| \leq \|\mathbf{w} - \mathbf{t}\|$.

Para estos problemas, existe una versión γ, siendo γ un factor de aproximación:

SVP$_\gamma$: Debemos encontrar un vector no nulo del retículo de longitud como máximo $\gamma\lambda_1(\mathcal{L})$ para un $\gamma \geq 1$ dado. Es decir, es una versión del SVP donde se busca un vector que no necesariamente sea el más corto, pero cuya longitud sea menor que γ veces la longitud del vector más corto.

SIVP$_\gamma$: Dada una base del retículo B del retículo n-dimensional $\mathcal{L}(\mathcal{B})$, encontrar n vectores linealmente independientes v_1,..., $v_n \in \mathcal{L}(\mathcal{B})$ tales que $max_{i\in[1,n]}\|v_i\| \leq \gamma(n)\lambda_n(\mathcal{L}(\mathcal{B}))$.

GapSVP: El problema GapSVP es un problema de decisión relacionado con SVP$_\gamma$. En GapSVP, se distingue entre dos casos:

- YES-instance: La longitud del vector más corto en el retículo es menor o igual que una cierta longitud d.
- NO-instance: Todos los vectores no nulos en el retículo tienen una longitud mayor o igual a γd.

En otras palabras, el problema GapSVP pregunta si el vector más corto del retículo es más corto que una cierta longitud d, o si todos los vectores no nulos son más grandes que un factor γ veces esa longitud. El objetivo es decidir cuál de los dos casos es verdadero para un retículo dado.

Con una correcta elección de parámetros, estos problemas parecer ser intratables en tiempo polinómico, incluso con un ordenador cuántico, cuando n (la dimensión del retículo) es grande. No se conoce un

algoritmo que los resuelva en tiempo polinómico. Por esto tienen una gran utilidad en la construcción de esquemas criptográficos, especialmente a través de dos problemas fundamentales: Learning With Errors (LWE) y Short Integer Solution (SIS). Estos proporcionan la base para construir primitivas criptográficas seguras y eficientes, resistentes a ataques tanto clásicos como cuánticos. En las secciones posteriores veremos cómo se pueden implementar[2].

El factor de aproximación γ puede ser una función de n. Cabe resaltar la siguiente idea: la versión γ de estos problemas obviamente será más fácil de resolver cuanto más grande sea γ. La figura 8.8 muestra, para los problemas SVP_γ y CVP_γ, una gráfica del tiempo que se tardan en resolver estos problemas con el algoritmo más eficiente conocido, según va creciendo γ.

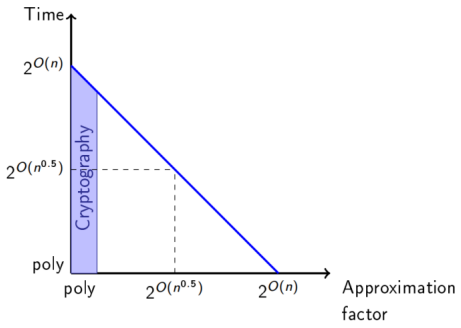

Figura 8.8: Tiempo de resolución del problema en función de γ. [50]

Por lo tanto, para su uso en criptografía, es habitual que el parámetro γ sea una función polinómica de las dimensiones del retículo (parte azul en la figura).

8.1.1. Añadir estructura algebraica

La siguiente sección se basa en las siguientes notas de un curso de verano en criptografía postcuántica [50] y [51] y el taller [49].

A todos los problemas de retículos detallados en la sección anterior se les puede añadir una estructura algebraica, lo que nos permite mejorar la eficiencia de los esquemas criptográficos. Es decir, podemos trabajar sobre un retículo con una estructura algebraica determinada, como por ejemplo los anillos de polinomios, y definir los problemas sobre este retículo.

Por ejemplo: sea $f \in \mathbb{Z}[x]$ un polinomio mónico de grado n, y consideremos el anillo cociente $\mathbb{Z}[x]/\langle f \rangle$. Utilizando el conjunto $\{g \bmod f : g \in \mathbb{Z}[x]\}$, y la identificación de polinomios con vectores, el anillo cociente $\mathbb{Z}[x]/\langle f \rangle$ es isomorfo (como grupo aditivo) al retículo de enteros $\mathbb{Z}^n$, y cualquier ideal $I \subseteq \mathbb{Z}[x]/\langle f \rangle$ define un subretículo entero correspondiente $\mathcal{L}(I) \subseteq \mathbb{Z}^n$.

Definición 8.1.1. *Un **retículo ideal** es un retículo entero $\mathcal{L}(B) \subseteq \mathbb{Z}^n$ tal que $B = \{g \mod f : g \in I\}$ para algún polinomio mónico f de grado n e ideal $I \subseteq \mathbb{Z}[x]/\langle f \rangle$.*

[2]Nótese que para una correcta implementación de LWE y SIS, se deben seleccionar cuidadosamente los parámetros del problema, como la dimensión n, el módulo q, y la distribución del ruido o el tamaño del vector para SIS. Los parámetros son los que al final van a determinan tanto la seguridad como la eficiencia de los esquemas criptográficos.

Se denota ideal-X al problema X restringido a retículos ideales, por ejemplo, ideal-SVP.

También podemos usar otra estructura: los módulos. Se denota module-X_k al problema X restringido a retículos sobre módulos de rango k. Por ejemplo, module-SVP$_k$.

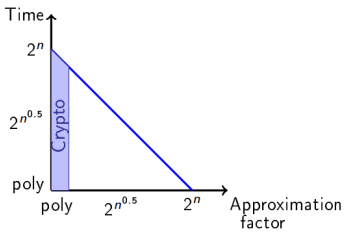

Figura 8.9: SVP y Module-SVP$_k$ con $k \geq 2$.

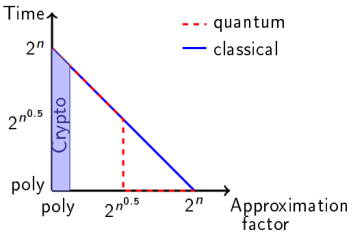

Figura 8.10: Ideal-SVP sobre cuerpos ciclotómicos. [16]

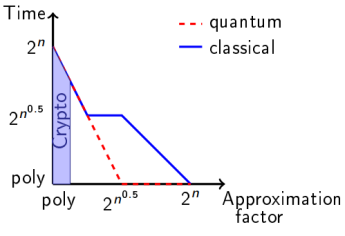

Figura 8.11: Ideal-SVP con preprocesado. [52] [6]

Figura 8.12: Complejidad del SVP sobre retículos con estructura algebraica. [49]

Como se puede ver en 8.9, el problema SVP para módulos de rango $k \geq 2$ es tan difícil de resolver como el problema SVP para retículos cualesquiera.

En cuanto al problema SVP sobre ideales, como se ve en las figuras 8.10 y 8.11, se puede utilizar la estructura algebraica para obtener algoritmos de resolución notablemente más eficientes, especialmente cuando se utilizan algoritmos cuánticos. No obstante, cuando el factor de aproximación γ es una función polinómica de n, la seguridad es la misma o casi la misma para el SVP sin estructura. Por tanto, en este caso, *añadir una estructura algebraica parece no disminuir la dificultad de estos problemas y por tanto no compromete la seguridad de los criptosistemas basados en ellos.* Veremos posteriormente cómo podemos desarrollar esta idea. Hay que tener en mente siempre que el hecho de que no se haya encontrado hasta ahora un ataque que aproveche esto no significa que no se vaya a encontrar en el futuro, pero puede ser un indicativo de que tal vez no sea tan fácil y es una idea que vale la pena explotar.

8.2. LWE

Sección basada en [47] y [55]. El problema Learning With Errors (LWE) es un problema fundamental en la criptografía basada en retículos. Consiste en la recuperación de una pequeña cantidad de información sobre un vector en un retículo, cuando se proporcionan múltiples muestras de ese vector, perturbadas por ruido aleatorio. Formalmente, dado un conjunto de vectores $(\mathbf{a}_i, c_i)$ donde $\mathbf{a}_i$ son vectores en un retículo $\mathcal{L}$ y c_i son productos escalares de $\mathbf{a}_i$ con un vector secreto $\mathbf{s}$, junto con ruido e_i, el objetivo es recuperar $\mathbf{s}$ dado $(\mathbf{a}_i, c_i + e_i)$.[3] Veámoslo de forma más rigurosa:

Sea $Z_q = \mathbb{Z}/q\mathbb{Z}$ el anillo de enteros módulo q, es decir, el conjunto de enteros no negativos menores que q con las operaciones $+$ y $\cdot$ módulo q. Podemos formar un sistema de ecuaciones lineales

$$A \cdot \mathbf{s} = \mathbf{b} \tag{8.8}$$

[3]La idea de perturbar un sistema es una técnica muy usada en criptografía. También se utilizaba en criptografía basada en códigos.

donde $A \in \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \in \mathbb{Z}_q^m$ y $\mathbf{b} \in \mathbb{Z}_q^n$. Por ejemplo, consideremos el siguiente sistema:

$$A = \begin{pmatrix} 10 & 4 & \dots \\ 3 & 3 & \dots \\ 1 & 5 & \dots \\ 1 & 1 & \dots \\ 1 & 2 & \dots \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 10 \\ 3 \\ 8 \\ \vdots \end{pmatrix}. \tag{8.9}$$

Las ecuaciones asociadas quedan así:

$$10s_1 + 3s_2 + 5s_3 + 1s_4 = 10 \tag{8.10}$$
$$4s_1 + 1s_2 + 1s_3 + 2s_4 = 3 \tag{8.11}$$
$$\vdots \tag{8.12}$$
$$3s_1 + 1s_2 + 1s_3 + 5s_4 = 8 \tag{8.13}$$

Resolver este sistema de ecuaciones puede realizarse eficientemente utilizando el algoritmo de Gauss. Sin embargo, agregar valores de error pequeños $e \in \mathbb{Z}_q^n$ al sistema de ecuaciones hace que resolver el sistema y recuperar el vector solución s sea muy difícil. Este hecho está fundamentado en la relación con los problemas difíciles de retículos, ya que este sistema se puede ver como un CVP:

Si tenemos

$$A \cdot \mathbf{s} + \mathbf{e} = \mathbf{b} \tag{8.14}$$

donde $A \in \mathbb{Z}_q^{n \times m}$, $\mathbf{b} \in \mathbb{Z}_q^n$ y los vectores pequeños $\mathbf{s} \in \mathbb{Z}_q^m$, $\mathbf{e} \in \mathbb{Z}_q^n$, el vector más cercano a $\mathbf{b}$ es *casi siempre*[4] el vector del retículo $A \cdot \mathbf{s}$ con distancia $\mathbf{e}$.

Para dar una intuición de la relación entre LWE y el problema del SVP, consideremos el retículo

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} \mid (A \,|\, I_n \,|\, -\mathbf{b}) \cdot \mathbf{x} = \mathbf{0} \bmod q\} \tag{8.15}$$

donde el operador $|$ denota la concatenación e I_n denota la matriz de identidad $n \times n$. Se puede observar que el vector $(\mathbf{s}, \mathbf{e}, \mathbf{1})$ es un elemento de $\mathcal{L}$ al verificar que

$$(A|I_n| - \mathbf{b}) \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \\ \mathbf{1} \end{pmatrix} = A \cdot \mathbf{s} + \mathbf{e} - \mathbf{b} = \mathbf{b} - \mathbf{b} = \mathbf{0} \bmod q \tag{8.16}$$

se cumple. Se puede demostrar que el vector $(\mathbf{s}, \mathbf{e}, \mathbf{1})$ es en realidad un vector de mínima longitud en $\mathcal{L}$ y, por lo tanto, es una solución SVP para $\mathcal{L}$. Esto significa que obtener el vector $(\mathbf{s}, \mathbf{e}, \mathbf{1})$ directamente nos da el secreto $\mathbf{s}$ así como el vector de error $\mathbf{e}$ y, por lo tanto, resuelve el sistema LWE.

LWE Decisional (dLWE)

El problema LWE también puede reformularse como un problema de decisión, generalmente abreviado como dLWE. Dada una muestra LWE $(\mathbf{A}, \mathbf{b})$ como se define anteriormente (donde $\mathbf{s}$ y $\mathbf{e}$ se mantienen en secreto), la tarea es adivinar si los valores de $\mathbf{b}$ han sido calculados como $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ con valores de error pequeños $\mathbf{e}$, o si han sido elegidos arbitrariamente. Ambas variantes son igualmente difíciles.

[4] Nota: es casi siempre ya que se trata de un algoritmo probabilístico.

Esquemas de cifrado basados en LWE

Supongamos que queremos transmitir un mensaje que consiste en un solo bit para ver el funcionamiento del LWE (este ejemplo puede ser trivialmente extendido para transmitir una cadena de bits de cualquier longitud deseada).

Consideremos un LWE $A \cdot \mathbf{s} + \mathbf{e} = \mathbf{b}$, donde $A \in \mathbb{Z}_q^{n \times m}$ es elegida uniformemente al azar, $\mathbf{s} \in \mathbb{Z}_q^m$ y $\mathbf{e} \in \mathbb{Z}_q^n$ son elegidos de una distribución de errores, es decir, sus valores son bastante pequeños. Supongamos que los valores A y $\mathbf{b}$ son públicos mientras que los valores correspondientes $\mathbf{s}$ y $\mathbf{e}$ se mantienen en secreto. El problema LWE entonces establece que es difícil calcular $\mathbf{s}$ o $\mathbf{e}$.

Para construir el esquema de cifrado real, muestrearemos aleatoriamente los valores adicionales $\mathbf{r} \in \mathbb{Z}_q^n$ así como los errores $\mathbf{e}_1 \in \mathbb{Z}_q^m$ y $\mathbf{e}_2 \in \mathbb{Z}_q^n$. Con eso, construimos el sistema de ecuaciones

$$u = A^T \cdot r + e_1 \in \mathbb{Z}_q^m \tag{8.17}$$

$$v = b^T \cdot r + e_2 \in \mathbb{Z}_q \tag{8.18}$$

que puede ser representado de forma matricial como

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A^T \\ \mathbf{b}^T \end{pmatrix} \cdot \mathbf{r} + \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{pmatrix}. \tag{8.19}$$

Es fácil ver que esto es, de nuevo, el problema LWE. Conociendo $(A, \mathbf{b}, \mathbf{u}, \mathbf{v})$ es muy difícil calcular cualquiera de los otros valores. Además, el problema de LWE decisional establece que incluso es difícil diferenciar entre los valores $\mathbf{u}, \mathbf{v}$ calculados de la manera descrita anteriormente y $\mathbf{u}, \mathbf{v}'$ con algún valor arbitrario $\mathbf{v}'$. Esto es fundamental en nuestro sistema de cifrado.

Por ahora, asumamos que simplemente enviaríamos $(\mathbf{u}, \mathbf{v})$ de vuelta al destinatario, quien (conociendo $\mathbf{s}$) podría entonces calcular el valor $\mathbf{s}^T \cdot \mathbf{u} = \mathbf{s}^T \cdot (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1)$. Teniendo en cuenta que los valores de error son relativamente pequeños, observamos que $\mathbf{s}^T \cdot \mathbf{u} \approx \mathbf{s}^T \cdot \mathbf{A}^T \cdot \mathbf{r}$, y también que $\mathbf{v} = \mathbf{b}^T \cdot \mathbf{r} + \mathbf{e}_2 \approx \mathbf{b}^T \cdot \mathbf{r} \approx (\mathbf{A} \cdot \mathbf{s})^T \cdot \mathbf{r} = \mathbf{s}^T \cdot \mathbf{A}^T \cdot \mathbf{r}$. Así, descartando los valores de error, obtenemos que $\mathbf{s}^T \cdot \mathbf{u} \approx \mathbf{v}$.

Esto significa que hemos encontrado una manera de transmitir indirectamente aproximadamente el mismo valor de dos formas separadas y lo hemos hecho de manera inadvertida por una tercera persona: sin conocimiento de $\mathbf{s}$ no se puede deducir qué tan cerca están exactamente estos valores entre sí.

El truco es ocultar el mensaje en uno de estos valores. Cuando el mensaje es 0, simplemente transmitiremos $\mathbf{v}' = \mathbf{v}$. Sin embargo, en caso de que sea 1, transmitiremos $\mathbf{v}' = \mathbf{v} + \frac{q}{2}$ (estamos operando en $\mathbb{Z}_q$, por lo que este es el valor "opuesto" a 0). El receptor puede entonces calcular $\mu = \mathbf{v}' - \mathbf{s}^T \cdot \mathbf{u}$. Si μ está cerca de cero (módulo q), el mensaje era 0, si está más cerca de $\frac{q}{2}$, el mensaje era 1.

Veamos el proceso de manera más formal. Sea $\text{round}_n(\cdot)$ el redondeo al múltiplo más cercano de n. Para un mensaje de un solo bit codificado como $\mu \in \{0, \lfloor \frac{q}{2} \rfloor\}$, el texto cifrado es $(\mathbf{u}, \mathbf{v}')$ con

$$u = A^T \cdot r + e_1 \tag{8.20}$$
$$v' = b^T \cdot r + e_2 + \mu \tag{8.21}$$

a partir del cual el receptor puede calcular

$$\begin{aligned}
&\text{round}_{\lfloor \frac{q}{2} \rfloor}(\mathbf{v}' - \mathbf{s}^T \cdot \mathbf{u}) \\
&= \text{round}_{\lfloor \frac{q}{2} \rfloor} \left(\mathbf{b}^T \cdot \mathbf{r} + \mathbf{e}_2 + \mu - \mathbf{s}^T (\mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_1) \right) \\
&= \text{round}_{\lfloor \frac{q}{2} \rfloor} \left((\mathbf{A} \cdot \mathbf{s} + \mathbf{e})^T \cdot \mathbf{r} + \mathbf{e}_2 + \mu - \mathbf{s}^T \mathbf{A}^T \cdot \mathbf{r} - \mathbf{s}^T \mathbf{e}_1 \right) \\
&= \text{round}_{\lfloor \frac{q}{2} \rfloor} \left((\mathbf{As})^T \cdot \mathbf{r} + \mathbf{e}^T \cdot \mathbf{r} + \mathbf{e}_2 + \mu - (\mathbf{As})^T \cdot \mathbf{r} - \mathbf{s}^T \mathbf{e}_1 \right) \\
&= \text{round}_{\lfloor \frac{q}{2} \rfloor} (\mu + \mathbf{e}^T \cdot \mathbf{r} + \mathbf{e}_2 - \mathbf{s}^T \mathbf{e}_1) \\
&= \mu.
\end{aligned}$$

Para que la última igualdad se cumpla (y, por lo tanto, el descifrado tenga éxito), necesitamos que el efecto general del término de error $(\mathbf{e}^T \cdot \mathbf{r} + \mathbf{e}_2 - \mathbf{s}^T \mathbf{e}_1)$ se mantenga por debajo de $\frac{q}{4}$. En la práctica, los esquemas relevantes utilizan una distribución de errores y un módulo q donde esto no siempre es el caso para tener tamaños de criptograma razonables. La probabilidad de fallo es extremadamente pequeña, por lo que es generalmente despreciable en la práctica. Sin embargo, se debe tener cuidado de que los atacantes no puedan aprender nada sobre la clave secreta al crear intencionalmente textos cifrados que causen fallos en el descifrado.

El criptosistema que hemos descrito puede ser trivialmente extendido para encapsular cadenas de bits de longitud fija ℓ ejecutando el mismo protocolo ℓ veces en paralelo. Debido a su simplicidad, se considera que tiene el menor potencial de ataques que los que vamos a ver a continuación: Ring-LWE y Module-LWE. Sin embargo, esto se paga con costos de comunicación aproximadamente 15 veces más altos que los dos siguientes.

Ring-LWE y Module-LWE

Para mejorar la eficiencia de LWE podemos añadir una estructura algebraica. Retomando la idea que habíamos visto en la figura 8.12, esto parece una muy buena solución ya que la dificultad de los problemas de retículos cuando γ es una función polinómica apenas se ve afectada al añadir una estructura, y por tanto la robustez de sistemas criptográficos basados en estos problemas tampoco se verá afectada. Esta fue la motivación que se tuvo para desarrollar Ring-LWE y Module-LWE.

Ring-LWE fue propuesto por primera vez en 2010 en [37], por lo que es bastante reciente. Los cálculos tienen lugar en un anillo de polinomios $R_q := \mathbb{Z}_q[x]/f(x)$ para algún polinomio $f(x)$. Por lo tanto, se utiliza la multiplicación de polinomios en lugar de la multiplicación de matrices.

Por tanto:

1. Se elige un anillo R_q y un módulo de error q.
2. Dado un vector secreto $\mathbf{s}$ en R_q^n, se genera un vector aleatorio $\mathbf{a}$ y se calcula $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q$, donde $\mathbf{e}$ es un vector de error.

3. El problema consiste en recuperar $\mathbf{s}$ a partir de $(\mathbf{a}, \mathbf{b})$, dados solo $\mathbf{a}$ y $\mathbf{b}$.

Permite una mayor flexibilidad algebraica que el LWE al trabajar con anillos en lugar de cuerpos. La elección del anillo y del polinomio $f(x)$ afecta la complejidad de resolver el problema.

Module-LWE es una variante que mejora aún más Ring-LWE, propuesta en 2012 ([38]). Utiliza la misma estructura que el LWE simple, pero con la diferencia de que ahora los escalares son reemplazados por elementos del anillo R_q como en Ring-LWE. En consecuencia, los vectores se convierten en elementos de los módulos, que son una generalización de los espacios vectoriales sobre anillos, de ahí el nombre.

Permite aún más flexibilidad estructural al no limitarse a anillos de polinomios, por lo que es útil cuando se necesita construir esquemas criptográficos sobre estructuras algebraicas más complejas que las que permite Ring-LWE.

Ambos problemas, Ring-LWE y Module-LWE, a pesar de ser muy recientes, han servido como base para el diseño de algoritmos criptográficos postcuánticos debido a su resistencia aparente a los ataques basados en computación cuántica, proporcionando una base sólida para la seguridad futura de la criptografía, y se seguirá trabajando en nuevos esquemas que tomen estos problemas como base.

	Plain LWE	Ring-LWE	Module-LWE
$\mathbf{A}$	$\mathbb{Z}_q^{n\times m}$	$\mathbb{Z}_q[x]/f$	$(\mathbb{Z}_q[x]/f)^{n\times m}$
$\cdot$	Multiplicación matricial	Multiplicación de polinomios	Multiplicación matricial
$\mathbf{s}$	$\mathbb{Z}_q^m$	$\mathbb{Z}_q[x]/f$	$(\mathbb{Z}_q[x]/f)^m$
$\mathbf{b}, \mathbf{e}$	$\mathbb{Z}_q^n$	$\mathbb{Z}_q[x]/f$	$(\mathbb{Z}_q[x]/f)^n$

Cuadro 8.1: Comparativa entre Plain LWE, Ring-LWE y Module-LWE.

Hemos visto que, en general, en los criptosistemas, añadir una estructura definida tiene la ventaja de que los hace más eficientes pero a la vez la desventaja de que eso suele hacerlos más inseguros. Sin embargo, esto no es el caso para Ring-LWE y Module-LWE (véase [50]). Los ataques que utilizan el hecho de que haya una estructura algebraica subyacente generalmente no se ven beneficiados de la estructura específica del anillo utilizado en Ring-LWE y en Module-LWE, ya que los ataques más eficientes son los que aprovechan la naturaleza estadística del problema (la distribución del ruido) y la capacidad de los algoritmos de encontrar vectores cortos en un retículo de manera eficiente, que son independientes de la estructura del anillo. Por tanto, conservan la ventaja de mayor eficiencia sin comprometer la seguridad, lo que les hace excelentes candidatos para su implementación, como en los algoritmos Kyber y Dilithium, que utilizan esta estructura y hasta ahora han resistido todos los ataques y fueron los ganadores del concurso del NIST. Estos son los denominados CRYSTALS (Cryptographic Suite for Algebraic Lattices).

8.2.1. Kyber

Kyber es un KEM seguro contra ataques de texto cifrado elegido (CCA) derivado de un esquema de cifrado de clave pública seguro contra ataques de texto plano (CPA) basado en Module-LWE, por lo que utiliza las ventajas que ofrece la estructura de los módulos. La idea fue publicada en [9]. Para $n, q \in \mathbb{N}$, el anillo subyacente es $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, es decir, el anillo de polinomios hasta grado $n-1$ con coeficientes en $\mathbb{Z}_q$. El módulo correspondiente es R_q^k con rango $k \in \mathbb{N}$.

Se requiere un espacio de ruido B, donde tomar un valor de B nos da un número entero aleatorio pequeño en el rango $\{-4, \ldots, 4\}$. Con esto obtendremos el error. Además, para la construcción del KEM,

se requieren funciones hash seguras H_1, H_2 y una función de derivación de clave segura KDF (Key Derivation Function).

El texto plano encriptado por Kyber es un elemento del anillo $r \in R_q$, por lo que la cadena de bits de entrada $m \in \{0,1\}^{256}$, mediante la función **toRing**, se convierte en un elemento del anillo, $r = \mathbf{toRing}(m)$, es decir, un polinomio, de la siguiente manera:

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \xrightarrow{\mathbf{toRing}} \begin{pmatrix} 0 \\ 0 \\ \lceil q/2 \rceil \\ \vdots \\ 0 \\ \lceil q/2 \rceil \end{pmatrix} \iff 0 + 0 \cdot x + \frac{q}{2} \cdot x^2 + \ldots + 0 \cdot x^{n-2} + \frac{q}{2} \cdot x^{n-1}. \tag{8.22}$$

Incluso después de haber agregado un vector de error, el polinomio original se puede reconstruir fácilmente. La operación inversa **fromRing** reconstruye una cadena de bits a partir de un elemento de anillo dado mediante división de los coeficientes por $\frac{q}{2}$ y redondeo posterior.

Análogamente al esquema de cifrado basado en LWE, la generación de claves de Kyber es un problema LWE particular $A\mathbf{s}+\mathbf{e} = \mathbf{b}$ generando coeficientes A para el sistema de ecuaciones lineales y muestreando un vector de solución $\mathbf{s}$ así como un vector de error $\mathbf{e}$.

Algoritmo 1. Generación de claves de Kyber PKE: keyGen
Entrada: ninguna
1. Generar $A \in R_q^{k \times k}$ 2. Muestrear $\mathbf{s} \in R_q^k$ con coeficientes de B 3. Muestrear $\mathbf{e} \in R_q^k$ con coeficientes de B 4. Calcular $\mathbf{b} = A\mathbf{s} + \mathbf{e}$
Salida: clave pública pk $= (A, \mathbf{b})$, clave secreta $\mathbf{s}$

El vector de solución $\mathbf{s}$ funciona como la clave secreta mientras que A y el vector $\mathbf{b} = A\mathbf{s}+\mathbf{e}$ se utilizan como la clave pública. Calcular $\mathbf{s}$ a partir de la clave pública es un problema LWE.

Algoritmo 2: Cifrado Kyber PKE: enc
Entrada: clave pública pk $= (A, \mathbf{b})$, mensaje $m \in \{0,1\}^{256}$
1. Muestrear $\mathbf{r} \in R_q^k$ con coeficientes de B 2. Muestrear $\mathbf{e}_1 \in R_q^k$ con coeficientes de B 3. Muestrear $\mathbf{e}_2 \in R_q$ con coeficientes de B 4. Calcular $\mathbf{u} = A^T\mathbf{r} + \mathbf{e}_1$ 5. Calcular $\mathbf{v} = \mathbf{b}^T\mathbf{r} + \mathbf{e}_2 + \mathbf{toRing}(m)$
Salida: texto cifrado $\mathbf{c} = (\mathbf{u}, \mathbf{v})$

Conociendo $\mathbf{s}$, la reconstrucción del mensaje m es posible a través del procedimiento de descifrado PKE de Kyber:

Algoritmo 3. Descifrado PKE de Kyber: dec
Entrada: clave secreta $\mathbf{s}$, texto cifrado $c = (\mathbf{u}, \mathbf{v})$
Calcular $m^* = \mathbf{v} - \mathbf{s}^T\mathbf{u}$
Salida: mensaje $m =$ **fromRing**(m^*)

Aplicar la operación **fromRing** (m^*) reconstruye el m original con una probabilidad muy alta. El esquema de cifrado de Kyber es un algoritmo probabilístico que devuelve el mensaje original m con una probabilidad muy alta dependiendo de la cantidad de ruido dentro de los vectores muestreados.

También podemos usar Kyber como método de generación de claves:

Algoritmo 4. Generación de claves de Kyber KEM
Entrada: ninguna
1. Generar $\sigma \in \{0,1\}^{256}$ 2. Generar $(\text{pk}, \mathbf{s}) = \text{PKE.keyGen}()$
Salida: clave pública pk, clave secreta $\text{sk} = (\mathbf{s}, \sigma)$

En la encapsulación KEM, el valor $\mathbf{r}$ se utiliza como una semilla en el cifrado de clave pública (PKE) para generar valores de manera determinista durante el proceso de cifrado. Aunque normalmente no se desea un algoritmo de cifrado determinista, en un KEM es necesario para que el receptor pueda repetir el procedimiento de cifrado exactamente igual que el emisor. Es decir, queremos que el receptor genere los mismos vectores de error que el origen, y para ello se transmite la semilla del generador pseudoaleatorio $\mathbf{r}$. Denotamos la versión determinista del algoritmo de encriptación basado en la semilla $\mathbf{r}$ dada por $\text{PKE.enc}_r(\text{pk}, m)$. Esto garantiza que tanto el emisor como el receptor obtengan el mismo resultado del cifrado. Además, el mensaje m se ha sometido a un hash antes de ser introducido en el algoritmo de encriptación PKE.

Algoritmo 5. Encapsulación Kyber KEM
Entrada: clave pública pk
1. Generar mensaje $m \in \{0,1\}^{256}$ 2. Calcular $(K', \mathbf{r}) = H_1(H_2(m)\|\|H_2(\text{pk}))$ 3. Calcular $c = \text{PKE.enc}_r(\text{pk}, H_2(m))$ 4. Calcular $K = \text{KDF}(K'\|\|H_2(c))$
Salida: encapsulación c, secreto compartido K

El procedimiento de desencapsulación calcula los valores requeridos de manera análoga al procedimiento de encapsulación.

Algoritmo 6. Desencapsulación Kyber KEM
Entrada: clave pública pk, clave secreta $\text{sk} = (\mathbf{s}, \sigma)$, encapsulación c
1. Calcular $H_m = \text{PKE.dec}(\mathbf{s}, c)$ 2. Calcular $(K', \mathbf{r}') = H_1(H_m\|\|H_2(\text{pk}))$ 3. Calcular $c' = \text{PKE.enc}_r(\text{pk}, H_m)$ 4. Si $c = c'$, establecer $K = \text{KDF}(K'\|\|H_2(c))$ 5. Si $c \neq c'$, establecer $K = \text{KDF}(\sigma\|\|H_2(c))$
Salida: secreto compartido K

En el esquema PKE de Kyber, el mensaje m está incrustado dentro de la diferencia de los vectores $\mathbf{v}$ y $\mathbf{s}^T\mathbf{u}$, es decir,

$$\mathbf{v} - \mathbf{s}^T \cdot \mathbf{u} = \mathbf{toRing}(m) + (\mathbf{e}^T\mathbf{r} + \mathbf{e}_2 - \mathbf{s}^T\mathbf{e}_1),$$

donde $\mathbf{e}, \mathbf{e}_1, \mathbf{e}_2$ son vectores de error aleatorios. Hay varias combinaciones diferentes de valores de estos términos de error que corresponden al mismo m. Sin embargo, en el KEM, la aleatoriedad se vuelve determinista al derivarla de un $\mathbf{r}$ elegido, por lo que hay un conjunto único de valores $(\mathbf{e}, \mathbf{e}_1, \mathbf{e}_2)$ para cada m. Esta propiedad establece la seguridad CCA requerida del KEM. Cuando un adversario envía un texto cifrado aleatorio al procedimiento de desencapsulación, siempre descifrará a un mensaje m, pero la probabilidad de que el adversario haya elegido el texto cifrado específico (generado por los términos "aleatorios" correctos) correspondiente a m es despreciable.

8.2.2. Dilithium

Al igual que Kyber, utiliza las ventajas de una estructura algebraica. Basado en [20]. Para $n, q \in \mathbb{N}$, el anillo subyacente es $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, es decir, el anillo de polinomios hasta grado $n-1$ con coeficientes en $\mathbb{Z}_q$ y relación $X^n = -1$. El módulo correspondiente es R_q^ℓ con rango $\ell \in \mathbb{N}$. Además, Dilithium requiere una función hash segura H.

La generación de claves es casi idéntica a la generación de claves de Kyber. Se genera una matriz $A \in R_q^{k \times \ell}$ con $k \in \mathbb{N}$, un vector secreto $\mathbf{s} \in R_q^\ell$ y un término de error $\mathbf{e} \in R_q^k$. A y $\mathbf{b}$ son públicos mientras que $\mathbf{s}$ se mantiene privado.

Algoritmo 7. Generación de Claves de Dilithium: keyGen
Entrada: ninguna
Generar $A \in R_q^{k \times \ell}$ Muestrear $s \in R_q^\ell$ con coeficientes pequeños Muestrear $e \in R_q^k$ con coeficientes pequeños Calcular $\mathbf{b} = A\mathbf{s} + \mathbf{e}$
Salida: Clave pública $pk = (A, \mathbf{b})$, clave secreta $\mathbf{s}$

El proceso de firma de Dilithium es probabilístico. En el primer paso se muestrea un vector aleatorio $\mathbf{y} \in R_q^\ell$. Como veremos en el proceso de verificación, para lograr la corrección usaremos la versión redondeada de $A\mathbf{y}$ mediante una función *round()*. Esta función toma un vector dado de polinomios y redondea cada coeficiente de cada polinomio. La firma se forma calculando un par (z, c), donde c se forma al aplicar la función hash al mensaje m y al valor redondeado $A\mathbf{y}$. La función hash H mapea una entrada a un polinomio con coeficientes en $\{-1, 0, 1\}$.

Debido a que la dependencia de z de la clave secreta $\mathbf{s}$ conduce potencialmente a problemas de seguridad graves, z no se emite directamente. En su lugar, para eliminar las dependencias estadísticas entre z y s, Dilithium sigue un enfoque de muestreo de rechazo. En caso de que z sea invalidado ("rechazado"), el algoritmo se reinicia desde el paso 1.

Algoritmo 8. Generación de Firmas de Dilithium
Entrada: Clave pública $pk = (A, b)$, clave secreta s, mensaje $m \in \{0,1\}^*$
Hasta que z sea válido: Muestrear $y \in R_q^\ell$ con coeficientes pequeños $w = \text{round}(Ay)$ Calcular $c = H(m \parallel w)$ Calcular $z = y + cs$
Salida: Firma $\sigma = (z, c)$

Dada una firma correcta σ, es posible recuperar w mediante el siguiente cálculo:

$$\text{round}(Az - bc) = \text{round}(A(y + cs) - (As + e)c) = \text{round}(Ay + Acs - Acs - ce) = \text{round}(Ay - ce) = w.$$

Para recuperar w de hecho, el último paso requiere $\text{round}(Ay - ce) = \text{round}(Ay)$. Dado que c y e tienen coeficientes pequeños, su producto ce no influye en el resultado del redondeo. Para verificar la firma, podemos usar un w' recuperado para recalcular $c' = H(m \parallel w')$ y compararlo con el valor de firma proporcionado c. Observemos que si z no ha sido calculado usando la clave secreta s, es decir, por $z = y + cs$, los términos Acs no se cancelarían en la ecuación anterior, lo que conduciría a un $w' \neq w$ incorrecto. Por lo tanto, el valor c' también sería incorrecto, lo que llevaría a un rechazo de la firma proporcionada.

Algoritmo 9. Verificación de Dilithium
Entrada: Clave pública $pk = (A, b)$, mensaje $m \in \{0,1\}^*$, firma $\sigma = (z, c)$
Calcular $w' = \text{round}(Az - bc)$ Calcular $c' = H(m \parallel w')$
Salida: Válido si $c = c'$, sino inválido

Los algoritmos basados en LWE (como Kyber o Dilithium) son algoritmos probabilísticos, ya que LWE involucra la resolución de un sistema de ecuaciones lineales con ruido aleatorio. Este ruido se elige de una distribución (por ejemplo una gaussiana), y por tanto la solución del problema generalmente se obtiene utilizando algoritmos probabilísticos.

8.3. Short Integer Solution (SIS)

El problema de Short Integer Solution (SIS) es otro problema difícil que juega un papel fundamental en la criptografía basada en retículos. Fue introducido por Miklós Ajtai en 1996 ([2]) y es la base de muchos esquemas criptográficos seguros contra ataques cuánticos. Esta sección está basada en [47]. Vamos a empezar definiendo algunos términos:

Operador de desplazamiento rotacional y retículos cíclicos: El operador de desplazamiento rotacional en $\mathbb{R}^n$ $(n \geq 2)$ se denota como rot y se define como:

$$\forall \mathbf{x} = (x_1, \ldots, x_{n-1}, x_n) \in \mathbb{R}^n : \text{rot}(\mathbf{x}) = (x_n, x_1, \ldots, x_{n-1}). \tag{8.23}$$

Retículos cíclicos: Un retículo cíclico es un retículo cerrado para el operador de desplazamiento rotacional. Formalmente, se definen de la siguiente manera:
Un retículo $\mathcal{L} \subseteq \mathbb{Z}^n$ es cíclico si $\forall \mathbf{x} \in \mathcal{L} : \text{rot}(\mathbf{x}) \in \mathcal{L}$.

Ejemplos:

- $\mathbb{Z}^n$ es un retículo cíclico.
- Los retículos correspondientes a cualquier ideal en el anillo de polinomios cociente $R = \mathbb{Z}[x]/(x^n - 1)$ son cíclicos. Consideremos el anillo de polinomios cociente $R = \mathbb{Z}[x]/(x^n - 1)$ y sea $p(x)$ algún

polinomio en R, es decir, $p(x) = \sum_{i=0}^{n-1} a_i x^i$ donde $a_i \in \mathbb{Z}$ para $i = 0, \ldots, n-1$. Se define el isomorfismo que toma un polinomio de R y nos da la $n-$upla de $\mathbb{Z}^n$ de los coeficientes del polinomio:

$$\rho : R \to \mathbb{Z}^n, \quad \sum_{i=0}^{n-1} a_i x^i \mapsto (a_0, \ldots, a_{n-1}). \tag{8.24}$$

Sea $I \subset R$ un ideal. El retículo correspondiente al ideal $I \subset R$, denotado como $\mathcal{L}_I$, es un subretículo de $\mathbb{Z}^n$ y se define como:

$$\mathcal{L}_I := \rho(I) = \left\{ (a_0, \ldots, a_{n-1}) \mid \sum_{i=0}^{n-1} a_i x^i \in I \right\} \subset \mathbb{Z}^n. \tag{8.25}$$

Problema de la solución de enteros cortos (SIS): El problema SIS consiste en, dado un gran número de elementos uniformemente aleatorios de cierto grupo aditivo finito grande, encontrar una combinación entera no trivial lo suficientemente corta de ellos que sume cero. Más formalmente, SIS está parametrizado por los enteros positivos n y q que definen el grupo $\mathbb{Z}_q^n$, un real positivo β, y un número m de elementos del grupo.

Solución Entera Corta ($\mathbf{SIS}_{n,q,\beta,m}$): Dados m vectores $\mathbf{a}_i$ uniformemente aleatorios en $\mathbb{Z}_q^n$, formando las columnas de una matriz $A \in \mathbb{Z}_q^{n \times m}$, encontrar un vector entero no nulo $\mathbf{z} \in \mathbb{Z}^m$ de norma $\|\mathbf{z}\| \leq \beta$ tal que

$$z_1 \cdot \begin{pmatrix} | \\ \mathbf{a_1} \\ | \end{pmatrix} + z_2 \cdot \begin{pmatrix} | \\ \mathbf{a_2} \\ | \end{pmatrix} + \cdots + z_m \cdot \begin{pmatrix} | \\ \mathbf{a_m} \\ | \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{0} \\ | \end{pmatrix} \in \mathbb{Z}_q^n \tag{8.26}$$

es decir,

$$\big(a_{ij}\big)_{1 \leq i \leq m, 1 \leq j \leq n} \begin{pmatrix} | \\ \mathbf{z} \\ | \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{0} \\ | \end{pmatrix} \in \mathbb{Z}_q^n. \tag{8.27}$$

Definimos la función $f_A(\mathbf{z}) := A\mathbf{z} = \sum_i \mathbf{a}_i \cdot z_i = 0$ en $\mathbb{Z}_q^n$.

Algunas observaciones simples pero útiles sobre el problema SIS:

- Sin la restricción sobre $\|\mathbf{z}\|$, es fácil encontrar una solución mediante la eliminación gaussiana. De manera similar, debemos tomar $\beta < q$ porque de lo contrario $\mathbf{z} = (q, 0, \ldots, 0) \in \mathbb{Z}^m$ siempre sería una solución válida (pero trivial).
- Cualquier solución para una matriz A puede convertirse trivialmente en una para cualquier extensión $[A|A_0]$, simplemente agregando el vector solución con ceros (lo que deja inalterada la norma de la solución). Por tanto, podemos ignorar las columnas $\mathbf{a}_i$ según se desee: el problema SIS solo puede volverse más fácil a medida que m aumenta, y solo puede volverse más difícil a medida que n aumenta.
- El límite de la norma β y el número m de vectores $\mathbf{a}_i$ deben ser lo suficientemente grandes como para garantizar que exista una solución. Esto ocurre siempre que $\beta \geq \sqrt{m\overline{m}}$ y $m \geq \overline{m}$, donde $\overline{m}$ es el entero más pequeño mayor que $n \log_2 q$,[5] por un argumento de palomar: primero, por la

[5] El número de bits necesarios para codificar cada componente de $\mathbf{z}$ es $log_2 q$, y tenemos n componentes

observación anterior podemos asumir sin pérdida de generalidad que $m = \overline{m}$. Luego, debido a que hay más de q^n vectores $\mathbf{x} \in \{0,1\}^m$, debe haber dos $\mathbf{x}$, $\mathbf{x}_0$ distintos tales que $A\mathbf{x} = A\mathbf{x}_0 \in \mathbb{Z}_q^n$, por lo que $A(\mathbf{x} - \mathbf{x}_0) = \mathbf{0}$ y por tanto $\mathbf{z} = \mathbf{x} - \mathbf{x}_0 \in \{0, \pm 1\}^m$ es una solución con una norma de como máximo β.

- El argumento anterior nos da un detalle muy importante: la familia de funciones inducidas $\{f_A : \{0,1\}^m \to \mathbb{Z}_q^n\}$ es *resistente a colisiones*, asumiendo la dificultad del problema SIS correspondiente. Esto se debe a que una colisión $\mathbf{x}, \mathbf{x}_0 \in \{0,1\}^m$ para f_A inmediatamente produce una solución de SIS para A. Nótese que el dominio $\{0,1\}^m$ es arbitrario; puede ser reemplazado por prácticamente cualquier otro conjunto lo suficientemente grande de vectores enteros lo suficientemente cortos.

[48]: La relación entre el SIS y los problemas de retículos se pueden ver en el siguiente ejemplo: Sea $A \in \mathbb{Z}^{n \times m}$ que define un retículo q-ario:

$$\mathcal{L}^{\perp}(A) = \{\mathbf{z} \in \mathbb{Z}^m : A\mathbf{z} = 0\}. \tag{8.28}$$

Las soluciones cortas $\mathbf{z}$ se encuentran en el círculo azul:

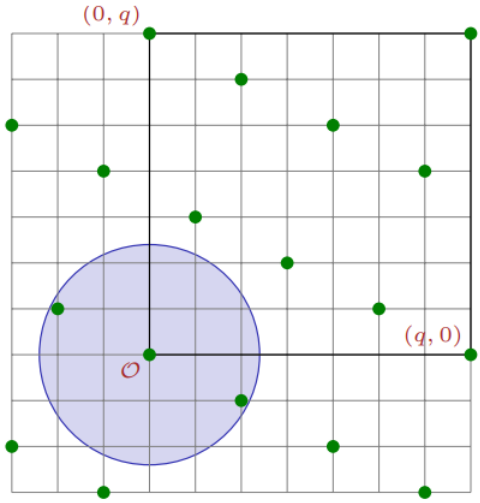

Figura 8.13: Retículo q-ario $\mathcal{L}^{\perp}(A)$. Dentro del círculo azul están las soluciones del SVP. [48]

8.3.1. Aplicación del SIS: Firma digital

Generamos una clave A con una *trapdoor* secreta T. Para firmar un mensaje μ, usamos T para muestrear un vector corto $\mathbf{z} \in \mathbb{Z}^m$ tal que $A\mathbf{z} = H(\mu) \in \mathbb{Z}_q^n$. Se elige $\mathbf{z}$ de una distribución que no revele nada sobre la clave secreta. Para verificar la firma $H(\mu)$, comprobamos que $A\mathbf{z} = H(\mu)$ y que $\mathbf{z}$ es suficientemente corto. La seguridad de esto es que si queremos firmar un nuevo mensaje μ^*, necesitamos encontrar un vector corto $\mathbf{z}^*$ tal que $A\mathbf{z}^* = H(\mu^*)$. Esto es el problema SIS, que sabemos que es difícil.

8.4. El algoritmo LLL

El algoritmo LLL (Lenstra-Lenstra-Lovász), publicado en [34], es un algoritmo muy utilizado en criptografía que permite reducir bases de retículos, produciendo una base con vectores más cortos y más cercanos a ser ortogonales.

Definición 8.4.1. ***Ortogonalización de Gram-Schmidt****: Dada una base* $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$*, la ortogonalización de Gram-Schmidt produce un conjunto de vectores ortogonales* $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*$ *tal que:*

$$\mathbf{b}_i = \sum_{j=1}^{i} \mu_{i,j} \mathbf{b}_j^* \tag{8.29}$$

donde $\mu_{i,j}$ *son coeficientes de proyección.*

Definición 8.4.2. ***Reducción de Lovász****: La base* $\mathbf{B}$ *es reducida de Lovász si:*

$$|\mu_{i,j}| \leq \frac{1}{2} \quad \textit{para todos los } 1 \leq j < i \leq n \tag{8.30}$$

y

$$\delta \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_i + \mu_{i,i-1}\mathbf{b}_{i-1}^*\|^2. \tag{8.31}$$

para un parámetro δ *típicamente tomado como* $\delta = \frac{3}{4}$.

El algoritmo

Entrada: Base $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$.

Salida: Base reducida $\mathbf{B}' = \{\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_n\}$.

1. Inicialización:
   Calcular la ortogonalización de Gram-Schmidt de $\mathbf{B}$ para obtener $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*$ y los coeficientes $\mu_{i,j}$.

2. Reducción de Gram-Schmidt:
   for $i = 2$ to n
     for $j = i - 1$ to 1
       Actualizar $\mathbf{b}_i$ restando la proyección en $\mathbf{b}_j^*$:

$$\mathbf{b}_i := \mathbf{b}_i - \text{round}(\mu_{i,j})\mathbf{b}_j \tag{8.32}$$

       Recalcular $\mu_{i,j}$ y actualizar $\mathbf{b}_i^*$ si es necesario.
     end for

3. Condición de Lovász:
   if $\delta\|\mathbf{b}_{i-1}^*\|^2 > \|\mathbf{b}_i^*\|^2 + \mu_{i,i-1}^2\|\mathbf{b}_{i-1}^*\|^2$
     Intercambiar $\mathbf{b}_i$ y $\mathbf{b}_{i-1}$.
     Recalcular la ortogonalización de Gram-Schmidt para los vectores intercambiados.
     Recalcular $\mu_{i,j}$ y $\mathbf{b}_i^*$ según sea necesario.
   end if
   end for

Es decir, estamos repitiendo los pasos 2 y 3 hasta que toda la base cumpla las condiciones de reducción de Lovász, y cuando esto se cumple ya hemos obtenido la base reducida.

Ejemplo

Consideremos una base en $\mathbb{R}^2$: $\{\mathbf{b}_1 = (1,1), \mathbf{b}_2 = (3,2)\}$

1. **Ortogonalización de Gram-Schmidt:**

$$\mathbf{b}_1^* = \mathbf{b}_1 = (1,1).$$

$$\mu_{2,1} = \frac{\langle \mathbf{b}_2, \mathbf{b}_1^* \rangle}{\langle \mathbf{b}_1^*, \mathbf{b}_1^* \rangle} = \frac{5}{2}.$$

$$\mathbf{b}_2^* = \mathbf{b}_2 - \mu_{2,1}\mathbf{b}_1^* = (3,2) - \frac{5}{2}(1,1) = \left(\frac{1}{2}, -\frac{1}{2}\right).$$

2. **Reducción de Gram-Schmidt:**

$$\text{Redondear } \mu_{2,1} = \frac{5}{2} \text{ a } 2.$$

$$\mathbf{b}_2 := \mathbf{b}_2 - 2\mathbf{b}_1 = (3,2) - 2(1,1) = (1,0).$$

3. **Condición de Lovász:**

Recalcular la ortogonalización de Gram-Schmidt:

$$\mathbf{b}_2^* = (1,0).$$

Verificar la condición de Lovász:

Se cumple porque $\delta\|\mathbf{b}_1^*\|^2 \leq \|\mathbf{b}_2^*\|^2$.

El resultado final es una base reducida $\{(1,1),(1,0)\}$, que tiene vectores más cortos y más cercanos a ser ortogonales.

Aplicaciones del LLL

El LLL se utiliza en el criptoanálisis de varios sistemas criptográficos, especialmente aquellos basados en problemas de retículos, ya que al reducir bases de retículos y encontrar vectores cortos, se puede comprometer la seguridad del sistema. También se utiliza para reducir el tamaño de las claves en ciertos sistemas criptográficos, manteniendo la seguridad del sistema. Nótese que aunque el algoritmo LLL resuelve los problemas de retículos en tiempo polinómico, lo hace solo para factores de aproximación subexponenciales ([47], sección 2.2).

8.5. Reducción del caso peor al caso medio

La criptografía requiere problemas para los cuales las instancias aleatorias (extraídas de una distribución de probabilidad especificada), o en otras palabras, los casos medios de un problema, sean difíciles de resolver, ya que en la práctica es con lo que vamos a trabajar, con las instancias aleatorias. Esto es cualitativamente diferente de la noción de dificultad en el caso peor que se considera generalmente en la teoría de algoritmos y NP-completitud, donde un problema se considera difícil si simplemente existen algunas instancias intratables.

Los problemas que parecen difíciles en el caso peor a menudo resultan ser más fáciles en el caso promedio, especialmente para distribuciones que producen instancias con cierta estructura adicional, como la existencia de una clave secreta para descifrado. La criptografía basada en retículos presenta una propiedad notable: la complejidad del caso medio del SIS y del LWE es al menos tan alta como la complejidad del caso peor de problemas de retículos relacionados con el SVP. Es sorprendente que se pueda establecer una relación entre el caso medio de un problema (instancias generadas de manera aleatoria o promedio) y el caso peor de otro. Esto se logra mediante una reducción del problema de retículos difícil al SIS y al LWE, de manera que el caso medio del SIS y el LWE es al menos tan difícil como el caso peor de problemas de retículos.

Esta fue la idea de Ajtai, que en su artículo [2] introdujo el problema del SIS (en el caso promedio) y demostró que resolverlo es al menos tan difícil como resolver problemas de retículos en el caso peor.

En particular, el trabajo de Ajtai introdujo la primera función criptográfica que se fundamenta en la dificultad computacional del peor caso estándar. Estos resultados permiten diseñar construcciones criptográficas y demostrar que son seguras, a menos que todos los problemas de retículos (incluyendo el peor caso) sean fáciles de resolver.[6] Este enfoque sigue siendo crucial en aplicaciones criptográficas hasta el día de hoy.

13 años después, Regev en [54], utilizando esta misma idea, demostró que resolver LWE en el caso promedio es al menos tan difícil como resolver ciertos problemas de retículos en el caso peor.

En los siguientes apartados se esbozan las demostraciones de estas dos reducciones. Las demostraciones completas están en los dos artículos citados, pero en este trabajo solo nos centraremos en la idea general de la demostración, que viene muy clara en [35], por lo que me he basado en estos tres documentos.

8.5.1. Reducción del SIS

El problema SIS es un problema de caso promedio que se utiliza en construcciones criptográficas basadas en retículos. Como hemos dicho, Ajtai en su artículo [2] presentó una familia de funciones unidireccionales basadas en el problema SIS y demostró que es seguro en el caso promedio si SVP_γ es difícil en el peor de los casos.

Teorema 8.5.1. *Para cualquier $m = poly(n)$[7], cualquier $\beta > 0$ y cualquier $q \geq \beta \cdot poly(n)$, resolver $SIS_{n,q,\beta,m}$ con probabilidad no despreciable es al menos tan difícil como resolver el $GapSVP_\gamma$ y los $SIVP_\gamma$ (entre otros) en retículos arbitrarios de dimensión n (es decir, en el peor de los casos) con probabilidad abrumadora, para algún $\gamma = \beta \cdot poly(n)$.*

La demostración de este teorema se basa en una sucesión de reducciones que se puede ver bien en la siguiente imagen:

[6] Téngase en cuenta que muchos problemas numéricos utilizados en criptografía como el logaritmo discreto también admiten reducciones de caso peor/caso promedio pero solo dentro de un grupo fijo. Una reducción de este tipo nos da una distribución sobre un grupo que es tan difícil como el caso peor para el mismo grupo, pero no dice nada sobre si el grupo en sí es difícil, o qué grupos son más difíciles. De hecho, la complejidad de estos problemas parece variar bastante según el tipo de grupo (por ejemplo, grupos multiplicativos de enteros módulo un primo o de otros cuerpos finitos, grupos de curvas elípticas, etc.). Por tanto, esta reducción de retículos es mucho más potente al ser válida de forma general.

[7] La notación $m = \mathrm{poly}(n)$ significa que m es una función polinómica de n.

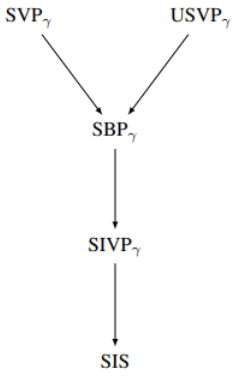

Figura 8.14: Reducciones al problema SIS desde problemas difíciles de retículos (SVP_γ, $USVP_\gamma$ y SBP_γ). El problema de retículos intermedio en las reducciones es la γ-aproximación del problema del vector independiente más corto ($SIVP_\gamma$). [35]

La demostración de Ajtai se basa en reducir SVP_γ y $USVP_\gamma$[8] a SBP_γ[9], luego reducir este al $SIVP_\gamma$ y por último, el $SIVP_\gamma$ en el SIS. La demostración completa está en su artículo original, pero es demasiado extensa así que solo mostraré los pasos principales y algunas ideas claves utilizadas en la misma.

Reducción de SVP$_\gamma$ a SBP$_\gamma$: Dado un retículo $\mathcal{L}$ y un algoritmo que resuelve SBP$_\gamma$, se puede usar este algoritmo para encontrar una base corta del retículo. Una vez que se tiene una base corta, el vector más corto en esta base es una buena aproximación del vector más corto en el retículo. Este proceso se repite, refinando la base hasta que se obtenga un vector que cumpla con las condiciones de SVP$_\gamma$.

Reducción de USVP$_\gamma$ a SBP$_\gamma$: En el caso de USVP$_\gamma$, se asume que el vector más corto es único. Similar a la reducción anterior, se usa el algoritmo de SBP$_\gamma$ para encontrar una base corta del retículo. Dada la unicidad del vector más corto, se puede asegurar que el vector más corto encontrado en la base es el vector más corto del retículo (si hubiera un vector más corto en el retículo que no estuviera en la base, la base no sería la más corta posible, lo cual contradiría la definición de SBP$_\gamma$). Por tanto, este vector cumple con las condiciones de USVP$_\gamma$.

SBP_γ está relacionado con $SIVP_\gamma$ porque dado un conjunto de vectores de $\mathcal{L}$ linealmente independientes $\mathbf{r_1}, ..., \mathbf{r_n} \in \mathcal{L}$, se puede construir en tiempo polinómico una base $\{\mathbf{s_1}, ..., \mathbf{s_n}\}$ de $\mathcal{L}$ tal que $\text{máx}_{i=1}^{n} ||\mathbf{s_i}|| \leq n \, \text{máx}_{i=1}^{n} ||\mathbf{r_i}||$. Por lo tanto, la tarea se reduce a reducir el $SIVP_\gamma$ a SIS. Para ello, se asume que existe un algoritmo probabilístico de tiempo polinómico $\mathcal{A}$ que resuelve SIS con una probabilidad no despreciable. El siguiente paso es transformar una instancia difícil de SIVP$_\gamma$ en una instancia aleatoria de SIS y demostrar que, si existe tal solución $\mathcal{A}$ para SIS, esta da lugar a un algoritmo probabilístico de tiempo polinómico $\mathcal{B}$ que resuelve SIVP$_\gamma$ para un factor polinómico. Esta solución luego se transforma en una solución para SBP$_\gamma$, así como para SVP$_\gamma$ y USVP$_\gamma$.

Se denota $M = \text{máx}_i ||\mathbf{a_i}||$ y $bl(\mathcal{L})$ la longitud de la base más corta de $\mathcal{L}$. La clave está en encontrar una cota superior para M (que va a depender de $bl(\mathcal{L})$) para así garantizar que se van acortando iterativamente los vectores más largos a la mitad para lograr $\frac{M}{2}$. Repitiendo estos pasos obtenemos vectores de la longitud deseada. Cada iteración de este proceso es la siguiente:

[8] El problema del vector más corto único (USVP) es una variante del SVP. En USVP, se asume que existe un único vector más corto en el retículo.

[9] El problema de la base más corta (SBP) es un problema donde el objetivo es encontrar la base más corta posible. La longitud de la base se define como $\text{máx}_{i=1}^{n} \|\mathbf{b_i}\|$, siendo $\mathbf{b_i}$ los vectores de la base.

1. Construir un paralelepípedo casi cúbico: Con esto nos referimos a, partiendo de los vectores del retículo $\mathbf{a_1}, \ldots, \mathbf{a_n}$, construir otros vectores del retículo $\mathbf{f_1}, \ldots, \mathbf{f_n}$ de modo que sean casi ortogonales entre sí y tengan longitudes similares, pero restringiendo la longitud máxima $\text{máx}_{1 \leq i \leq n} ||\mathbf{f_i}|| \leq n^3 M$, y con estos vectores formar un paralelepípedo $W = P(\mathbf{f_1}, \ldots, \mathbf{f_n})$ que sea casi un hipercubo[10]. Para ilustrarlo mejor veamos un ejemplo:

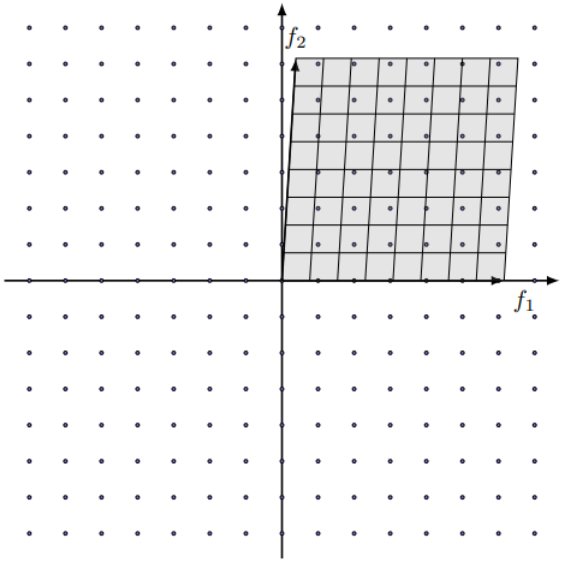

Figura 8.15: En un retículo $\mathcal{L} = \mathbb{Z}^2$, el paralelepípedo casi cúbico W formado por los vectores independientes $\{\mathbf{f_1}, \mathbf{f_2}\}$. [35]

2. Inducir una instancia de SIS casi uniforme: Luego seccionamos W en q^n paralelepípedos pequeños no superpuestos que tienen la forma $w_j = (\sum_{i=1}^{n} \frac{t_j^i}{q} \mathbf{f_i}) + \frac{1}{q} W$, donde $t_j^i \in [0, q)$ es un entero. Ahora, muestreamos m vectores aleatorios del retículo $\mathcal{L}$, luego los reducimos módulo W para asegurarnos de que estén dentro del paralelepípedo más grande. Denotamos estos vectores reducidos como $\mathbf{x_{i_1}}, \ldots, \mathbf{x_{i_m}}$. Si $\mathbf{x_{i_k}}$ está en un paralelepípedo más pequeño $w_j = (\sum_{i=1}^{n} \frac{t_j^i}{q} f_i) + \frac{1}{q} W$, entonces tomamos $(t_j^1, \ldots, t_j^n)$ y lo colocamos como una columna de una matriz A. La idea es que cada uno de los w_j se selecciona con casi la misma probabilidad, por lo que tenemos una matriz $n \times m$ aleatoria A. En nuestro ejemplo de la figura 8.15, se divide W en q^2 piezas más pequeñas, cada una de las

[10] El interés en que el paralelepípedo sea casi cúbico radica en varias razones clave relacionadas con la estructura geométrica del retículo y la eficiencia de los algoritmos utilizados para resolver problemas en criptografía basada en retículos. Estas razones son:

a) Uniformidad en la distribución de vectores, ya que cuando el paralelepípedo es casi cúbico, sus lados tienen longitudes similares y los vectores que lo forman son casi ortogonales. Esto asegura que al dividir el paralelepípedo en partes más pequeñas, cada sub-paralelepípedo tendrá aproximadamente el mismo volumen. Esto permite que los vectores aleatorios muestreados dentro del paralelepípedo grande se distribuyan casi uniformemente entre los sub-paralelepípedos. La uniformidad es crucial para garantizar la seguridad del esquema criptográfico.

b) Tener vectores que son casi ortogonales simplifica los cálculos y las transformaciones matemáticas necesarias para construir la base de vectores $\mathbf{f_1}, \ldots, \mathbf{f_n}$. La ortogonalidad facilita la aplicación de algoritmos de reducción de base, como el algoritmo LLL.

c) A medida que se va reduciendo la longitud de los vectores en cada iteración, un paralelepípedo casi cúbico garantiza que la reducción sea eficiente y que la longitud de los vectores se disminuya de manera uniforme. Esto permite que la cota superior M se reduzca de forma predecible en cada iteración.

cuales es alcanzada con igual probabilidad por vectores aleatorios del retículo reducidos dentro de W.

La intuición clave es que si W se corta en pequeñas regiones no superpuestas de manera uniforme, entonces los vectores aleatorios del retículo dentro de W inducirán una distribución casi uniforme sobre las piezas w_j. Esto implica que la matriz A es una instancia aleatoria de SIS.

3. Reducir a la mitad la longitud del vector: Ahora asignamos la matriz obtenida en el paso anterior A al algoritmo probabilístico de tiempo polinómico $\mathcal{A}$ para que produzca una solución SIS $(h_1, \ldots, h_m) \in \mathbb{Z}^m$. Por último se demuestra que el vector $\mathbf{u} = \sum_{i=1}^{n} h_i \xi_i$ es solo de la mitad del tamaño de los vectores iniciales, es decir, $||\mathbf{u}|| \leq \frac{M}{2}$ y que son no nulos.

Todos los detalles en profundidad están en el artículo original de Atjai [2].

Por tanto, ya hemos visto ahora la reducción del caso peor de retículos al caso medio del SIS. Veamos lo mismo ahora para el LWE:

8.5.2. Reducción del LWE

El trabajo de Regev ([54]), publicado en 2009, mostró una reducción del caso peor al caso medio para el problema LWE. Regev demostró que resolver LWE en el caso promedio (instancias aleatorias) es al menos tan difícil como resolver ciertos problemas de retículos en el caso peor. Esos problemas de retículos son el SVP y el SIVP. Para esta reducción se utiliza una distribución gaussiana sobre retículos, lo que permite manejar los errores de forma controlada. Después, se usa la matriz generadora del retículo para convertir las instancias de SVP y SIVP en instancias de LWE.

Esta reducción establece una base sólida para la seguridad de los criptosistemas basados en LWE, ya que garantiza que resolver LWE es tan difícil como resolver problemas de retículos, que son reconocidos por su dificultad.

Teorema 8.5.2. ***Teorema:*** *Para cualquier $m = poly(n)$, cualquier módulo $q \leq 2^{poly(n)}$, y cualquier distribución de error Gaussiana (discretizada) χ de parámetro $\alpha q \geq 2\sqrt{n}$ donde $0 < \alpha < 1$, resolver el problema de decisión LWE$_{n,q,\chi,m}$ es al menos tan difícil como resolver con un ordenador cuántico GapSVP$_\gamma$ y SIVP$_\gamma$ en retículos arbitrarios de n dimensiones, para algún $\gamma = \tilde{O}(n/\alpha)$.*

La distribución de la que se extraen los errores (DGS, discrete gaussian sampling) es una distribución en $\mathbb{Z}_p$ que tiene la forma de una gaussiana discreta centrada en 0 con una desviación estándar αp, como se muestra en la Figura 8.16. Además, la probabilidad de obtener un 0 (es decir, sin error) es aproximadamente $1/(\alpha p)$.

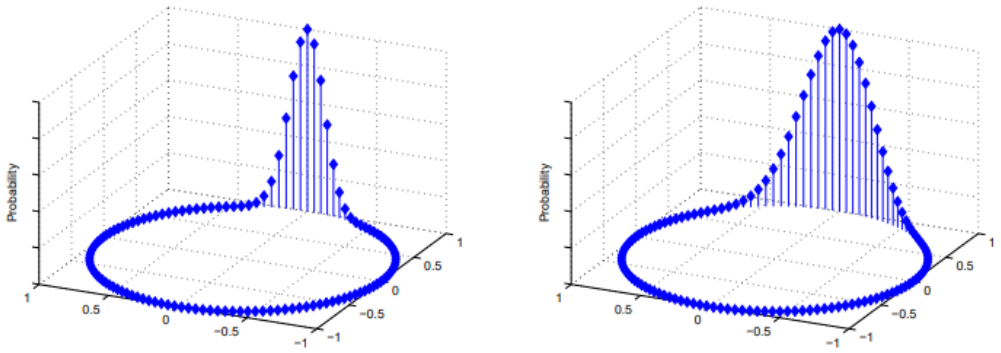

Figura 8.16: La distribución de la que se extraen los errores para $p = 127$ con $\alpha = 0{,}05$ (izquierda) y $\alpha = 0{,}1$ (derecha). Los elementos de $\mathbb{Z}_p$ están dispuestos en una circunferencia. [54]

La demostración del teorema utiliza una construcción iterativa. Esencialmente, esto significa que en lugar de encontrar "inmediatamente" vectores muy cortos en el retículo, la reducción se realiza en pasos en los que en cada paso se encuentran vectores del retículo cada vez más cortos.

Hasta la fecha no existe ningún algoritmo clásico de tiempo polinómico para este tipo de problemas de retículos. Incluso se podría conjeturar que no existe tampoco un algoritmo cuántico. Entonces, se puede interpretar el teorema como que, basado en esta conjetura, el problema LWE es difícil. Esta conjetura está fundamentada en el hecho de que no se conocen algoritmos cuánticos para problemas de retículos que superen a los algoritmos clásicos, aunque esta es una de las preguntas abiertas más importantes en el campo de la computación cuántica.

De hecho, también se podría interpretar nuestro teorema principal como una forma de refutar esta conjetura: si se encuentra un algoritmo eficiente para LWE, entonces también se obtiene un algoritmo cuántico para aproximar problemas de retículos en el caso peor. Un resultado así tendría una importancia tremenda por sí mismo, pues pondría en jaque toda la criptografía basada en retículos, no solo la basada en LWE. También es posible que se demuestre que el teorema principal también es válido para el caso clásico, aunque por ahora no se ha descubierto.

Los pasos principales de la demostración que hizo Regev se esbozan en la Figura 8.17, en la que se describe el flujo de reducciones desde problemas de retículos más complejos (GapSVP y SIVP) hasta problemas más manejables (LWE), utilizando tanto métodos clásicos como cuánticos.

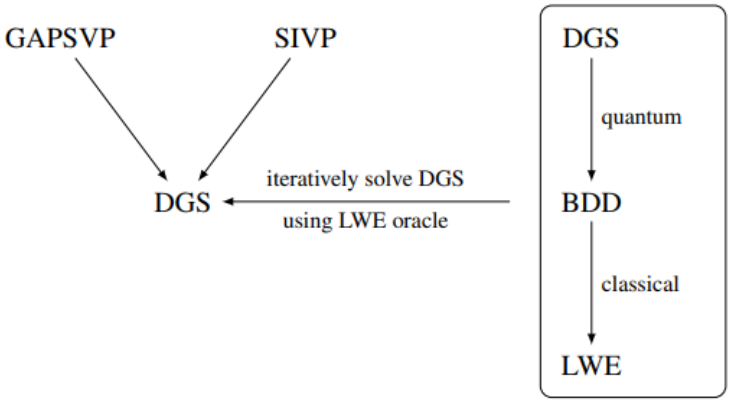

Figura 8.17: Reducciones de GapSVP y SIVP a LWE. [35]

La estrategia implica resolver DGS (generar vectores en un retículo según una distribución gaussiana discreta) de manera iterativa, apoyándose en un oráculo LWE y un algoritmo cuántico para mejorar las soluciones a problemas de retículos. Entonces, resolver GapSVP y SIVP puede reducirse a resolver el problema DGS.

Resolver DGS iterativamente usando un oráculo LWE

La clave para resolver DGS para una pequeña escala r cercana a su límite inferior es aplicar iterativamente una subrutina que gradualmente reduzca la escala. Esta subrutina proporciona muestras gaussianas discretas a un oráculo LWE para resolver BDD (Bounded Distance Decoding)[11] de manera clásica, y el resultado se usa luego por un algoritmo cuántico para producir muestras gaussianas discretas más cortas.

Reducción de DGS a BDD y de BDD a LWE

En el recuadro se muestra una reducción clásica de BDD a LWE, sugiriendo que si se puede resolver LWE, entonces se puede resolver BDD. La reducción de DGS a LWE implica primero resolver DGS usando un algoritmo cuántico y luego utilizar un oráculo LWE para resolver BDD de manera clásica.

Nota: durante la realización de este trabajo de fin de grado, se publicó un preprint ([13]) en el que se presenta un supuesto algoritmo cuántico en tiempo polinómico para resolver el LWE. Usando las reducciones de problemas de retículos a LWE, se obtendrían algoritmos cuánticos en tiempo polinómico para resolver el problema aproximado del vector más corto decisional (GapSVP_γ) y el problema aproximado del vector independiente más corto (SIVP_γ) con γ una función polinómica de n. Tan solo es un preprint y es posible que se encuentre algún error en el algoritmo (de hecho, ya se ha detectado un error en el artículo), pero la criptografía es un campo que está en pleno desarrollo y los algoritmos que hasta ahora son seguros pueden dejar de serlo en cualquier momento.

[11] Recordemos que BDD es el problema en el que se busca encontrar el vector de retículo más cercano a un punto dado cuando la distancia está acotada.

8.5.3. Relación del LWE con códigos

Una idea muy interesante que plantea Regev en su trabajo ([54], página 34) es la siguiente: el problema LWE puede ser presentado de manera equivalente como el problema de decodificar códigos lineales aleatorios. Si nos fijamos en el problema, en realidad estamos haciendo lo mismo. Esto establece una conexión importante entre la teoría de códigos y los problemas de retículos en el contexto de la criptografía postcuántica.

Se define LWE tal y como lo hemos hecho: sea $m = \text{poly}(n)$ arbitrario y $\mathbf{s} \in \mathbb{Z}_p^n$ un vector dado. Sea $Q \in \mathbb{Z}_p^{m \times n}$ una matriz aleatoria y el vector $\mathbf{t} = Q\mathbf{s} + \mathbf{e} \in \mathbb{Z}_p^m$ donde cada coordenada del vector de errores $\mathbf{e} \in \mathbb{Z}_p^m$ se elige independientemente de una distribución gaussiana como la definida en 8.16 (recordemos que esa distribución tiene la propiedad de que cada valor es 0 con una probabilidad de aproximadamente $\frac{1}{\alpha p}$). El problema consiste en recuperar $\mathbf{s}$.

El peso de Hamming de $\mathbf{e}$, es decir, el número de entradas no nulas en $\mathbf{e}$, es aproximadamente $m(1-\frac{1}{\alpha p})$. Esto se debe a que cada coordenada de $\mathbf{e}$ es 0 con probabilidad $\frac{1}{\alpha p}$ y por tanto distinta de 0 con probabilidad $1 - \frac{1}{\alpha p}$.

Por lo tanto, la distancia de Hamming de $\mathbf{t}$ a $Q\mathbf{s}$ es aproximadamente $m(1 - \frac{1}{\alpha p})$. Además, se puede observar que para un m lo suficientemente grande, para cualquier otra palabra $\mathbf{s}' \neq \mathbf{s}$ la distancia de Hamming de $\mathbf{t}$ a $Q\mathbf{s}'$ es aproximadamente $m(1 - \frac{1}{p})$. Esto se debe a que, para cualquier otra palabra $\mathbf{s}'$, los valores de $\mathbf{t}$ estarán distribuidos aleatoriamente respecto a $Q\mathbf{s}'$, lo que implica que la probabilidad de coincidencia en cada coordenada es $\frac{1}{p}$. Por lo tanto, la fracción de coordenadas en las que $\mathbf{t}$ y $Q\mathbf{s}'$ difieren es aproximadamente $1 - \frac{1}{p}$, resultando en una distancia de Hamming de $m(1 - \frac{1}{p})$.

Con esto obtenemos que aproximar el problema de la palabra código más cercana dentro de factores menores que

$$\frac{(1 - \frac{1}{p})}{(1 - \frac{1}{\alpha p})} \tag{8.33}$$

en códigos aleatorios es tan difícil como aproximar problemas de retículos en el peor caso de manera cuántica. Esto da una idea de la enorme dificultad de decodificar códigos lineales aleatorios, lo cual sigue siendo una pregunta abierta.

Como vemos, los códigos y el problema LWE están íntimamente relacionados. Pero además, podemos construir retículos a partir de códigos ([21], páginas 10 y 127). La criptografía basada en retículos y la criptografía basada en códigos tienen mucha relación y los avances en una pueden llevar a avances en la otra.

8.6. Cifrado homomórfico a partir de retículos

Una de las grandes ventajas que presenta la criptografía basada en retículos es que permiten la construcción de un cifrado completamente homomórfico, es decir, podremos realizar operaciones matemáticas en los datos cifrados sin necesidad de descifrarlos primero. Esto es posible debido a la propia estructura algebraica de los retículos: permiten definir operaciones como sumas y multiplicaciones que conservan

ciertas propiedades clave, como la distributividad y la cerradura bajo estas operaciones.

Esto es extremadamente útil en aplicaciones donde se desea realizar cálculos sobre datos mientras se mantienen encriptados para preservar la privacidad.

Capítulo 9

Conclusiones

Tras haber estudiado las diferencias entre los principales enfoques de la criptografía postcuántica (retículos, códigos correctores, funciones hash y polinomios multivariable), los problemas en los que basan su seguridad frente a ordenadores cuánticos y las implementaciones concretas de cada uno, es interesante realizar una pequeña comparativa de todos ellos.

9.0.1. Retículos

Los sistemas basados en retículos, como el LWE o el SIS, son actualmente los más estudiados y propuestos como candidatos sólidos debido a la aparente dificultad de los problemas de retículos, que nos garantiza la seguridad tanto para computadoras clásicas como para cuánticas. Además se pueden aplicar para diversas construcciones criptográficas, incluyendo cifrado, firmas digitales y protocolos de intercambio de claves. Además permiten el cifrado homomórfico completo. Debido a todas estas características fueron los ganadores del concurso del NIST.

Una de las desventajas que presentan los retículos en comparación con los códigos, los esquemas multivariable o las funciones hash es el tamaño de claves. Sin embargo, hemos visto que se ha descubierto una propiedad esencial de los retículos: *añadir una estructura algebraica parece que no compromete la seguridad y mejora la eficiencia.* Como siempre en criptografía, es posible que se encuentre algún ataque que se aproveche de esta estructura, pero por ahora no se ha encontrado. Esto nos va a permitir solventar este inconveniente de la longitud de las claves. Es una idea bastante reciente, pero ya se está empezando a explotar de manera fructífera.

9.0.2. Códigos correctores

Hemos detallado la importancia de los códigos correctores en la criptografía, tanto para corregir mensajes que atraviesan un canal ruidoso como para la construcción de criptosistemas. Los criptosistemas que se basan en códigos correctores de errores, como McEliece, tienen una larga trayectoria y se cree que son muy seguros debido a la aparente dificultad de los problemas de decodificación de códigos aleatorios. Han resistido hasta ahora, así que su seguridad contra ataques tanto clásicos como cuánticos parece también bastante garantizada. Hemos visto concretamente tres criptosistemas: McEliece, Neiderreiter y HyMES, comparando las ventajas y desventajas de cada uno.

9.0.3. Funciones hash

Son generalmente simples de implementar y ofrecen una gran eficiencia en términos de tiempo de ejecución y tamaño de firma. La seguridad se basa en la resistencia a la colisión de las funciones hash, y basando esto en retículos obtenemos las garantías de seguridad de los retículos, que, al menos por ahora, son muy seguros. Las funciones hash son también un componente indispensable en criptografía.

9.0.4. Polinomios multivariable

Los sistemas basados en polinomios multivariable ofrecen un rendimiento muy rápido en la verificación y generación de firmas, y algunos esquemas son altamente compactos en términos de tamaño de clave y firma. No obstante, la seguridad de estos sistemas no está tan bien estudiada como la de los sistemas basados en retículos o códigos, lo que genera incertidumbre sobre su fiabilidad a largo plazo. Por lo tanto, es conveniente estudiar más este enfoque para tener una idea más clara de la solidez que pueden ofrecer.

A la hora de elegir qué esquema criptográfico seguir, siempre hay que hacer un balance entre seguridad y eficiencia. Tras haber visto las diferentes opciones de criptosistemas postcuánticos, podemos elegir el que más se adapte a las necesidades concretas de cada caso. Si buscamos una alta seguridad, los retículos parece ser una opción muy prometedora. Para aplicaciones que pueden tolerar tamaños de clave grandes y también se necesite una alta seguridad, McEliece puede ser una opción muy acertada. Para aplicaciones con restricciones de tamaño de clave, las funciones hash y los polinomios multivariable pueden ser más adecuados, siempre y cuando se seleccione cuidadosamente el esquema para evitar vulnerabilidades. Es muy importante trabajar en todos estos enfoques hasta conseguir los esquema más seguros en los que basar toda nuestra criptografía, y así estar preparados para el momento en el que los ordenadores cuánticos puedan mantener el entrelazamiento en un número suficiente de qubits y el peligro que prometen se convierta en realidad.

Bibliografía

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. https://doi.org/10.4007/annals.2004.160.781.

[2] M. Ajtai. Generating hard instances of lattice problems. *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 99–108, 1996.

[3] J.-P. Aumasson. *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, USA, 2017.

[4] A. Barg. Complexity issues in coding theory. In V. Pless and W. Huffman, editors, *Handbook of Coding Theory*, chapter 7, pages 649–754. Elsevier, 1998.

[5] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[6] M. Bernard, C. Pierrot, D. Stehlé, and G. Villard. Twisted-PHS: using the product formula to solve approx-SVP in ideal lattices. *Acta Cryptographica*, 2(1):1–23, 2020. https://doi.org/10.1007/s12095-020-00420-7.

[7] B. Biswas and N. Sendrier. McEliece Cryptosystem Implementation: Theory and Practice. In *PQCrypto*, Paris - Rocquencourt, France, 2008. INRIA.

[8] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, Feb. 2024. https://doi.org/10.1038/s41586-023-06927-3.

[9] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 353–367, 2018.

[10] D. Castelvecchi. IBM releases first-ever 1,000-qubit quantum chip. *Nature*, 2023. https://www.nature.com/articles/d41586-023-03854-1.

[11] P.-L. Cayrel, C. T. Gueye, O. Ndiaye, and R. Niebuhr. Critical attacks in code-based cryptography. *International Journal of Information and Coding Theory*, 3, 01 2015.

[12] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on Post-Quantum Cryptography. Technical Report NISTIR 8105, National Institute of Standards and Technology (NIST), 2016. https://csrc.nist.gov/publications/detail/nistir/8105/final.

[13] Y. Chen. Quantum Algorithms for Lattice Problems. Cryptology ePrint Archive, Paper 2024/555, 2024. https://eprint.iacr.org/2024/555.

[14] A. Childs. Lecture 2: The HSP and Shor's algorithm for discrete log. Lecture Notes for CO 781, University of Waterloo, 2008. Winter 2008.

[15] I. V. Chizhov and M. A. Borodin. The failure of McEliece PKC based on Reed-Muller codes. *IACR Cryptology ePrint Archive*, 2013:287, 2013.

[16] R. Cramer, L. Ducas, C. Peikert, and O. Regev. Short Stickelberger class relations and application to ideal-SVP. In *Advances in Cryptology - EUROCRYPT 2017*, volume 10211 of *Lecture Notes in Computer Science*, pages 486–517. Springer, 2017. https://doi.org/10.1007/978-3-319-56614-6_17.

[17] L. De Feo. Mathematics of Isogeny Based Cryptography. *Université de Versailles & Inria Saclay*, 2021. http://www.di.ens.fr/~defeo/.

[18] T. Debris-Alazard. Code-based Cryptography: Lecture Notes. 2023. https://arxiv.org/abs/2304.03541.

[19] Y. Dong, H. Liu, Y. Fu, X. Che, C. Liu, L. Sun, and G. Wen. Improving the Success Rate of Quantum Algorithm Attacking RSA Encryption System. *Research Square*, 03 2022.

[20] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehle. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. Cryptology ePrint Archive, Paper 2017/633, 2017. https://eprint.iacr.org/2017/633.

[21] W. Ebeling. *Lattices and Codes*. Springer, Berlin, Heidelberg, revised edition, 2002. Advanced Lectures in Mathematics, A course partially based on lectures by F. Hirzebruch.

[22] S. Edem, G. Vivek, and G. Rani. Role of Hash Function in Cryptography. In *International Journal of Advanced Engineering Research and Science (IJAERS)*, volume 3, pages 10–13, 01 2016. https://ijaers.com/detail/role-of-hash-function-in-cryptography/.

[23] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[24] A. Fraenkel and Y. Yesha. Complexity of problems in games, graphs and algebraic equations. *Department of Applied Mathematics, The Weizmann Institute of Science*, 30(3), 1981.

[25] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[26] T. Haner, M. Roetteler, and K. M. Svore. Factoring using 2n+2 qubits with Toffoli based modular multiplication. *Quantum Information and Computation*, 17(7&8):673–684, June 2017. https://doi.org/10.26421/QIC17.7-8-7.

[27] H. Imai and K. Kobara. Semantically Secure McEliece Public-Key Cryptosystems: Conversions for McEliece PKC. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 01)*, pages 19–35, Cheju Island, Korea, 2001.

[28] R. Impagliazzo and A. Wigderson. Derandomizing the XOR Lemma. *Journal of Computer and System Sciences*, 55(1):1–14, 1997. University of California and Hebrew University.

[29] M. F. Irène Marquez-Corbella, Nicolas Sendrier. MOOC on Code-Based Cryptography. https://learninglab.inria.fr/en/mooc-code-based-cryptography/.

[30] L. W. Jesko Hüttenhain. Topics in Post-Quantum Cryptography Lattice-Based Methods. *IEEE Transactions on Information Theory*, 57(1):402–417, January 18th, 2011.

[31] C. Kaufman, R. Perlman, and M. Speciner. *Network security: private communication in a public world, second edition*. Prentice Hall Press, USA, second edition, 2002.

[32] A. Y. Kitaev. Quantum measurements and the Abelian Stabilizer Problem. 2008. arXiv:quant-ph/9511026.

[33] C. Lavor, L. R. U. Manssur, and R. Portugal. Grover's Algorithm: Quantum Database Search. 2003. https://arxiv.org/abs/quant-ph/0301079.

[34] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[35] Y. Li, K. S. Ng, and M. Purcell. A Tutorial Introduction to Lattice-based Cryptography and Homomorphic Encryption, 2022. https://arxiv.org/abs/2208.08125.

[36] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A Modest Proposal for FFT Hashing! *Cryptology ePrint Archive, Report 2008/389*, 2008. https://eprint.iacr.org/2008/389.pdf.

[37] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010. https://doi.org/10.1007/978-3-642-13190-5_1.

[38] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Paper 2012/230, 2012. https://eprint.iacr.org/2012/230.

[39] W. Macharia. Cryptographic Hash Functions. *ResearchGate*, May 2021. https://www.researchgate.net/publication/351837904_Cryptographic_Hash_Functions.

[40] R. J. McEliece. A Public-Key Cryptosystem Based on Algebraic Coding Theory. *IEEE Transactions on Information Theory*, 24(6):709–710, 1978.

[41] L. Minder and A. Shokrollahi. Cryptanalysis of the Sidelnikov cryptosystem. In *EUROCRYPT 2007*, pages 347–360. Springer, 2007. https://infoscience.epfl.ch/record/55095.

[42] A. Mitra. What is the Diffie-Hellman Key Exchange Protocol and how does it work? *The Security Buddy*, 2017. https://www.thesecuritybuddy.com/encryption/how-does-diffie-hellman-key-exchange-protocol-work/.

[43] R. Niebuhr and P.-L. Cayrel. Broadcast attacks against code-based encryption schemes. In *Proceedings of the Workshop on Weaknesses in Elliptic Curve Cryptography (WEWORC)*, Weimar, Germany, 2011. https://www.researchgate.net/publication/224205562_Broadcast_Attacks_Against_Code-Based_Encryption_Schemes.

[44] H. Niederreiter and G. Solomon. Public key cryptosystems based on algebraic coding theory. *IEEE Transactions on Information Theory*, 32(6):793–800, 1986.

[45] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. 2010. Cambridge University Press, 10th Anniversary Edition.

[46] D. Pei, D. Hu, and W. Qi. *Multivariate Public Key Cryptosystems*. 2006.

[47] C. Peikert. A Decade of Lattice Cryptography, feb 2016. Published February 17, 2016.

[48] C. Peikert. Post-Quantum Cryptography, March 2022. Tutorial, QIP 2022, University of Michigan.

[49] A. Pellet-Mary. Algebraic lattices for cryptography. Foundations and Applications of Lattice-Based Cryptography Workshop, July 25–28 2022.

[50] A. Pellet-Mary. Lattice-based crypto, part 1: Algorithmic problems over lattices. In *Summer School in Post-Quantum Cryptography*, Budapest, 2022.

[51] A. Pellet-Mary. Lattice-based crypto, part 2: Protocols and structured cryptography. In *Summer School in Post-Quantum Cryptography*, Budapest, 2022.

[52] A. Pellet-Mary, D. Stehlé, and P. Zimmermann. Approx-SVP in Ideal Lattices with Pre-processing. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11477 of *Lecture Notes in Computer Science*, pages 141–171. Springer, 2019. https://doi.org/10.1007/978-3-030-17653-2_5.

[53] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[54] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[55] M. Richter. A (somewhat) gentle introduction to lattice-based post-quantum cryptography. *Cryptology ePrint Archive*, 2017:437, 2017. https://eprint.iacr.org/2017/437.

[56] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[57] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct. 1997. http://dx.doi.org/10.1137/S0097539795293172.

[58] V. M. Sidelnikov and S. O. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2:439–444, 1992.

[59] J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 2009.

[60] H. Singh. Code Based Cryptography: Classic McEliece. 2020. Scientific Analysis Group, Defence RD Organisation, Delhi – 110 054.

[61] C. Wieschebrink. Cryptanalysis of the Niederreiter public key scheme based on GRS subcodes. In *Post-Quantum Cryptography*, volume 6061 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2010.

[62] N. S. Yanofsky. *An Introduction to Quantum Computing*, pages 145–180. Springer Netherlands, Dordrecht, 2011. https://doi.org/10.1007/978-94-007-0080-2_10.