BENEMERITA UNIVERSIDAD AUTONOMA DE PUEBLA

Facultad de Ciencias de la Computación

TESIS

“SEGURIDAD EN DISPOSITIVOS MÓVILES USANDO EL ALGORITMO DE CIFRADO DE FLUJO”

QUE PARA OBTENER EL TÍTULO DE LICENCIADO EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN
PRESENTA

Osvaldo Contreras Martínez

Asesor
Dra. Barbara Emma Sánchez Rinza

Puebla, Pue. Octubre,2024

Contenido

CAPÍTULO 1
ALGORITMOS DE CIFRADO DE FLUJO

1.1 Algoritmo

La palabra algoritmo proviene de la traducción al latín del término árabe Al-Khowarizimi, que significa originario de Khwarizm, nombre antiguo del mar de Aral. De otra parte, Al-Khowarizimi es el nombre con que se conoció a un matemático y astrónomo persa que escribió un tratado sobre manipulación de números y ecuaciones en el Siglo IX.

Puede definirse como “una serie de operaciones detalladas y no ambiguas, a ejecutar paso a paso, y que conducen a la solución de un problema”. [1]

Para Becerra, un algoritmo es el camino para solucionar un problema en un computador, es un grupo de instrucciones que se escriben de acuerdo con reglas sintácticas suministradas por un lenguaje de programación.

Mientras que, para Hérmes, un algoritmo es un procedimiento general con el que se obtiene la respuesta a todo problema apropiado mediante un simple cálculo de acuerdo con un método especificado y posteriormente menciona que este procedimiento debe estar claramente especificado de manera que no haya lugar a que intervenga la imaginación y la creatividad de quien lo ejecuta.

Un algoritmo debe tener al menos las siguientes características:

Ordenado: Presentan una secuencia clara y precisa para poder llegar a la solución.

Finito: Contienen un número determinado de pasos.

Concreto: Ofrecen una solución determinada para la situación o problema planteados.

Definido: El mismo algoritmo debe dar el mismo resultado al recibir la misma entrada [2].

1.2 Cifrado

El cifrado en ciberseguridad es la conversión de datos de un formato legible a un formato codificado. Los datos cifrados solo se pueden leer o procesar luego de descifrarlos.

El cifrado es la base principal de la seguridad de datos. Es la forma más sencilla e importante para garantizar que la información de un sistema de computadora no pueda robarla ni leerla alguien que desee utilizarla con fines maliciosos.

El cifrado de seguridad de datos es un elemento ampliamente utilizado por usuarios individuales y grandes corporaciones para proteger la información de los usuarios enviada entre un navegador y un servidor. Esa información podría incluir todo, desde datos de pagos hasta información personal. Se utiliza software de cifrado de datos, también conocido como algoritmo de cifrado o una clave, para desarrollar un esquema de cifrado que, en teoría, solo puede deshacerse mediante grandes cantidades de potencia informática.

Los dos principales tipos de cifrado son el simétrico y el asimétrico Estas son las principales diferencias entre ambos:

- El cifrado **simétrico** utiliza la misma clave tanto para cifrar los datos como para descifrarlos más tarde, ver figura 1.2.1. El remitente debe entregar previamente la clave al destinatario y cualquier persona con dicha clave puede acceder a la información. Por eso el cifrado simétrico también se conoce como cifrado de clave privada: si la clave se hace pública, deja de ser seguro.

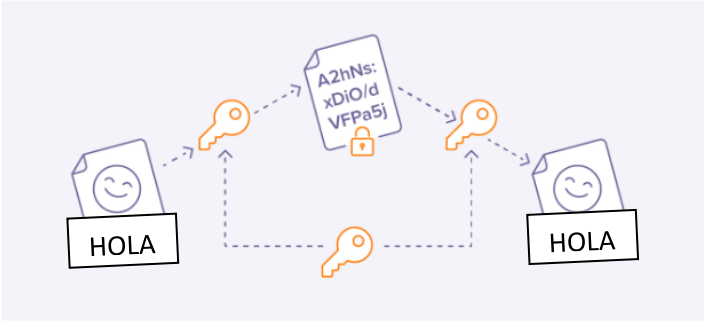

Figura 1.2.1 El cifrado simétrico utiliza una única clave de cifrado para codificar y descodificar los datos

- El cifrado **asimétrico** es más seguro, ya que utiliza dos claves: una pública para cifrar los datos y otra privada para descifrarlos. La clave pública se puede compartir con cualquiera, motivo por el cual a este método se lo conoce como cifrado de clave pública. Solo aquellos con la clave privada pueden devolver los datos cifrados a su estado de texto plano inteligible [2,3].

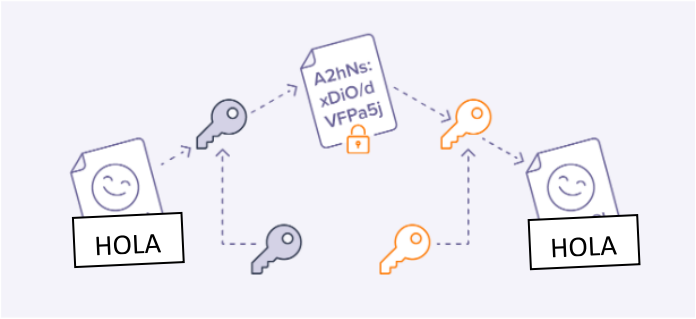

Figura 1.2.2 El cifrado asimétrico es más seguro ya que utiliza dos claves de cifrado

1.3 Criptoanálisis

El criptoanálisis es el intento de descifrar el código, obteniendo acceso no autorizado a los datos cifrados.

El criptoanálisis es un componente importante del proceso de creación de criptosistemas sólidos. La dependencia de la «seguridad a través de la oscuridad» puede resultar en el uso de criptosistemas débiles si los creadores no consideraron todos los posibles vectores de ataque.

En cambio, los algoritmos criptográficos de uso común en la actualidad se han publicado para revisión criptoanalítica. Los que actualmente se consideran “confiables” y de uso común son aquellos para los que aún no se ha descubierto un ataque efectivo.

La definición de criptoanálisis establece que es el estudio detallado de varios métodos utilizados para comprender o decodificar información cifrada sin acceso a la información confidencial que generalmente se requiere para hacerlo. También se lo conoce como descifrado de código [3].

1.4 Cifrado de flujo

Un cifrado de flujo es un cifrado de clave simétrica en el que los dígitos de texto plano se combinan con un flujo de dígitos de cifrado pseudoaleatorio (flujo de claves). En un cifrado de flujo, cada dígito de texto sin formato se cifra uno a la vez con el dígito correspondiente del flujo de claves, para dar un dígito del flujo de texto cifrado, ver figura 1.4.1. Dado que el cifrado de cada dígito depende del estado actual del cifrado, también se conoce como cifrado de estado. En la práctica, un dígito es típicamente un bit y la operación de combinación es un exclusivo-o (XOR).

El cifrado básico requiere tres componentes principales: un mensaje, documento o dato, una llave y un algoritmo de cifrado.

La clave que se utiliza normalmente con un cifrado de flujo se conoce como cojín de una sola vez. Matemáticamente, una libreta de un solo uso es irrompible porque siempre tiene al menos exactamente el mismo tamaño que el mensaje que está encriptando [5].

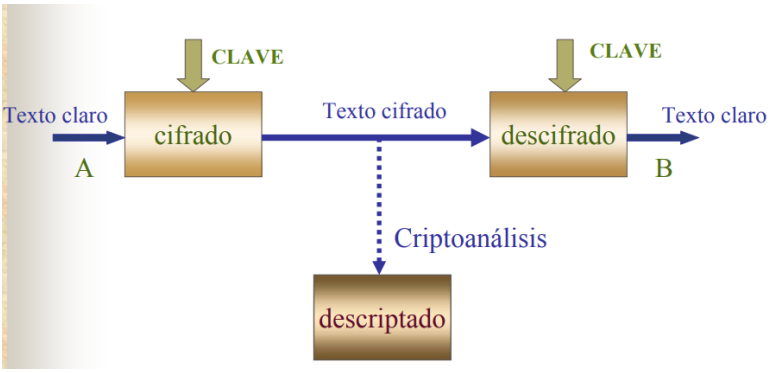

Figura 1.4.1 Esquema de un procedimiento criptográfico

A muy alto nivel, los cifrados de flujo pueden trabajar de dos formas:

- Mediante mantener un estado secreto. Después de haber sido inicializados con una llave y un nonce, el algoritmo mantiene un estado interno que se va actualizando en cada nueva llamada.
- Mediante un contador. Este tipo de algoritmos recibe a parte de la llave y el nonce, un contador que se incrementa en cada llamada, de esta manera no es necesario mantener un estado interno secreto.

a. Evitando errores

Si se está usando un cifrado de flujo, debes evitar en todo momento re-usar el nonce, recuerda que la única razón de existencia del nonce es ser usado una única vez con la misma llave. Esta es la forma más fácil de usar mal los cifrados de flujo, y elimina completamente la seguridad teórica que puedan ofrecer.

b. Ataques más importantes sobre algoritmos criptográficos

Los principales ataques a este tipo de algoritmos buscan debilidades en la estructura de este que le permitan descubrir partes de la secuencia de cifrado. Una de las características fundamentales es el periodo de la clave de cifrado, ya que si es muy corto y se descubre una parte de la clave se puede emplear en sucesivos periodos del algoritmo.

Complejidad lineal. Una técnica empleada para atacar estos algoritmos es el uso de un registro de desplazamiento lineal con realimentación (linear feedback shift register) para replicar parte de una secuencia. A partir de esta técnica aparece la complejidad lineal de una secuencia, que será el tamaño del registro que necesitemos para replicarla.

Ataques de correlación. Otros ataques intentan recuperar parte de una secuencia de cifrado ya empleada. Dentro de estos ataques hay una clase que podemos denominar divide y vencerás que consiste en encontrar algún fragmento característico de la secuencia de cifrado y atacarla con un método de fuerza bruta y comparar las secuencias generadas con la secuencia de cifrado real. Este método lleva a lo que se denomina ataques de correlación y ataques de correlación rápidos [4].

1.5 LFSR

Los cifrados de flujo binario a menudo se construyen utilizando registros de desplazamiento de retroalimentación lineal (LFSR) porque se pueden implementar fácilmente en hardware y se pueden analizar matemáticamente fácilmente.

LFSR significa linear feedback shift register, que se traduce como: registro de desplazamiento con retroalimentación lineal. Es un registro de desplazamiento en el cual la entrada es un bit proveniente de aplicar una función de transformación lineal a un estado anterior.

El valor inicial se denomina semilla y, como la forma de operar el registro es determinista, la secuencia de valores producidos está completamente determinada por el estado actual o el estado anterior. La secuencia tiene un periodo de repetición, es decir que la secuencia vuelve a generarse y se repite indefinidamente. Cuando el periodo de repetición es máximo, ese LFSR tiene interés criptográfico.

a) ¿Cómo trabaja?

Veamos un ejemplo, tenemos la secuencia {16,14,13,11}

La secuencia tap de un LFSR se puede representar como un polinomio mod 2. Esto significa que los coeficientes del polinomio deben ser 1's o 0's. Esto se llama polinomio de realimentación o característica polinomial.

Por ejemplo, si los taps están en las posiciones de los bits: 16, 14, 13 y 11, el polinomio LFSR resultante es:

$$x^{11} + x^{13} + x^{14} + x^{16} + 1$$

Las salidas que influyen en la entrada se denominan taps. Son las que aparecen en el polinomio.

- Si el polinomio es primitivo, sí y solo sí, el LFSR es máximo, o lo que es lo mismo, tiene periodo máximo.
- El LFSR sólo será máximo si el número de taps es par.
- Los valores de tap en un LFSR máximo son coprimos.
- Puede haber más de una secuencia tap que haga máximo al LFSR para esa longitud determinada.
- Una vez encontrada una secuencia tap máxima, automáticamente sigue otra. Si la secuencia tap, en un LFSR n-bit, es {n,A,B,C}, entonces la secuencia mirror correspondiente es {n,n-A,n-B,n-C}. Por ejemplo, la secuencia tap {32,3,2,1}, tiene su homólogo {32,29,30,31}. Ambos dan como resultado periodo máximo.

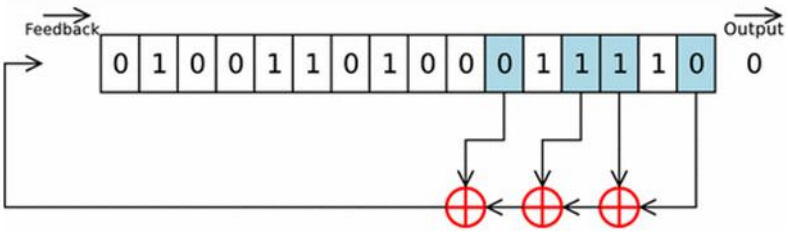

Figura 1.5.1 Funcionamiento de LFSR

Sin embargo, el uso de LFSR por sí solos es insuficiente para proporcionar una buena seguridad.[12] Se han propuesto varios esquemas para aumentar la seguridad de las LFSR.

b) Funciones de combinación no lineales

Un enfoque consiste en utilizar n LFSR en paralelo, combinando sus salidas mediante una función booleana binaria de n entradas (F).

Debido a que los LFSR son inherentemente lineales, una técnica para eliminar la linealidad es alimentar las salidas de varios LFSR paralelos en una función booleana no lineal para formar un generador de combinación. Varias propiedades de dicha función de combinación son críticas para garantizar la seguridad del esquema resultante, por ejemplo, para evitar ataques de correlación.

c) Generadores controlados por reloj

Normalmente, los LFSR se escalonan con regularidad. Un enfoque para introducir la no linealidad es hacer que el LFSR se sincronice de manera irregular, controlado por la salida de un segundo LFSR. Dichos generadores incluyen el generador de parada y arranque, el generador de pasos alternos y el generador de contracción.

Un generador de pasos alternos comprende tres LFSR, que llamaremos LFSR0, LFSR1 y LFSR2 por conveniencia. La salida de uno de los registros decide cuál de los otros dos se utilizará; por ejemplo, si LFSR2 emite un 0, LFSR0 se sincroniza, y si emite un 1, LFSR1 se sincroniza en su lugar. La salida es el OR exclusivo del último bit producido por LFSR0 y LFSR1. El estado inicial de los tres LFSR es la clave.

El generador intermitente [10] consta de dos LFSR. Un LFSR se sincroniza si la salida de un segundo es 1, de lo contrario, repite su salida anterior. Luego, esta salida se combina (en algunas versiones) con la salida de un tercer LFSR sincronizado a una velocidad regular.

El generador de contracción adopta un enfoque diferente. Se utilizan dos LFSR, ambos cronometrados regularmente. Si la salida del primer LFSR es 1, la salida del segundo LFSR se convierte en la salida del generador. Sin embargo, si el primer

LFSR genera 0, la salida del segundo se descarta y el generador no genera ningún bit. Este mecanismo sufre de ataques de sincronización en el segundo generador, ya que la velocidad de la salida es variable de una manera que depende del estado del segundo generador. Esto se puede aliviar almacenando la salida en búfer.

d) Generador de filtros

Otro enfoque para mejorar la seguridad de un LFSR es pasar todo el estado de un solo LFSR a una función de filtrado no lineal [5].

1.6 Tipos de cifrado de flujo

Un cifrado de flujo genera elementos sucesivos del flujo de claves basándose en un estado interno. Este estado se actualiza de dos maneras:

Si el estado cambia independientemente de los mensajes de texto plano o de texto cifrado, el cifrado se clasifica como un cifrado de flujo síncrono.

Si el estado se actualiza en función de los cambios anteriores de los dígitos del texto cifrado, el cifrado se clasifica como un cifrado de flujo auto sincronizado.

- **Cifrado de flujo sincrónico**

En un cifrado de flujo síncrono se genera un flujo de dígitos pseudoaleatorios independientemente de los mensajes de texto plano y de texto cifrado, y luego se combina con el texto plano (para cifrar) o con el texto cifrado (para descifrar). En la forma más común, se utilizan dígitos binarios (bits), y el flujo de claves se combina con el texto plano utilizando la operación exclusiva o (XOR). Esto se denomina cifrado de flujo aditivo binario.

En un cifrado de flujo síncrono, el emisor y el receptor deben estar sincronizados para que el descifrado tenga éxito. Si se añaden o eliminan dígitos del mensaje durante la transmisión, se pierde la sincronización. Para restablecer la sincronización, se pueden probar sistemáticamente varios desvíos para obtener el descifrado correcto. Otro método consiste en marcar el texto cifrado con marcadores en puntos regulares de la salida.

Sin embargo, si un dígito se corrompe en la transmisión, en lugar de añadirse o perderse, sólo se ve afectado un único dígito del texto plano y el error no se propaga a otras partes del mensaje. Esta propiedad es útil cuando la tasa de error de transmisión es alta; sin embargo, hace menos probable que el error se detecte sin otros mecanismos. Además, debido a esta propiedad, los cifradores de flujo síncronos son muy susceptibles a los ataques activos: si un atacante puede cambiar un dígito en el texto cifrado, podría ser capaz de realizar cambios predecibles en el correspondiente bit del texto plano; por ejemplo, voltear un bit en el texto cifrado hace que el mismo bit sea volteado (Toggled) en el texto plano.

- **Cifrado de flujo auto sincronizado**

Los cifradores de flujo auto sincronizados son otra técnica que utiliza parte de los N dígitos del texto cifrado anterior para calcular el flujo de claves. Estos esquemas se conocen también como cifrado de flujo asíncrono o autoclave de texto cifrado (CTAK). La idea de la auto sincronización se patentó en 1946 y tiene la ventaja de que el receptor se sincroniza automáticamente con el generador del flujo de claves tras recibir N dígitos del texto cifrado, lo que facilita la recuperación si se pierden o se añaden dígitos al flujo de mensajes. Los errores de un solo dígito tienen un efecto limitado, ya que sólo afectan a un máximo de N dígitos del texto plano. Es algo más difícil realizar ataques activos a los cifradores de flujo auto sincronizados que a sus homólogos sincrónicos.

Un ejemplo de cifrado de flujo auto sincronizado es un cifrado de bloques en modo de retroalimentación de cifrado (CFB). [6(1)]

A continuación, se muestra algunas ventajas y desventajas:

Ventajas del cifrado de flujo:

1. **Velocidad y eficiencia**: El cifrado de flujo tiende a ser más rápido que el cifrado de bloque, ya que no requiere esperar a que se complete un bloque de datos para comenzar el cifrado. Es adecuado para sistemas donde el tiempo es crucial, como transmisiones en tiempo real.[11]
2. **Capacidad de cifrar datos de longitud variable**: A diferencia del cifrado de bloque, el cifrado de flujo no depende de un tamaño de bloque fijo, por lo que

es más eficiente para cifrar datos que llegan en tiempo real o en flujos continuos.[12]

3. **No necesita relleno (padding)**: Como el cifrado de flujo opera bit por bit o byte por byte, no es necesario rellenar bloques incompletos como en el cifrado de bloque, lo que simplifica la operación.[13]
4. **Uso en aplicaciones de bajo ancho de banda**: Es más eficiente en entornos con limitaciones de recursos o con ancho de banda limitado debido a su menor complejidad computacional.[11]

Desventajas del cifrado de flujo:

1. **Vulnerabilidad ante ataques de reutilización de clave**: Si se utiliza la misma clave para cifrar más de un mensaje, el cifrado de flujo puede ser vulnerable a ataques como el ataque de texto cifrado XOR.[13]

   La seguridad depende en gran medida de que la clave sea única para cada sesión de cifrado.[12]
2. **Sensibilidad a errores**: Un solo error en el texto cifrado puede propagar errores en el texto claro descifrado, lo que afecta a la integridad de los datos.[11]
3. **Difícil de implementar correctamente**: La correcta implementación de los generadores de secuencias cifrantes (como LFSRs) puede ser compleja, y cualquier error en su diseño puede comprometer la seguridad.[13]

1.7 Ejemplos de cifrado de flujo

Veamos algunos ejemplos de cifrados de flujo, sus características y sus usos, así como qué tan seguros son.

A) RC4

RC4 es un Algoritmo de cifrado simétrico, pertenece al cifrado de secuencia (streamcipher, también conocido como cifrado de flujo) en el algoritmo de cifrado simétrico, es una clave de longitud variable, cifrado de flujo orientado a bytes.

RC4 es un tipo de cifrado de flujo, que es un cifrado de secuencia. El algoritmo de cifrado RC4 es un grupo de algoritmos de cifrado de longitud variable diseñado por Ron Rivest en 1987. Al principio, el algoritmo era un secreto comercial y no se hizo público hasta 1994. Debido a que RC4 tiene las ventajas de un algoritmo simple, una velocidad de operación rápida y una fácil implementación de hardware y software, ha sido ampliamente utilizado en algunos protocolos y estándares.

- **Características del algoritmo RC4:**

(1) el algoritmo es simple y fácil de implementar en el software, la velocidad de cifrado es rápida y la seguridad es relativamente alta

(2) la longitud de la clave es variable, generalmente se utilizan 256 bytes.

Antes de presentar el principio del algoritmo RC4, veamos algunas variables clave en el algoritmo:

1、Secuencia clave: La clave del algoritmo RC4 es generar una secuencia de claves correspondiente de acuerdo con el texto sin formato y la clave. La longitud de la secuencia de claves corresponde a la longitud del texto sin formato, lo que significa que la longitud del texto sin formato es de 500 bytes, y la secuencia de claves también es de 500 bytes. Por supuesto, el texto cifrado también tiene 500 bytes, porque Byte i en texto cifrado = byte i en texto sin formato ^ byte i en secuencia clave.

2、Vector de estado S: La longitud es 256, S (0), S (1)..... S (255). Cada unidad es un byte. Cada vez que se ejecuta el algoritmo, S incluye una combinación de permutación de 8 bits de 0-255, excepto que se cambia la posición del valor.

3、Vector temporal T: La longitud también es 256, y cada unidad también es un byte. Si la longitud de la clave es de 256 bytes, el valor de la clave se asigna directamente a T; de lo contrario, cada byte de la clave se asigna a su vez.

4、Clave K: La longitud es de 1 a 256 bytes. Tenga en cuenta que la longitud de la clave, keylen, no está necesariamente relacionada con la longitud del texto sin formato y la longitud del flujo de claves. Por lo general, la longitud de la clave es de 16 bytes (128 bits).

El principio de RC4 se divide en tres pasos como se muestra en la Tabla 1.7.1:

1. Inicializar S y T	2. Permutacion inicial S	3. Generar una secuencia clave r= 0 para len do

for i=0 to 255 do
S[i] =i;
T[i]=K[imodkeylen];
	for i=0 to 255 do
	j=(j+S[i]+T[i])mod256;
	swap(S[i],S[j]);
		r es la longitud del texto plano, r bytes
		i=(i+1) mod 256;
		j=(j+S[i])mod 256;
		swap(S[i],S[j]);
		t=(S[i]+S[j])mod 256;
		k[r]=S[t]; **[7]**

Tabla 1.7.1 Pasos de RC4

B) A5/1

A5/1 es un algoritmo cifrador de flujo usado para proporcionar privacidad en la comunicación al aire libre en el estándar GSM, es decir, el algoritmo que cifra la conversación entre 2 terminales GSM cuando el mensaje viaja por el aire. Inicialmente fue mantenido en secreto, pero llegó al dominio público debido a sus debilidades y a la ingeniería inversa. Varias debilidades serias han sido identificadas en el algoritmo.

Funciona como un cifrador de flujo de datos, y el flujo cifrado se consigue por medio de la operación xor de tres registros (controlado por medio de un reloj) con el flujo a cifrar. El bit que controla el reloj de cada uno de los registros es un bit resultante de una función mayoritaria entre los tres bits centrales de cada registro (i.e. a cada paso habrá dos o tres registros en movimiento). La longitud de los tres registros (, ,) 1 2 3 R R R son 19, 22 y 23 bits respectivamente (por lo que la longitud de la llave es de 64 bits) .

Los polinomios que representan la función f(x) para cada uno de los registros son:

$$f(x_1) = x^{19} + x^{18} + x^{17} + x^{14} + 1$$

$$f(x_2) = x^{22} + x^{21} + 1$$

$$f(x_3) = x^{23} + x^{22} + x^{21} + x^{8} + 1$$

La Figura 1.7.1 muestra el diagrama general de funcionamiento del algoritmo A5/1

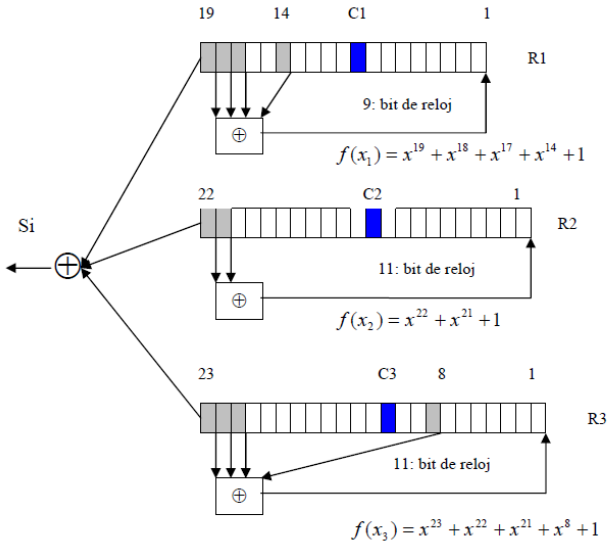

Figura 1.7.1 Descripción del Algoritmo A5 [8]

C) Grain-128a

Grain es un cifrado de flujo propuesto para eSTREAM en 2004 por Martin Hell, Thomas Johansson y Willi Meier. Se incluyó en la cartera final de eSTREAM para Profile 2, es decir, para implementaciones de hardware con capacidad limitada. El algoritmo existe en 2 versiones: la primera se llama Grain v1 y utiliza una clave de 80 bits y un vector de inicialización (IV) de 64 bits, la segunda se llama Grain - 128 y utiliza una clave de 128 bits y un VI de 80 bits.

Grain 128a consta de dos partes grandes: función de salida previa y MAC. La función de salida previa tiene un tamaño de estado interno de 256 bits, que consta de dos registros de tamaño de 128 bits: NLFSR y LFSR. El MAC admite longitudes de etiqueta variables w tales que 0 <w leq 32. El cifrado utiliza una clave de 128 bits.

El cifrado admite dos modos de funcionamiento: con o sin autenticación, que se configura a través del IV_ {0} tal que si IV_ {0} = 1 entonces la autenticación del mensaje está habilitada, y si IV_ {0} = 0 la autenticación del mensaje está deshabilitada, ver Figura 1.7.2.

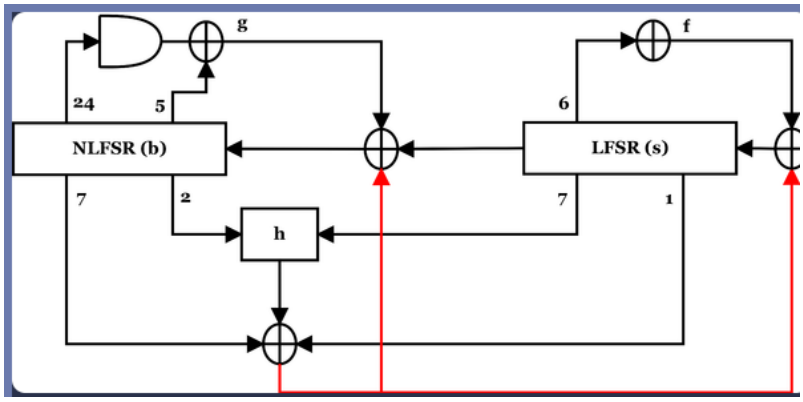

Figura 1.7.2 Diagrama que muestra el procedimiento de inicio de la salida previa que alimenta el flujo de salida previa a las funciones g y f.

D) salsa20

El salsa20 es un cifrado de flujo desarrollado por Daniel J. Bernstein en 2005 y seleccionado para el portafolio de proyectos eSTREAM para profile 1 (implementación de software). Se basa en una función pseudo-aleatoria basada en operaciones de adición de 32 bits, or exclusivo (XOR) y rotación de bits, que utiliza teclas de 256 bits (pero también se pueden usar teclas de 128 bits) y nonce de 64 bits. La función funciona realizando 20 pasos, de ahí el nombre del algoritmo.

El Salsa20 fue desarrollado centrándose en dos puntos principales: el primer punto fue satisfecho usando simples operaciones de adición del módulo 32, operaciones XOR y rotaciones de palabras de 32 bits. El salsa20 expande la clave de 256 bits y el nonce de 64 bits en una cadena de claves de 2 y 70 bytes. Para cifrar o descifrar un mensaje de n bytes, el algoritmo combina estos bytes con los primeros n bytes de la cadena de claves a través de XOR, descartando los restantes. El flujo de claves se genera en bloques de 512 bits, y cada bloque consiste en un hash independiente de la clave, el nonce y el número de bloque expresado con un contador de 64 bits. Gracias a este contador, el flujo de claves también se puede usar con acceso aleatorio, lo que permite al usuario usar cualquier bloque que desee.

El salsa20 debe su nombre al hecho de que realiza 20 pasos de mezcla de su entrada. También hay versiones con el número de pasos reducidos a 8 y 12

llamados, respectivamente, salsa20/8 y salsa20 / 12. Estas versiones se han introducido, no para reemplazar el algoritmo original, sino para obtener el mejor rendimiento en los benchmarks corridos al eSTREAM: el salsa20/12 se selecciona para ser incluido en la cartera final de los algoritmos seleccionados por el proyecto, gracias al excelente equilibrio entre rendimiento y seguridad [8].

El primer ataque realizado positivamente contra salsa20 es por Paul Crowley, quien en 2005 utilizó criptoanálisis diferencial truncado para violar una versión reducida con solo 5 pasos (salsa20/5) del algoritmo, con un tiempo de cálculo estimado de 2 165 [9].

E) SEAL

Es un generador de secuencia diseñado en 1993 para IBM por Phil Rogaway y Don CopperSmith, cuya estructura está especialmente pensada para funcionar de manera eficiente en computadores con una longitud de palabra de 32 bits.

Su funcionamiento se basa en un proceso inicial en el que se calculan los valores para unas tablas a partir de la clave, de forma que el cifrado propiamente dicho puede llevarse a cabo de una manera realmente rápida.

El tamaño de la tabla está establecido por la longitud del clave multiplicado por los 32 bits antes mencionados.

Se comporta como un generador pseudoaleatorio inicializado por una clave raíz de 160 bits. Esta clave se amplía mediante tablas internas para producir un flujo de claves de 32 bits. Las tablas internas con números de 3 se calculan con SHA-0 para Seal versión 1.0 y SHA-1 para versión 3.0.

Luego, las tablas se combinan en dos pasos para producir una salida de 32 bits. Las operaciones elementales durante estos dos pasos son las XOR y las adiciones de módulo 2 32, lo que induce ciertas debilidades.

Finalmente, cada clave de 32 bits así obtenida es luego combinada por un XOR de 32 bits del mensaje, a la manera de una máscara desechable.

Por lo tanto, la seguridad del sello se basa en la seguridad de los algoritmos de la familia SHA-0, ver Figura 1.7.3.

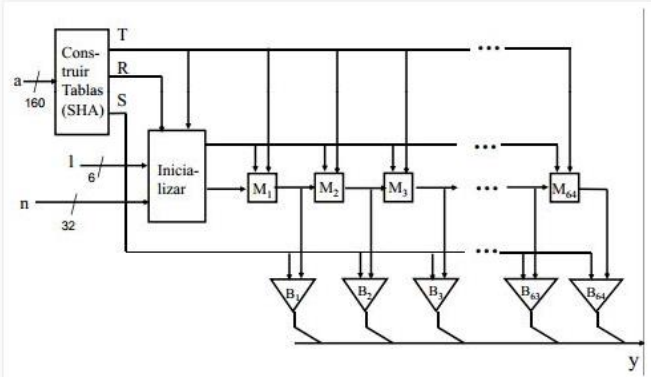

Figura 1.7.3 Algoritmo de la familia SHA-0 [14]

Ventajas de SEAL

- Útil en aplicaciones donde criptosistemas de flujo tradicionales no lo son.
- La mayor parte de criptosistemas de flujo pueden generar una secuencia de bits en un solo sentido:

  -conociendo la llave y una posición i, la única forma de determinar el i -esimo bit generado es generando todos los bits hasta el i –esimo bit buscado.
- Con un criptosistema como SEAL

  -se puede acceder a cualquier bit de la llave generada.
- Uso: asegurar un disco duro

  -encriptar por, y cada, sectores de 512 bytes.

  -con SEAL es posible encriptar el contenido del sector n, con un XOR del contenido con la llave k(n).

Ataques

En 1997 se publicó un ataque de Cryptanalyse_χ^2 . Este ataque permite distinguir un flujo Seal de un flujo pseudoaleatorio y permite encontrar una gran parte de las tablas internas en una versión simplificada del algoritmo.

1.8 Comparativa

A continuación, se muestra en la Tabla 1.8.1 una tabla comparativa de diferentes ejemplos de cifrado de flujo:

Algoritmo	Año de creación	Primer ataque	Complejidad	Longitud de clave efectiva	Velocidad	Uso principal	Requerimientos de hardware
RC4	1987	1995 (RC4 Weak Keys)	O(n) (donde n es la longitud de la clave)	40-256 bits	Rápido en software	Protocolos de cifrado como WEP y SSL (obsoleto)	Bajo, implementable en software general
A5/1	1987 (usado en GSM)	1999 (Ataque a GSM)	O(2^40)	64 bits	Rápido en hardware	Comunicación móvil (GSM)	Bajo, optimizado para hardware móvil
Grain-128a	2011	2011 (Ataque parcial)	O(2^64)	128 bits	Muy rápido en hardware	Dispositivos IoT y aplicaciones de baja potencia	Muy bajo, diseñado para hardware embebido
Salsa20	2005	2007 (Ataques relacionados con claves)	O(2^256)	256 bits	Muy rápido en software	Criptografía en sistemas modernos, VPNs	Bajo, eficiente en hardware y software
SEAL	1993	Sin ataques significativos conocidos	O(2^128)	160 bits	Muy rápido en software (SPARC)	Aplicaciones embebidas y sistemas específicos	Bajo a moderado, diseñado para procesadores SPARC
HC-128	2004	Sin ataques prácticos conocidos	O(2^128)	128 bits	Muy rápido en software	Aplicaciones de software criptográfico	Bajo, ideal para software y sistemas ligeros
Rabbit	2003	2008 (Ataques relacionados con claves)	O(2^128)	128 bits	Muy rápido en software	Protocolos de seguridad y cifrado en aplicaciones	Bajo, altamente optimizado para software

Tabla 1.8.1 tabla comparativa de diferentes algoritmos de flujo

A continuación, se dará un breve explicación sobre la tabla que cuenta con 8 columnas:

Algoritmo: Nombre del algoritmo de cifrado de flujo.

Año de creación: El año en que se desarrolló el algoritmo, lo que puede dar una idea de su antigüedad y evolución en el campo de la criptografía.

Primer ataque: Indica el año y el tipo de ataque que se realizó por primera vez contra el algoritmo. Esto es importante para evaluar la seguridad del algoritmo en el tiempo.

Complejidad: Refleja la dificultad de romper el cifrado en términos de recursos computacionales necesarios. Por ejemplo, O(n) indica que la complejidad aumenta linealmente con la longitud de la clave, mientras que O(2^40) indica que el ataque requeriría una cantidad exponencial de esfuerzo.

Longitud de clave efectiva: La longitud de las claves que el algoritmo permite, lo que impacta directamente en la seguridad del cifrado. Longitudes más largas generalmente indican un mayor nivel de seguridad.

Velocidad: Indica qué tan rápido es el algoritmo, especialmente en diferentes tipos de hardware (software, hardware especializado). Los algoritmos más rápidos son preferidos en aplicaciones donde el rendimiento es crítico.

Uso principal: Describe los contextos o aplicaciones donde se utiliza el algoritmo. Esto puede ayudar a identificar qué algoritmos son más comunes en ciertas industrias o tecnologías.

Requerimientos de hardware: Indica la eficiencia del algoritmo en diferentes plataformas. Algunos algoritmos están optimizados para hardware específico, mientras que otros funcionan bien en plataformas de software general. [6]

Esta tabla y explicación te permiten comparar de manera clara y efectiva diferentes algoritmos de cifrado de flujo en términos de sus características técnicas y aplicaciones prácticas. La elección del algoritmo adecuado dependerá del contexto de uso, requerimientos de seguridad, y recursos disponibles.

1.9 Diferencias cifrado de flujo vs bloque

Aunque tanto los cifrados de flujo como los cifrados de bloque pertenecen a la familia de cifrados simétricos, existen algunas diferencias clave. Los cifrados de

bloque cifran bloques de bits de longitud fija, mientras que los cifrados de flujo combinan bits de texto sin formato con un flujo de bits de cifrado pseudoaleatorio utilizando la operación XOR. Aunque los cifrados de bloque utilizan la misma transformación, los cifrados de flujo utilizan transformaciones variables según el estado del motor. Los cifrados de flujo generalmente se ejecutan más rápido que los cifrados de bloque. En términos de complejidad del hardware, los cifrados de flujo son relativamente menos complejos. Los cifrados de flujo son la preferencia típica sobre los cifrados de bloque cuando el texto sin formato está disponible en cantidades variables (por ejemplo, una conexión wifi-segura), ya que los cifrados de bloque no pueden operar directamente en bloques más cortos que el tamaño del bloque. Pero a veces, la diferencia entre los cifrados de flujo y los cifrados de bloque no es muy clara. La razón es que, cuando se utilizan ciertos modos de operación, se puede usar un cifrado de bloque para actuar como un cifrado de flujo al permitirle cifrar la unidad de datos más pequeña disponible, ver Figura 1.9.1 [8].

1.9 Comparativa

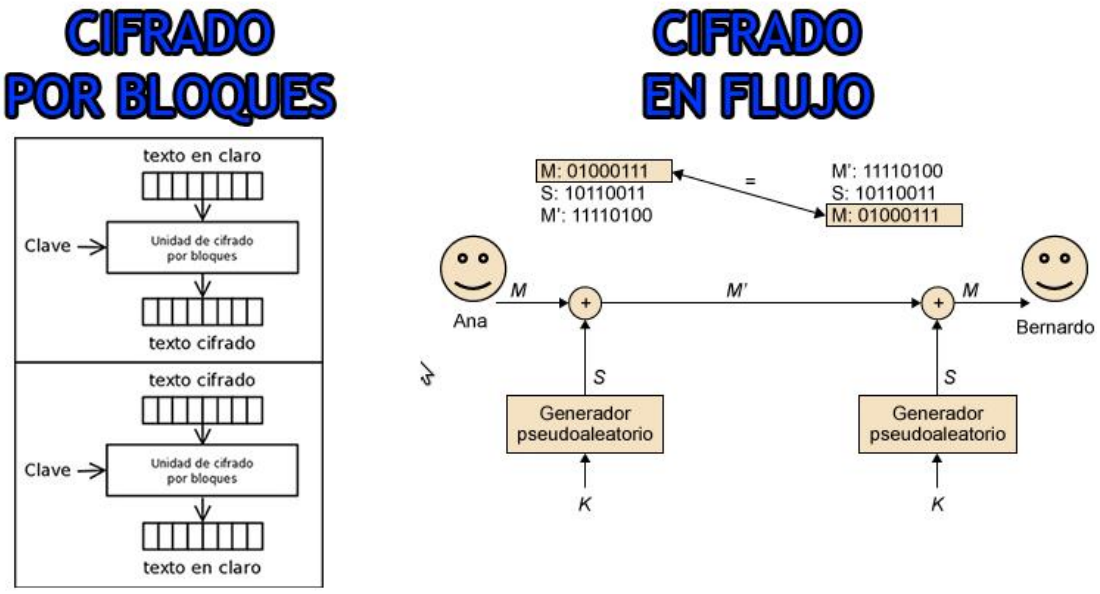

Figura 1.9.1 Diferencias entre cifrado de bloque y cifrado de flujo

Los cifrados de flujo se utilizan a menudo por su velocidad y simplicidad de implementación en hardware y en aplicaciones donde el texto sin formato viene en

cantidades de longitud desconocida, como una conexión inalámbrica segura. Si se usara un cifrado de bloque (que no opera en un modo de cifrado de flujo) en este tipo de aplicación, el diseñador debería elegir entre la eficiencia de transmisión o la complejidad de implementación, ya que los cifrados de bloque no pueden trabajar directamente en bloques más cortos que su tamaño de bloque. Por ejemplo, si un cifrado de bloque de 128 bits recibe ráfagas de texto sin formato de 32 bits, tres cuartas partes de los datos transmitidos serían de relleno. Los cifrados de bloque se deben utilizar en el robo de texto cifrado o en la terminación de bloque residual modo para evitar el relleno, mientras que los cifrados de flujo eliminan este problema al operar naturalmente en la unidad más pequeña que se puede transmitir (generalmente bytes).

Otra ventaja de los cifrados de flujo en la criptografía militar es que el flujo de cifrado se puede generar en una caja separada que está sujeta a estrictas medidas de seguridad y se alimenta a otros dispositivos, como un aparato de radio, que realizará la operación xor como parte de su función. Este último dispositivo se puede diseñar y utilizar en entornos menos estrictos.

ChaCha se está convirtiendo en el cifrado de flujo más utilizado en software;3 otros incluyen: RC4, A5/1, A5/2, Chameleon, FISH, Helix, ISAAC, MUGI, Panama, Phelix, Pike, Salsa20, SEAL, SOBER, SOBER-128, y WAKE [3].

CAPÍTULO 2

ANALISIS Y DISEÑO

2.1 Objetivo General

Cada día las aplicaciones tienen mayor necesidad de intercambiar mensajes, con el fin de integrar aplicaciones o de comunicar algo a otras aplicaciones, este tipo de intercambio de mensajes ha crecido a un más, sea cual sea el motivo por el cual un mensaje es enviado de un punto a otro es indispensable asegurarnos que los mensajes sean enviados en un canal seguro y que el mensaje enviado sea el mismo que se reciba del otro lado sin ninguna alteración, también existen escenario en los que los mensajes contienen información altamente confidencial como datos de nuestros clientes o cuentas bancarias por lo que somos responsables de asegurarnos que la información continué siendo confidencial.

El objetivo es diseñar e implementar un software que permita el uso del algoritmo de cifrado de flujo en los mensajes y que permita ser enviados entre usuarios a través de sus dispositivos móviles.

2.2 Objetivos Específicos

- Analizar los diferentes tipos de sistemas operativos para dispositivos móviles.
- Determinar el sistema o sistemas operativos en donde se diseñará el software para su funcionamiento en el dispositivo móvil.
- Evaluar el software en funcionamiento y que cumpla con el uso del Algoritmo de cifrado de flujo.

2.3 Funcionamiento del algoritmo de cifrado de flujo

Recordando la propuesta de cifrado hecha por Vernam en 1917, los cifradores de flujo usarán:

- Un algoritmo de cifrado basado en la función XOR.
- Una secuencia cifrante binaria y pseudoaleatoria denominada S, que se obtiene a partir de una clave secreta K compartida por el emisor y el receptor, y un algoritmo generador determinístico.
- El mismo algoritmo para el descifrado debido el carácter involutivo de la función XOR.

En concreto, cada bit de entrada al sistema de cifrado (mensaje en claro) se combinará, usando la función lógica XOR, con el bit correspondiente del flujo clave S, para dar lugar al bit correspondiente al flujo de salida (mensaje cifrado). El receptor hará el mismo proceso de combinación con la XOR para obtener el flujo descifrado.

Características de la secuencia cifrante S:

- Período:
  - ✓ La clave deberá ser tanto o más larga que el mensaje. En la práctica se usará una semilla K de unos 120 a 250 bits en cada extremo del sistema para generar períodos superiores a 1035.
- Distribución de bits:
  - ✓ La distribución de bits de unos (1s) y ceros (0s) deberá ser uniforme para que represente a una secuencia pseudoaleatoria.

2.4 Descripción del algoritmo XOR (OR exclusivo)

El cifrado XOR es un algoritmo de cifrado basado en el operador binario XOR, el cual es una operación OR exclusiva (XOR).

El operador XOR

Para entender el funcionamiento del operador XOR podemos observar la Tabla 2.3.1

A	B	A XOR B
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Tabla 2.3.1 Tabla XOR

Además, se puede observar que se cumple que:

$$A \oplus 0 = A,$$

$$A \oplus A = 0,$$

$$(B \oplus A) \oplus A = B \oplus 0 = B$$

Si aplicamos a una cadena A XOR 0 = A, nos dará como resultado la misma cadena. Si aplicamos a una cadena B XOR A = C y a este resultado le volvemos a aplicar C XOR A = B, nos dará como resultado la cadena inicial, es decir B.

Ejemplo

- **Cifrado cadena-clave (8bits = 1byte = 1 carácter)**

Supóngase que se tiene la palabra "FISI" (01000110 01001001 01010011 01001001) y la ciframos con la clave (01010101) que su valor en ASCII es 85 y está representado como "U", entonces tenemos así:

01000110 01001001 01010011 01001001

01010101 01010101 01010101 01010101

00010011 00011100 00000110 00011100

- **Cifrado cadena-clave (n caracteres)**

Si tenemos una cadena de entrada "FISI" (01000110 01001001 01010011 01001001) y clave "1234" (00110001 00110010 00110011 00110100)

01000110 01001001 01010011 01001001

00110001 00110010 00110011 00110100

01110111 01111011 01100000 01111101

En esto consiste básicamente el cifrado XOR, el cual tenemos como entrada el texto a cifrar "FISI" (B) y la clave de cifrado "1234" (A), para descifrar el texto solamente tendríamos que volver a aplicar un XOR al texto cifrado "w{`}" (C) , que nos dará como resultado la cadena inicial "FISI" (B).

- **Descifrado cadena-clave (n caracteres)**

Tenemos como entrada el texto cifrado "w{`}" (01110111 01111011 01100000 01111101) y la clave utilizada anteriormente "1234" (00110001 00110010 00110011 00110100) y le aplicamos XOR:

01110111 01111011 01100000 01111101

00110001 00110010 00110011 00110100

01000110 01001001 01010011 01001001

Como resultado hemos obtenido la cadena "FISI" (01000110 01001001 01010011 01001001) que es la cadena que inicialmente ingresamos.

CAPITULO 3

3.1 Diagrama de Flujo

Un Diagrama de Flujo es una representación gráfica de un algoritmo, que muestra los pasos o procesos necesarios para alcanzar la solución de un problema. Es esencial para la correcta implementación de un programa, ya que a partir de este diagrama se estructura el código en un lenguaje de programación específico.

A continuación, se detalla el diagrama de flujo de nuestra aplicación, mostrado en la **Figura 3.1.1**. Esta aplicación permite cifrar y descifrar mensajes, así como enviarlos a través de correo electrónico.

Descripción del diagrama:

1. **Inicio**: El usuario interactúa con un botón de “Iniciar”, que lo redirige a un menú principal.
2. **Menú principal**: Este menú ofrece tres opciones:
    - **Cifrar**: Solicita al usuario ingresar el mensaje y la llave de cifrado. Luego, utiliza el algoritmo XOR para cifrar el mensaje. Tras esto, se ofrecen dos opciones: “Regresar” al menú principal o “Enviar” el mensaje cifrado a un correo electrónico junto con un asunto.
    - **Descifrar**: El usuario ingresa un mensaje cifrado y la llave correspondiente. Al hacer esto, el sistema ofrece las opciones de “Regresar” al menú principal o “Descifrar” el mensaje, mostrándolo en su forma original.
    - **Regresar**: Devuelve al usuario al botón de inicio.

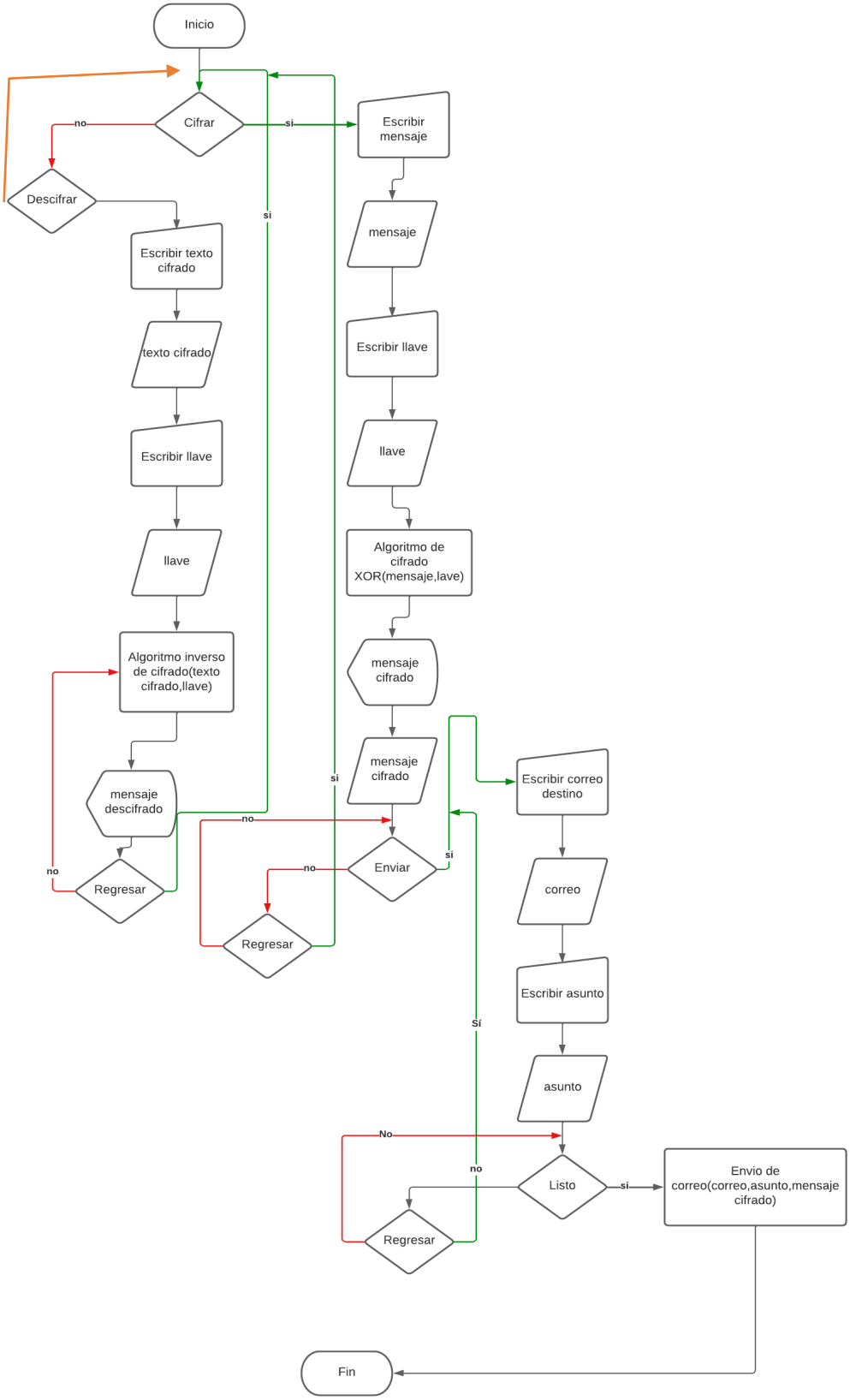

Figura 3.1.1 Diagrama de Flujo de la aplicación algoritmos de flujo

Además, en la Figura 3.1.2, La figura presentada muestra un diagrama de flujo simplificado del algoritmo de cifrado de flujo, que describe las interacciones básicas entre los usuarios y las acciones principales del sistema. El diagrama se divide en dos partes: el proceso de **cifrado** y el proceso de **descifrado**, ambos conectados al algoritmo central.

Explicación de los elementos del diagrama:

1. **Algoritmo cifrado de flujo (parte superior)**:
    - Es el núcleo del sistema, representado en la parte superior del diagrama, que conecta las dos funciones principales (cifrado y descifrado).
    - Este algoritmo es responsable de realizar las operaciones de cifrado o descifrado de los mensajes según la interacción del usuario.
2. **Proceso de cifrado (izquierda)**:
    - Esta parte del diagrama está dedicada al **usuario emisor** que desea cifrar un mensaje.
    - El usuario introduce un mensaje y una clave que serán procesados por el algoritmo de cifrado.
    - Después de aplicar el cifrado, el mensaje se convierte en un texto cifrado listo para ser enviado de manera segura.
3. **Proceso de descifrado (derecha)**:
    - Aquí se encuentra el **usuario receptor** del mensaje cifrado.
    - Este usuario ingresa el mensaje cifrado junto con la clave de cifrado utilizada originalmente para poder descifrar el mensaje y obtener el texto legible.
    - El algoritmo toma estos datos y realiza la operación inversa al cifrado, revelando el mensaje original.
4. **Usuarios (parte inferior)**:
    - Los usuarios representados por figuras sencillas (círculos y líneas) son los participantes en este proceso. Ambos están involucrados en las etapas de cifrado y descifrado.

- Aunque las figuras se etiquetan solo como "Usuario", el contexto indica que el usuario de la izquierda es quien cifra (emisor) y el usuario de la derecha es quien descifra (receptor).

Conexión y flujo:

- Las flechas bidireccionales indican que el algoritmo puede operar en ambas direcciones: cifrado y descifrado. Los usuarios interactúan con el algoritmo según la tarea que desean realizar, ya sea convertir un mensaje en texto cifrado o revertir este proceso.

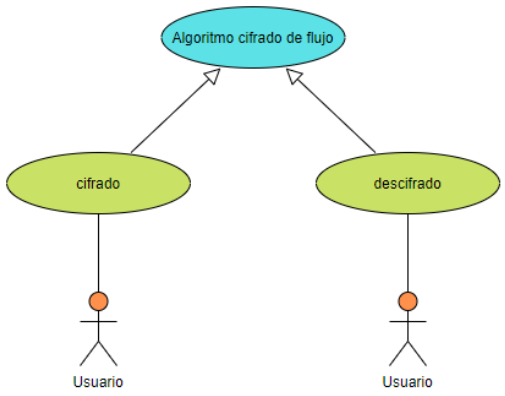

Figura 3.1.2 Diagrama de usuarios

3.2 Diagrama de bloque

Ahora se analiza un sencillo diagrama de bloques que es una representación gráfica de un proceso. Cada bloque representa un paso en el que tiene una entrada y una salida. El objetivo de los bloques es realizar una transformación de aquello que entra y diferenciarlo de lo que sale también de resumir todo un procedimiento o un proceso que realizan las máquinas.

Un **Diagrama de Bloques** es una representación gráfica de un proceso. Cada bloque simboliza un paso que tiene una entrada y una salida, lo que permite visualizar claramente la transformación que ocurre en cada etapa.

En la Figura 3.2.1, observamos un diagrama con dos usuarios: el emisor y el receptor. Cada uno realiza diferentes tareas dentro del proceso de cifrado y descifrado de mensajes.

Descripción del diagrama:

1. **Emisor**:
    - Escribe el mensaje a cifrar.
    - Ingresa la llave de cifrado.
    - Presiona el botón "Cifrar", que transforma el mensaje en su versión cifrada.
    - Selecciona la opción de "Enviar", enviando el mensaje cifrado por correo electrónico.
2. **Receptor**:
    - Recibe el mensaje cifrado.
    - Copia el mensaje en el campo correspondiente.
    - Introduce la misma llave utilizada para el cifrado.
    - Presiona el botón "Descifrar" para ver el mensaje en su formato original.

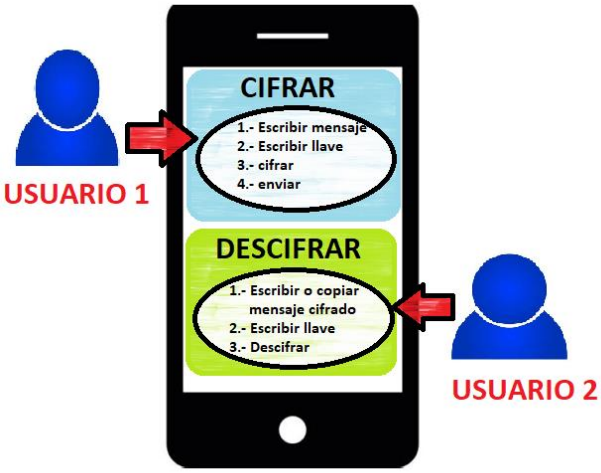

Figura 3.2.1 Diagrama de bloque de la aplicación algoritmo de cifrado de flujo

3.3 Código de funcionamiento

Introducción

Este código implementa un algoritmo de cifrado simple utilizando la operación XOR (o "O-exclusivo") para cifrar y descifrar cadenas de texto. La clase cifrado contiene un método llamado encode que se encarga de aplicar este algoritmo. El cifrado XOR es un enfoque básico en la criptografía, donde cada carácter del mensaje se combina con una clave utilizando la operación XOR bit a bit. Este método de cifrado tiene la ventaja de ser reversible, es decir, aplicando la misma clave al texto cifrado, se puede recuperar el texto original.

Este tipo de algoritmo es adecuado para demostraciones o aplicaciones sencillas, pero no debe ser utilizado para cifrado robusto en entornos críticos debido a su vulnerabilidad ante ataques más avanzados. A continuación, se muestra en la Figura 3.3.1:

Código

```
package com.example.algoritmodeseguridad;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
public class cifrado {
    public String encode(String str, String key) {
        String mensaje = "";
        //int llave=Integer.parseInt(key);}
        char[] keyChars = key.toCharArray();
        char[] chars = str.toCharArray();
        int keyLength = keyChars.length;
        /*
        for(int i=0;i< chars.length;i++)
        {
            char c = (char)(chars[i] ^ llave);
            mensaje +=c;
        }
        */
        for (int i = 0; i < chars.length; i++) {
            char c = (char) (chars[i] ^ keyChars[i % keyLength]);
            mensaje += c;
        }
        return mensaje;
    }
}
```

Figura 3.3.1 Código java cifrado y descifrado.

Explicación del Código

1. Paquete y bibliotecas

El paquete com.example.algoritmodeseguridad organiza el código dentro del proyecto. Se importan algunas librerías, aunque en este código no se están utilizando activamente (ByteArrayOutputStream y IOException).

2. Clase cifrado

Esta clase contiene el método encode, que implementa el algoritmo de cifrado y descifrado basado en la operación XOR.

3. Método encode

El método encode toma dos parámetros:

- str: El mensaje de texto que se desea cifrar o descifrar.

- key: La clave que se utilizará para aplicar la operación XOR.

El resultado cifrado se almacenará en la variable mensaje.

4. Conversión de cadenas a arreglos de caracteres

El mensaje (str) y la clave (key) se convierten en arreglos de caracteres (char[]). Esto es necesario para poder operar sobre cada carácter individualmente.

5. Obtener la longitud de la clave

Se guarda la longitud de la clave para utilizarla en el proceso de cifrado, ya que la clave se reutilizará cuando sea más corta que el mensaje.

6. Bucle para cifrar/descifrar

El bucle recorre cada carácter del mensaje original (chars[i]) y lo cifra utilizando la operación XOR con el carácter correspondiente de la clave (keyChars[i % keyLength]). Si la longitud del mensaje excede la longitud de la clave, la clave se reutiliza cíclicamente (por eso se utiliza el operador de módulo %).

- **XOR**: La operación XOR compara los bits de dos caracteres. Si son iguales, el resultado es 0; si son diferentes, el resultado es 1. Esta operación es reversible: al aplicar XOR nuevamente con la misma clave, se recupera el texto original.

El resultado cifrado se va concatenando en la variable mensaje.

7. Retorno del mensaje cifrado

El método retorna el mensaje cifrado o descifrado (dependiendo de si se pasa un mensaje claro o uno ya cifrado). Dado que el proceso es reversible, el mismo método se utiliza tanto para cifrar como para descifrar.

Conclusión

El código implementa un algoritmo de cifrado básico utilizando la operación XOR entre cada carácter del mensaje y una clave. Aunque es un enfoque simple y fácil de implementar, este tipo de cifrado es susceptible a ataques si no se utiliza adecuadamente. Por ejemplo, si la clave es corta o predecible, el mensaje cifrado puede ser vulnerable a ataques de criptoanálisis. Este tipo de algoritmo se puede utilizar para fines educativos o en aplicaciones donde la seguridad no sea crítica.

CAPITULO 4

4.1 Wireframes

Para la realización de nuestra aplicación se utilizarán wireframes, según el diccionario de Cambridge; un wireframe es un plano básico para un sitio web que muestra el tipo de información que contendrá y cómo estará dispuesta, pero no incluye características de diseño como color o detalles particulares.

Aunque cabe recalcar que estas representaciones se utilizan también para diseño de aplicaciones de dispositivos móviles, por lo tanto, para mostrar y facilitar el desarrollo, esta herramienta no será útil ya que se mostrará la estructura y funcionalidad de la interfaz de usuario.

Además, los wireframes son útiles para validar conceptos, probar la usabilidad y obtener retroalimentación temprana de los usuarios, antes de invertir tiempo y recursos en el desarrollo de un diseño final.

Se comienza mostrando el diseño del inicio de la App:

Para comenzar se muestra el título de la App, en donde la ventana mostrara un botón de “Inicio” y un fondo de pantalla, como se muestra en la Figura 4.1. Al dar click sobre el botón “Iniciar”, se cambiará de ventana.

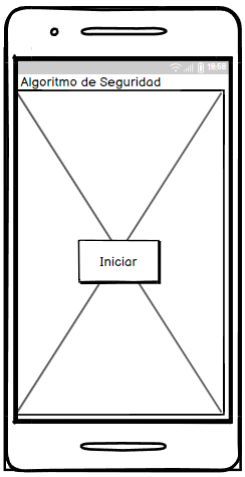

Figura 4.1 Ventana de inicio de la App

Ahora se muestra el menú principal en donde solo se tendrán 3 opciones, como se observa en la Figura 4.2, las opciones son “CIFRAR”, “DESCIFRAR” y “REGRESAR”, esta ventana también contendrá un fondo de pantalla, que haga resaltar los colores y lo 3 botones.

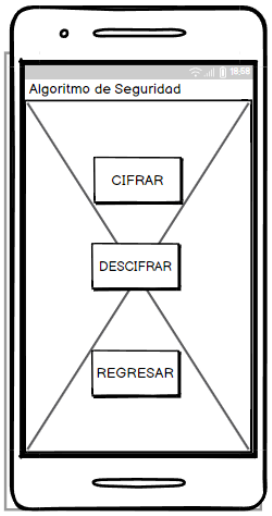

Figura 4.2 Ventana de menú de la App

Al seleccionar la opción de “CIFRAR” se desplegará otra ventana que contendrá lo que se observa en la Figura 4.3, esta contendrá un cuadro de texto en donde se escribirá el mensaje que el usuario desee cifrar. A continuación, se mostrar otro cuadro de texto más pequeño en donde se pedirá la llave para hacer el cifrado.

Mas abajo se encuentra un botón con el título “CIFRAR” que al dar click utilizara la llave para realizar el cifrado del mensaje y mostrara el texto cifrado debajo del botón que es el mensaje que se enviara.

Para realizar el envío se muestra un botón con el título “ENVIAR” que nos enviara a otra ventana, también se cuenta con un botón para regresar al menú principal.

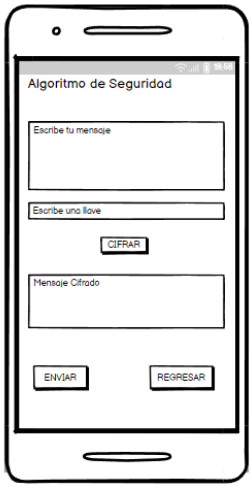

Figura 4.3 Ventana de Cifrado

Al hacer click en el botón “ENVIAR” se mostrará una ventana como en la Figura 4.4.

En la nueva vista se puede observar un cuadro de texto en donde se pedirá al usuario ingresar el correo electrónico al que se enviara el mensaje cifrado y además otro cuadro de texto en donde se podrá ingresar el asunto del mensaje, también se mostrara el mensaje cifrado que se enviara al destinatario, y para finalizar se cuenta con un botón con el título “LISTO”, que al seleccionarlo se abrirá la aplicación de correo electrónico, que se tenga instalada , en donde la información del correo al que se enviara el mensaje y el asunto se mostrara en la app, solo para enviar al destino.

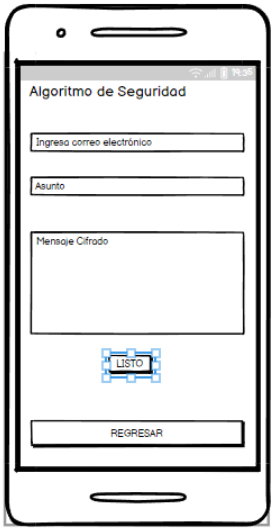

Figura 4.4 Ventana de correo

Al recibir el mensaje cifrado del emisor se mostrara en la bandeja de entrada de nuestro app de correo electrónico, se tendrá que copiar el mensaje cifrado y en el menú principal de esta aplicación se selecciona la opción “DESCIFRAR”, que nos mostrara la ventana en la Figura 4.5 donde hay un cuadro de texto para pegar el mensaje cifrado, otro cuadro de texto para la llave que descifrara el mensaje , después un botón que al darle click mostrara el mensaje descifrado en un cuadro de texto y un botón para regresar al menú principal.

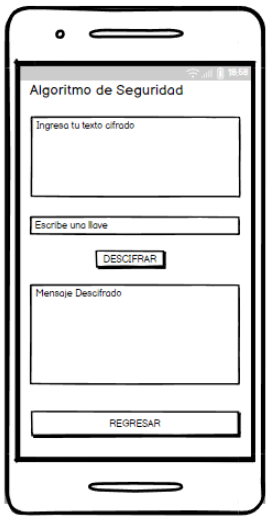

Figura 4.5 Ventana de Descifrado

Estos wireframes nos permitirán llevar una programación en el diseño más ordenada y evitar errores en el diseño. De haber alguna modificación sobre los bocetos se documentará, así como las razones del cambio.

4.2 Guía de Diseño

En este parte se definirán patrones de diseño, así como las directivas que se utilizarán para garantizar que usuario tenga una experiencia intuitiva, atractiva y accesible.

En los siguientes párrafos se detallarán las especificaciones sobre colores, fuentes, iconos e imágenes que se deben utilizar, también los componentes ya que esta parte servirá como referencia para el diseño y que se siga un diseño al momento de realizar la programación.

Para comenzar definiremos algunos fondos de pantalla entre los que se muestra en las Figuras 4.2.1, Figura 4.2.2, Figura 4.2.3, 4.2.4, 4.2.5 y 4.2.6 de las cuales se elegirán 2, uno para el inicio y otro para el menú principal, por lo tanto, se tienen las siguientes opciones:

Estas opciones se escogieron por su fondo que son colores en su mayoría blanco y negro lo que dará mayor libertad de escoger más colores para los demás elementos que la aplicación tendrá, además de que tienes un tema alusivo a la seguridad que es el principal objetivo para el desarrollo de la aplicación.

1)

Figura 4.2.1 Opción 1

2)

Figura 4.2.2 Opción 2

3)

Figura 4.2.3 Opción 3

4)

Figura 4.2.4 Opción 4

5)

Figura 4.2.5 Opción 5

6)

Figura 4.2.6 Opción 5

Ahora se definirá la paleta de colores, que se utilizará, esta paleta se basará en transmitir confianza, estabilidad, seguridad, autoridad y algo destacado para los textos.

También que la combinación de estos colores refleje una interfaz atractiva y coherente para el usuario, así como reforzar una experiencia intuitiva, de manera que cada color sea un complemento que mejore el uso de la aplicación.

Entre la paleta de colores, se seleccionaron colores que demuestre una interfaz confiable e intuitiva para el usuario; entre los objetivos para la elección de los colores esta:

Transmitir confianza y seguridad: Utilizar colores que evoquen confianza y seriedad, lo que en consecuencia el usuario sienta que su información está protegida.

Accesibilidad: Garantizar que la paleta de colores cumpla con la accesibilidad, haciendo la aplicación intuitiva.

A continuación, se muestra la paleta de colores en la Tabla 4.2.1:

COLOR	NOMBRE	HEXADECIMAL	DESCRIPCION
	Azul oscuro	#0A3D62	Representa confianza, seguridad y estabilidad.
	Azul claro	#3498DB	Representa la tecnología y modernidad.
	Gris oscuro	#2C3E50	Representa seriedad y profesionalismo, mantener equilibrio.
	Gris claro	#ECF0F1	Representa una proporción de colores al tener un contraste suave en varios colores oscuros y ayuda a la legibilidad.
	Verde	#27AE60	Representa éxito y seguridad, útil para confirmar acciones seguras y exitosas.
	Rojo	#E74C3C	Representa advertencia sobre errores y acciones destructivas, llamando la atención del usuario de manera efectiva.
	Amarillo	#F1C40F	Representa que el usuario debe estar alerta sobre información que necesita ser tan severo como el rojo.

Tabla 4.2.1 Colores para utilizar en la App

Una vez que tenemos los colores, elegiremos algunos iconos esta parte es crucial para hacer la aplicación clara, intuitiva y profesional, cada icono deberá representar el uso adecuado de una aplicación de cifrado.

Ahora se mostrarán algunos iconos que pueden ser utilizados en la en diferentes secciones y/o funciones de la aplicación, junto con los propósitos:

Icono de candado: Transmite la idea de protección y privacidad.

Icono de llave: Simboliza el control y acceso.

Icono de alerta: Llamar la atención en situaciones que requieren precaución.

Icono de verificación: Proveer retroalimentación positiva y confirmación visual.

Estos iconos deben mejorar la experiencia del usuario, cada uno de ellos debe ser un complemento y no entorpecer las acciones del usuario y mucho menos que la interfaz confunda al usuario.

Para continuar se elegirá algunas tipografías para la aplicación, teniendo en cuenta que este aspecto es crucial ya que dará la percepción de confiabilidad y la experiencia del usuario en general, entre las que se usaran se muestra en la Tabla 4.2.2:

NOMBRE	DESCRIPCION	EJEMPLO
ROBOTO	Es muy utilizada en aplicaciones Android y es moderna.	**Osvaldo Contreras Martínez**
OPEN SANS	Es versátil y adecuada para interfaces.	Osvaldo Contreras Martínez
LATO	Fácil de leer, buena para ser accesible	Osvaldo Contreras Martínez
HELVETICA	Clásica y confiable, con excepción de que no es gratuita.	

Tabla 4.1.2 Tabla de tipografías

CAPITULO 5

5.1 Implementación de diseño

Para este capítulo se llevará a cabo la implementación del diseño, se utilizarán los recursos que se describieron en el capítulo anterior, como se mencionó el diseño va a desempeñar un papel especial tanto en la funcionalidad y efectividad del software y en la creación de una experiencia que inspire confianza y facilite el uso intuitivo del usuario.

INICIO

El código para el diseño se puede consultar en el ****Apéndice A****, donde se detalla su implementación en XML.

Objetivo: El código nos muestra los elementos de la ventana principal, que es el fondo de pantalla y un botón para iniciar, para esta ventana se utilizaron los elementos del capítulo anterior como se muestra en la Figura 5.1:

Figura 5.1 Ventana de Inicio

MENÚ

El código para el diseño se puede consultar en el ****Apéndice B****, donde se detalla su implementación en XML.

Objetivo: Mostrar un menú para que el usuario puede elegir entre las opciones de cifrar y descifrar dependiendo de lo que el usuario necesite, de la misma manera se utilizó lo que se mencionó en el capítulo anterior, el fondo de pantalla, colores e iconos. A continuación, se muestra Figura 5.2:

Figura 5.2 Menú de usuario

CIFRAR

El código para el diseño se puede consultar en el ****Apéndice C****, donde se detalla su implementación en XML.

Objetivo: En esta ventana se le muestra al usuario un apartado para que escriba su mensaje, además de escribir la llave y el botón para mostrar el mensaje ya cifrado. En caso de que el mensaje le convenza al usuario un botón para enviar el mensaje ya cifrado, como se muestra en la Figura 5.3:

Figura 5.3 Ventana de cifrado

ENVIAR

El código para el diseño se puede consultar en el ****Apéndice D****, donde se detalla su implementación en XML.

Objetivo: Aquí se pide al usuario el correo destinatario, así como algún asunto que se requiera enviar y además se le muestra en mensaje ya cifrado y al darle click al botón de listo se enviara en mensaje al correo destino, sabiendo esto se muestra la interfaz, en la Figura 5.4:

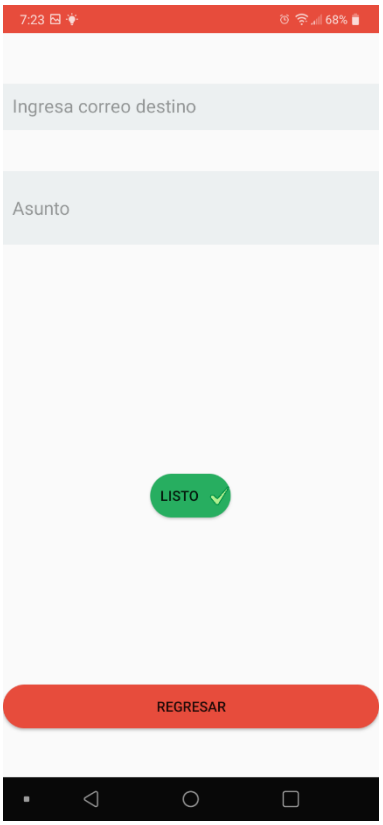

Figura 5.4 Ventana de enviar

DESCIFRAR

El código para el diseño se puede consultar en el ****Apéndice E****, donde se detalla su implementación en XML.

Objetivo: En el caso de que se haya un mensaje cifrado, la app muestra la ventana en donde el usuario podrá descifrar su mensaje, desde su correo puede pegar el mensaje y escribir su llave, el mensaje descifrado se mostrara en la pantalla para poder leerse, en la Figura 5.5 se muestra cómo se quedó la ventana:

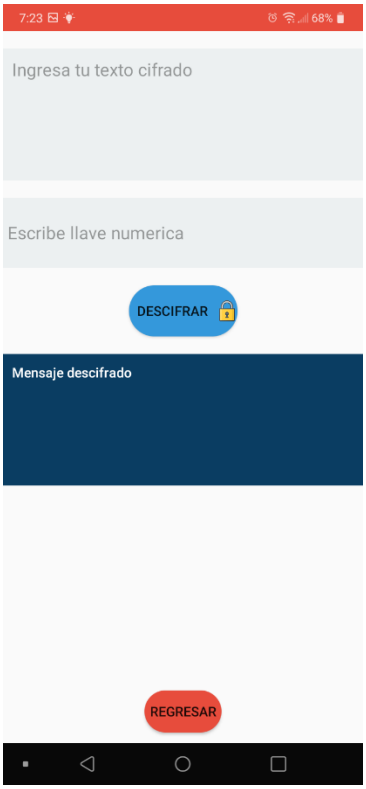

Figura 5.5 Ventana descifrar

5.2 CONCLUSION

La implementación del diseño siguiendo las especificaciones del Capítulo 4 garantiza una interfaz coherente, intuitiva y atractiva para el usuario. Cada sección ha sido detalladamente codificada para reflejar los wireframes y la guía de diseño previamente establecidos.

CAPÍTULO 6

6.1 Introducción a las Pruebas

Las pruebas de software son una parte fundamental del desarrollo de software, asegurando la calidad y fiabilidad del producto final. [15] enfatiza que "las pruebas de software no solo verifican que el software cumple con los requisitos especificados, sino que también aseguran que el software es fiable y está libre de defectos significativos". Las pruebas efectivas pueden identificar y corregir errores antes de que el software sea lanzado al mercado, reduciendo así el costo y el tiempo de desarrollo al prevenir defectos costosos y proporcionando una experiencia de usuario positiva.

Tipos de Pruebas Realizadas

Pruebas Funcionales

Las pruebas funcionales se centran en verificar que cada función del software opera de acuerdo con los requisitos especificados. [15] destaca que "las pruebas funcionales son cruciales para asegurar que las características del software funcionen correctamente y cumplan con todas las especificaciones funcionales".

Pruebas de Usabilidad

Las pruebas de usabilidad evalúan qué tan fácil es para los usuarios interactuar con el software.[15] explica que "las pruebas de usabilidad permiten mejorar la experiencia del usuario, asegurando que el software sea intuitivo y fácil de usar". Estas pruebas pueden revelar problemas que no se detectan mediante pruebas funcionales.

Pruebas de Rendimiento

Las pruebas de rendimiento examinan cómo se comporta el software bajo diversas condiciones de carga. [15] menciona que "las pruebas de rendimiento ayudan a identificar cuellos de botella y áreas de mejora, asegurando que el software sea rápido, estable y escalable" (p. 102). Estas pruebas son vitales para asegurar que el software puede manejar el número esperado de usuarios y transacciones.

6.2 Plan de Pruebas

Descripción del Plan de Pruebas

El plan de pruebas tiene como objetivo garantizar que la aplicación cumpla con los requisitos definidos y que funcione de manera efectiva y segura en dispositivos móviles. Este plan se enfoca en la validación de que los mensajes cifrados sean enviados y recibidos sin alteraciones y que la confidencialidad de la información se mantenga.

Objetivos del Plan de Pruebas

- **Verificar la integridad de los mensajes cifrados**: Asegurar que los mensajes enviados sean los mismos que se reciban, sin ninguna alteración.
- **Garantizar la confidencialidad de la información**: Probar que los datos altamente confidenciales, como la información de clientes o cuentas bancarias, permanezcan protegidos durante el intercambio.
- **Evaluar el funcionamiento del algoritmo de cifrado de flujo**: Confirmar que el software implemente correctamente el algoritmo de cifrado de flujo en dispositivos móviles.

Estrategias de Pruebas

- **Pruebas Unitarias**: Se realizarán pruebas unitarias para verificar el correcto funcionamiento de cada componente del sistema.
- **Pruebas de Integración**: Evaluar cómo interactúan los diferentes módulos del sistema.
- **Pruebas de Sistema**: Verificar el comportamiento del sistema completo en diferentes entornos operativos.
- **Pruebas de Aceptación del Usuario (UAT)**: Realizar pruebas con usuarios finales para asegurar que el software cumple con sus expectativas y requisitos.

Criterios de Aceptación

- **Exactitud de los Mensajes**: Todos los mensajes cifrados y enviados deben ser descifrados correctamente en el destino sin alteraciones.
- **Confidencialidad**: Los datos confidenciales deben permanecer seguros y sin accesos no autorizados.
- **Compatibilidad**: El software debe funcionar en los sistemas operativos móviles definidos (Android).
- **Usabilidad**: La interfaz debe ser intuitiva y fácil de usar para los usuarios.

6.3 Pruebas Funcionales

Describir las pruebas funcionales realizadas para asegurar que cada característica de la aplicación funciona según lo esperado.

Casos de Prueba y Resultados

- **Caso de Prueba 1: Cifrado y Envío de Mensajes**
    - **Descripción**: Probar el cifrado de un mensaje y su envío.

- **Resultado Esperado**: El mensaje cifrado debe ser recibido y descifrado correctamente.
- **Entradas del Caso de Prueba**:
  - Mensaje **de prueba**: "Hola Mundo"
  - Clave **de cifrado**: "clave123"
- **Pasos que seguir**:
  i. En la Figura 6.3.1 se muestra cómo se eligió la opción cifrar, se escribió el mensaje y la clave, se puede observar el mensaje cifrado.

Figura 6.3.1 Cifrando mensaje

  ii. Después se selecciona enviar el mensaje, y se debe especificar el correo y el asunto, como en la Figura 6.3.2

Figura 6.3.2 Enviando mensaje

i. Cuando se selecciona “LISTO”, se muestra la aplicación del correo para enviarlo y lo reciba el receptor, como se observa en la Figura 6.3.3.

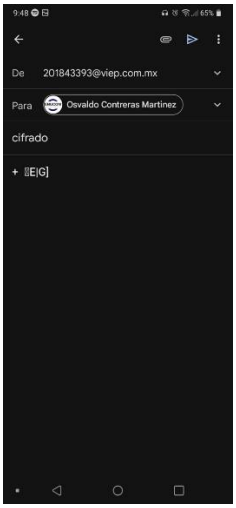

Figura 6.3.3 Aplicación de correo

i. Se recibe el correo, con el mensaje cifrado, se copia el mensaje y se selecciona la opción de descifrar en la aplicación y se escribe la clave y se descifra, así como se muestra en la Figura 6.3.4.

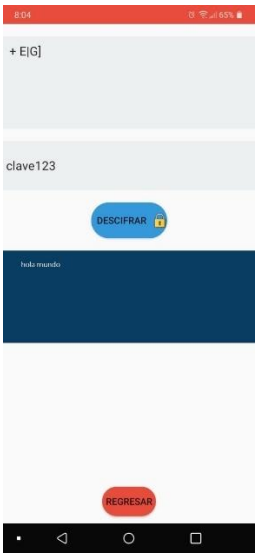

Figura 6.3.4 Descifrando el mensaje

- **Resultado Real**: Al cifrar el mensaje funciona correctamente y al enviarlo funciona de manera correcta, ahora al recibirlo y copiarlo y poner la misma clave con la que fue cifrado el mensaje se pudo leer sin complicaciones.

6.4 Pruebas de Usabilidad

Detalles sobre las pruebas de usabilidad, incluyendo metodología y participantes.

Se creo un link para llenar un formulario y se envió la aplicación en .apk, a varias personas con las siguientes preguntas:

¿Cuál es su edad?

18-25

26-35

36-45

46-55

56+

¿Cuál es su nivel de experiencia con aplicaciones móviles?

Principiante

Intermedio

Avanzado

¿Fue fácil navegar por la aplicación?

Muy fácil

Fácil

Neutro

Difícil

Muy difícil

¿Encontró alguna dificultad para comprender cómo usar la aplicación?

Si

No

Si respondió "Sí", por favor describa la dificultad:

¿Las características principales de la aplicación funcionaron según lo esperado?

Si

No

¿Experimentó algún error o problema técnico?

Si

No

Si respondió "Sí", por favor describa el error:

¿Qué funcionalidades adicionales le gustaría ver en la aplicación?

¿Cómo calificaría la velocidad de la aplicación?

Muy rápida

Rápida

Neutra

Lenta

Muy lenta

¿Qué tan satisfecho está con la aplicación en general?

Muy satisfecho

Satisfecho

Neutro

Insatisfecho

Muy Insatisfecho

¿Recomendaría esta aplicación a otros usuarios?

Si

No

Por favor, escriba cualquier comentario adicional o sugerencia de mejora:

Estas preguntas se realizaron para saber la opinión de diferentes usuarios, así como saber un poco sobre ellos y que tan fácil o difícil es de usar la aplicación.

Para validar que los usuarios fueran reales se pidió un correo para llenar el formulario después de probar la aplicación, como se ve en la Figura6.4.1, si no se ingresaba correo se solicitaba forzosamente.

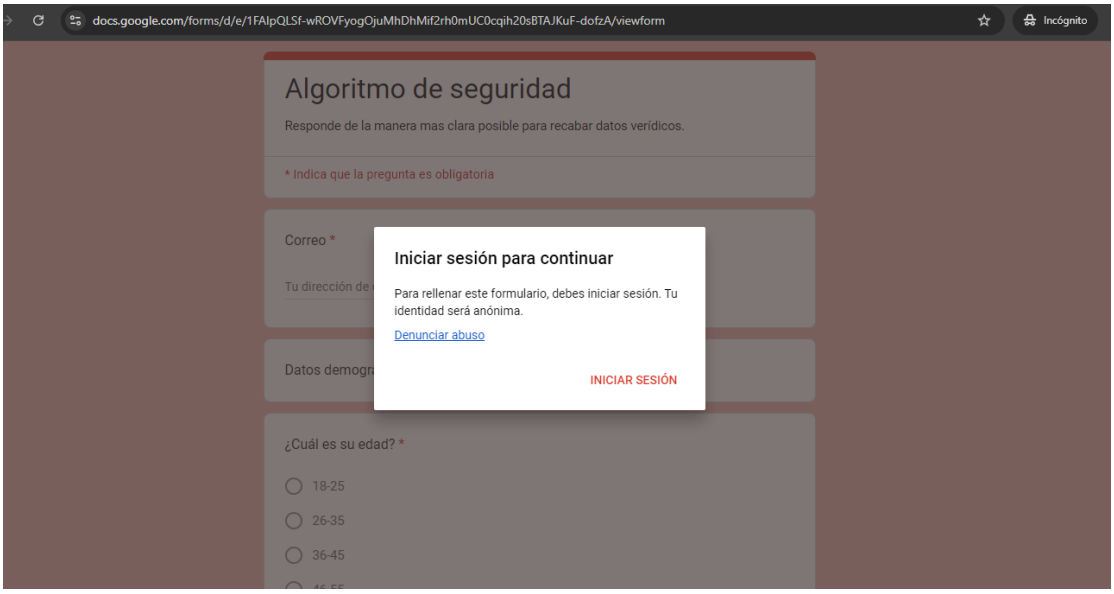

Figura 6.4.1 Prueba de correos

Resultados y Análisis

- **Metodología**: Para evaluar la facilidad de uso de la aplicación, se llevaron a cabo pruebas de usabilidad con 6 participantes divididos en tres grupos:

novatos (poca experiencia con aplicaciones móviles), **intermedios** (usuarios regulares de dispositivos móviles) y **avanzados** (usuarios frecuentes de aplicaciones móviles). Los participantes realizaron las siguientes tareas: cifrar un mensaje, descifrar un mensaje y navegar por el menú principal de la aplicación. Las pruebas fueron observadas directamente y los participantes también completaron una breve entrevista posterior a la prueba con las preguntas anteriores.

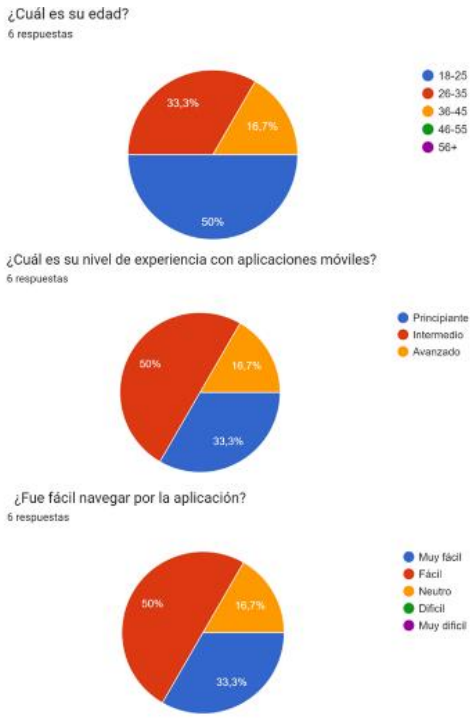

Figura 6.4.2 Respuestas

La Figura 6.4.2 muestra tres gráficos circulares que representan los resultados de una encuesta con seis respuestas relacionadas con la usabilidad de una aplicación móvil.

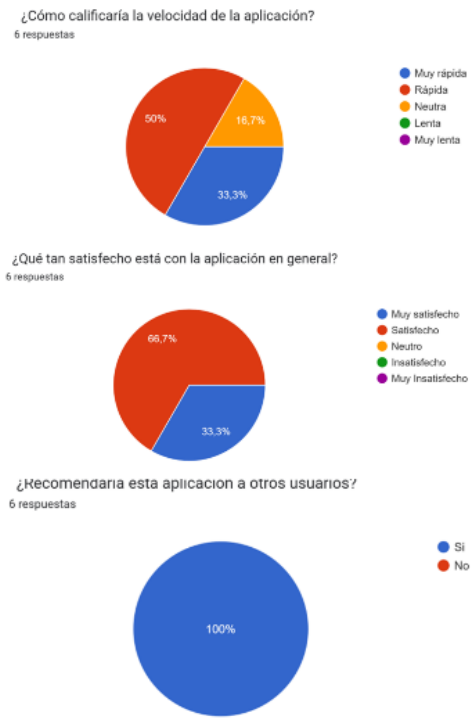

Figura.6.4.3 Respuestas 2

En la Figura 6.4.3 se siguen mostrando los gráficos con las preguntas correspondientes, e igual con las para llegar a una conclusión sobre la aplicación.

Figura 6.4.4 Respuestas finales

Y para concluir esta parte de la usabilidad se pidió la opinión de los usuarios sobre la aplicación y esto también nos ayuda a saber que está bien en el desarrollo de la aplicación y que se puede mejorar.

Cabe mencionar que todas las respuestas se fueron monitoreando para que se lleve esta aplicación a un público más grande, pero respecto a estas pruebas en personas cercanas se puede obtener las siguientes sugerencias, que para seguir mejorando este proyecto se tomaran en cuenta, no todos los usuarios opinaron:

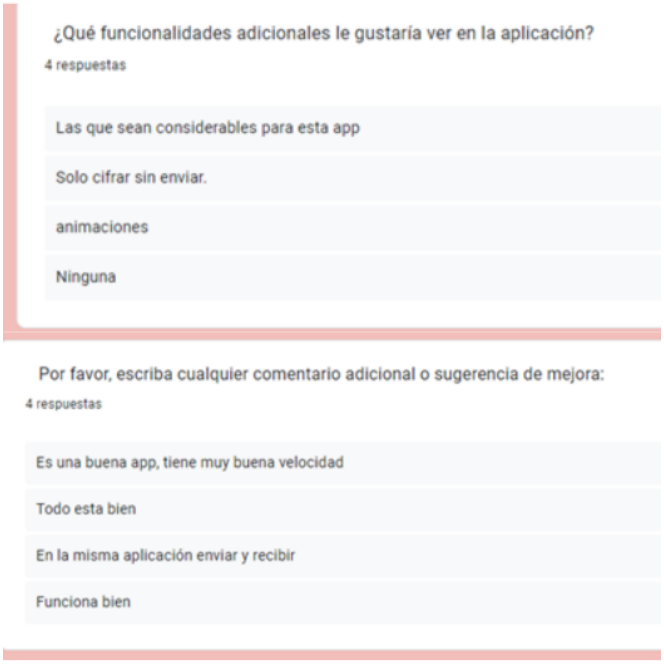

Figura 6.4.5 Sugerencias

Resultados:
Los participantes con experiencia **avanzada** completaron las tareas rápidamente, sin errores, y calificaron la aplicación como intuitiva. Los participantes **intermedios** encontraron algunas dificultades iniciales con la navegación, pero lograron completar las tareas después de intentarlo un par de veces. Los **novatos** necesitaron más tiempo para entender la función de cifrado y descifrado, pero finalmente lograron enviar y recibir mensajes sin complicaciones.

- **Tiempo promedio de cifrado**: 2 minutos para usuarios avanzados, 3 minutos para usuarios intermedios y 5 minutos para usuarios novatos.

- **Errores comunes**: Usuarios novatos no entendieron inicialmente el uso de la llave de cifrado.
- **Comentarios**: "La aplicación es simple, pero el concepto de la llave para cifrar necesita más explicación."

Conclusión:
La aplicación es **intuitiva** para usuarios con diferentes niveles de experiencia. Sin embargo, se sugiere incluir una breve guía o tutorial para usuarios que no están familiarizados con conceptos como el cifrado y el uso de llaves, para mejorar la experiencia de usuarios novatos.

6.5 Pruebas de Rendimiento

Descripción de las pruebas de rendimiento realizadas para evaluar la eficiencia de la aplicación.

Resultados y Análisis

- **Pruebas de Carga**: Dado que la aplicación no utiliza un modelo cliente-servidor, las pruebas de carga se enfocaron en evaluar cómo se comporta la aplicación cuando se somete a un alto número de operaciones locales en un solo dispositivo. Simulamos operaciones de cifrado y descifrado con un dispositivo de gama media-baja además de mantener varias aplicaciones en ejecución para verificar la estabilidad de la aplicación.
- **Pruebas de Estrés**: Para las pruebas de estrés, sometimos la aplicación a condiciones extremas, como procesar mensajes cifrados y descifrados de gran longitud y realizar operaciones continuas durante largos periodos de tiempo.
- **Metodología**: Se cifrará un texto grande, que será una canción popular y se mantendrán aplicaciones ejecutándose además que la clave será de longitud grande también:

Texto: Estas son las mañanitas
Que cantaba el rey David
Hoy, por ser día de tu santo
Te las cantamos aquí

Despierta, mi bien, despierta
Mira que ya amaneció
Ya los pajaritos cantan
La luna ya se metió

Clave: 123456789101112131415

A continuación, se muestran las características del equipo en el que se realizaran las pruebas de estrés y de cargar para validar el dispositivo y sus características, así se muestra en la figura 6.4.6, cabe mencionar que esta aplicación incluso debería funcionar en un dispositivo con versiones de Android más rezagadas.

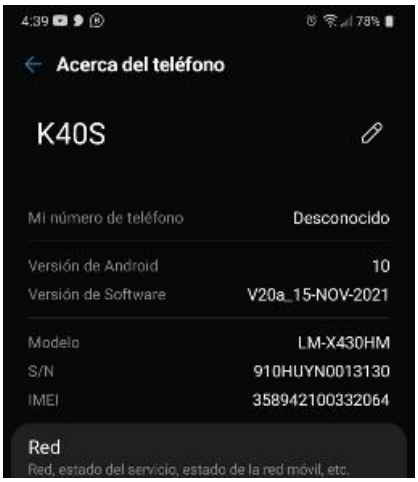

Figura 6.4.6 Dispositivo Móvil

Ahora se muestra las aplicaciones ejecutándose en el dispositivo al momento de cifrar el mensaje, así como el reproductor de música y varias aplicaciones que se pueden visualizar en la parte superior del dispositivo:

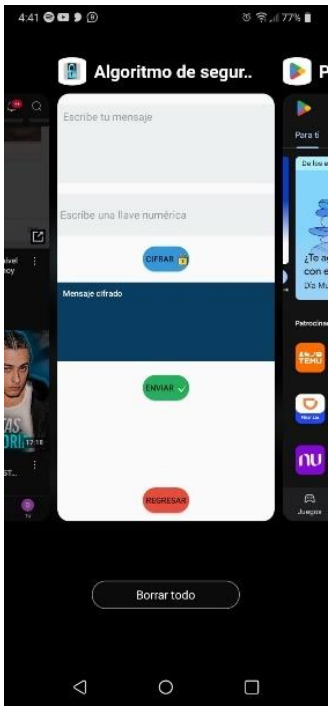

Figura 6.4.7 Aplicaciones ejecutándose

Ahora en la Figura 6.4.8 se puede observa como se cifra el mensaje para enviarlo al usuario receptor, como se puede notar el mensaje ha sido cifrado sin ningún inconveniente, al igual que el envió que abre la aplicación de correo electrónico.

Figura 6.4.8 Cifrado de mensaje

Se ha recibido el mensaje cifrado en el correo especificado, como se muestra en la siguiente Figura 6.4.9:

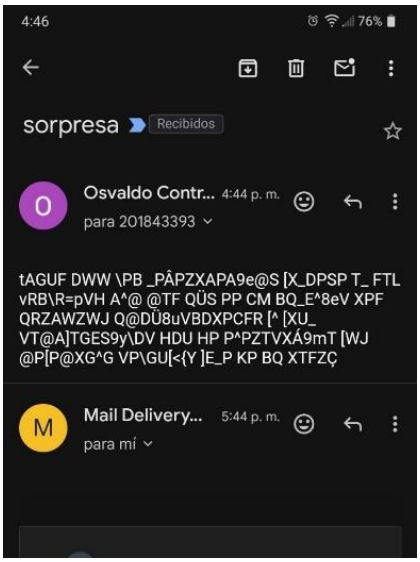

Figura 6.4.9 Recepción de mensaje

Finalmente se muestra el mensaje descifrado en la aplicación, de la siguiente manera en la Figura 6.4.10:

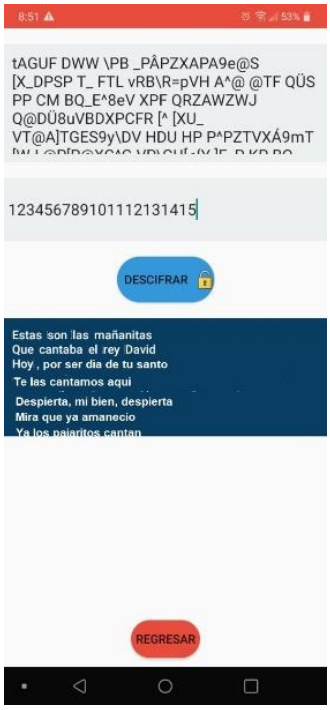

Figura 6.4.10 Descifrado

Resultados: La aplicación funcionó correctamente bajo estas condiciones, sin experimentar fallas importantes ni pérdida de rendimiento significativa.

Conclusión

Los resultados de las pruebas de rendimiento muestran que la aplicación es eficiente y robusta, capaz de manejar múltiples operaciones consecutivas y condiciones extremas sin afectaciones en su rendimiento.

CAPÍTULO 7

7.1 Resumen del Proyecto

El proyecto tuvo como objetivo diseñar e implementar un software en el sistema operativo Android que permitiera el cifrado y envío seguro de mensajes utilizando un algoritmo de cifrado de flujo. Los objetivos iniciales incluían la selección del sistema operativo adecuado, la implementación eficiente del algoritmo, y la garantía de que los mensajes cifrados se mantuvieran seguros durante la transmisión.

Los objetivos fueron alcanzados exitosamente: se seleccionó Android como la plataforma de desarrollo debido a su amplia adopción y flexibilidad; se implementó el algoritmo de cifrado basado en la función XOR, asegurando que los mensajes cifrados fueran recibidos sin alteraciones.

Entre los desafíos principales, se destacó la integración del algoritmo de cifrado en un entorno móvil y la optimización del rendimiento de la aplicación. Estos desafíos se superaron mediante la adopción de prácticas de codificación eficientes y la realización de pruebas exhaustivas en diferentes dispositivos y versiones de Android.

7.2 Lecciones Aprendidas

El proceso de desarrollo de esta aplicación reveló la importancia de una planificación detallada y la necesidad de realizar pruebas exhaustivas en las primeras etapas del proyecto. Una lección clave aprendida fue la necesidad de considerar la seguridad desde el diseño inicial, asegurando que la implementación del algoritmo de cifrado fuera tanto efectiva como segura.

Recomiendo a futuros desarrolladores que trabajen en proyectos similares, que realicen un análisis exhaustivo de los requisitos de seguridad y que dediquen tiempo a la optimización del código para garantizar que la aplicación funcione de manera eficiente en una variedad de dispositivos. También es esencial documentar cada paso del proceso de desarrollo para facilitar futuras actualizaciones o migraciones a otras plataformas.

7.3 Impacto del Proyecto

La aplicación desarrollada ha tenido un impacto significativo en la seguridad de la comunicación entre usuarios, resolviendo el problema de la transmisión segura de mensajes en dispositivos móviles. La implementación del algoritmo de cifrado de flujo ha garantizado que los mensajes transmitidos sean confidenciales y lleguen sin alteraciones.

Los usuarios que han probado la aplicación han expresado su satisfacción con la facilidad de uso y la tranquilidad que proporciona saber que sus mensajes están protegidos. Estos testimonios reflejan el éxito del proyecto en cumplir con su objetivo de mejorar la seguridad en la comunicación móvil.

7.4 Trabajo Futuro

Aunque la aplicación ha alcanzado sus objetivos principales, existen áreas de mejora que podrían explorarse en el futuro. Una posible mejora es optimizar aún más el rendimiento del algoritmo de cifrado en dispositivos de gama baja, así como ampliar la compatibilidad de la aplicación con futuras versiones de Android.

En cuanto a futuras funcionalidades, se podría considerar la integración de cifrado para mensajes multimedia, la incorporación de autenticación biométrica para mayor seguridad, y la expansión de la aplicación a otros sistemas operativos móviles.

El mantenimiento a largo plazo del proyecto incluirá la corrección de posibles errores, la actualización continua del algoritmo de cifrado y la adaptación a las nuevas versiones del sistema operativo Android.

7.5 Conclusión

En conclusión, este proyecto ha sido una contribución importante al campo de la seguridad en la comunicación móvil, demostrando que es posible cifrar y transmitir mensajes de manera segura en dispositivos Android. A nivel personal y profesional, el desarrollo de esta aplicación ha sido una experiencia enriquecedora que ha mejorado mis habilidades técnicas y mi comprensión de los desafíos de la seguridad en las aplicaciones móviles.

Apéndice A: código Inicio

INICIO

```
<?xml                    version="1.0"                    encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:layout_width="809dp"
        android:layout_height="933dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.5"
        app:layout_constraintHorizontal_bias="0.5"
        app:srcCompat="@drawable/fondo"                                    />

    <Button
        android:id="@+id/Inicio"
        android:layout_width="175dp"
        android:layout_height="102dp"
        android:background="@drawable/rounded_button"
        android:fontFamily="@font/roboto_bold"
        android:onClick="inicio"
        android:text="Iniciar"
        android:textAppearance="@style/TextAppearance.AppCompat.Display2"
        android:textColor="@color/black"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.5"
        app:layout_constraintHorizontal_bias="0.5"
        tools:text="Iniciar"                                               />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Apéndice B: Código Menú

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Menu"
    android:background="@color/black">

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="150dp"
        android:background="@drawable/rounded_button"
        android:fontFamily="@font/lato"
        android:onClick="cifrar1"
        android:text="@string/cifrar"
        android:textColor="@color/white"
        android:textSize="20sp"
        app:layout_constraintBottom_toTopOf="@+id/button3"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:ignore="UsingOnClickInXml" />

    <Button
        android:id="@+id/button3"
        android:layout_width="133dp"
        android:layout_height="52dp"
        android:layout_marginTop="50dp"
        android:background="@drawable/rounded_button3"
        android:fontFamily="@font/lato"
        android:onClick="descifrar"
        android:text="@string/descifrar"
        android:textColor="@color/black"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.5" />
```

```
<Button
        android:id="@+id/button4"
        android:layout_width="123dp"
        android:layout_height="53dp"
        android:layout_marginTop="80dp"
        android:background="@drawable/rounded_button2"
        android:backgroundTint="@color/rojo"
        android:fontFamily="@font/lato"
        android:onClick="regresar"
        android:text="@string/regresar"
        android:textColor="@color/white"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/button3"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button3"
        app:layout_constraintVertical_bias="0.137"                    />

    <ImageView
        android:id="@+id/imageView3"
        android:layout_width="655dp"
        android:layout_height="944dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.5"
        app:layout_constraintHorizontal_bias="0.5"
        app:srcCompat="@drawable/fmenu"                               />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Apéndice C: Código Cifrar

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Cifrar">

    <EditText
        android:id="@+id/et1"
        android:layout_width="0dp"
        android:layout_height="150dp"
        android:layout_marginTop="20dp"
        android:gravity="start|top"
        android:hint="Escribe tu mensaje"
        android:inputType="textMultiLine"
        android:background="#ECF0F1"
        android:padding="10dp"
        app:layout_constraintBottom_toTopOf="@+id/et2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/et2"
        android:layout_width="0dp"
        android:layout_height="80dp"
        android:hint="Escribe una llave numérica"
        android:padding="5dp"
        android:background="#ECF0F1"
        android:layout_marginTop="20dp"
        android:inputType="text"
        app:layout_constraintBottom_toTopOf="@+id/button5"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/et1" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="Cifrar"
        android:onClick="cifrar"
        android:drawableEnd="@drawable/ic_bloquear_layer"
        android:background="@drawable/rounded_button3"
        app:layout_constraintTop_toBottomOf="@+id/et2"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
```

```
    <TextView
        android:id="@+id/tv"
        android:layout_width="0dp"
        android:layout_height="150dp"
        android:text="Mensaje                                                cifrado"
        android:textColor="@color/white"
        android:background="#0A3D62"
        android:padding="10dp"
        android:textAppearance="@style/TextAppearance.AppCompat.Body2"
        app:layout_constraintTop_toBottomOf="@+id/button5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="20dp"                                    />

    <Button
        android:id="@+id/button9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enviar"
        android:onClick="enviar"
        android:background="@drawable/rounded_button"
        android:drawableEnd="@drawable/ic_verificar_layer"
        app:layout_constraintTop_toBottomOf="@+id/tv"
        app:layout_constraintStart_toStartOf="@+id/button6"
        app:layout_constraintEnd_toEndOf="@+id/button6"
        android:layout_marginTop="26dp"                                    />

    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Regresar"
        android:onClick="regresar2"
        android:background="@drawable/rounded_button2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginBottom="16dp"                                 />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Apéndice D: Código Enviar

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".mensajeC">

    <Button
        android:id="@+id/button7"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="30dp"
        android:onClick="regresar3"
        android:text="regresar"
        android:background="@drawable/rounded_button2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.969" />

    <EditText
        android:id="@+id/ecorreo"
        android:layout_width="0dp"
        android:layout_height="50dp"
        android:layout_marginTop="20dp"
        android:gravity="start|top"
        android:hint="Ingresa correo destino"
        android:background="#ECF0F1"
        android:padding="10dp"
        android:inputType="textEmailAddress"
        app:layout_constraintBottom_toTopOf="@+id/eAsunto"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/eAsunto"
        android:layout_width="0dp"
        android:layout_height="80dp"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="445dp"
        android:ems="10"
        android:hint="Asunto"
        android:background="#ECF0F1"
        android:padding="10dp"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toTopOf="@+id/button7"
        app:layout_constraintEnd_toEndOf="parent"
```

```
    <TextView
        android:id="@+id/tvContenido"
        android:layout_width="0dp"
        android:layout_height="110dp"
        android:layout_marginTop="60dp"
        android:layout_marginBottom="434dp"
        app:layout_constraintBottom_toTopOf="@+id/button7"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="@+id/eAsunto"
        app:layout_constraintTop_toBottomOf="@+id/eAsunto"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/button8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="80dp"
        android:onClick="listo"
        android:text="Listo"
        android:background="@drawable/rounded_button"
        android:drawableEnd="@drawable/ic_verificar_layer"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvContenido" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Apéndice E: Código Descifrar

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".descifrar">



    <EditText
        android:id="@+id/ecifrado"
        android:layout_width="0dp"
        android:layout_height="150dp"
        android:layout_marginTop="20dp"
        android:gravity="start|top"
        android:hint="Ingresa tu texto cifrado"
        android:inputType="textMultiLine"
        android:background="#ECF0F1"
        android:padding="10dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"  />

    <EditText
        android:id="@+id/ekey"
        android:layout_width="0dp"
        android:layout_height="80dp"
        android:hint="Escribe llave numerica"
        android:padding="5dp"
        android:background="#ECF0F1"
        android:layout_marginTop="20dp"
        android:inputType="text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/ecifrado" />

    <Button
        android:id="@+id/button11"
        android:layout_width="124dp"
        android:layout_height="58dp"
        android:layout_marginTop="20dp"
        android:background="@drawable/rounded_button3"
        android:drawableEnd="@drawable/ic_bloquear_layer"
        android:onClick="descifra"
        android:text="Descifrar"
        app:layout_constraintEnd_toEndOf="@+id/ekey"
        app:layout_constraintStart_toStartOf="@+id/ekey"
        app:layout_constraintTop_toBottomOf="@+id/ekey" />
```

```
    <TextView
        android:id="@+id/tvDescifra"
        android:layout_width="0dp"
        android:layout_height="150dp"
        android:text="Mensaje                                    descifrado"
        android:textColor="@color/white"
        android:background="#0A3D62"
        android:padding="10dp"
        android:textAppearance="@style/TextAppearance.AppCompat.Body2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button11"
        android:layout_marginTop="20dp"/>

    <Button
        android:id="@+id/button10"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="12dp"
        android:onClick="regresar"
        android:text="Regresar"
        android:background="@drawable/rounded_button2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"                    />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Referencias

[1]. Luis Joyanes Aguilar,” Fundamentos de programación: algoritmos, estructura de datos y objetos ”, McGraw-Hill/Interamericana Editores, S.A. de C.V., 2020, Edición 5, ISBN 6071514681, 9786071514684

[2]. Anívar Chaves Torres,”Aprenda a Diseñar Algoritmos”, Universidad Nacional Abierta y a Distancia, 2017, ISBN 9586516229, 9789586516228

[3]. ALEGRE RAMOS, MARIA DEL PILAR, GARCÍA-CERVIGÓN HURTADO, ALFONSO, ”Seguridad informática”, Editorial Paraninfo,2011,

ISBN 8497328124, 9788497328128

[4]. Álvaro Gómez Vieites,” Enciclopedia de la Seguridad Informática. 2ª edición”, Grupo Editorial RA-MA, 2011, ISBN 8499643949, 9788499643946

[5]. Enrique Mandado Pérez, Enrique Mandado y Yago Mandado,” Sistemas Electrónicos Digitales”, Marcombo, 2007,ISBN 8426714307, 9788426714305

[6]. Leandro Alegsa,”Cifrador de flujo”, *Enciclopedia Alegsa*, 20/12/2021, vol.1, No.1.

[7]. Ariel Maiorano,” Criptografía: Técnicas de desarrollo para profesionales”, Alpha Editorial, 2009,ISBN 6077073792, 9786077073796

[8]. MarcomboS.A.-ACCESO RÁPIDO-Mundo electrónico,” Telecomunicaciones Móviles”, Marcombo, 1998,ISBN 8426711499, 9788426711496

[9]. María Ángeles Caballero Velasco, Diego Cilleros Serrano,” El libro del Hacker. Edición 2022”, Comercial Grupo ANAYA, S.A., 2021, ISBN 8441544441, 9788441544444

[10]. Beth, T.; Piper, F. C. (1985). «The Stop-and-Go-Generator». En Beth, Thomas, ed. *Advances in Cryptology*. Lecture Notes in Computer Science (en inglés) (Springer): 88-92. ISBN 978-3-540-39757-1. doi:10.1007/3-540-39757-4_9. Consultado el 26 de enero de 2021.

[11]. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC Press.

[12]. Schneier, B. (1996). *Applied cryptography: Protocols, algorithms, and source code in C* (2nd ed.). John Wiley & Sons.

[13]. Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson Education.

[14]. National Institute of Standards and Technology. (1993). *Secure Hash Standard (SHS)*. FIPS PUB 180.

[15]. Chauhan, N. (2010). *Software Testing: Principles and Practices*. Oxford University Press.