Breves Notas sobre
Codificación y Criptología

Jorge L. Ortega Arjona
Departamento de Matemáticas
Facultad de Ciencias, UNAM

Noviembre 2007

Índice general

1. Códigos de Corrección de Errores
Imágenes del Espacio **7**

2. Criptografía de Llave Pública
Secretos Insolubles **13**

3. La Teoría de Shannon
Códigos Elusivos **21**

Prefacio

Las *Breves Notas sobre Codificación y Criptología* introducen en forma simple y sencilla a algunos de los temas relevantes en el área de la Teoría de Códigos y Criptología. No tiene la intención de substituir a los diversos libros y publicaciones formales en el área, ni cubrir por completo los cursos relacionados, sino más bien, su objetivo es exponer brevemente y guiar al estudiante a través de los temas que por su relevancia se consideran esenciales para el conocimiento básico de esta área, desde una perspectiva del estudio de la Computación.

Los temas principales que se incluyen en estas notas son: Códigos de Corrección de Errores, Criptografía de Llave Pública y la Teoría de Shannon. Estos temas se exponen haciendo énfasis en los elementos que el estudiante (particularmente el estudiante de Computación) debe aprender en las asignaturas que se imparten como parte de la Licenciatura en Ciencias de la Computación, Facultad de Ciencias, UNAM.

Jorge L. Ortega Arjona
Noviembre 2007

Capítulo 1

Códigos de Corrección de Errores

Imágenes del Espacio

Una sonda espacial se dirige a un planeta distante. En algunos días, pasará cerca de la superficie del planeta, y tomará varios cientos de fotografías, registrándolas internamente. Cuando se dirige más allá, en el espacio, comienza a transmitir las imágenes a estaciones receptoras en la Tierra, a varios millones de kilómetros de distancia.

Cada fotografía se divide en líneas horizontales, como una imagen de televisión, y cada línea se divide en pixeles (*picture elements*) representando una de las 32 posibilidades de escala de grises – el blanco es 0, el negro 31, y los valores intermedios son tonalidades de gris.

Cada pixel se codifica en un mensaje que se envía de regreso a la Tierra. Aun cuando viaja a gran velocidad, la señal puede requerir algún tiempo para recorrer la enorme distancia. Y durante este viaje, puede "ensuciarse". Cuando alcanza la antena receptora, la debilidad de la señal puede hacer que se le confunda con ruido de fondo.

De tal modo, cuando un pixel individual, codificado como un patrón digital de ceros y unos, alcanza la estación, su forma original, por ejemplo:

$$01001100011101010101111001010101$$

puede haber cambiado a:

$$010\mathit{1}1110\mathit{1}01110\mathit{00}0101\mathit{0}11001\mathit{1}101\mathit{1}1$$

donde los caracteres en cursiva indican los errores que han ocurrido en el mensaje. Se podría pensar que 5 bits de información serían suficientes para especificar uno de los 32 niveles de gris posibles. Sin embargo, la cadena anterior tiene 32 bits y no 5. Es parte de un código diseñado para hacer tantos errores corregibles.

De hecho, la cadena es parte del código utilizado en las series Mariner de los vehículos de reconocimiento en Marte. Este código es esencialmente el código (32,5) Reed-Muller, definido como sigue:

Sea:

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

y se define:

$$H_{n+1} = H_n \otimes H_1$$

donde $\otimes$ denota el producto cartesiano de matrices: reemplace cada elemento $+1$ de H_n por H_1, y cada elemento -1 por $-H_1$. De tal modo, esta ecuaciones definen las *matrices Hadamard*.

El código de telemetría del Mariner consiste de renglones de M_5, que se obtienen de reemplazar 1 por 0 y -1 por 1 en H_5. De tal modo, la matriz M_5 se muestra efectivamente como una malla cuadrada en la que los unos se representan por cuadrados negros y los ceros por cuadrados blancos, como se muestra en la figura 1.1.

Los renglones de M_5 representan las 32 palabras del código posibles a ser transmitidas por el Mariner, y tienen una propiedad muy interesante: cualquier par de renglones difieren exactamente en 16 posiciones. Por ejemplo, si se comparan los renglones segundo y tercero, se nota que en las siguientes posiciones un renglón tiene un 0, mientras que el otro tiene un 1, o viceversa:

2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31

Cuando se recibe un nuevo código de 32 bits, la computadora los compara con los renglones de M_5 y el renglón que más se le parezca es seleccionado

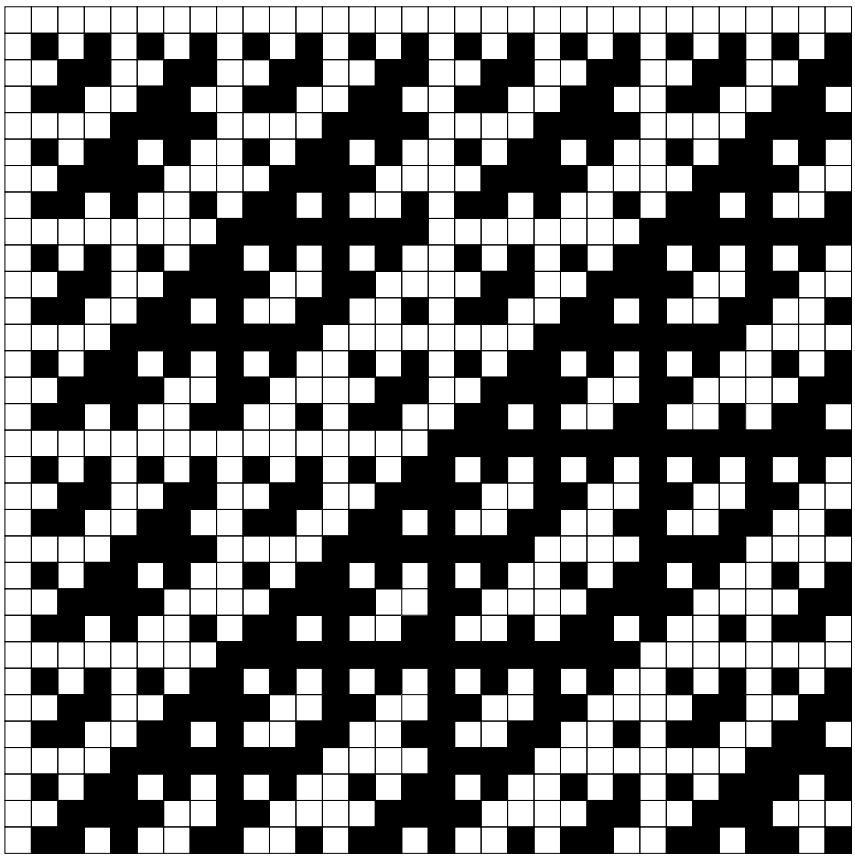

Figura 1.1: El código Reed-Muller como M_5, un patrón blanco y negro

como la palabra transmitida. Naturalmente, si solo ha habido un error en la palabra, difiere entonces de los renglones de M_5 en solo un dígito, y no puede confundirse con ningún otro renglón. Si suceden dos errores, tal afirmación se mantiene. De hecho, pueden ocurrir hasta siete errores sin peligro de confusión acerca de cuál palabra fue transmitida por el Mariner.

Pero si ocurren ocho errores, entonces la palabra recibida puede diferir tanto de otro renglón de M_5 como del renglón que le corresponde. Esto se debe al hecho de que esos dos renglones difieren sólo en 16 posiciones, y los ocho errores pueden haber resultado en una palabra que está tan "cerca" de un renglón incorrecto de M_5 como del renglón correcto.

El número de posiciones en las que dos palabras binarias o vectores difieren se conoce como *distancia de Hamming* entre ellos. Un conjunto de palabras del código que están mutuamente a una distancia de Hamming d o más permite a los usuarios detectar y corregir hasta $\lfloor (d-1)/2 \rfloor$ errores. La distancia de Hamming del código de telemetría del Mariner es 16, de tal modo que hasta siete errores pueden detectarse y corregirse.

Todo esto aún deja abierto precisamente cómo cada pixel se codifica como uno de los renglones de M_5. Ciertamente, el nivel de grises que representa al pixel puede estar codificado como un número binario de 5 bits. Sería tentador solo transmitir ese patrón de ceros y unos, pero los errores podrían dañar fuertemente la información: sería simplemente imposible saber cuáles fueron los bits originalmente transmitidos.

De tal modo, el nivel de grises x, se genera a bordo como un número binario

$$(x_4, x_3, x_2, x_1, x_0)$$

que se codifica como el k-ésimo renglón de M_5, donde k es el número binario con esos bits. En Tierra, como ya se ha considerado, cada palabra recibida se compara contra los renglones de M_5, y la mejor se selecciona y su correspondiente vector de 5 bits se genera y escribe en un archivo. Cuando las señales marcan que el final de una imagen se ha recibido, el archivo se lee mediante una unidad de despiegue gráfico para su inspección visual.

El proceso de decodificación es realmente mucho más complicado que esto porque la velocidad de envío de mensajes de la sonda se encuentra sobre los 16000 bits/s. La computadora en Tierra debe por lo tanto trabajar muy rápido para transformar esta información a los 500 pixels por segundo re-

sultantes. De hecho, la carga de este trabajo y posteriores misiones Mariner se realizan en una computadora de propósito especial que realiza una versión discreta de la transformada rápida de Fourier (*Fast Fourier Transform* ó FFT).

El código descrito se le llama "de corrección de errores" (*error-correcting*) porque permite que el receptor de un mensaje de tal modo codificado pueda corregir (hasta un cierto número de) errores que pueden haber ocurrido en el mensaje durante su largo viaje por el espacio. Un código de "detección de errores" (*error-detecting*) es aquél que permite al receptor detectar (un cierto número de) errores. Ambos códigos pueden entenderse en términos de una distancia de Hamming. Si la distancia de Hamming entre una palabra transmitida y la misma palabra cuando es recibida es h, entonces han ocurrido h errores. Si la distancia de Hamming entre dos palabras codificadas (como se transmiten) es siempre d o más, sin embargo, entonces h puede ser tan grande como $d-1$, y el receptor aún sabe que han ocurrido errores. Si exactamente ocurren d errores, entonces es posible que una palabra del código haya corrompido a otra.

Si se detectan así los errores, ¿qué puede hacer el receptor? El receptor simplemente selecciona la palabra código en la lista de palabras que se acerque más a la palabra recibida en términos de la distancia de Hamming. La selección es la correcta si no han ocurrido mas de $\lfloor (d-1)/2 \rfloor$ errores.

Capítulo 2

Criptografía de Llave Pública
Secretos Insolubles

La secrecía en las comunicaciones es una necesidad no solamente de militares y agentes de inteligencia, sino de muchas empresas comerciales e industriales que dependen de información confiable. Hoy por hoy, probablemente el 90 % de la información sensitiva o secreta se genera en estos sectores. Tan solo imagínese el flujo de información crediticia entre las varias instituciones financieras.

Las enormes cantidades de información sensitiva o secreta hacen imperativo que la confidencialidad sea garantizada por sistemas de encripción computacionalmente factibles, y sin embargo, muy difíciles de desencriptar cuando se intercepta la información.

Tradicionalmente, el proceso de encriptado involucra dos algoritmos de transformación T y T^{-1} que operan sobre un mensaje. Por ejemplo, en la figura 2.1 el mensaje X puede ser un enunciado en lenguaje natural, y la transformación T puede ser la substitución por cada letra del alfabeto de la k-ésima letra después de esa misma (cuando se alcanza el fin del alfabeto, se añade el caracter en blanco, y se recomienza con la A). Por ejemplo, con $k = 4$, la palabra SECRET se transforma en WIGVIX. Si el receptor y el emisor del mensaje saben ambos el algoritmo y la llave k, entonces cifrar y decifrar el mensaje son operaciones bastante sencillas, especialmente para una computadora. Si el algoritmo se vuelve de conocimiento público, sin embargo, no le toma mucho tiempo a un criptoanalista descubrir la llave k una vez que un mensaje particular ha sido interceptado: se requiere meramente intentar $k = 1, 2, 3, \ldots$ hasta que la transformación inversa (substituir

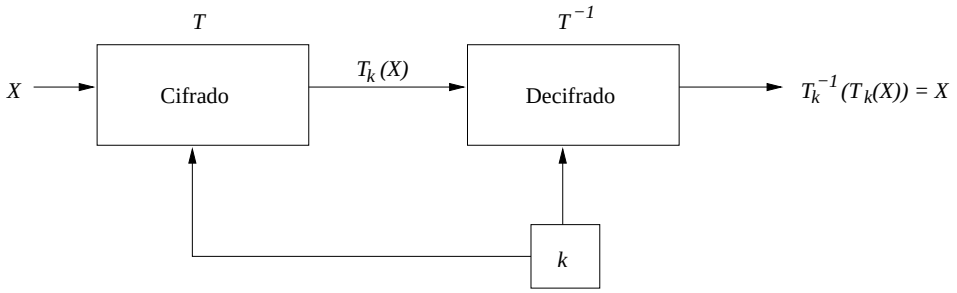

Figura 2.1: Un sistema tradicional de cifrado

la k-ésima letra previa a la que se decifra) arroja un mensaje intelegible. En este caso en particular, aun cuando el algoritmo de encripción no fuera público, no le tomaría mucho tiempo a un criptoanalista "romper el código". Sin embargo, se dispone de algoritmos de encripción mucho más elaborados para X, transformaciones que son tan complicadas que aun cuando se hagan públicas, conocer la llave resulta esencial para decifrar el mensaje. Así, los sistemas tradicionales de encripción requieren que las llaves se distribuyan (usualmente, por mensajería) a los varios usuarios del sistema. Con cambios frecuentes en las llaves, esto puede representar un alto cosumo de tiempo y dinero.

En la criptografía de llave pública, se le da a cada usuario del sistema una llave k que no requiere cambiarse nunca, y puede listarse en un directorio público. El receptor del mensaje en este tipo de sistemas (véase la figura 2.2), sin embargo, cuenta con una llave privada k' que se utiliza para implementar la transformación inversa T^{-1}. Naturalmente, existe una relación entre k y k', y de hecho, tal relación puede revelarse sin comprometer la seguridad de k'. La idea esencial es que k' es muy difícil de generar dada una k. Pero k es muy sencilla de obtener a partir de k', tan fácilmente que en cuestión de minutos es posible obtener una llave pública k para un nuevo usuario del sistema.

Uno de los primeros sistemas con llaves públicas criptográficas fue desarrollado por Martin Hellman, Ralph Merkle y Whitfield Diffie en la Universidad de Stanford, a finales de los 1970s. Se basaba en un problema computacionalmente difícil llamado el Problema de la Suma del Subconjunto (*subset-sum problem*):

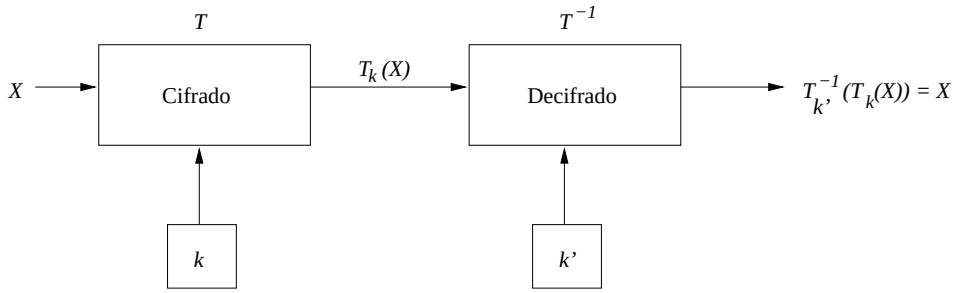

Figura 2.2: Un sistema de llave pública

> Dados $n+1$ enteros positivos $a_1, a_2, \ldots, a_n$ y B, encontrar un subconjunto de a_i que sumado de como resultado B.

Por ejemplo, supóngase que se tienen $n = 5$ enteros 5, 8, 4, 11 y 6, y que $B = 20$. ¿Se puede encontrar un subconjunto de los cinco enteros cuya suma sea 20?

Otra forma de ver este problema es imaginar que los n enteros representan las alturas de bloques que entran de forma justa en una caja con altura B (figura 2.3). ¿Se puede encontrar un subconjunto de bloques e exactamente llene la caja?

En este ejemplo, no toma mucho tiempo tras intentar algunas combinaciones, descubrir que $5+4+11 = 20$. En general, sin embargo, nadie conoce un algoritmo que garantice resolver este problema en tiempo polinomial. De hecho, se trata de un problema NP-completo. Así, cualquier algoritmo que garantice que finalmente llegará al resultado, debe intentar una fracción significativa de las 2^n posibles combinaciones de enteros para descubrir qué subconjunto (si lo hay) suma B.

He aquí cómo el sistema criptográfico basado en este problema funciona: supóngase que un usuario tiene alguna información confidencial para transmitir al receptor. Al usuario se le ha dado una llave pública que consiste de n enteros $a_1, a_2, \ldots, a_n$. La información se transmite como una cadena de dígitos binarios, que se dividen en bloques de longitud n. El bloque $x = (x_1, x_2, \ldots, x_n)$ se mapea al entero:

$$B_x = \sum_{i=1}^{n} x_i a_i$$

y se transmite.

Para hacer el proceso de encripción más concreto, supóngase que el mensaje es SECRET y que se codifica con ASCII de 7 bits:

S	E	C	R	E	T
1010011	1000101	1000011	1010010	1000101	1010100

Por conveniencia en este ejemplo se toma $n = 7$ (aun cuando pueden y de hecho se usan valores mucho más grandes), y por lo tanto se cifra la primera letra S como:

$$1 \times (901) + 0 \times (568) + 1 \times (803) + 0 \times (39) + 0 \times (450) + 1 \times (645) + 1 \times (1173)$$

donde (901, 568, 803, 39, 450, 645, 1173) es la llave pública $(a_1, a_2, \ldots, a_n)$ y (1,0,1,0,0,1,1) el bloque x que se cifra. El entero resultante $B_x = 3522$ se transmite sobre un canal inseguro. Cualquiera que quiera conocer qué mensaje está encriptado en 3522, aun cuando sepa que involucra algunas combinaciones de las llaves públicas, tendrá que intentar una fracción de las $2^7 = 128$ posibles combinaciones antes de obtener la correcta. Si n se hace

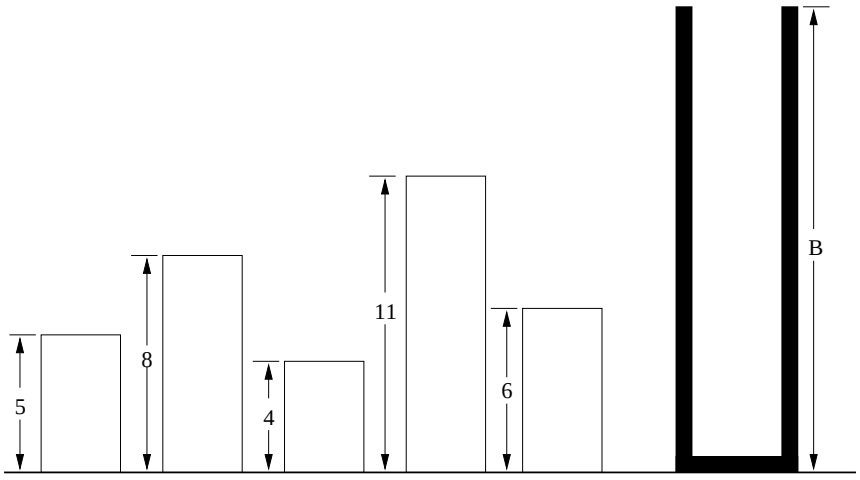

Figura 2.3: El Problema de la Suma del Subconjunto

mucho más grande que 7, el tiempo para hacer esto se vuelve prohibitivo. ¿Porqué dedicar 100 años de tiempo de cómputo para decifrar una pieza de información que, aunque es secreta hoy, será irrelevante cuando ya se haya decifrado? De hecho, con muy poco incremento de n, el interceptor del mensaje puede enfrentar un problema que puede llevar el tiempo de vida del Universo en resolver.

Ahora bien, cuando el mensaje B_x llega a su destino, el receptor usa una llave privada $(a'_1, a'_2, \ldots, a'_n)$ y dos enteros especiales w y m para decifrar el mensaje.

La llave pública se obtiene originalmente en base de la información privada como:

$$a_i = w \cdot a'_i \, mod \, m$$

Para recobrar los bits del mensaje x_i, el receptor formula una versión especial del Problema de la Suma del Subconjunto:

Cuál subconjunto de $(a'_1, a'_2, \ldots, a'_n)$ suma B'_x, donde:

$$B'_x = B_x \cdot w^{-1} \, mod \, m$$

Aquí, w^{-1} es el inverso de w en el campo de los enteros módulo m, es decir $w \cdot w^{-1} \equiv 1 \, mod \, m$.

La solución del Problema de la Suma del Subconjunto en el lado receptor es propiamente la transformación inversa T^{-1}. Por tanto, si la llave privada del receptor es (1, 2, 5, 11, 32, 87, 141), y considerando los enteros especiales $w = 901$ y $m = 1234$, entonces el problema es descubrir qué subconjunto de los enteros de la llave privada suma a:

$$\begin{aligned} B'_x &= 3522 \times (901)^{-1} \, mod \, 1234 \\ &= 3522 \times 1171 \, mod \, 1234 \\ &= 234 \end{aligned}$$

Este problema resulta ser resuelto de forma fácil y rápida. Los enteros en el receptor se han arreglado de modo que cada entero es mayor que la suma de los enteros que lo preceden en la secuencia. No es difícil notar que el siguiente algoritmo resuelve el problema en tiempo lineal:

SUMSUB

1. $sum \leftarrow 0$
2. **for** $i = n$ **step** 1 **until** 1 **do**

   a) **if** $a_i + sum \leq B'_x$
   then $sum \leftarrow sum + a_i$, $subset(i) \leftarrow 1$
   else $subset(i) \leftarrow 0$

3. **if** $sum = B'_x$ **then** output *subset*
   else output "fail"

Este algoritmo inicia con los enteros más grandes (más a la derecha) de la llave privada, tratando de ajustarlos dentro de la caja (siguiendo el ejemplo anterior de los bloques y la caja), y eliminando aquéllos bloques que no se ajusten. Funciona debido a que cada a_i es mayor que la suma de los enteros que le siguen en orden derecha a izquierda.

Aplicando *SUMSUB* para el ejemplo (1, 2, 5, 11, 32, 87, 141) y $B'_x = 234$, se encuentra rápidamente el subconjunto (1,0,1,0,0,1,1). Evidentemente, este representa la cadena ASCII 1010011 para la S, la primera letra del mensaje SECRET.

¿Porqué hasta aquí todo funciona tan bien? Quizá el álgebra de T y T^{-1} proporcione una simple explicación:

$$\begin{aligned} B'_x &= \sum_{i=1}^{n} x_i a_i \\ &= \sum_{i=1}^{n} x_i \cdot wa'_i \bmod m \end{aligned}$$

Pero

$$\begin{aligned} B'_x &= B_x \cdot w^{-1} \bmod m \\ &= \sum_{i=1}^{n} x_i \cdot wa'_i w^{-1} \bmod m \\ B'_x &= \sum_{i=1}^{n} x_i a'_i \end{aligned}$$

Es decir, el mensaje x codificado en B_x como $\sum_{i=1}^{n} x_i a_i$ resulta codificarse en ${B'}_x$ como $\sum_{i=1}^{n} x_i {a'}_i$.

Los códigos de llave pública tienen solo dos posibles vulnerabilidades, más allá de descubrir la llave privada:

- Un algoritmo que resuelva problemas NP-completos de forma rápida.
- Un algoritmo que resuelva el problema particular (NP-completo) en el cual se base un sistema criptográfico.

La primera vulnerabilidad se considera muy improbable por expertos en Teoría de la Complejidad, pero la segunda puede suceder si el problema particular no tiene las salvedades teóricas adecuadas. Esto es precisamente lo que sucedió con el sistema criptográfico de Hellman-Merkle-Diffie que se describe anteriormente. Resulta que aun cuando el problema "público" de la Suma del Subconjunto parecía ser una versión general, incluyendo aquéllos casos en que presumiblemente hacían al problema computacionalmente insoluble, no lo era del todo en realidad. Adi Shamir, del MIT, descubrió que era mucho más un caso especial (más como el Problema de la Suma del Subconjunto con la llave privada) que lo que sus diseñadores creyeron. Shamir encontró un algoritmo de tiempo polinomial que resolvía hasta la versión pública del Problema de la Suma del Subconjunto.

En los 1980s, un nuevo sistema criptográfico de llave pública substituyó al esquema Hellman-Merkle-Diffie. Conocido como criptosistema RSA (acrónimo de Rivest, Shamir y Adleman), este sistema depende de la insolubilidad aparente en tiempo polinomial del problema de la factorización. Si un número de n bits es no primo, ¿cuánto le debe tomar a una computadora para resolverlo en un número polinomial de pasos? Si, como muchos piensan, no puede resolverse en un número polinomial de pasos, entonces el criptosistema tiene oportunidad de tener éxito.

En RSA, dos enteros e y n se dan como llaves públicas. Si un mensaje m se convierte a un entero menor que n, entonces m puede encriptarse de acuerdo a la fórmula:

$$c = m^e \, mod \, n$$

El receptor cuenta con una llave privada que consiste de dos factores primos p y q de n. Es decir, $n = pq$. Esto significa que el receptor del mensaje puede fácilmente encontrar un entero d tal que:

$$ed = 1 \, mod \, \phi(n)$$

donde $\phi(n) = (p-1)(q-1)$. Para decifrar el mensaje c, el receptor obtiene $c^d \, mod \, n$. Este número puede analizarse como sigue:

$$\begin{aligned} c^d \, mod \, n &= m^{ed} \, mod \, n \\ &= m^{kQ(n)+1} \, mod \, n \qquad \text{(para algún entero } k\text{)} \\ &= m \cdot m^{kQ(n)} \, mod \, n \\ &= m \, mod \, n \qquad \text{(ya que } m^{kQ(n)} = 1 \, mod \, n\text{)} \\ &= m \qquad \text{(ya que } m < n\text{)} \end{aligned}$$

Ya que n puede ser elegido con factores arbitrarios p y q, el problema de la factorización que explota el criptosistema RSA es completamente general y no abierto a un ataque especial, como el esquema anterior. Sin embargo, continúan realizándose esfuerzos para obtener algoritmos de factorización rápida. Hasta ahora, uno de ellos requiere:

$$O(e^{(log \, n \, log \, log \, n)1/2}) \text{pasos}$$

La cuestión del impacto de la investigación teórica sobre los criptosistemas de llave pública ha hecho que ciertas autoridades que dependen de tales sistemas se pongan entendiblemente nerviosas. En 1981, la Agencia Nacional de Seguridad de los Estados Unidos solicitó un estudio acerca del asunto.

Capítulo 3

La Teoría de Shannon
Códigos Elusivos

Cualquier mensaje que consiste de palabras puede codificarse como una secuencia de ceros y unos. Estos símbolos pueden transmitirse por algún medio: alambre, ondas de radio, u otros. En todos los casos, el mensaje está sujeto a corrupción. Un cero puede inadvertidamente cambiar a un uno y viceversa. Lo que sea la fuente de interferencia o "ruido", como los teóricos de la información le llaman, es decir, la tendencia de un bit a cambiar, puede deberse a lo que algunos autores llaman un "demonio de ruido". Con cierta probabilidad p, el demonio altera cada bit que se transmite.

Una forma de engañar al demonio es transmitir tres ceros por cada cero que se pretende transmitir, a la vez de transmitir tres unos por cada uno que se desea enviar. Suponiendo que el receptor está sincronizado con el emisor, de modo que el principio de cada triada es conocido, la regla de decodificación es muy simple, como se muestra en la siguiente tabla:

Recibido	**Significa**
000	0
100, 010, 001	0
011, 101, 110	1
111	1

¿Cuál es la probabilidad de que un mensaje bajo este esquema se corrompa? Es la probabilidad de que al menos dos bits de cada tres cambien. Por tanto, si 000 se envía, puede volverse 110, 101, 011 ó 111 con las respectivas

probabilidades p^2q, pqp, qp^2 ó p^3, donde $q = 1 - p$. Sumando estas probabilidades arroja la fórmula $3p^2 - 2p^3$, que significa que si p es menor que 5, el nuevo esquema de tripletas garantiza una probabilidad mucho más baja que el éxito del demonio. Por ejemplo, si $p = 0{,}1$ entonces la probabilidad de que el demonio corrompa el mensaje es de 0.32.

El código anterior utiliza simplemente redundancia para eliminar errores. Se encuentran disponibles otros métodos más sofisticados. Por ejemplo, se pueden agrupar los bits del mensaje en pares y transmitir el par junto con otros bits extra de verificación, de acuerdo con el siguiente esquema:

Bits del mensaje a_1a_2	**Bits de verificación** a_3a_4
0 0	0 0
0 1	1 1
1 0	0 1
1 1	1 0

El primer bit de verificación a_3 simplemente repite el segundo bit del mensaje a_2: $a_3 = a_2$. El segundo bit de verificación se le conoce como bit de paridad (*check sum*). Se refiere a la suma lógica de los primeros dos bits: $a_4 = a_1 \oplus a_2$.

No hay garantía de que para cualquier esquema de codificación, el demonio falle. Si tiene éxito en cambiar suficientes bits, no hay esquema de decodificación que pueda recuperar el mensaje original. Por tal razón, el mensaje se decodifica por un método de máxima similaridad (*maximum-likelihood*). Por cada cadena recibida de cuatro bits, ¿cuál es la interpretación más similar de los dos primeros bits? El siguiente algoritmo toma cuatro bits recibidos b_1, b_2, b_3 y b_4 como entrada, y da como salida b_1 y b_2, alterados de acuerdo con los errores probables detectados por el algoritmo:

1. **input** b_1, b_2, b_3, b_4
2. **if** $b_4 \neq b_1 \oplus b_2$ **then**
   a) **if** $b_3 \neq b_2$ **then** $b_2 \leftarrow \bar{b_2}$
   b) **else** $b_1 \leftarrow \bar{b_1}$
3. **output** b_1, b_2

La operación del algoritmo de decodificación se muestra en el diagrama de hipercubo de la figura 3.1. Cada vértice representa una cadena posible de 4 bits a recibirse. Cuatro de los vértices se han marcado. Estos representan las cuatro palabras probablemente sin corrupción.

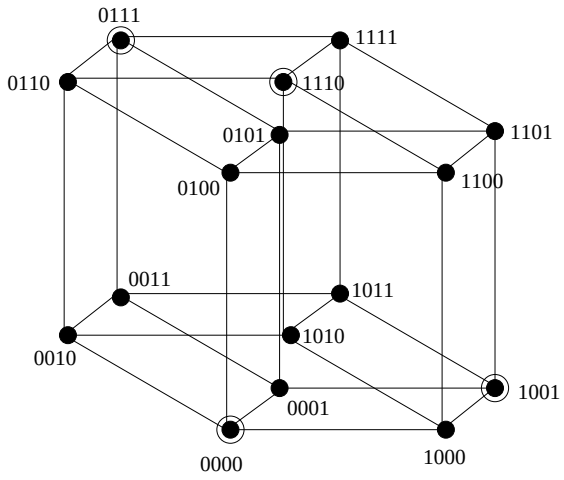

Figura 3.1: Un código de detección de errores en un hipercubo.

Los vértices representan cadenas que difieren en un solo bit, y que se conectan mediante un arco. Si se analiza la acción del algoritmo en cada una de las 10 posibles palabras recibidas, se descubre que cada palabra es reinterpretada como la palabra marcada más cercana. El concepto de distancia que se aplica aquí es la distancia de *Hamming*, es decir, el mínimo número de arcos que deben atravesarse para ir de un vértice (palabra posible) a otro. La distancia de Hamming es claramente el número de bits alterados en el proceso de comunicación. La decodificación por máxima similaridad selecciona para cada posible palabra en el mensaje, a la palabra marcada con la mínima distancia de Hamming a partir de la palabra del mensaje.

El esquema de codificación y decodificación descrito hasta aquí permite al usuario evitar todos los errores, excepto dos: dos vértices en el diagrama tienen distancia de Hamming 1 de las palabras marcadas. Es por lo tanto posible engañar al algoritmo cuando el demonio crea un solo error en el mensaje. Pero la probabilidad de que esto suceda es $p/6$ ya que sólo 2 de los 12 mensajes que contienen un solo error puede malinterpretarse. En algunos casos cuando dos o más errores ocurren, el algoritmo puede todavía recupe-

rar el mensaje original. De esto, sigue que la probabilidad para engañar al algortimo es en realidad menor que $p/6$. Cuando $p = 0{,}1$, tal probabilidad se vuelve algo así como 0.016, lo que es substancialmente mejor que el esquema anterior.

En el primer ejemplo, se envía un mensaje de un bit y dos bits de verificación. La probabilidad de fallar por el demonio del ruido es de $3p^2 - 2p^3$. En el segundo ejemplo, se envía un mensaje de dos bits con dos bits de verificación. La probabilidad de un error irrecuperable es menor que $p/6$ más la probabilidad de dos o más errores, es decir, $3p^2 - 2p^3$. Pero en el segundo caso dos bits de mensaje se transmiten, y por lo tanto, la probabilidad de recuperación por bit es menor que $p/12 + 3p^2/2 - p^3$. En caso de que $p = 0{,}1$, esto tiene un valor de 0.0198, que es definitivamente una mejora.

Hay claras ventajas de enviar palabras codificadas que son relativamente más largas. Un teorema fundamental descubierto por Claude Shannon en los laboratorios Bell, en 1948, hace esta ventaja explícita. Desafortunadamente, el teorema no da ninguna clave de cómo construir códigos que exploten tal ventaja.

En lo que sigue, se supone que los códigos bajo discusión tienen palabras de longitud n. El demonio modifica los mensajes con una probabilidad uniforme de p por bit. Supóngase que el código C tiene m palabras $X_1, X_2, \ldots, X_m$ y que el algoritmo para la decodificación de máxima similitud incorrectamente decodifica la palabra X_i con probabilidad p_i. Entonces la probabilidad de decodificar incorrectamente una palabra promedio en C es:

$$\frac{1}{m} \sum_{i=1}^{m} P_i$$

Se denota como $P(n)$ a la mínima de tales probabilidades sobre todos los códigos C de tamaño m y longitud n. El Teorema fundamental de Shannon describe condiciones bajo las cuales esta probabilidad de acerca a cero conforme n se aproxima al infinito. Especificamente, la tasa de información (*information rate*) r de un código es simplemente $log_2 m$ (la cantidad de información incorporada al conjunto de todas las palabras del código) dividido por la longitud n del código. Si r es mayor que cero y menor que $p \log p + q \log q$ y si $m = 2^{nr}$ entonces:

$$P(n) \to 0 \text{ conforme } n \to \infty$$

Es decir, el teorema de Shannon especifica que para cualquier número positivo pequeño ϵ, hay una longitud de palabra n y un código C tal que la probabilidad de fallar en la decodificación es menor que ϵ. Más aún, el teorema es generoso en su permisibilidad de palabras: el número disponible es exponencial en n.

La demostración del Teorema de Shannon es algo larga y técnica, pero la idea esencial es fácilmente descriptible. La demostración procede mediante seleccionar m palabras del código aleatoriamente del conjunto de todas las palabras binarias de longitud n. Este conjunto constituye una clase de espacio en el que la distancia entre dos palabras es la distancia de Hamming. En la demostración del Teorema de Shannon, cada palabra aleatoriamente seleccionada se rodea de una "esfera" de radio ρ. Este es tan solo todo el conjunto de palabras de n bits que tienen una distancia de Hamming ρ o menor del código de la palabra que actúa como centro de la esfera.

La regla de decodificación usada en la demostración es una forma estricta de máxima similitud. Cuando se recibe una palabra, se encuentra la palabra que esté más cerca en términos de la distancia de Hamming. Si la palabra recibida cae dentro de ρ de la palabra, se selecciona ésta última como el mensaje. Si ρ se escoge para ser un cierto polinomio simple en n, resulta que la probabilidad de una palabra recibida cayendo fuera de su esfera ρ es arbitrariamente pequeño, al menos para n lo suficientemente grande. Así, la probabilidad de una falla en la decodificación por el método de máxima similitud es al menos pequeño: tiende a 0 conforme n tiende a infinito.

Pero, ¿porqué, entonces, es tan difícil encontrar buenos códigos? ¿porqué no escoger uno aleatoriamente? La respuesta recae en el tamaño que n debe tener antes de que la probabilidad de falla deseada ϵ se alcance. Es demasiado grande para ser práctica. Al mismo tiempo, la demostración del Teorema de Shannon sólo abarca el código promedio en el espacio de palabras de n bits. Meramente garantiza que se puede hacer lo posible hasta lograr ϵ con n bits. Quizá una n mucho menor es suficiente. Frecuentemente, así es, pero los códigos más cortos deben encontrarse por otros métodos que continúan interesando a los teóricos.

Bibliografía

[1] B. Bosworth. *Codes, Ciphers, and Computers: An Introdution to Information Security.* Hayden, Rochelle Park, NJ, 1980.

[2] D.E.R. Denning. *Cryptography and Data Security.* Addison-Wesley, 1983.

[3] R.W. Hamming. *Coding and Information Theory.* Prentice-Hall, 1980.

[4] E.C. Posner. *Combinatorial Structures in Planetary Reconnaissance.* Error Correcting Codes (H.B. Mann, ed.) Wiley, New York, 1969.

[5] W.W. Peterson and E.J. Weldon, Jr. *Error-Correcting Codes.* MIT Press, 1981.