# Una breve comprensión de la Criptografía
## Conceptos y aplicaciones.

Marcos Daniel Calderón Calderón

Centro de Investigación en Matemáticas
marcos.calderon@cimat.mx

**Resumen** En este documento se hace un pequeño análisis de la Criptografía.

## 1. Capítulo2. Cifrado de flujo.

Este capítulo da una introducción a los cifrados de flujo:

- Los pros y los contras de los cifrados de flujo.
- Generadores de números aleatorios y pseudoaleatorios.
- Un cifrado irronplible: la libreta de un solo uso (OTP).
- Registros lineales de retroalimentación en turno y Trivium, un cifrado de flujo moderno.

### 1.1. Introducción.

**Comparación de los cifrados de bloque con los cifrados de flujo.** La criptografía simétrica se divide en cifrados de bloque y cifrados de flujo, los cuales son fáciles de distinguir. A continuación se presenta una breve descripción de cada uno de los dos tipos de criptografía simétrica.

**Cifrado de flujo.** Se encarga de encriptar bits de manera individual. Esto se logra mediante la adición de un bit del flujo que representa a la llave a un bit de texto plano. Hay cifrados de flujo sincronizados donde el flujo de la llave depende únicamente de la llave, el cifrado asíncrono es aque donde el flujo de la llava también depende del texto cifrado.

Ejemplo 1 En este ejemplo, se aplicará un proceso de encriptado y desencriptado por medio del esquema de cifrado de flujo. El problema tiene los siguientes componentes:

$$Caracter \quad a \quad encriptar: \quad A(65_{10}) = 1000001_2$$

$$Flujo \quad de \quad llave: \quad (s_0, ...s_6) = 0101100_2$$

Los pasos para el proceso de cifrado y descifrado del ejemplo anterior son los siguientes:

- Aplicamos la operación XOR a las cadenas de bits involucradas, el resultado obtenido es el siguiente: $m = 1101101_2$
- Podemos pensar que $m$ es el mensaje cifrado que se va a enviar al receptor.
- Una vez que el receptor recibe el mensaje, su tarea es descifrar el texto cifrado, para hacerlo, vuelve a aplicar la operación XOR al mensaje recibido y con el flujo de llave $(s_0, ...s_6)$, el resultado obtenido es igual al valor inicial cifrado: $1000001_2 = A$

**Cifrado de bloque.** Encripta un bloque entero de los bits del texto plano al mismo tiempo con la misma llave. Esto quiere decir que la encriptación de cualquier bit del texto plano en un bloque dado, depende de los demás bits del texto plano que se encuentren en el mismo bloque. En la práctica, la inmensa mayoría de los cifrados de bloque tienen una longitud de bloque de 128 bits (16 bytes) como el advanded encryption standard (AES), o una longitud de bloque de 64 bits (8 bytes) como el data encription standard (DES) o el algoritmo triple DES (3DES).

A continuación, mencionamos los pros y los contras de los cifrados de bloque contra los cifrados de flujo.

- En la práctica, y en particular para encriptación de la comunicación en internet, el cifrado de bloque es utilizado con mayor frecuencia que los cifrados de flujo.
- Como los cifrados de flujo tienen a ser pequeños y rápidos, estos son particularmente relevantes para aplicaciones con pocos recursos computacioneales, por ejemplo, teléfonos celulares y otros dispositivos embebidos. Un ejemplo de este cifrado es el método A5/1, el cual es parte de el estándar para teléfonos móviles GSM y es utilizado para encriptación de voz. También, los cifrados de flujo algunas veces son utlizados para encriptar tránsito en internet, por ejemplo el método RC4.
- Tradicionalmente, se ha hasumida que los cifrados de flujo tienen a encriptar de una manera más eficiente que los cifrados de bloque. Eficiencia en sofware para cifrado de flujo significa que son necesaria pocas instrucciones del procesador para encriptar un bit de texto plano. Eficiencia en hardware para cifrado de flujo significa que se necesita un espacio más pequeño de chip que los cifrados de block para encriptar el mismo rango de datos. De cualquier manera, cifrados de bloque modernos como el AES también son muy eficientes en software. También en hardware, hay cifrados de bloque altamente eficientes, que son tan buenos como los cifrados de flujo más compactos.

**Encriptación y decriptación de los cifrados de flujo.** Como se ha mencionado arriba, cifrados de bloque encriptan bits del texto plano de manera individual. La cuestión ahora es, ¿Cómo trabaja la encriptación de un bit individual? La respuesta es sorprendentemente simple: Cada bit $x_i$ es encriptado añadiendo una flujo de llave secreta $s_i$ módulo 2.

Definición 1 (Encriptación y Decriptación del cifrado de flujo.)
El texto plano, el texto cifrado y el flujo de la llave consisten de bits individuales, esto es: $x_i, y_i, s_i \in 0, 1$.
**Encriptación:** $y_i = e_{s_i} \equiv x_i + s_i mod 2$
**Decriptación:** $x_i = d_{s_i} \equiv y_i + s_i mod 2$

Podemos notar los siguientes aspectos:

- Encriptacion y decriptacion son las mismas funciones.
- Se una una adición módulo 2 como paso de encriptación.
- La naturaleza de los flujos de bits $s_i$.

A continuación, discutimos cada uno de los aspectos mencionados.

**¿Porqué la encriptación y la decriptación son la misma función?** La razón de que los dos procesos estén representados por la misma función puede ser mostrada fácilmente. Se debe probar que la función de decriptación produce el bit $x_i$ del texto plano otra vez. Sabemos que el bit del texto cifrado $y_i$ fué calculado usando la función de encriptación $y_i \equiv x_i + s_i mod 2$, insertamos esta expresión de encriptaciópn en la función de decriptación:

$$\begin{aligned} d_{s_i}(y_i) &\equiv y_i + s_i \pmod 2 \\ &\equiv (x_i + s_i) + s_i \pmod 2 \\ &\equiv x_i + s_i + s_i \pmod 2 \\ &\equiv x_i + 2s_i \pmod 2 \\ &\equiv x_i + 0 \pmod 2 \\ &\equiv x_i \pmod 2 \end{aligned}$$

Lo interesante es que la expresión $2s_i \pmod 2$ siempre tiene el valor cero porque $2 \equiv 0 \pmod 2$.

**¿Porqué la adición módulo 2 es una buena función de encriptación?** Si se trabaja con la aritmética módulo 2, los únicos posibles valores son 0 y 1. Así podemos tratar la aritmética móduclo 2 como funciones booleanas AND, OR y NAND. La tabla de verdad para la suma módulo 2 es la misma que la función XOR. Esto nos da una conclusión importante: **La adición en módulo 2 es equivalente a la operación XOR.**

La operación XOR es muy importante en criptografía moderna. Pero podría surgir la duda de porqué es tam importante XOR y no son tan importantes otras operaciones como AND. La respuesta es la siguiente, si analizamos la operación XOR para un bit del texto plano $x_i = 0$, obtenemos la siguiente respuesta: Dependiendo del bit de llave, el texto cifrado $y_i$ puede ser cero ($s_i = 0$) o uno ($s_i = 1$). SI el bit de llave se comporta de una manera aleatoria, es decir que tiene un 50 % de posibilidades de ser 0 ó 1, entonces los textos cifrados pueden ocurrir con un 50 % de verosimilitud. De igual manera, si nosotros encriptamos el bit de texto plano $x_i = 1$, otra vez ocurre que dependiendo del valor del bit de llave $s_i$, hay un 50 % de posibilidades de que el texto cifrado sea 1 ó 0.

Podemos concluir que la función XOR es perfectamente balanceada, esto significa que si observamos los valores de la salida, hay exactamente un 50 % de posibilidad de cambio para cualquier valor de los bits de entrada. Otra cosa a notar es que XOR es invertible, esto no ocurre con las operaciones AND y NAND.

**¿Cuál es exactamente la naturaleza de el flujo de la llave?** La manera en cómo se generan los valores $s_i$ es un asunto central para la seguridad de los sistemas de cifrado. De hecho la seguridad de un flujo cifrado depende completamente del flujo de la llavbe. Los bits $s_i$ del flujo de la llame no son los bits de la llave. Entonces ¿cómo obtenemos el flujo de la llave? Generar un flujo para la llave es lo que hace que un cifrado de flujo sea eficidente. Este es un tópico principar. Sin embargo, ya podemos adivinar que un requisito principal para los bits de flujo de la llave es que se puedan representar como una secuencia aleatoria para un atacante. Si lo anterior no ocurriera así, el atacante puede adivinar las claves y podrá realizar el descifrado.

### 1.2. Número aleatorios y flujos de cifrado indescifrables.

**Generadores de números aleatorios.** El proceso de encriptado y desencriptado de un cifrado de flujo es un proceso extremadamente simple. La seguridad de los cifrados ded flujo depende mucho de un flujo de una llave "adecuada" $s_0, s_1, s_2, ...$ Como la aleatoriedad juega un papel fundamental es importante conocer los fundamentes de los generadores de números pseudoaleatorios.

**True Random Number Generators TRNG.** los TRNG tiene la caracterísitcda de que su salida no puede ser reproducida. Pr ejemplo, si nosotros lanzamos una moneda 100 veces y registramos el resultado de la secuencia en 100 bits, será virtualmente imposible que alguien, an algún lugar de la tierra genere la misma secuencia se bits. La posibilidad de éxito es de $\frac{1}{2^{100}}$, es una probabilidad muy pequeña. Los TRNG están basados en procesos físicos, como lanzar un dado, una moneda, ruido de semiconductores.

**Generadores de números psedudoaleatorios (PRNG).** Los PRNGs generan secuencias que son calculadas partiendo de un valor inicial. COn frecuencia son calculadas de manera recursiva por medio de la siguiente manera:

$$s_0 = seed$$

$$s_{i+1} = f(s_i), \quad i = 0, 1, ...$$

Un ejemplo muy popular es el *generador lineal congruencial*, que tienen la siguiente forma:

$$s_0 = seed$$

$$s_{i+1} = a(s_i) + b \mod m \quad i = 0, 1, ...$$

donde a, b, m son enteros constantes. Debemos de recordar que los PRNGs no son totalmente aleatorios porque son completamente determinísticos. Una función muy utilizada en C es la función *rand()*. Esta función tiene los parámetros:

$$s_0 = 12345$$

$$s_{i+1} = 1103515245(s_i) + 12345 \quad \text{mód } 2^{31} \quad i = 0, 1, ...$$

Un requerimiento de los PRNGs es que posean un buen comportamiento estadístico, esto significa que sus salidas se aproximen a una secuencia de números aleatorios verdaderos. Hay muchos test matemáticos que pueden verificar el comportamiento estadístico de las secuencias, por ejemplo, la prueba de la chi cuadrada.

**Generadores de números psedudoaleatorios criptográficamente seguros.**

*Generadores de números psedudoaleatorios criptográficamente seguros (CSPRNGs)* son un tipo especial de PRNG los cuales poseen la siguiente propiedad acidiona: un CSPRNG es un PRNG es cual es impredecible. Informalmente, esto quiere decir que dados n bits de salida de el flujo de la llaba $s_i, s_{i+1}, ..., s_{i+n-1}$, donde $n$ es algún entero, es computacionalmente imposible calcular la subsecuencia de bits $s_{i+n}, s_{i+n+1}, ....$ Una definición más exacta es que dados $n$ bits consecutivos de un flujo de llave, no hay un algoritmo en tiempo polinomial que pueda predecir el siguiente bit $s_{n+1}$ con posibilidad de exito mayor a 50 %. Otra propieda de los CSPRNG es que dada una secuencia superior, es computacionalmente inviable calcular los bits precedentes $s_{i-1}, s_{i-2}, ....$

La impredicibilidad de los CSPRNGs sólo se utiliza en criptografía. En otras situaciones donde los números pseudoaleatorios sean necesarios, la impredicibilidad no se ocupa.

**The one-time pad.** Ahora, discutiremos que pasa si usamos tres tipos de números aleatorios como generadores para el flujo de la llave $s_0, s_1, s_2, ...$ de un cifrado de flujo. Primero, definimios lo que un cifrado perfecto debe ser:

Definición 2 (Seguridad incondicional.) Un criptosistema es incondicionalmente seguro si no puede ser roto incluso con recursos computacionales infinitos.

Seguridad incondicional se basa en teoría de la información y asume que no hay límites en el poder computacional del atacante. Esta difinición es sencilla de entender, pero los requerimientos son muy complejos para que un cifrado sea incondicionalmente seguro. Podemos hacer un experimento: Hay que asumir que tenemos un un algoritmo de encriptación simétrico con una longitud de llave de 10,000 bits, y que el único ataque que funciona es el de una búsqueda exhaustiva de la llave, mejor conocido como la fuerza bruta. Con 18 bits es mas que suficiente para garantizar una seguridad elevada. Pero ¿un cifrado con 10,000 bits es incondicionalmente seguro? La respuesta es ¡No!. Como un atacatne tiene recursos computacionales infinitos, podemos asumir quie el atacante tiene $2^{10000}$ computadoras disponibles y que cad acomputadora revisa

exactamente una llave. Este planteamniento nos dará la llave correcta en un período de tiempo. Por supuesto, no existe la posibilidad de que esistan $2^{10000}$ computadoras, por ejemplo, se estima que hay sólo $2^{266}$ átomos en el universo. El cifrado puede definirse como computacionalmente seguro, pero no incondicional. A continuación, mostraremos una manera de contruir un cifrado ceguro incondicionalmente que es muy simple. Este cifrado es llamado One-Time Pad.

Definición 3 (One-Time Pad (OTP)) Es un tipo de cifrado con la siguientes características:

- El flujo de la llave $s_0, s_1, s_2, ...$ es generado por un generador de números aleatorios verdadero, y
- el flujo de la llave es únicamente conocida para autenciar la compunicación entre el emisor y el receptor.
- Cada bit del flujo de la llave $s_i$ es utilizado sólo una vez.

Es fácil mostrar que el OTP es incondicionalmente seguro. Para cada bit del texto dicfrado, obtenemos una ecuación de esta forma:

$$y_0 \equiv x_0 + s_0 \mod 2$$
$$y_1 \equiv x_1 + s_1 \mod 2$$
$$\ldots$$

Cada relación individual es una ecuación lineal módulo 2 con dos incógnitas. Esto es imposible de resolver. Si el atacante conoce el valor para $y_0$, el no puede determinar el valor de $x_0$, De hecho, $x_0 = 0$ y $x_0 = 1$ tienen las mismas posibilidades de ocurrir. La situación es idéntica para la segunda ecuación y todas las subsecuentes. La situación es diferente si los valores $s_i$ no son realmente aleatorios. Incluso aunque aún es difícil resolver el sistema de ecuaciones, no se puede demostrar que este sistema sea seguro.

**Cifrados de flujo prácticos.** En un sistema práctico de cifrado de flujo, se reemplaza el flujo de la llave verdaderamente aleatoria por un generador de números pseudoaleatorios, donde la llave k sirve como una semiila. Antes de hablar sobre cifrados de flujo en la vida real, debería ser importante recordar que los sistemas de cifrado prácticos no son incondicionalmente seguros. De hecho todos los algoritmos de encriptación prácticos (cifrado de flujo, cifrado de bloque y algoritmos de llave pública) no son incondicionalmente seguros. Lo mejor que podemos esperar es que sean conputacionalmente seguros, esto se define de la siguiente manera:

Definición 4 (Seguridad computacional.) Un criptosistema es computacionalmente seguro si el mejor algoritmo conocido para romperlo necesita de al menos t operaciones.

La definición anterior es adecuada, pero hay aún varios problemas con esto. Primero, con frecuencia no sabemos cuál es el mejor algoritmo para hacer un ataque. Un ejemplo clásico es el esquema de llave pública RSA, el cual puede ser roto al factorizar enteros largos. Incluso aunque muchos algoritmos de factorización son conocidos, nosotros no sabemos si existe uno mejor.Incluo si hay un límite inferior sobre la complejidad de un ataque es conocido, no sabimoes si hay otro ataque más poderoso. Lo mejor que podemos hacer en la práctica es diseñar criptosistemas para los cuales se asume que son computacionalmente seguros. Para cifrados simétricos, esti significa que usualmente uno espera que no hay un método de ataque con una complejidad mejor que una búsqueda de llave exhaustiva.

Para muchso esquemas de cifrado, lo único que deben conocer el emisor y el receptor es la clave secreta de a lo mas 100 bits de longitud, y que además, no tenga que ser tan grande como el mensaje que queremos encriptar. Ahora analizaremos cuidadosamente las propiedades del flujo de la llave $s_0, s_1, s_2, ...$ que es generada por el emisor y el receptor. Obviamente, se necesita algún tipo de generador de número aleatorio para obtener el flujo de la llave. Primero, debemos de notar que no se puede usar un TRNG ya que, por definición, Alice y Bob (a partir de este momento, Alice será el emisor y Bob será el receptor) no podrán generar el mismo flujo de la llave. Lo que necesita es un generador deberminístico de números pseudoaleatorios.

**Construyendo flujos de llave para PRNGs.**
Aquí hay una idea que parece buena, pero en realidad no lo es: Muchos PRNGs tienen buenas proopiedades estadísticas, lo cual es necesario para un flujo de cifrado fuerte. Si se aplican pruebas estadísticas a la secuencia del flujo de la llava, la salida debería comportarse como la secuencia de bits generadas por el lanzamiento de una moneda.Por lo tanto, es tentador suponer de un PRNG puede ser utilizada para generar el flujo de la llave. Pero no es suficiente para que un flujo de un cifrado sea seguro. Oscar es inteligente, considera el siguiente ataque:

Ejemplo 2 Supongamos que tenemos un PRNG basado en el generador lineal congruencial:

$$S_0 = seed$$

$$S_{i+1} \equiv AS_i + Bmodm, i = 0, 1, ...$$

donde nosotros elegimos un flujo $m$ que tenga 100 bits de largo y $S_i, A, B \in 0, 1, ..., m-1$. Debemos de notar que este PRNG puede tener propiedades estadisticas excelentes si elegimos los parámetros cuidadosamente. El módulo $m$

es parte de el esquema de encriptación y es públicamente conocido. La llave secreta comprende los valores $(A, B)$ y posiblemente la semilla $S_0$, cada uno con una longitud de 100. Lo que nos da una longitud de llave de 200 bits, lo cual es más que eficiente para protegerse contra un ataque de fuerza bruta. Como se trataa de un cifrado de flujo, Alice puede encriptar:

$$y_i \equiv x_1 + s_1 \mod 2$$

donde $s_i$ son los bit de la representación binaria de los símbolos de salida $S_j$ del PRNG. Pero Oscar puede de una manera muy fácil planear un ataque. Se asume que el conoce los primeros 300 bits del texto plano (se conocen los primeros 37.5 bytes). Como seguramente conoce el texto cifrado, el ahora puede calcular los primeros 300 bits del flujo de la clave como:

$$s_i \equiv y_i + x_i \mod m, \quad i = 1, 2, ..., 300$$

Cuando se aplican la operación anterior, inmediatamente obtenemos los primeros tres símbolos de salida de la PRNG: $S_1 = (s_1, ..., s_{100})$, $S_2 = (s_{101}, ..., s_{200})$ y $s_3 = (s_{201}, ..., s_{300})$. Con los resultados obtenidos, el atacante Oscar puede formar un sistema de ecuaciones de la sisguiente manera:

$$S_2 \equiv AS_1 + B \mod m$$

$$S_3 \equiv AS_2 + B \mod m$$

Lo anterior es un sistema lineal de ecuaciones sobre $\mathbf{Z}_m$ con dos incógnitas $A$ y $B$. Pero estos valores son la llave, y por lo tanto, podemos resolver el sistema de la siguiente manera:

$$A \equiv (S_2 - S_3)/(S_1 - S_2) \mod m$$

$$B \equiv S_2 - S_1(S_2 - S_3)/(S_1 - S_2) \mod m$$

En el caso de que $\gcd((S_1 - S_2), m) \neq 1$ podemos obtener múltiples soluciones porque este es un sistema de ecuaciones sobre $\mathbf{Z}_m$. Sin embargo, con una carta parte del texto plano conocido, la clave única puede ser detectada en casi todos los casos. POscar implemente intenta cifrar el mansaje con cada una de las múltiples soluciones entontradas. Por lo tanto, si conocemos pequeños fragmentos del texto plano, podemos calcular la clave y desencriptar el texto cifrado.

**Construyendo flujos de claves para CSPRNGs.**
Lo que se necesitas concer para prevenir el ataque mencionado en el ejemplo anterior es utilizar un CSPRNG, el cual asegura que el flujo de la clave es impredecible. Se debe recordar que esto quiere decir que dados los primeros $n$ bits de salida del flujo de la clave $s_1, s_2, ..., s_n$, es computacionalmente imposible calcular los bits $s_{n+1}, s_{n+2}, ....$ Desafortunadamente, casi todos los generadores de

números pseudoaleatorios que son utilizados para aplicaciones fuera de la criptografía no son criptográficamente seguros. Por lo tanto, en la práctica, debemos de utilizar generadores de números pseudoaleatorios diseñados especialmente para cifrados de flujo.

La cuestión ahora es cómo los cifrados de flujo deben ser actualmente. Hay muchas propuestas para cifrados de flujo en la literatura. Los tipos de cifrado pueden clasificarse como aquellos que optimizan el software, o aquellos que optimizan el hardware. En el primer caso, los sistemas de cifrado requieren pocas instrucciones del CPU para calculasr un bit del flujo de la clave. En el último caso, están basados en operaciones que pueden ser fácilmente ejecutadas en hardware. El estado del arte del diseño de cifrados de bloque está más avanzado que los cifrados de flujo. Actualmente parece ser más fácilo para los investigadores diseñar cifrados de bloque seguros qui cifrados de flujo.

**Cifrados de flujo pasados en desplazamientos de registros.** Como se ha mencionado anteriormente, sistemas de cifrado prácticos usan un flujo de clave que es generador por un generador de flujos de claves, el cual debe tener ciertas propiedades. Una manera elegande de hacer secuencias grandes de números pseudoaleatorios es desplazamientos lineales de registros con retroalimentacion (LFSRs). Los LFSRs son fácilmente implementados en hardware y la mayoría de los cifrados de flujo utilizan elos LFSRs. Un ejemplo es el cifrado $A5/1$ el cual es el estándar para la encriptación de voz en un GSM. Y como veremos, aunque un LFSR plano produce una secuencia con propiedades estadísticas muy buenas, es criptográficamente débiel. Sin embargo, la combinación de LFSRs como el $A5/1$ y el cifrado Trivium, puede hacer cifrados de flujo seguros. Se debe destacar que hay muchas formas de construir cifrados de flujo.

**LFSR.** Un LFSR se compone de elementos de alamecenamientos almacenados en flip-flops y una ruta de retroalimentacion. EL número de elementos de almacenamiento nos da el grado de los LFSR. En otras palabras, un LFSR con $m$ flip-flops se dice que tiene un grado $m$. La red de retroalimentación calcula la entrada del último flip-flop como una suma XOR de ciertos flip flopt del registro del desplazamiento.

## Ejemplo 3

Un LFSR se compone de elementos de almacenamiento de velocidad de reloj (flip-flops) y una ruta de realimentación. la número de elementos de almacenamiento nos da el grado de la LFSR. En otras palabras, una LFSR con m flip-flops se dice que es de grado m. La red de realimentación calcula la entrada del flip-flop pasado como XOR-suma de ciertos flip-flops en el registro de desplazamiento.

## Referencias

[RE1] Author: Christof Paar, Jan Pelzl Article/Book: Understanding Cryptrography Other info:Springer 2010