BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FAC. DE CIENCIAS DE LA COMPUTACIÓN

“ALGORITMOS DE BLOQUE IMPLEMENTADOS EN SEGURIDAD DE DISPOSITIVOS MÓVILES”

TESIS

PARA OBTENER EL GRADO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

MAURICIO DÍAZ AVILA

ASESORA DE TESIS

DRA. BÁRBARA EMMA SÁNCHEZ RINZA

OCTUBRE 2024

ÍNDICE

CAPÍTULO 1
ALGORITMO DE BLOQUES

1.1 Introducción

La criptografía se refiere a la disciplina que se encarga de utilizar códigos y métodos de cifrado para asegurar la privacidad de los mensajes. El proceso de cifrado transforma los mensajes de tal manera que solo la persona a quien están dirigidos pueda entender su contenido. Esta técnica se aplica frecuentemente para resguardar la información mientras se transmite [1].

El cifrado es un método reversible, lo que significa que cualquier transformación aplicada al mensaje debe poder deshacerse para recuperar el contenido original.

En la actualidad, hay dos clases principales de algoritmos de cifrado: simétricos y asimétricos.

- Los algoritmos simétricos son también conocidos como de "clave secreta", mientras que los asimétricos se llaman de "clave pública". La principal diferencia entre ellos, es que los simétricos disponen de una misma clave para cifrar y descifrar, y los asimétricos utilizan claves distintas para cada proceso [1].

El cifrado por bloques es un método que trabaja sobre segmentos de texto en lugar de procesar cada bit individualmente. Cada segmento tiene un tamaño uniforme y consiste de un número determinado de bits [1].

En esta tesis se explorará lo que es el cifrado por bloques, incluyendo sus principios de diseño y los modos en que opera. Igualmente, se analizarán algunos ejemplos de algoritmos, describiendo su funcionamiento e ilustrando su aplicación para facilitar su comprensión [1].

1.2 Objetivo general

Se busca hacer un análisis y evaluar la implementación del algoritmo de cifrado AES en dispositivos móviles, con el fin de determinar su efectividad, eficiencia y seguridad como método de protección de datos sensibles en la comunicación y almacenamiento.

Se busca estudiar el rendimiento del algoritmo en términos de consumo de recursos (energía, memoria y procesamiento), así como su resistencia a ataques comunes, y comparar su desempeño con otros algoritmos de cifrado de bloques. Además, se pretende explorar su impacto en la experiencia con el usuario y proponer mejoras y/o buenas prácticas para optimizar su implementación en entornos móviles modernos.

1.3 El cifrado de bloques

El cifrado de bloques, conocido también como ***Block Cipher***, es una técnica de encriptación que segmenta el texto original en unidades de tamaño fijo, donde cada unidad contiene la misma cantidad de bits, con este tipo de cifrado se trabaja individualmente con cada bloque de texto original, utilizando una clave para generar el bloque cifrado equivalente [2].

Un mensaje a cifrar, denominado texto claro, se compone de un número específico de bits que se organizan en bloques de tamaño uniforme. La encriptación por bloques transforma cada uno de estos bloques en otro bloque del mismo tamaño. Este método es el más empleado actualmente en el cifrado simétrico, especialmente por el apoyo del gobierno estadounidense [2].

Los cifradores de bloque que utilizan claves simétricas son elementos fundamentales en varios sistemas de criptografía, estos no solo aseguran la confidencialidad de manera independiente, sino que su versatilidad permite su uso en la creación de generadores de números aleatorios y cifradores de flujo, entre otras funciones [2].

En el proceso de descifrado, únicamente se necesita trabajar con un bloque cifrado para recuperar el texto original correspondiente. Un ejemplo destacado de este método es el estándar de cifrado de datos (*DES*) [2].

1.4 Descripción

Es una función que mapea un bloque de texto plano de n-bits a un bloque de texto cifrado de n-bits. Donde n es la longitud del bloque [2].

La función es parametrizada por una clave K de k-bits, tomando valores de un subconjunto del conjunto Vk de todos los vectores de k-bits. Asumiendo siempre que la clave es escogida al azar, los bloques de texto plano y texto cifrado de igual longitud evitan la expansión de los datos. Además, para permitir el cifrado y el

descifrado, la función debe de ser uno a uno; es decir, esta función debe ser invertible [2].

Aunque los cifradores de bloque suelen manejar texto claro en tamaños bastante grandes, por ejemplo, n ≥ 128, y los cifradores de flujo generalmente trabajan con unidades más pequeñas, no existe una diferencia precisa entre ambos. Esto se debe a que los cifradores de bloque pueden emplearse también para realizar cifrado en modo de flujo.

Así como se puede observar en el ejemplo de la **figura 1.4.1**, el cómo se invierte el cifrado y descifrado a si mismo [2].

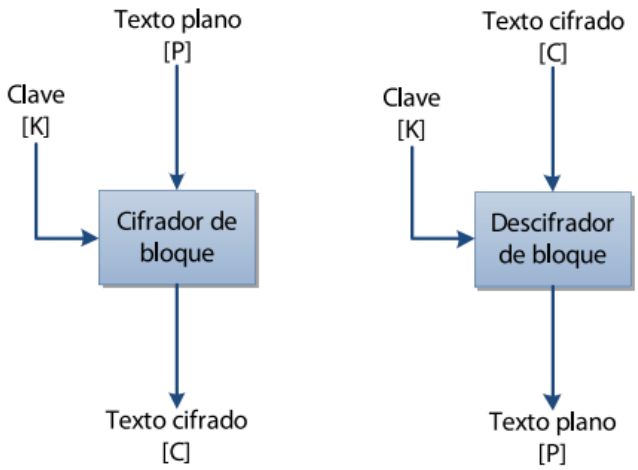


Figura 1.4.1: Cifrado y descifrado de bloque.

1.5 Algoritmo DES

El texto sin formato se presenta en bloques de 64 bits en tamaño. DES trabaja en cada bloque de texto sin formato a la vez. Por lo tanto, a cada bloque se le aplica una clave para que el texto cifrado de ese bloque en particular sea de tamaño 64 bits. Por otro lado, durante el descifrado, también se opera en un bloque de texto cifrado a la vez para producir sus 64 bits correspondientes de texto sin formato [2].

Se debe tener en cuenta que en DES, el algoritmo de descifrado es el mismo que el de cifrado.

En general, los cifrados de bloques normalmente operan en un bloque de bits a la vez en lugar de un bit a la vez. El cifrado de bit a bit es poco práctico ya que es un proceso largo, y dado que la encriptación por bloques se basa en algoritmos de computadora, se debe ser rápidos. Es por ello, que operar en un bloque de bits a la vez hace que el cifrado sea más rápido que el cifrado de flujo [2].

Se debe mencionar una pequeña limitación del cifrado de bloque. En el caso de texto repetido, generaría el mismo texto cifrado. Sin embargo, fue resuelto por el cifrado en cadena.

1.6 Cómo se diseña el algoritmo de bloques

Existen tres aspectos críticos para el diseño de un cifrado de bloque:

1. **Número de rondas**

El número de rondas es una forma de juzgar la fuerza del algoritmo de cifrado de bloques. Se puede pensar en que, cuanto mayor sea el número de rondas, será más difícil que el criptoanálisis rompa el algoritmo, incluso si la función F es relativamente poco robusta. El número de rondas esté allí para hacer molesta la tarea de vulnerar el algoritmo [2].

2. **Diseño de la función F**

La función F en el cifrado por bloques es tal cuál para que ningún criptoanálisis pueda revertir la sustitución, al igual que no debe ser lineal, ya que, si desencriptar fuera lineal, entonces no sería útil tenerlo. Dicho de otra forma, cuanta más no linealidad tenga la función, más difícil de romper será, también es vital asegurarse de que posee una propiedad de avalancha adecuada, la propiedad de avalancha estipula que, si se cambia un bit en la entrada, se deben cambiar varios bits en la salida [2].

La función también debe diseñarse de manera que tenga un criterio de independencia de bits que dice que los bits de salida deben cambiar de manera independiente sin importar qué bits están cambiando en la entrada [2].

3. **Algoritmo de programación clave**

Se prefiere que el esquema de generación de claves garantice la propiedad de avalancha e independencia del bit. Por ejemplo, el algoritmo de generación de claves en el esquema DES, divide una clave de 56 bits en dos partes de 28 bits cada una. De manera similar, en algoritmo *Serpent*, codifica una clave de 256 bits en 132 palabras y cada palabra es de 32 bits [2].

1.7 Funcionamiento de los cifrados por bloque

Como se mencionó previamente, este es un tipo de cifrado donde el texto plano se fragmenta primero en varios bloques, cada uno con un tamaño fijo. Básicamente, cada bloque contiene la misma cantidad de bits de información. En cada etapa, el proceso de encriptación trabaja sobre un solo bloque de texto plano y aplica la clave de cifrado para transformarlo en un bloque de texto encriptado [2].

Todos los bloques tienen la misma longitud. Un bloque de 160 bits de texto sin formato se cifra en 2 bloques de 64 bits cada uno y el tercer bloque tendrá el saldo de 32 bits. Luego, se rellenará con 32 bits para hacerlo una unidad porque debe tener el mismo tamaño [2].

Además, el descifrado opera en un solo bloque de texto cifrado para convertirlo de nuevo en texto sin procesar. Un ejemplo de cifrado de bloques muy útil DES, el algoritmo estándar de cifrado de datos desarrollado por el Instituto Nacional de Estándares y Tecnología es un cifrado de bloques con claves simétricas [2].

1.8 Beneficios del algoritmo de bloques

A diferencia de operar en un solo bit de datos, el criptosistema trata todo el bloque de datos simultáneamente, es muy rápido en comparación con muchas otras técnicas criptográficas. Dado que el tamaño del bloque es constante, esto no compromete la seguridad de los datos cifrados [2].

El tamaño se mantiene constante para todos los bloques.

No se puede tener un bloque más pequeño (lo que facilita que los hackers descifren su bloque a través de un ataque de diccionario) ni mayores, lo que conduciría a una operación no viable. [2].

1.9 Modos de operación

Para los mensajes de texto plano que exceden la longitud del bloque, se usan varios modos de operación para los cifradores de bloque. Por ejemplo, cuando un mensaje excede el tamaño del bloque, la forma más fácil de llevar a cabo el cifrado es dividiendo o particionando este mensaje en bloques de n-bits y así cifrar por separado uno a uno [3].

Este enfoque utilizado (conocido como ECB) es uno de los modos de operación disponibles para abordar problemas de texto sobrante.

Algunos de los modos más utilizados son:

- Electronics codebook (ECB).
- Cipher-block chaining (CBC).
- Cipher feedback (CFB).
- Output feedback (OFB).

Se explicará cada uno de manera individual, para comprender cómo es posible realizar las operaciones.

a) Electronics codebook (ECB).

Este es el modo de cifrado más básico. En ECB, el mensaje se segmenta en bloques, y cada uno se cifra por separado. La desventaja de este método radica en que bloques idénticos de texto plano generarán bloques idénticos de texto cifrado. Debido a esto, no garantiza una verdadera confidencialidad y no se recomienda para su uso en protocolos criptográficos [3].

Aquí se puede observar un ejemplo en imagen de cómo funciona el cifrado ECB (**figura 1.9.1**).

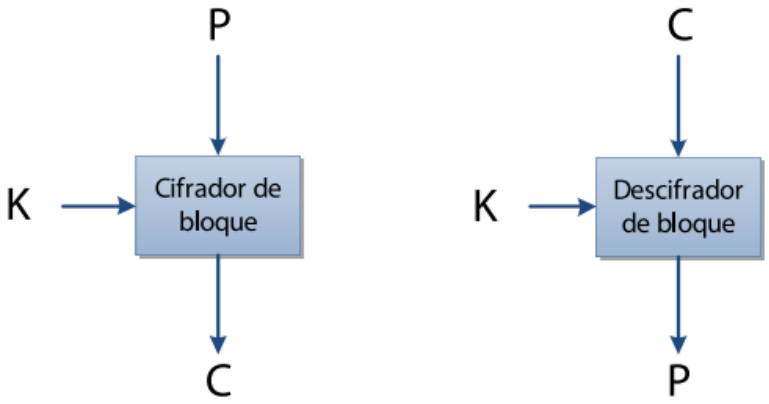


Figura 1.9.1: Electronics codebook (ECB).

La **figura 1.9.2** ilustra claramente cómo el método ECB puede exponer patrones en el texto. Una versión de mapa de bits de la imagen de la izquierda fue cifrada utilizando ECB, generando la imagen en el centro [3].

La imagen de la derecha muestra cómo se vería la misma imagen si fuera cifrada con CBC, CTR u otros métodos, apareciendo como ruido aleatorio. Sin embargo, es importante señalar que la apariencia aleatoria de la imagen de la derecha no garantiza que el cifrado sea seguro. Varios métodos inseguros de cifrado pueden producir resultados aparentemente aleatorios desde el inicio [3].

El modo ECB puede aumentar la vulnerabilidad de los protocolos sin protección de integridad a los ataques de repetición, ya que cada bloque se descifra de forma idéntica [3].

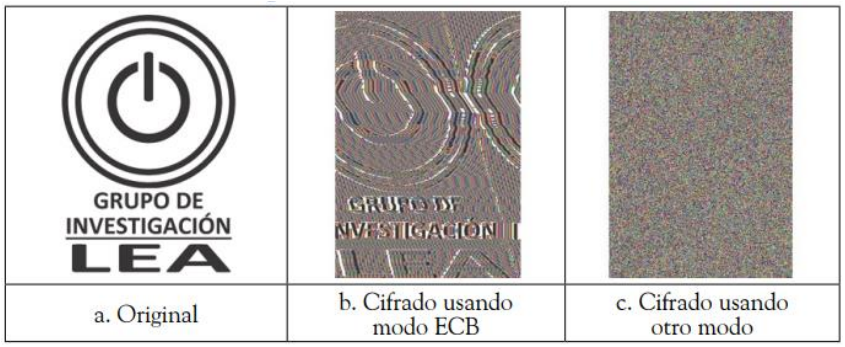


Figura 1.9.2: Patrones del modo ECB.

b) Cipher-block chaining (CBC)

A cada bloque de texto plano se le aplica la operación XOR con el bloque cifrado anterior antes de ser cifrado. Esto hace que cada bloque de texto cifrado dependa del texto plano total hasta el punto del mensaje actual. Para que cada uno sea único y utilizado sobre el vector de inicialización, se usa para evitar un cifrado idéntico en mensajes encriptados más largos [3].

En la **figura 1.9.3** se puede observar cómo funciona el cifrado CBC.

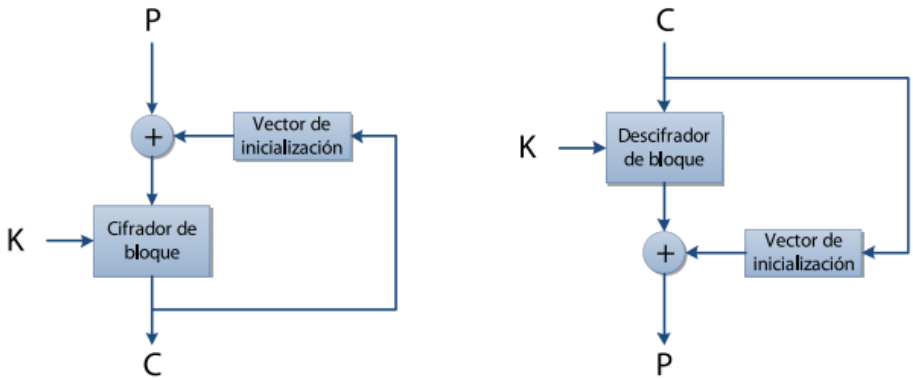


Figura 1.9.3: Cipher-block chaining (CBC).

Mediante esta técnica, cada bloque encriptado se retroalimenta para encriptar el siguiente bloque. El texto plano se combina mediante una operación XOR con el bloque encriptado previo, encriptando a continuación el resultado. El encriptado de cada bloque depende de todos los bloques anteriores [3].

Esto obliga a que, en el momento del desencriptado, todos los bloques se procesen de forma secuencial. Se requiere un vector de inicialización aleatorio, el cual no necesita ser confidencial, para procesar el primer bloque. Un error en un bloque causa una distorsión completa en dicho bloque [3].

El bloque siguiente presentará errores en las mismas posiciones que el bloque anterior, aunque los bloques posteriores ya no se verán afectados.

c) Cipher feedback (CFB).

Mientras el modo CBC procesa texto plano de n-bits en un tiempo (usando un cifrador de bloques de n-bits), algunas aplicaciones requieren que r-bits del texto plano sean encriptados y transmitidos sin ningún tipo de retraso, para algún r<n (frecuentemente r=1 o r = 8), es en este caso que se usa el cifrador de realimentación o Cipher Feedback (CFB) [3].

Se puede observar una muestra del cómo se usa el algoritmo CFB, en la **figura 1.9.4** y también su funcionamiento.

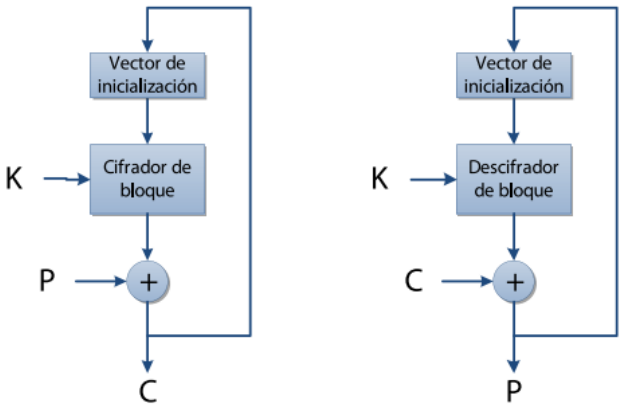


Figura 1.9.4: Cipher feedback (CFB).

El modo de retroalimentación de cifrado es una variante de cifrado por bloques en la que se utiliza el texto encriptado (o una parte de este) como retroalimentación para encriptar nuevamente. Este resultado se mezcla con el texto original mediante una operación XOR, generando así el siguiente bloque encriptado [3].

Cuando se retroalimentan n bits, el cifrado se denomina en modo de retroalimentación de n bits. Su uso más común está en la transmisión encriptada de datos. En protocolos basados en caracteres, se suelen usar n=8, mientras que, en protocolos orientados al bit, se toma n=1 [3].

El proceso de **cifrado** es el siguiente:

1. Se inicializa el registro de desplazamiento a IV, suponiendo que la longitud del registro de desplazamiento es en bits.
2. El encriptador y la clave secreta encriptan el registro de desplazamiento para obtener Ki (i = 1, 2, 3...).
3. La longitud del texto plano es m (m ≤ longitud) bits, que es XOR con los m bits más altos de K1 para obtener texto cifrado de m bits.
4. Se mueve el registro de desplazamiento a la izquierda con m bits y a continuación se tienen que llenar los m bits más bajos del registro de desplazamiento con el texto cifrado de m bits que se acaban de obtener.
5. Se tienen que repetir los pasos 2 a 4 hasta que todo el texto plano esté encriptado.

Dado que el modo, cifra el texto cifrado, por lo tanto, al descifrar, también se tiene que usar el cifrador para descifrar [3].

Se explicará a continuación el proceso de **descifrado**:

1. Se inicializa el registro de desplazamiento a IV, suponiendo que la longitud del registro de desplazamiento es en bits.

2. El encriptado y la clave secreta encriptan el registro de desplazamiento para obtener Ki (i = 1, 2, 3...).
3. La longitud del texto cifrado es m (m ≤ longitud) bits, que es XOR con los m bits más altos de K1 para obtener texto plano de m bits.
4. Mueva el registro de desplazamiento hacia la izquierda en m bits y complete el texto cifrado de m bits anterior en los m bits más bajos del registro de desplazamiento.
5. Repetir los pasos 2 a 4 hasta que se descifren todos los textos cifrados.

Al observar el paso de descifrado, podemos ver que el proceso puede ser paralelo. Convierte el cifrado de bloque en un modo de cifrado de flujo de sincronización automática. Los datos de texto sin formato pueden tener cualquier longitud de bit m, y el texto cifrado correspondiente también es m bits, por lo que no es necesario dividir el texto sin formato en bloques de datos de bit fijo, al igual que no es necesario rellenar todo el texto sin formato. Se puede cifrar bit a bit, por lo que se puede ver como una forma de utilizar cifrados de bloque para implementar cifrados de flujo [3].

Ventajas

1. El modo de texto sin formato está oculto.
2. El cifrado de bloque se convierte en el modo de transmisión.
3. Los datos más pequeños que el propio paquete se pueden cifrar y transmitir a tiempo.

Desventajas

1. No es adecuado para la computación en paralelo.
2. Transmisión de error: el daño a una unidad de texto sin formato afecta a múltiples unidades.
3. El único IV.

d) Output feedback (OFB).

Retroalimentación de salida, modo de retroalimentación de salida. El modo de retroalimentación de salida es similar al de retroalimentación de texto encriptado, con la diferencia de que la salida se transmite. El modo OFB también transforma el cifrado por bloques en un modo de cifrado continuo [3].

El diagrama de flujo de cifrado del modo OFB se observa en la **figura 1.9.5**.

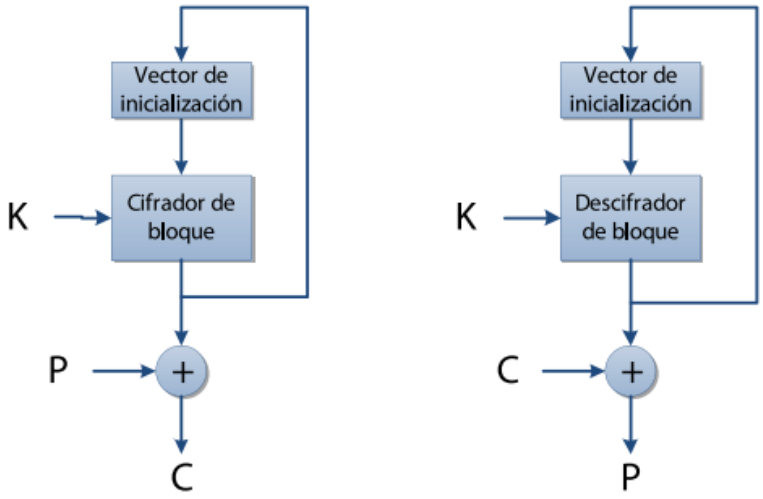


Figura 1.9.5: Output feedback (OFB).

El modo de operación OFB puede usarse en aquellas aplicaciones que se quiera evitar el error de propagación. Es similar al CFB y permite la encriptación de varios tamaños del bloque, pero difiere en que en este caso la salida del bloque de encriptación sirve como realimentación [4].

El proceso de **cifrado** es el siguiente:

1. Inicialice el registro de desplazamiento a IV, suponiendo que la longitud del registro de desplazamiento es en bits.

2. El encriptado y la clave secreta encriptan el registro de desplazamiento para obtener Ki (i = 1, 2, 3...).
3. La longitud del texto plano es m (m ≤ longitud) bits, que es XOR con los m bits más altos de K1 para obtener texto cifrado de m bits.
4. Mueva el registro de desplazamiento hacia la izquierda por m bits y llene los m bits altos de Ki que acaba de obtener en los m bits bajos del registro de desplazamiento.
5. Repita los pasos 2 a 4 hasta que todo el texto plano esté encriptado.

Al descifrar, también se usa el cifrador para descifrar.

El proceso de descifrado del modo OFB se muestra a continuación: se tiene que tener en cuenta que todo es diferente del proceso de cifrado [4].

Proceso de **descifrado**:

1. Inicialice el registro de desplazamiento a IV, suponiendo que la longitud del registro de desplazamiento es en bits.
2. El encriptado y la clave secreta encriptan el registro de desplazamiento para obtener Ki (i = 1, 2, 3...).
3. La longitud del texto cifrado es m (m ≤ longitud) bits, que es XOR con los m bits más altos de K1 para obtener texto plano de m bits.
4. Mueva el registro de desplazamiento a la izquierda en m bits, y llene los bits altos m del Ki anterior en los bits m bajos del registro de desplazamiento.
5. Repita los pasos 2 a 4 hasta que se descifren todos los textos ingresados.

Ventajas

1. El modo de texto sin formato está oculto.

2. El cifrado de bloque se convierte en el modo de transmisión.
3. Los datos más pequeños que el paquete se pueden cifrar y transmitir a tiempo.

Desventajas

1. No es lo correcto para la computación paralela.
2. Es posible un ataque activo en texto sin formato.
3. Transmisión de error: el daño de una unidad de texto sin formato afecta a varias unidades.

1.10 Posibles ataques

El propósito de un cifrador de bloques es garantizar la confidencialidad. La meta de un atacante, por el contrario, es obtener el texto original a partir del texto encriptado. Un cifrador de bloques se considera completamente comprometido si la clave es descubierta, y parcialmente comprometido si un atacante puede obtener una parte del texto original (sin descubrir la clave) a partir del texto cifrado [4].

Es fundamental que, al analizar la seguridad de un cifrador de bloques, se contemple a un atacante con las siguientes habilidades y conocimientos:

- **Acceso al canal del texto cifrado:** Se acepta que el adversario puede interceptar todos los datos transmitidos a través del canal de comunicación, es decir, tiene acceso completo a los mensajes cifrados

- **Conocimiento completo del sistema de cifrado:** En relación con el principio de Kerckhoffs, se asume que el adversario conoce todos los detalles de la función de encriptación, a excepción de la clave secreta. Esto implica que la seguridad del sistema descansa únicamente en la confidencialidad de la clave [4].

Los ataques son clasificados en base a la información que tenga acceso el criptoanalista, además del texto cifrado interceptado. Las clases de ataques más importantes para los cifradores de clave simétrica son los siguientes (para clave fija) [4].

- **Ataques de sólo texto cifrado**

En este caso, el atacante no tiene conocimiento alguno del contenido del mensaje y debe trabajar únicamente a partir del texto ya encriptado, sin acceso a información adicional.

Los sistemas de cifrado actuales no son vulnerables a ataques de 'solo texto encriptado', aunque en algunos casos podría asumirse que el mensaje presenta una 'tendencia' estática [4].

- **Ataques de texto plano conocidos**

El atacante conoce o puede adivinar el texto de alguna parte del texto cifrado. La tarea es obtener la clave utilizando sólo esta información [4].

Esto puede ser hecho utilizando el texto plano original o a través de algún atajo. Es decir, las parejas de texto plano y texto cifrado están disponibles.

- **Ataques de texto plano escogido**

El atacante puede tener cualquier texto encriptado con una clave desconocida. Y su tarea es determinar la clave utilizada para cifrar [4].

En otras palabras, el texto encriptado sirve como opción de texto plano para el adversario.

Un ataque de texto encriptado elegido se basa en el siguiente escenario: bajo la suposición de una clave desconocida, se permite que el atacante acceda a un par de texto plano y su correspondiente texto encriptado para recopilar información. Luego, se intenta utilizar esa información para descubrir la clave o el texto plano asociado a un nuevo texto encriptado [4].

Con pocas excepciones, la mejor defensa contra los ataques avanzados actuales en cifradores prácticos es la complejidad.

Los elementos que pueden diferenciarse dentro de esta complejidad son:

- **Complejidad de los datos:** hace referencia al número de unidades de datos de entrada requerido; por ejemplo, en un cifrador de bloque de 128 bits, la complejidad de los datos es de 128 bits [4].

- **Complejidad de almacenamiento:** hace referencia al número de unidades del almacenamiento requerido [4].

- **Complejidad de procesamiento:** hace referencia al número de operaciones exigido para procesar los datos de la entrada y/o almacenamiento de datos, por lo menos una unidad de tiempo por la unidad almacenamiento [4].

1.11 Algoritmo de cifrado de datos internacional (IDEA)

IDEA funciona con bloques de 64 bits empleando una clave de 128 bits y se compone de ocho transformaciones idénticas (llamadas rondas) y una transformación final (conocida como media ronda) [5].

El proceso de encriptado y desencriptado es similar. Gran parte de la seguridad de IDEA proviene de la combinación de operaciones de diferentes tipos: suma y multiplicación modular, junto con XOR bit a bit, que son algebraicamente 'incompatibles' entre sí de cierta manera [5].

El algoritmo utiliza tres operaciones en su proceso con las cuáles logra la confusión, se realizan con grupos de 16 bits y son las siguientes:

- Operación O-exclusiva (XOR) bit a bit (indicada con un círculo plus azul).
- Suma módulo 2^{16} (indicada con una caja plus verde).
- Multiplicación módulo $2^{16}+1$, donde la palabra nula (0x0000) se interpreta como 2^{16} (indicada con un círculo punteado rojo).

  Por ejemplo: $2^{16} = 65536$; $2^{16}+1 = 65537$, que es primo.

Después de realizar 8 rondas completas viene una 'media ronda', y cuyo resultado se puede ver en la **figura 1.11.1** [5].

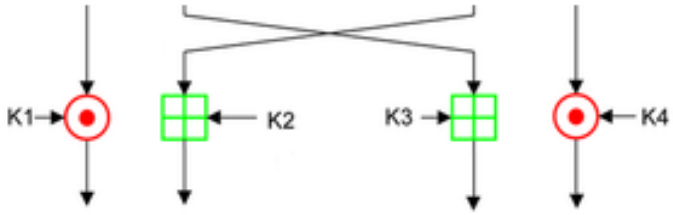


Figura 1.11.1: Salida de media ronda del algoritmo IDEA.

Este algoritmo presenta diferencias notables con el algoritmo DES, que lo hacen más atractivo:

- El espacio de claves es mucho más grande: $2^{128} \approx 3.4 \times 10^{38}$.
- Todas las operaciones son algebraicas.
- Es más eficiente que los algoritmos de tipo Feistel, porque a cada vuelta se modifican todos los bits de bloque y no solamente la mitad.
- Se pueden utilizar todos los modos de operación definidos para el algoritmo DES.

IDEA es un algoritmo bastante seguro, y hasta ahora se ha mostrado resistente a múltiples ataques, entre ellos el criptoanálisis diferencial. No presenta claves débiles, y su longitud de clave hace imposible en la práctica un ataque por la fuerza bruta [5].

Como ocurre con todos los algoritmos simétricos de cifrado por bloques, se basa en los conceptos de confusión y difusión, haciendo uso de las siguientes operaciones elementales [5].

- XOR.
- Suma modulo 2 (base 16).
- Producto modulo 2 (base 16) + 1.

1.12 Seguridad

Los diseñadores evaluaron la resistencia de IDEA frente al criptoanálisis diferencial y concluyeron que, bajo ciertos supuestos, es resistente. No se han identificado o registrado vulnerabilidades lineales o algebraicas exitosas [6].

Desde 2007, el mejor ataque para todas las claves logró comprometer el algoritmo reducido a 6 rondas (completo utiliza 8.5 rondas). Es importante notar que una 'ruptura' se define como cualquier ataque que requiere menos de 2^{128} operaciones; el ataque a 6 rondas necesita 2^{64}, textos planos conocidos y $2^{126,8}$ operaciones.

Bruce Schneier consideraba que IDEA era excelente en 1996 y comentó: "En mi opinión, es el algoritmo de bloque más seguro y efectivo disponible para el público en este momento" (Criptografía aplicada, 2ª ed.) (5). Sin embargo, en 1999 dejó de recomendarlo debido a la aparición de algoritmos más rápidos, algunos progresos en su análisis y temas relacionados con las patentes [6].

En 2012, se logró comprometer la versión completa de 8.5 rondas de IDEA, mediante un ataque de bi-ciclos estrechos, reduciendo su resistencia criptográfica en unos 2 bits, un efecto similar al ataque previo de bi-ciclos en AES, sin embargo, este ataque no afecta la seguridad práctica del propio algoritmo [6].

1.13 Estructura

La estructura general de IDEA sigue el diseño Lai-Massey, donde el XOR se emplea tanto para la suma como para la resta, al igual que utiliza una función semicircular que depende de la clave. Para procesar palabras de 16 bits (es decir, 4 entradas en lugar de 2 para el bloque de 64 bits), se aplica el esquema de Lai-Massey dos veces en paralelo, con dos funciones de ronda paralelas que se entrelazan [6].

Para garantizar una adecuada difusión, dos de los sub-bloques se intercambian después de cada ronda.

En la **figura 1.13.1** podremos ver un poco de la estructura del algoritmo IDEA [6].

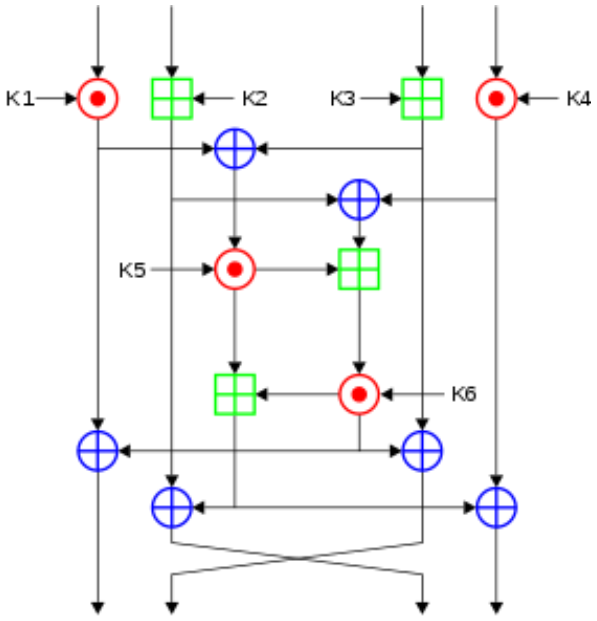


Figura 1.13.1: Estructura gráfica del algoritmo IDEA.

CAPÍTULO 2
ALGORITMO AES

2.1 Objetivo

Al comenzar este proyecto, se propuso una serie de pasos a seguir para conseguir implementar el AES y hacer estudios sobre su consumo de potencia para así poder trabajar sobre ella.

Este método se considera muy seguro y eficiente y sirve para cifrar los datos de todo tipo y por eso, se suele usar en varios protocolos y técnicas de transmisión. P.ej. la protección WPA2 de las redes Wifi utiliza también el Advanced Encryption Standard como p. ej. el estándar SSH o IPsec. Frecuentemente se usan los métodos AES también en la telefonía a través del internet (Voice over IP, VoIP) con el fin de asegurar los datos de señalización o también los datos útiles [6].

2.2 ¿Qué es el algoritmo AES?

Es fundamental entender cómo opera el AES, analizando las distintas transformaciones que se aplican a los datos originales [6].

Definido por el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés) bajo la Publicación de Estándares Federales de Procesamiento de Información 197, el Estándar de Cifrado Avanzado (AES) establece un algoritmo criptográfico aprobado por FIPS para la protección de datos electrónicos [6].

El algoritmo AES es un cifrado simétrico de bloques que permite tanto el encriptado como el desencriptado de datos. La encriptación convierte los datos en un formato no legible llamado texto encriptado, mientras que el desencriptado los transforma de vuelta al texto original. Utiliza claves criptográficas de 128, 192 o 256 bits para encriptar y desencriptar datos en bloques de 128 bits [6].

2.3 Criptografía asimétrica

La criptografía asimétrica, también llamada de clave pública o de dos claves, es un método criptográfico en el que se emplean dos claves distintas para el intercambio de mensajes. Ambas claves pertenecen a una misma persona; una es pública y se puede compartir libremente, mientras que la otra es privada y debe mantenerse en secreto. Este método funciona de tal manera que el emisor utiliza la clave pública del receptor para encriptar el mensaje antes de enviarlo, y luego el receptor emplea su clave privada para desencriptar el mensaje. De esta forma, se resuelve el problema de la criptografía simétrica relacionado con la seguridad en el intercambio de claves y garantiza la confidencialidad, ya que solo el destinatario puede acceder a la información [6].

Si el propietario del par de claves es el emisor y utiliza su clave privada para cifrar el mensaje, cualquier persona puede descifrarlo con su clave pública, lo que asegura la autenticación del mensaje, confirmando que el remitente es quien afirma ser [6].

La principal limitación de este sistema es la vulnerabilidad de las claves a los ataques de fuerza bruta. Mientras que en un sistema de clave simétrica se recomienda una longitud de clave de 128 bits, en criptografía de clave pública actualmente se sugieren claves de al menos 1024 bits para una mayor seguridad [6].

La **figura 2.3.1** pretende explicar de manera sencilla las diferencias entre criptografía simétrica y criptografía asimétrica.

Figura 2.3.1: Diferencias entre criptografía simétrica y asimétrica.

2.4 Definición del algoritmo

Como se mencionó previamente, AES es un cifrador por bloques de criptografía simétrica, es decir, funciona encriptando y desencriptando datos en bloques, utilizando la misma clave privada para ambos procesos.

El estándar, basado en el algoritmo Rijndael, divide los datos de entrada en bloques de 4 palabras de 32 bits, es decir, 4 × 32 = 128 bits. Cabe señalar que el algoritmo Rijndael también puede operar con bloques mayores de 192 y 256 bits, aunque estas opciones no están incluidas en el estándar [6].

Al observar la **figura 2.4.1**, se puede ver que tanto el texto original como el texto encriptado se segmentan en bloques de 128 bits, mientras que la longitud de la clave puede ser de 128, 192 o 256 bits. Más adelante se analiza cómo las diferentes longitudes de clave afectan el proceso de encriptado y desencriptado [6].

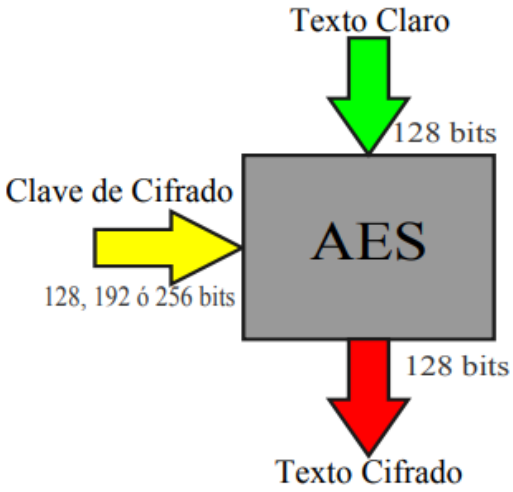


Figura 2.4.1: Cifrador AES.

La unidad fundamental de procesamiento en AES es el byte. Un byte es una secuencia de 8 bits, representada como {b7, b6, b5, b4, b3, b2, b1, b0}. Asimismo, un byte puede expresarse en un campo finito mediante un polinomio utilizando la siguiente fórmula [6]:

$$\sum_{i=0}^{7} b_i x^i$$

- El símbolo Σ representa una suma.
- La variable i es el índice de la suma, que comienza en 0 y termina en 7.
- b_i representa un coeficiente asociado a cada término en la serie.
- x^{i} es una potencia de la variable x, elevada al exponente i.

La fórmula general suma cada término de la forma $b_i x^i$, desde $i = 0$ hasta $i = 7$. Esto significa que estás sumando los siguientes términos:

$$b_0x^0 + b_1x^1 + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + b_6x^6 + b_7x^7$$

Este es un polinomio de grado 7 en la variable x, donde cada coeficiente bi determina el peso de cada potencia de x.

Por ejemplo, si partimos del siguiente byte {0010 1101}:

$$00101101 => b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0$$

Esto corresponde al polinomio:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

Sustituyendo los valores de los bits:

$$0x^7 + 0x^6 + 1x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0$$

Se eliminan los términos con coeficientes 0 (cero), daría como resultado:

$$\boldsymbol{x^5 + x^3 + x^2 + 1}$$

También es efectivo usar la notación hexadecimal para referirse a los bytes, dividiéndolos en dos grupos de cuatro bits, así como se muestra en la **tabla 2.4.2**.

Cadena de bits	Carácter hexadecimal	Cadena de bits	Carácter hexadecimal
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Tabla 2.4.2: Notación hexadecimal.

De esta manera, el byte {0010 1101}, puede ser representado por el número hexadecimal {2D}.

2.5 Funcionamiento del algoritmo AES

El algoritmo AES es un método de encriptación simétrica, lo que significa que emplea la misma clave tanto para encriptar como para desencriptar. Asimismo, es un algoritmo de cifrado en bloques que opera con segmentos de 128 bits. La clave puede tener diferentes longitudes: 128, 192 o 256 bits, y dependiendo de su tamaño, el algoritmo llevará a cabo un número específico de rondas, como se muestra en la **tabla 2.5.1** [7].

Longitud de Clave	Número de rondas
128	10
192	12
256	14

Tabla 2.5.1: Longitudes de claves.

Por lo tanto, podemos afirmar que AES es un cifrador de bloques iterativo, que aplica múltiples rondas de encriptación sobre una matriz de 4x4 bytes, denominada estado o *state* [7].

Tanto en el proceso de cifrado como en el de descifrado, en cada ronda se usa una subclave generada previamente mediante la función *KeyExpansion*, a partir de la clave de encriptación original. Esta función produce una subclave adicional al número total de rondas del algoritmo; esta subclave extra se emplea antes de iniciar el proceso completo de cifrado o descifrado en la matriz de estado, con el propósito de dificultar el criptoanálisis [7].

En todas las rondas, excepto en la última, el algoritmo realiza una serie de operaciones que van alterando la matriz de estado, añadiendo confusión y difusión al mensaje a encriptar.

A continuación, se plantean las operaciones que se aplican y después se describirán después más detalladamente [7].

- **Función SubBytes:** Tiene confusión en el proceso al realizar una sustitución byte a byte con propiedades óptimas no lineales.
- **Función ShiftRows:** Introduce difusión de información a la ronda mediante la rotación de las filas del estado.
- **Función MixColumns:** Permite un alto nivel de difusión mezclando las columnas entre sí.
- **Función AddRoundKey:** Incluye un grado de confusión que depende de la subclave de ronda.

2.6 Proceso de cifrado AES

El proceso implica aplicar cuatro funciones matemáticas reversibles a la información que se desea codificar. Estas funciones se repiten en cada ciclo. La información que se va a encriptar se organiza en una matriz de estado, sobre la cual se ejecutarán las transformaciones necesarias [8].

Antes de comenzar con la primera ronda, se aplica una transformación inicial a la matriz de estado mediante la función *AddRoundKey*, que consiste en una operación de OR exclusivo entre la primera subclave y la matriz de estado. Después de esta operación, se aplican cuatro transformaciones reversibles a la matriz de estado (SubBytes, ShiftRows, MixColumns y AddRoundKey), repitiendo estas cuatro operaciones hasta la penúltima ronda en lo que se denomina como ciclo estándar [8].

Finalmente, en la última ronda de encriptación, se aplican únicamente las funciones *SubBytes*, *ShiftRows* y *AddRoundKey*, en ese orden, sobre la matriz de estado resultante de los ciclos previos. El resultado de esta ronda final produce el bloque cifrado deseado [8].

- **Función SubBytes (subBytes())**

Tal y como se observa en la **figura 2.6.1**, consiste en una sustitución no lineal que se realiza a nivel de bytes. Se aplica independientemente sobre todos y cada uno de los bytes que conforman la matriz de estado, generando una nueva matriz de bytes [8].

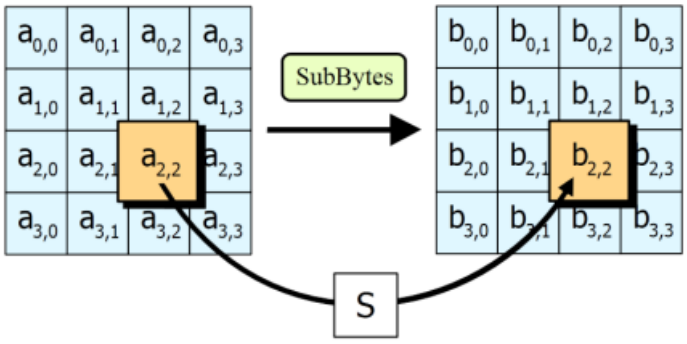


Figura 2.6.1: *SubBytes*.

Esta transformación consiste en la sustitución de cada byte por el resultado de aplicarle la tabla de sustitución *S-Box*, aparte de que es invertible y se construye mediante las siguientes dos transformaciones [8].

- Se sustituyen los elementos de la matriz de estado por sus inversos para la multiplicación en GF (2^8). Como el valor {00} necesita del inverso, en caso de aparecer, éste se sustituye por sí mismo.
- A continuación, se aplica la siguiente transformación afín en GF(2^8) al resultado de la anterior transformación [8].

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Nota: En esta transformación, cada valor representa el bit j del byte resultante de la primera operación, indicando su valor final. Así, cada valor refleja el bit j del byte que constituye el resultado de la transformación *SubBytes* [8].

Aplicando estas dos transformaciones a todos los valores de entrada posibles (256 valores, dado que se trata de un byte), se genera una tabla de sustitución llamada S-Box, que optimiza el proceso de encriptación.

- **Función *ShiftRows* (shiftRows())**

Esta operación consiste en un desplazamiento cíclico hacia la izquierda de los bytes en cada fila de la matriz de estado [8].

En este proceso, los bytes de la fila 0 permanecen en su posición original, mientras que los bytes de la fila 1 se desplazan una posición a la izquierda, los de la fila 2 se mueven dos posiciones, y los de la fila 3 avanzan tres posiciones a la izquierda. Por lo tanto, el resultado de esta transformación se representa en la **figura 2.6.2** [8].

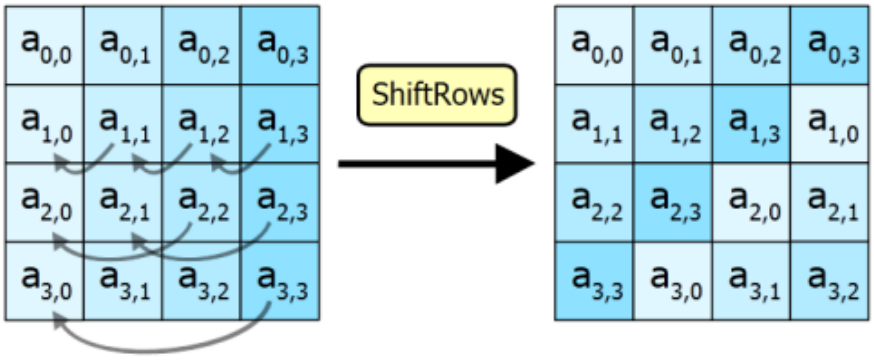


Figura 2.6.2: *ShiftRows*.

- **Función *MixColumns* (mixColumns())**

La transformación actúa sobre los bytes de una misma columna de la matriz de estado, lo que permite una mezcla de los bytes de las columnas, tal y como se muestra en la **figura 2.6.3** [8].

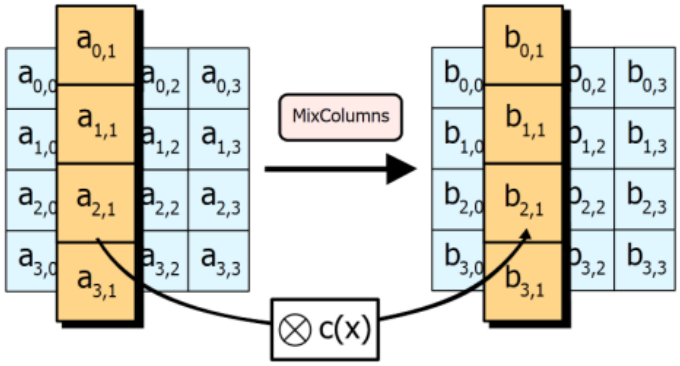


Figura 2.6.3: *MixColumns*.

En este proceso de transformación, las columnas de la matriz de estado son consideradas polinomios con coeficientes del campo GF(2^8). Estos polinomios se multiplican módulo M(x) = x^4 + 1 por el polinomio c(x) = {03}x^3 + {01}x^2 + {01}x + {02}.

Matricialmente, esta transformación se podría resumir de la siguiente manera; representando los coeficientes b_i, las columnas i de la matriz de estado de salida y los coeficientes de la matriz de estado de entrada, tal y como se muestra en la **figura 2.6.4** [8].

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Figura 2.6.4: Matriz de estado de entrada.

- **Función *AddRoundKey* (addRoundKey(subClave))**

Esta operación realiza un OR exclusivo entre los elementos de la matriz de estado que resultan de la transformación anterior y los elementos de la matriz correspondiente a la subclave de la ronda. Por esta razón, la matriz de la subclave tiene las mismas dimensiones que la matriz de estado, es decir, 4x4.

Esta función podría resumirse mediante la **figura 2.6.5** [8].

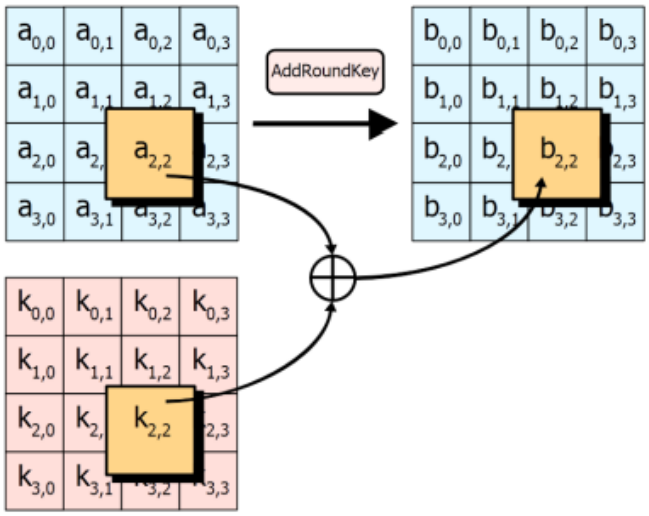


Figura 2.6.5: *AddRoundKey* (Operación OR - Exclusiva).

El bloque resultante de esta operación se convierte en la nueva matriz de estado para la siguiente ronda.

La importancia de esta función se basa en las subclaves. El algoritmo AES emplea distintas subclaves kj para que el resultado del proceso dependa totalmente de una información externa al sistema: la clave del usuario. Esto sigue el principio fundamental de la criptografía moderna, que indica que la seguridad de un algoritmo debe depender únicamente de la clave utilizada [8].

El mecanismo de la transformación *AddRoundKey* es simple; lo realmente interesante es entender el método para generar las distintas subclaves para cada

ronda (RoundKey), subclaves que se derivan de la clave principal k, para este propósito, se emplea la función *KeyExpansion* [8].

- **Función *KeyExpansion* (keyExpansion(clave))**

Como se mencionó anteriormente, la función *KeyExpansion* utiliza la clave de cifrado/descifrado para generar un vector de palabras de 4 bytes, que organizadas de a cuatro, formarán las subclaves de cada ronda. Este vector se conoce como Clave Expandida [8].

La clave principal contiene N_k palabras de 32 bits, con posibles valores para N_k de 4, 6, u 8. Las primeras N_k palabras de la clave expandida corresponden a la clave de cifrado original, mientras que las palabras restantes se calculan de manera recursiva a partir de las iniciales, con la función *KeyExpansion* dependiendo de N_k.

Tal como Daemen y Rijmen explicaron en su propuesta para AES, por razones de seguridad existen dos formas de ejecutar la función *KeyExpansion*, dependiendo del valor de Nk, que revisaremos en código C [8].

En esta descripción, la función *SubByte*(W) aplica la S-Box a cada byte de una palabra de 32 bits y devuelve los resultados en una nueva palabra. La función *RotByte*(W) rota la palabra W un byte hacia la izquierda [8].

De esta manera, en ambos códigos se puede observar que las primeras N_k palabras de la Clave Expandida corresponden a la clave principal. Las siguientes palabras, W(i), se calculan mediante una operación XOR entre la palabra anterior, *W(i-1)*, y la palabra $W(i - N_k)$. Si la posición es múltiplo de N_k (o en el caso de que N_k sea mayor que 6, cuando i – 4 es múltiplo de N_k), antes de realizar la XOR se aplica una transformación a *W(i - 1)*, que incluye el uso de las funciones *RotByte*(W) y *SubByte*(W), seguida de una operación XOR con una constante [8].

Las constantes aplicadas son generadas por la función Rcon de esta manera:

$$Rcon\ [i] = (RC\ [i], \{00\}, \{00\}, \{00\})$$

Siendo RC [i], un elemento perteneciente al Campo de Galois GF (2^8) con valor de $x^{(i - 1)}$ de manera que [8].

$$RC\ [1] = 1$$

$$RC\ [i] = x * (RC\ [i - 1]) = x^{(i - 1)}$$

2.7 Proceso de descifrado AES

Comprender el proceso de encriptación permite resumir fácilmente el de desencriptación. Este consiste en reemplazar cada operación del cifrado por su inversa y modificar el orden en que se aplican [9].

Así, el bloque a desencriptar se sitúa en la matriz de estado, la cual se somete a una serie de transformaciones. Antes de la primera ronda, se aplica la función *AddRoundKey* utilizando la última subclave. Luego, se realizan las rondas principales, que incluyen las operaciones InvShiftRows, *InvSubBytes*, *AddRoundKey* e *InvMixColumns* [9].

Finalmente, se ejecuta la ronda final, compuesta por las operaciones *InvShiftRows*, *InvSubBytes*, y *AddRoundKey*, esta última ronda produce el bloque desencriptado deseado [9].

Existen distintas variantes en el orden de aplicación de estas operaciones, todas llegando al mismo resultado final. Esto se debe a que, gracias a su funcionamiento interno, se puede intercambiar el orden entre las operaciones *InvShiftRows* y *InvSubBytes*, así como entre *AddRoundKey* e *InvMixColumns*, sin que ello afecte el resultado final [9].

Estas operaciones corresponden a las del proceso de encriptación, con algunos ajustes específicos que se explicarán a continuación.

- **Función *InvSubBytes* (invSubBytes())**

Para calcular la función inversa de *SubBytes*, bastará con aplicar una tabla inversa a la utilizada en el proceso de cifrado. Ésta se corresponde con la **tabla 2.7.1** [9].

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	9e	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	10	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Tabla 2.7.1: S-Box inversa (Caja de *SubBytes* inversos) [9].

- **Función *InvShiftRows* (invShiftRows())**

Para calcular la función *ShiftRows,* había que desplazar los bytes de las filas 1, 2 y 3, con un total de 1, 2 y 3 posiciones a la izquierda, para la función inversa habrá que desplazarlas hacia la derecha [9].

- **Función *InvMixColumns* (invMixColumns())**

En la función *InvMixColumns* las columnas de la matriz de estado se consideraban polinomios y se multiplican módulo $M(x) = x^4 + 1$ por el polinomio $c(x) = 3x^3 + x^2 + x + 2$. En la función inversa bastará con hacerlo por el inverso del polinomio c(x), el polinomio inverso (a(x)) se calcula mediante la definición de inverso [9]:

$$a(x) * c(x) = \{01\}$$

Y en este caso $a(x) = 11x^3 + 13x^2 + 9x + 14$.

Por lo tanto, la transformación se podría resumir matricialmente de la siguiente manera:

$$\begin{bmatrix} \boldsymbol{b_0} \\ \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} \mathbf{14} & \mathbf{11} & \mathbf{13} & \mathbf{09} \\ \mathbf{09} & \mathbf{14} & \mathbf{11} & \mathbf{13} \\ \mathbf{13} & \mathbf{09} & \mathbf{14} & \mathbf{11} \\ \mathbf{11} & \mathbf{13} & \mathbf{09} & \mathbf{14} \end{bmatrix} \begin{bmatrix} \boldsymbol{a_0} \\ \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix}$$

Representando los coeficientes b_i, las columnas i de la matriz de estado de salida y los coeficientes a_i, las de la matriz de estado de entrada [9].

- **Función *AddRoundKey* (addRoundKey(Subclave))**

La función inversa de la operación *AddRoundKey* es la misma función *AddRoundKey*. Esto es debido a que *AddRoundKey* realiza una operación OR-Exclusiva entre los elementos de la matriz de estado y los de la matriz de la subclave, y el resultado de una operación XOR es invertible con la propia operación X-OR [9].

- **Función *KeyExpansion* (keyExpansion(clave))**

Las distintas subclaves que se utilizan en el proceso de descifrado son las mismas que las del proceso de cifrado, con la única diferencia de que las distintas subclaves deben tomarse en orden opuesto. Por lo tanto, no es necesario modificar esta función, y bastará con ir utilizando las subclaves en orden inverso como se puede ver en el pseudocódigo del capítulo siguiente [9].

CAPÍTULO 3
DIAGRAMAS DE PROCESOS DE CIFRADO AES

3.1 Objetivo

Aplicar las técnicas de programación estructurada utilizando las herramientas de análisis estructurado de sistemas [10].

3.2 Diseños

El diagrama del proceso de cifrado en el sistema AES toma como entrada un archivo en formato BMP (imagen), lleva a cabo el cifrado o descifrado y genera como salida un archivo BMP con la imagen encriptada o desencriptada. Como se observa en la figura 3.2.1, no se emplea ningún método de compresión, ya que en ciertos países no se permite manipular información usando compresión, en cumplimiento con la Norma Oficial Mexicana 151-SCFI-2002. Además, esta decisión asegura que la imagen desencriptada sea idéntica a la original, sin pérdida alguna de información, ni siquiera de un bit. Por esta razón, se opta por el formato BMP, que es un formato sin pérdida. [10].

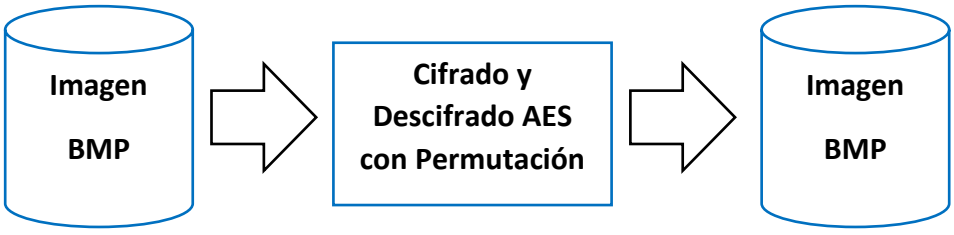


Figura 3.2.1: Estructura de las principales funciones del encriptado con AES y permutación variable.

3.3 Algoritmos

La **figura 3.3.1** muestra el algoritmo de cifrado AES con permutación variable, implementado en C++ para la aplicación de cifrado y descifrado.

El código se organiza en más de 80 funciones desarrolladas específicamente para cifrar y descifrar imágenes, siguiendo el estándar FIPS-197, y consta de casi 3,000 líneas de código en C++. Entre las funciones principales para el cifrado se encuentran: Cifrado_AES, SBytes, ShiftRows, MixColumns, las cuales son fundamentales en el proceso [11].

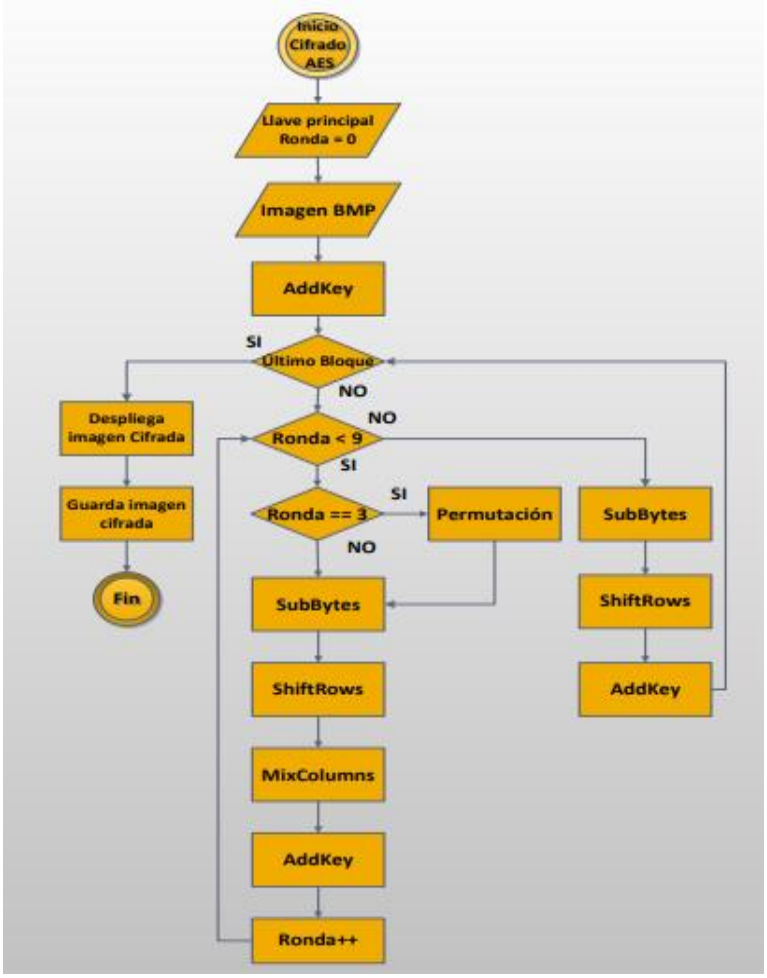


Figura 3.3.1: Algoritmo de Cifrado AES [11].

En la **figura 3.3.2** se incluyen funciones adicionales, como el agregado de claves y otras funciones complementarias que facilitan la interacción con el usuario mediante un sistema de menús. Estas funciones permiten gestionar archivos de datos e imágenes, su visualización, la administración de tablas, además de pruebas, generación de histogramas y medición de indicadores. Estas métricas permiten verificar la eficacia del sistema y evaluar su eficiencia en cuanto a tiempos de procesamiento, mostrando información sobre la dispersión de la imagen y el grado de recuperación una vez encriptada. [11].

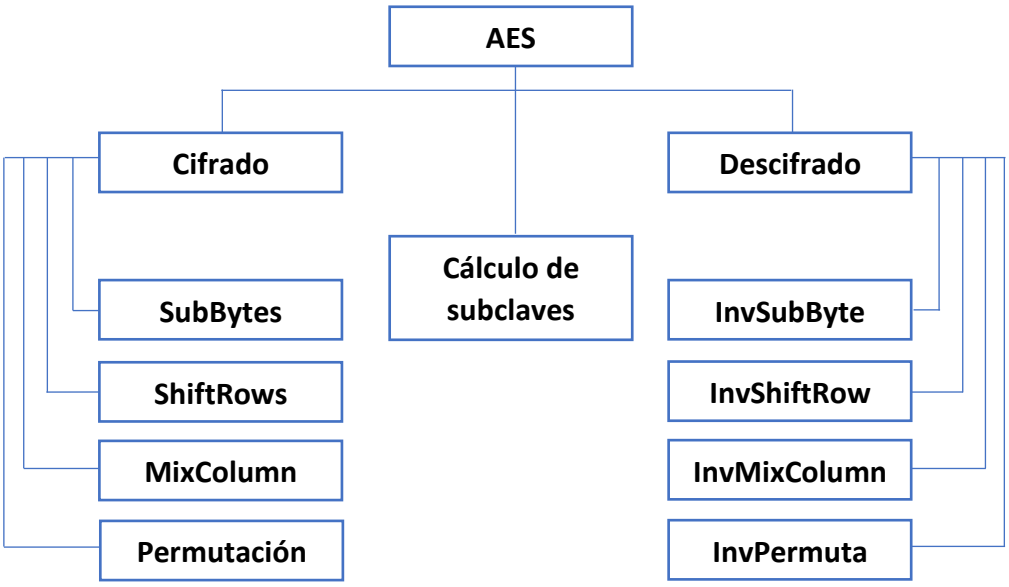


Figura 3.3.2: Estructura de las principales funciones del encriptado con AES y permutación variable [11].

Para el cifrado y descifrado, AES usa una función de ronda compuesta de cuatro transformaciones diferentes orientadas a los bytes [11]:

1) **SubBytes**: con una sustitución de bytes usando una tabla de sustitución (S-Box).

2) **ShiftRows**: cambio de las de la matriz de estado por distintas configuraciones.

3) **MixColumns**: mezcla de datos entre las columnas del vector de estado.

4) **AddRoundKey**: operación XOR entre la matriz de estado y la clave de ronda.

El proceso de encriptación consiste en aplicar repetidamente estas cuatro operaciones reversibles sobre la matriz de estado de 128 bits, y estas funciones deben ejecutarse en el orden indicado en la secuencia de la **figura 3.3.3** [12].

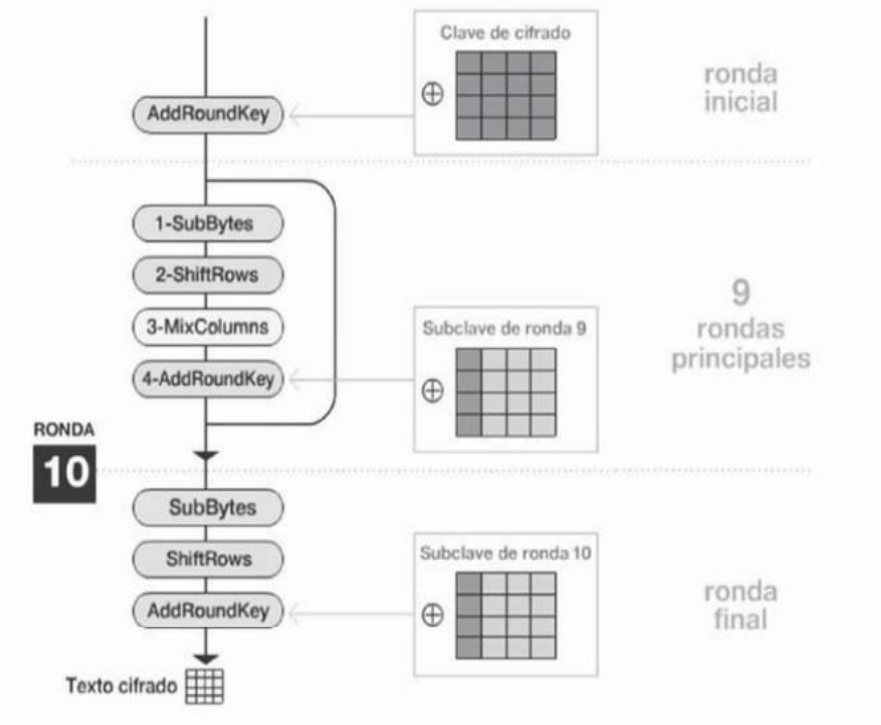


Figura 3.3.3: Funciones en secuencia [12].

Como se observa, el proceso inicia con una operación preliminar de *AddRoundKey*, que antecede a la primera ronda en la que se aplican las cuatro funciones en serie.

En la ronda final, se vuelven a aplicar todas las operaciones, excepto *MixColumns*, que se omite en este paso [12].

CAPÍTULO 4
CODIFICACIÓN DE LA APLICACIÓN MÓVIL

4.1 Objetivo

Mostrar el código implementado de la aplicación, para poder comprender más sobre las herramientas que se usaron, y el lenguaje de programación utilizado, para poder solucionar una tarea y una acción en específico.

4.2 Herramienta

Se utilizó Android Studio, es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y está basado en *IntelliJ IDEA*. Además del potente editor de códigos y las herramientas para desarrolladores de *IntelliJ*, Android Studio ofrece incluso más funciones que aumentan tu productividad cuando desarrollas aplicaciones para Android, como las siguientes [13]:

- Un sistema de compilación flexible basado en Gradle [13].
- Un emulador rápido y cargado de funciones.
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android.
- Aplicación de cambios para insertar cambios de código y recursos a la aplicación en ejecución sin reiniciarla.

4.3 Funcionamiento del cifrado AES

En el código de la **figura 4.3.1**, se puede observar, que se utilizó una clase en específico para poder realizar las tareas sobre el **cifrado AES**, este código usa como parámetro una variable llamada “claveEncriptacion”, que será de tipo byte y esta misma será la encargada de tomar un texto en su formato correcto, en este caso usamos el formato utf-8, para cualquier tipo de texto.

Al igual, usaremos una instancia para poder convertir ese texto a una clave de 16 dígitos, y esta será de tipo “String” o cadena, después retornaremos la clave ya creada.

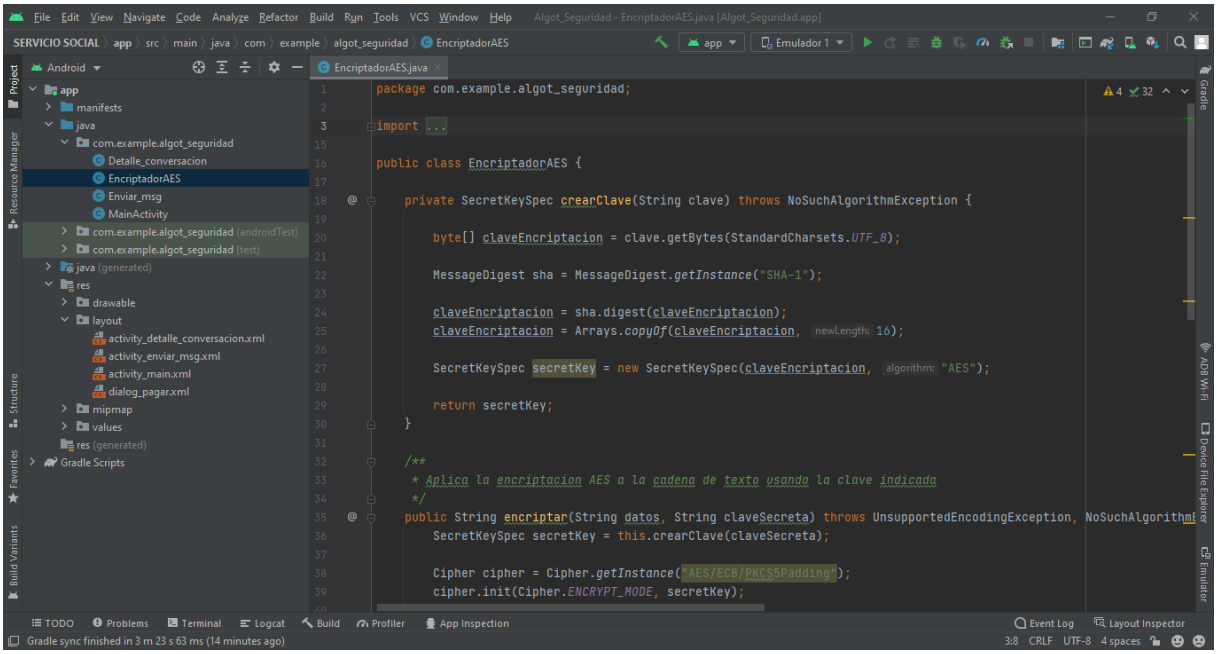


Figura 4.3.1: Tipo de clave.

En la **figura 4.3.2**, se aplicó la clave de 16 dígitos, y se unió con el tipo de cifrado AES, esta función se encargará de que no sea un valor nulo, e ira codificando una versión del SDK junto con la clave que se dio después del mensaje [13].

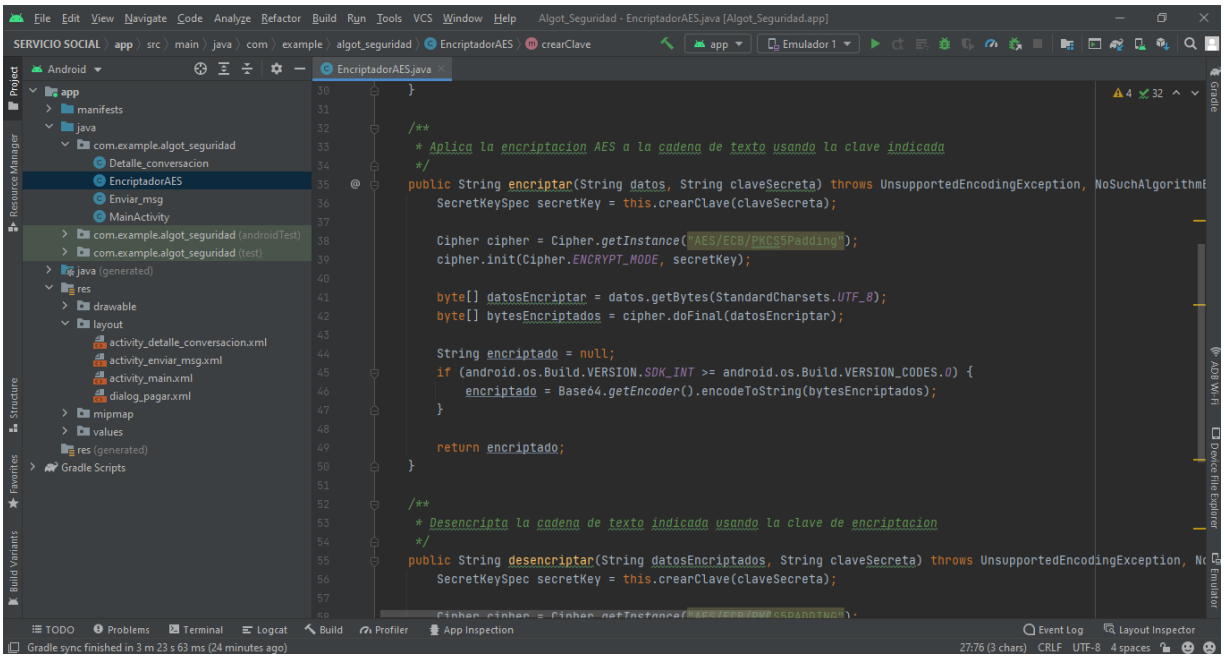


Figura 4.3.2: Cifrado sobre texto y clave.

Al igual que se creó una función para poder descifrar el mensaje dado, en la **figura 4.3.3**, se ve la función para descifrar los mensajes ya hechos con este método de cifrado.

Para este método de descifrado, se utilizará lo mismo, la clave y el texto, usando 'DECRYPT_MODE', podremos tomar la clave y el texto, y poderlo convertir a un lenguaje más común. Lo que hace esta función ('DECRYPT_MODE), es proporcionar la funcionalidad de un cifrado criptográfico para el cifrado y el descifrado. Forma el núcleo del marco Java Cryptographic Extension (JCE) [13].

La aplicación llama al método **Cipher** y le pasa el nombre de la transformación solicitada. Opcionalmente, se puede especificar el nombre de un proveedor [13].

Una **transformación** es una cadena que describe la operación (o conjunto de operaciones) que se realizará en la entrada dada, para producir alguna salida.

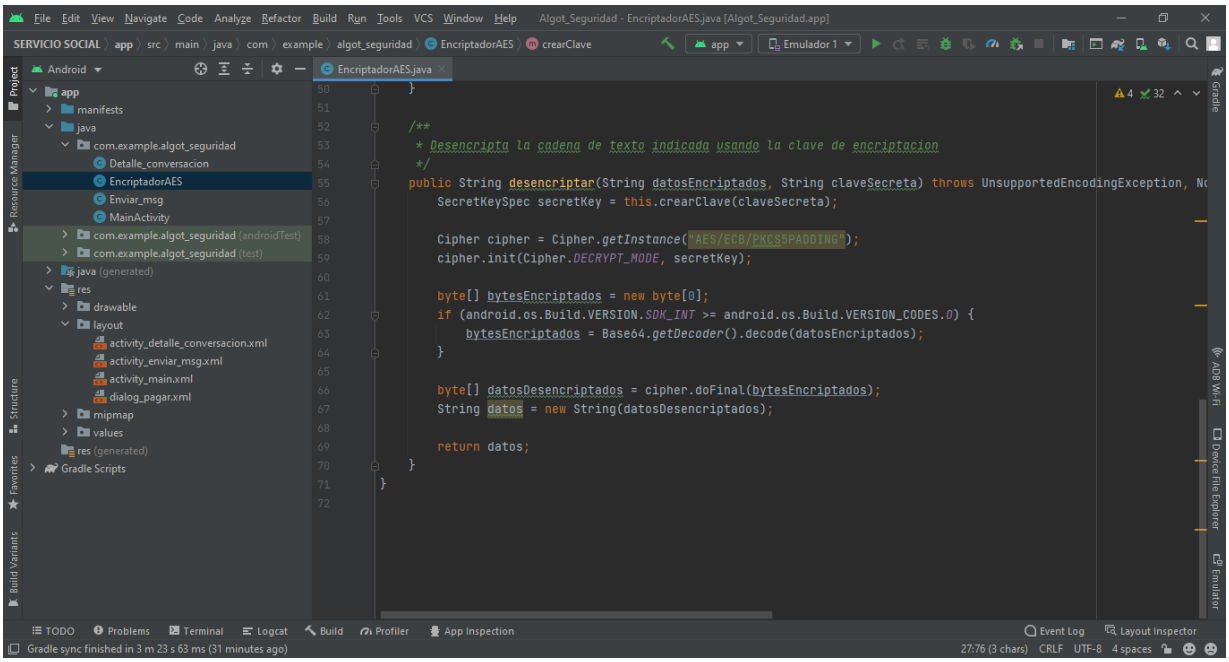


Figura 4.3.3: Descifrado con Cipher (AES).

4.4 Diseño de control

Con una ventana (**figura 4.4.1**), controlaremos nuestro código y textos de entrada y salida, en la pantalla principal se programó dos botones de transición, con los que nos permitirá el paso hacia la ventana que usará a la clase de cifrado y descifrado AES.

Estos botones, al igual que toda la aplicación, se programaron en lenguaje Java, nos guiamos del diseño y del nombre de cada ventana, para poder pasar de una pantalla a otra. Para el correcto funcionamiento se tuvo que crear una clase y otras dos ventanas.

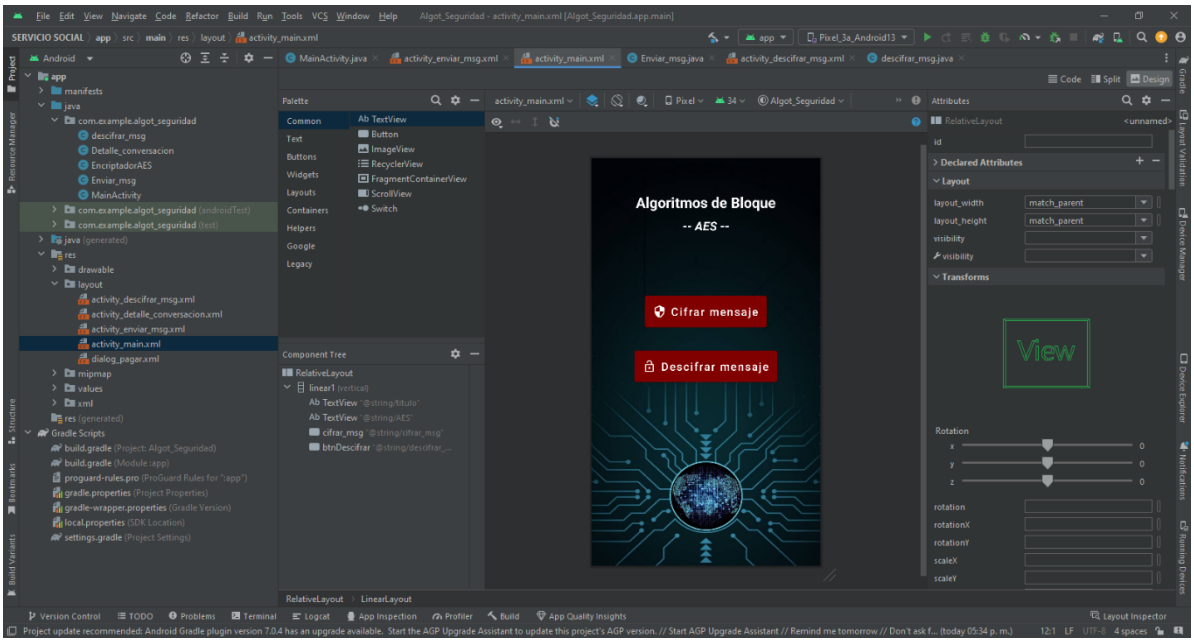
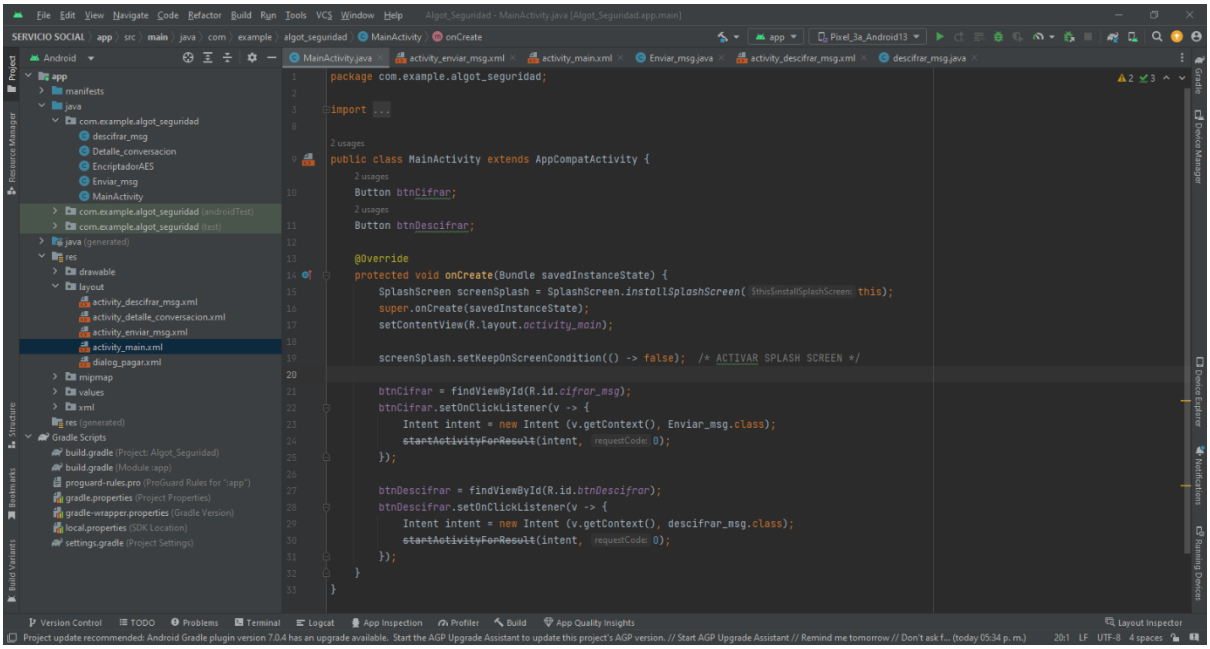


Figura 4.4.1: Pantalla principal

En la **figura 4.4.2**, veremos cómo se configuran los dos botones de transición con solo una función, esto es gracias al lenguaje JAVA.



```
package com.example.algot_seguridad;

import ...

public class MainActivity extends AppCompatActivity {
    Button btnCifrar;
    Button btnDescifrar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        SplashScreen screenSplash = SplashScreen.installSplashScreen( this);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        screenSplash.setKeepOnScreenCondition(() -> false);  /* ACTIVAR SPLASH SCREEN */

        btnCifrar = findViewById(R.id.cifrar_msg);
        btnCifrar.setOnClickListener(v -> {
            Intent intent = new Intent (v.getContext(), Enviar_msg.class);
            startActivityForResult(intent, 0);
        });

        btnDescifrar = findViewById(R.id.btnDescifrar);
        btnDescifrar.setOnClickListener(v -> {
            Intent intent = new Intent (v.getContext(), descifrar_msg.class);
            startActivityForResult(intent, 0);
        });
    }
}
```

Figura 4.4.2: Código de transición.

Al cambiar a la pantalla siguiente, así como se muestra en la **figura 4.4.3**, nos daremos cuenta de que ya pide el texto y clave con la que se usará el cifrado AES.

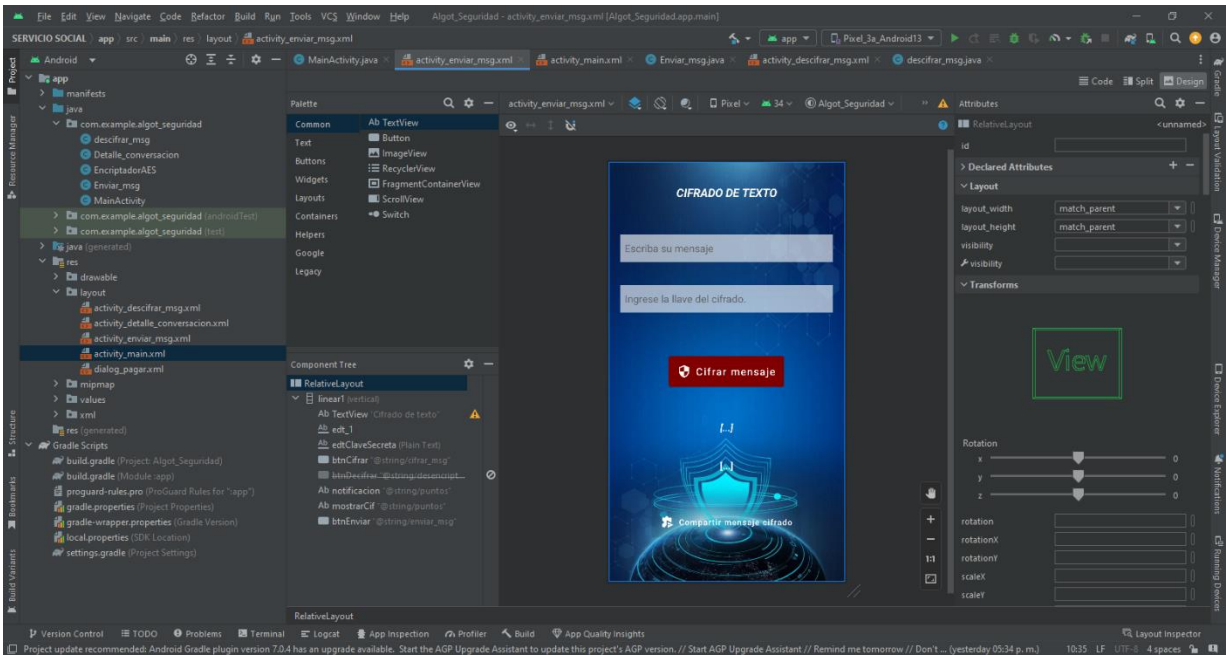


Figura 4.4.3: Texto y clave.

Como se puede notar, usaremos dos cajas de texto y dos botones, el primer botón se usará para mandar a llamar a la clase ‘cifrado’, y el segundo botón será para mandar el texto ya cifrado por algún medio de comunicación, en nuestro caso utilizaremos el correo electrónico, este será pedido por una caja de texto controlada por código, el cual será mostrada después de cifrar algún texto.

El código se puede ver en la **figura 4.4.4**, **figura 4.4.5**, **figura 4.4.6** y **figura 4.4.7**, así como las funciones declaradas y las acciones de cada botón.

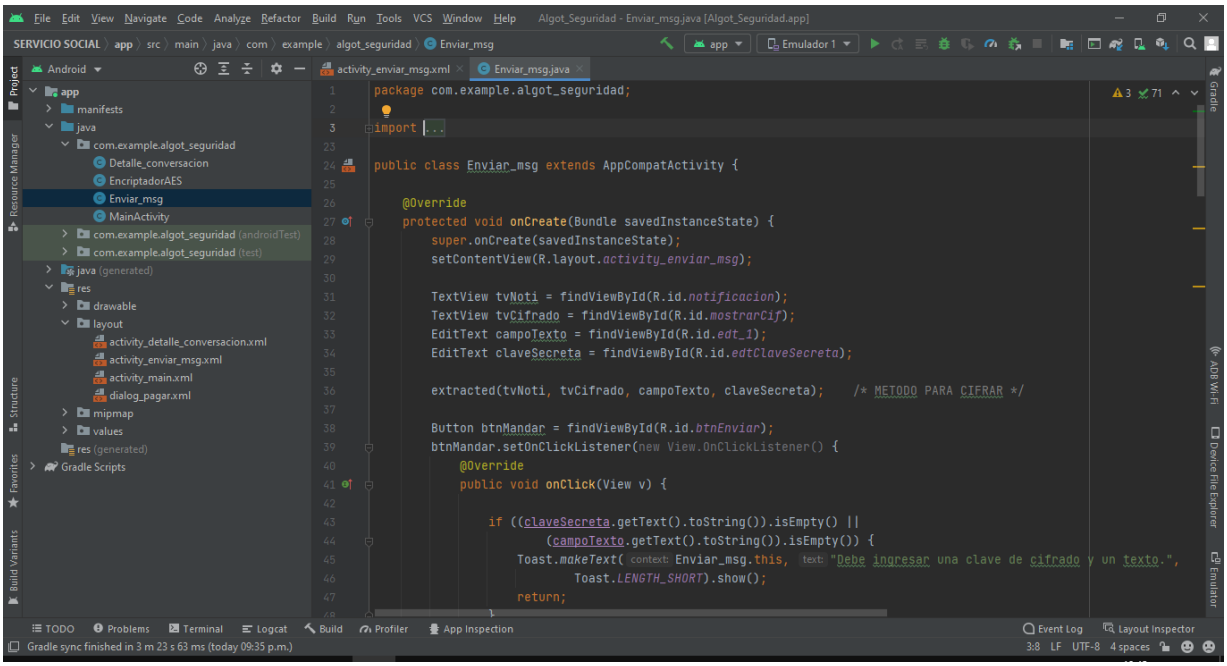


Figura 4.4.4: Declaraciones de cajas.

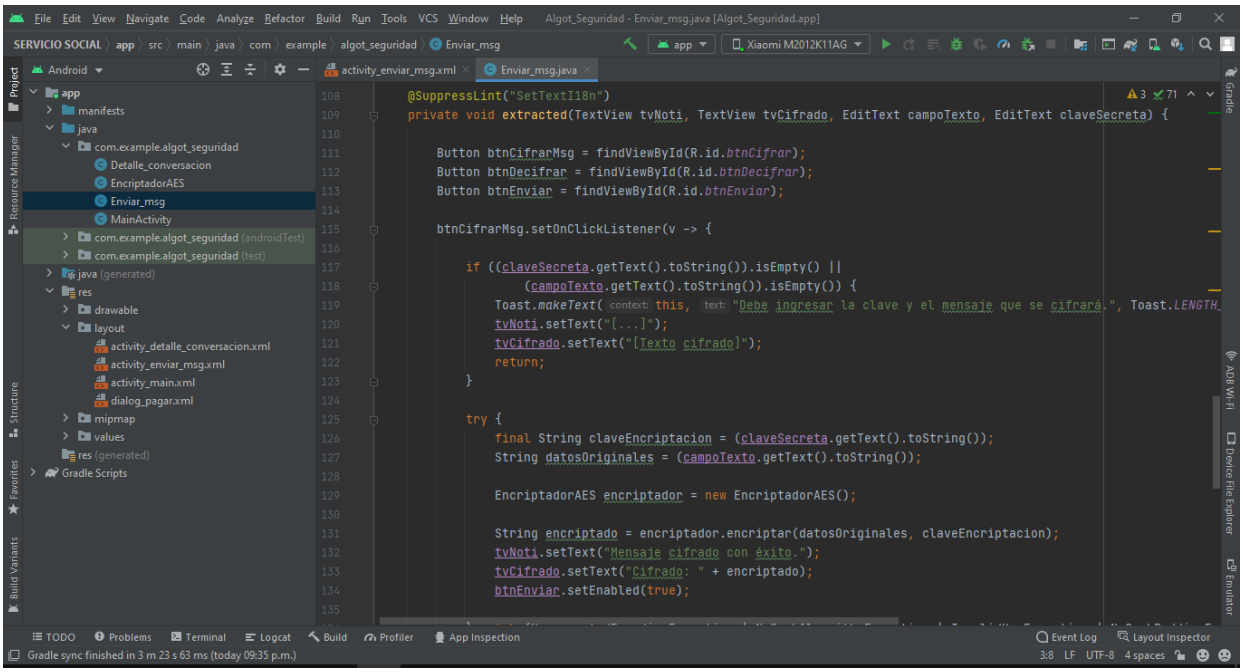
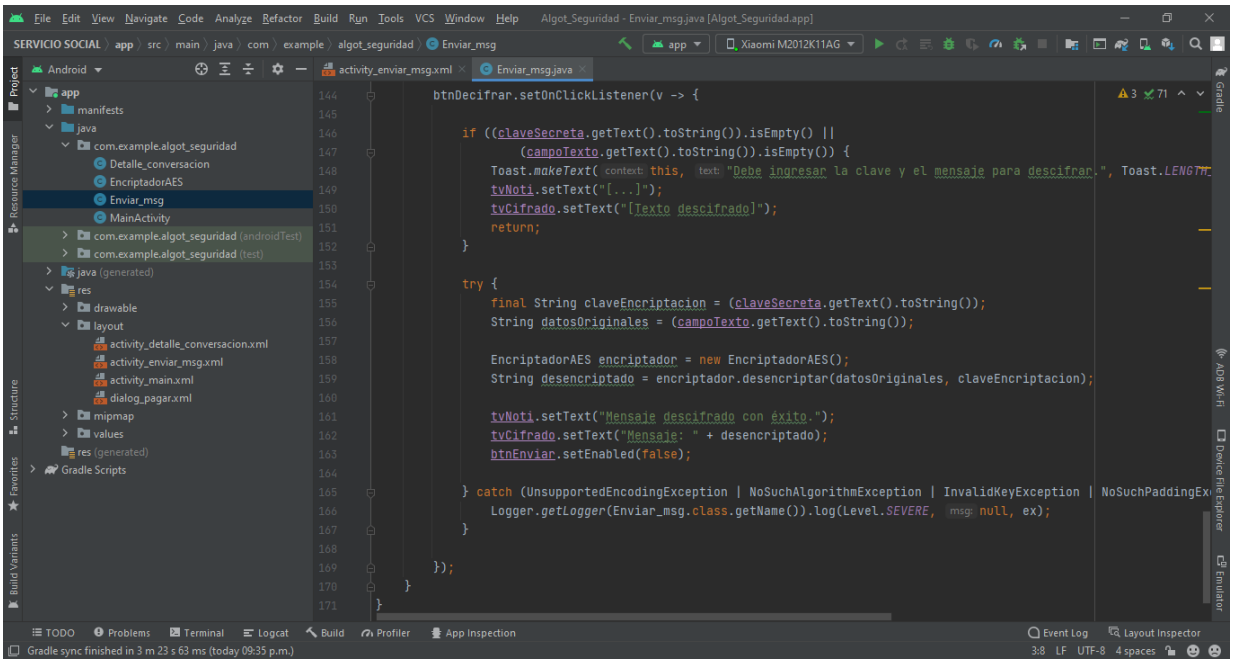
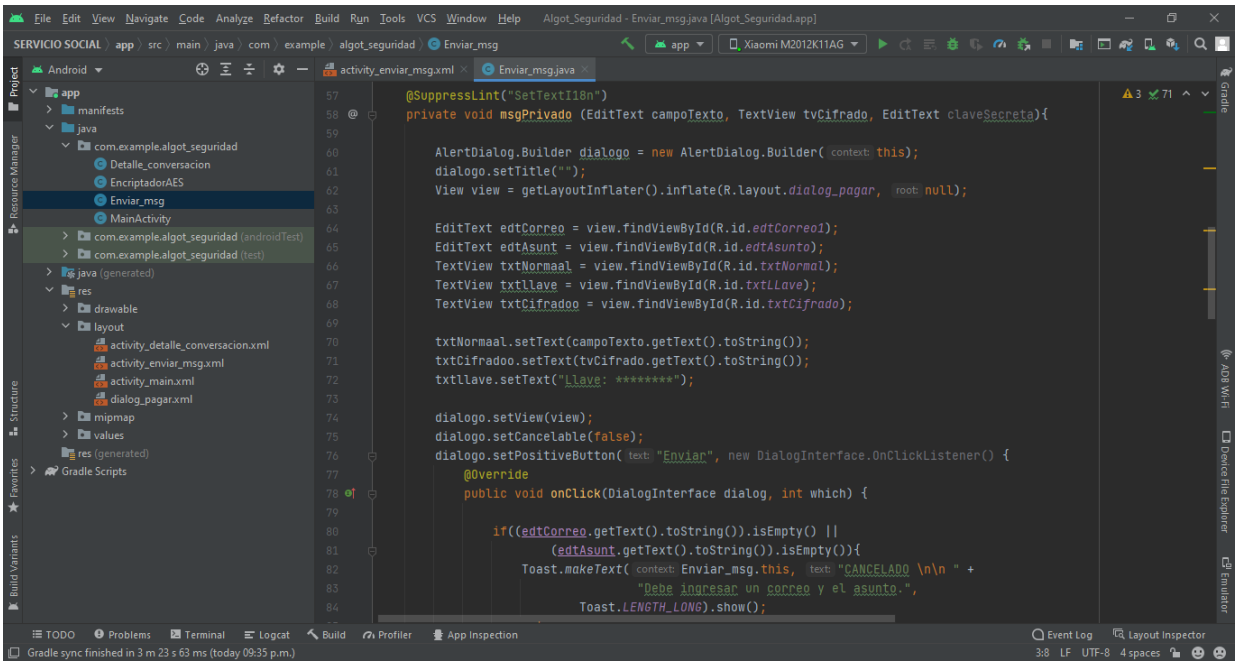


Figura 4.4.5: Extracción de mensajes.



```
btnDecifrar.setOnClickListener(v -> {

    if ((claveSecreta.getText().toString()).isEmpty() ||
            (campoTexto.getText().toString()).isEmpty()) {
        Toast.makeText( context: this, text: "Debe ingresar la clave y el mensaje para descifrar.", Toast.LENGTH
        tvNoti.setText("[...]");
        tvCifrado.setText("[Texto descifrado]");
        return;
    }

    try {
        final String claveEncriptacion = (claveSecreta.getText().toString());
        String datosOriginales = (campoTexto.getText().toString());

        EncriptadorAES encriptador = new EncriptadorAES();
        String desencriptado = encriptador.desencriptar(datosOriginales, claveEncriptacion);

        tvNoti.setText("Mensaje descifrado con éxito.");
        tvCifrado.setText("Mensaje: " + desencriptado);
        btnEnviar.setEnabled(false);

    } catch (UnsupportedEncodingException | NoSuchAlgorithmException | InvalidKeyException | NoSuchPaddingEx
        Logger.getLogger(Enviar_msg.class.getName()).log(Level.SEVERE, msg: null, ex);
    }

});
}
}
```

Figura 4.4.6: Acción de botones.



```
@SuppressLint("SetTextI18n")
private void msgPrivado (EditText campoTexto, TextView tvCifrado, EditText claveSecreta){

    AlertDialog.Builder dialogo = new AlertDialog.Builder( context: this);
    dialogo.setTitle("");
    View view = getLayoutInflater().inflate(R.layout.dialog_pagar, root: null);

    EditText edtCorreo = view.findViewById(R.id.edtCorreo1);
    EditText edtAsunt = view.findViewById(R.id.edtAsunto);
    TextView txtNormaal = view.findViewById(R.id.txtNormal);
    TextView txtllave = view.findViewById(R.id.txtLlave);
    TextView txtCifradoo = view.findViewById(R.id.txtCifrado);

    txtNormaal.setText(campoTexto.getText().toString());
    txtCifradoo.setText(tvCifrado.getText().toString());
    txtllave.setText("Llave: ********");

    dialogo.setView(view);
    dialogo.setCancelable(false);
    dialogo.setPositiveButton( text: "Enviar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

            if((edtCorreo.getText().toString()).isEmpty() ||
                    (edtAsunt.getText().toString()).isEmpty()){
                Toast.makeText( context: Enviar_msg.this, text: "CANCELADO \n\n " +
                                "Debe ingresar un correo y el asunto.",
                        Toast.LENGTH_LONG).show();
```

Figura 4.4.7: Ingreso de correo electrónico.

Al ir al botón siguiente con nombre “Descifrar mensaje”, nos mostrará el diseño de la **figura 4.4.8**, con dos cajas de texto, y un botón de descifrado.

Aquí tendrán que ingresar el código AES generado por la ventana anterior, y la llave de descifrado

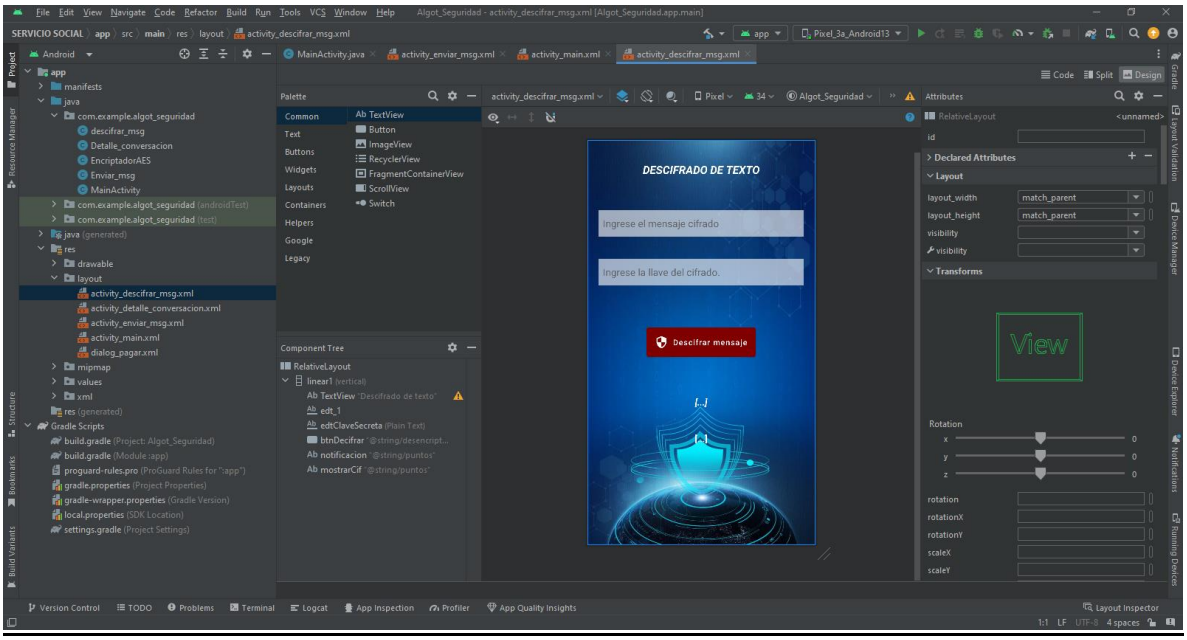


Figura 4.4.8: Envío de información.

Y así como se muestra en la **figura 4.4.9**, se mostrará el cuadro de dialogo, con el que se pedirá el correo electrónico y se hará el paso de parámetros del texto cifrado y la llave, pero la llave única no se podrá ver, porque solo los usuarios conocerán esa llave.

Y como se muestra en el código, esta información será enviada a la aplicación de correos para el envío del mensaje ya preparado hacia su destino.

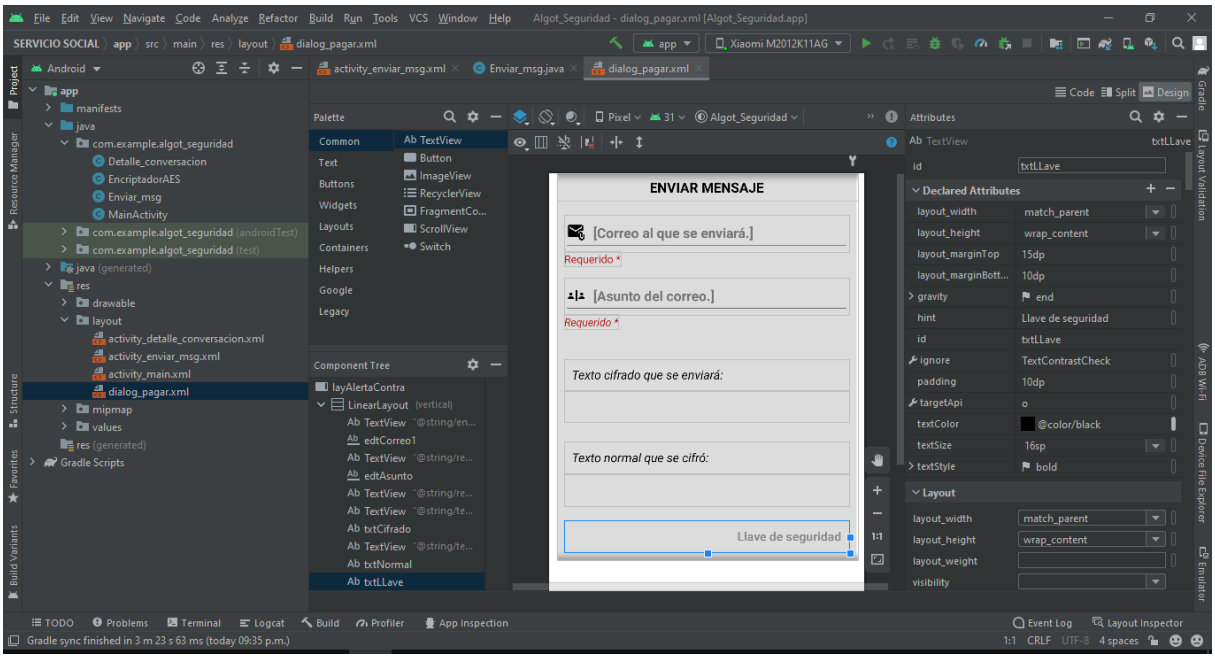


Figura 4.4.9: Envío de información.

CAPÍTULO 5

PRUEBAS DE LA APLICACIÓN

5.1 Objetivo

El objetivo principal de las pruebas es verificar y validar que el sistema desarrollado cumple con los requisitos funcionales y no funcionales especificados en el diseño. Esto incluye asegurar que la aplicación sea fiable, eficiente y fácil de usar para los usuarios finales. Las pruebas se centrarán en los siguientes aspectos:

Funcionalidad: Asegurar que todas las funciones especificadas se ejecutan correctamente y que los resultados producidos son precisos y coherentes con las especificaciones del diseño.

Rendimiento: Evaluar la eficiencia de la aplicación en términos de tiempo de respuesta y uso de recursos, garantizando que opera de manera correcta.

Usabilidad: Confirmar que la interfaz de usuario es intuitiva y que los usuarios pueden interactuar con la aplicación de manera efectiva y sin confusión.

Compatibilidad: Asegurar que la aplicación funciona correctamente en diferentes dispositivos y sistemas operativos.

Estabilidad: Evaluar que la aplicación se comporta de manera consistente y no falla bajo condiciones normales de uso.

A través de estas pruebas, se espera identificar y corregir cualquier defecto o área de mejora, garantizando que la aplicación final sea robusta y de alta calidad.

5.2 Desarrollo

Estás pruebas se realizaron con 3 dispositivos Android, las cuales contaban con las siguientes especificaciones:

	1	2	3
Versión de Android	9	13	14
Almacenamiento	60 GB	256 GB	512 GB
Almac. RAM	6 GB	8 GB	12 GB
Modelo de celular	22071219CG	M2012K11AG	RBN-NX3
Procesador	MediaTek Helio G99	Qualcomm Snapdragon 870	Qualcomm Snapdragon 480 plus
Pantalla	TFTLCD	AMOLED 120 Hz	Super AMOLED
Dispositivo	Honor X8a 5G	Poco F3 5G	Samsung Galaxy A24

Para poder realizar las pruebas de usabilidad y compatibilidad correctamente, se subió la aplicación a la tienda online llamada Play Store, en dónde te verifican la usabilidad de cada aplicación, así como la compatibilidad que tiene con cada dispositivo registrado en su sistema.

Se eligió esta tienda online debido a la seguridad y flexibilidad que se obtiene al subir una aplicación, así como la practicidad para modificaciones o correcciones de la misma.

De acuerdo con la compatibilidad en Play Store, el 90% de los dispositivos registrados pueden solventar los requerimientos que necesita la aplicación móvil para un control correcto y efectivo.

Las pruebas se comenzaron descargando la aplicación de forma directa; instalando el archivo APK y dando los permisos necesarios al dispositivo.

Al momento de abrir la aplicación, nos mostrará la pantalla de carga (**figura 5.2.1**), en la cual podemos observar el tiempo estimado que hace el celular al cargar. Haciendo las pruebas, está carga fue máximo de 1 segundo, demostrando el rendimiento desde un principio.

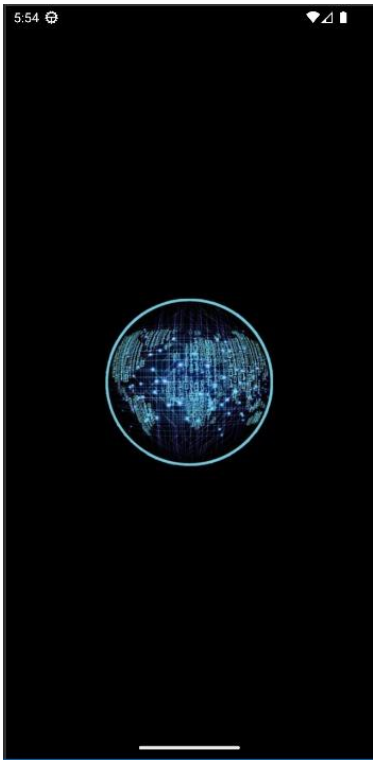


Figura 5.2.1: Pantalla de carga

Figura 5.2.2: Pantalla de inicio

Después de pasar de la primera pantalla, nos mostrará el inicio (**figura 5.2.2**) en dónde confirmaremos la funcionalidad y usabilidad, esto verificando que la aplicación sea intuitiva, manejable, sin confusiones y de manera efectiva.

Ya mostrándonos los botones, podremos ingresar y nos encontraremos en pantalla los campos requeridos para realizar un cifrado de mensajes de forma comprensible y precisa, así como se muestra en la **figura 5.2.3**.

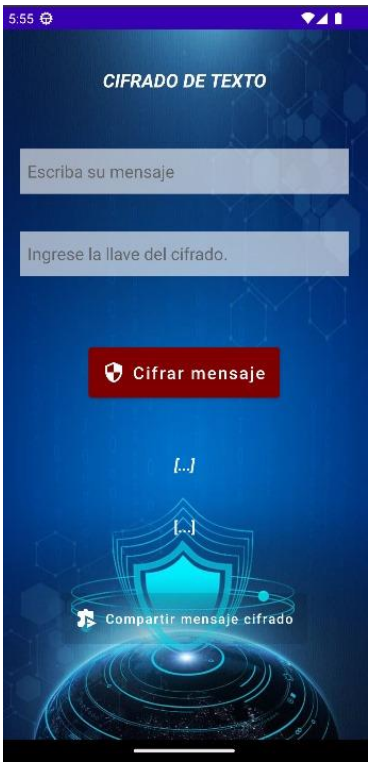


Figura 5.2.3: Cifrado de mensajes

Figura 5.2.4: Primer prueba

En la **figura 5.2.4** podemos observar lo que es un ejemplo de usabilidad, en dónde se tomará un mensaje y una clave de cifrado.

Nuestro mensaje de ejemplo fue “Esta es la primera prueba del cifrado AES”, y la clave con la que ocultaremos nuestro mensaje será “clave1”. Al dar seguimiento al botón de cifrado, nos mostrará el resultado del **algoritmo AES,** dándonos una serie de números, signos y letras que harán referencia a nuestro mensaje y clave.

Ya sea que se quiera copiar la referencia, o usar el botón “Compartir mensaje cifrado”, la aplicación móvil le mostrará una nueva pantalla como se puede apreciar en la **figura 5.2.5**, la cual te pedirá el correo destinatario y el asunto del envío. Al momento de enviar, te abrirá la aplicación de correos con el usuario emisor que hayas elegido (**figura 5.2.6**).

Nota: Por seguridad, no se enviará la clave o llave de cifrado, ya que, la persona de destino tendrá que saber cuál es esa llave para poder descifrar su mensaje.

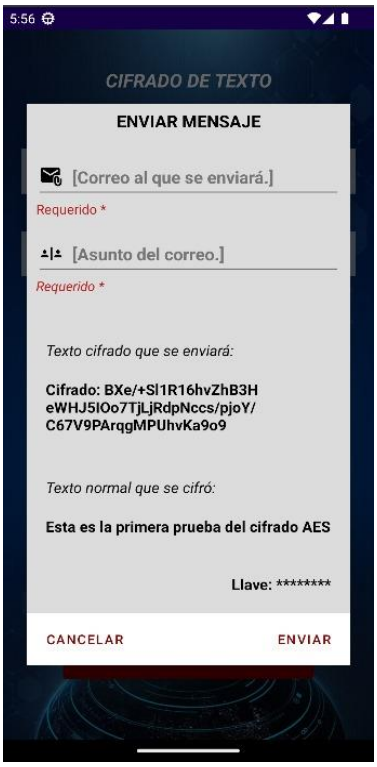


Figura 5.2.5: Envío de cifrado.

Figura 5.2.6: Enviado por correo

Así como usamos el algoritmo AES en la aplicación móvil para el cifrado de mensajes con una llave o clave especial, podremos hacer uso de ella para el descifrado de los mismos.

Este apartado en un inicio se pensó agregar en la misma pantalla de cifrado porque se piden los mismos campos, sin embargo, no es recomendable, puesto que no sería una buena práctica de una interfaz intuitiva.

Al inicio de la aplicación como se muestra en la **figura 5.2.2**, tendremos el botón “Descifrar mensaje”, este nos enviará a una nueva pantalla como se muestra en la **figura 5.2.7**, con la que nos pedirá los parámetros específicos para poder mostrarnos el mensaje que nos hayan enviado. Nos aparecerán dos campos, uno para la cadena de texto, y otro para la llave que nos dará acceso a la respuesta que queremos.

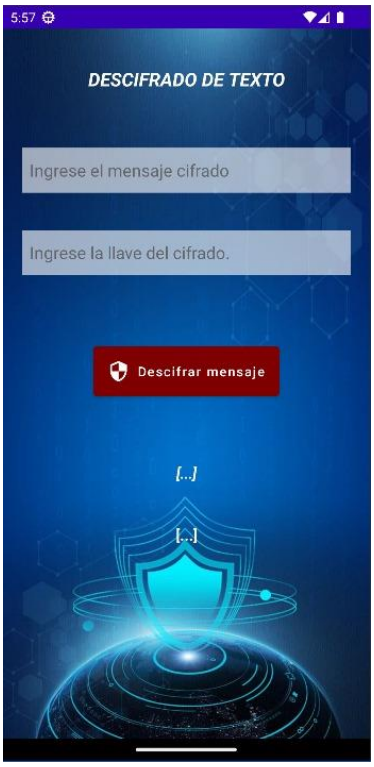


Figura 5.2.7: Descifrado de mensajes.

Figura 5.2.8: Campos de descifrado.

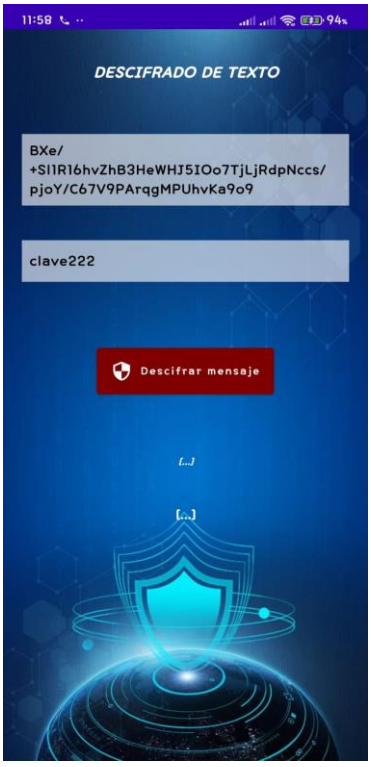


Figura 5.2.9: Distinta llave.

Figura 5.2.10: Descifrado correcto.

Como podemos observar en la **figura 5.2.8**, se ingresaron los parámetros que se requieren, después de oprimir el botón principal, puede haber diferentes situaciones, por ejemplo:

1. Si la contraseña o el mensaje ingresado es incorrecto, como se muestra en la **figura 5.2.9**, no nos devolverá nada.
2. Si tanto la llave como el mensaje es correcto, como en la **figura 5.2.10**, nos dará los resultados del descifrado AES.

Así como no hay opción y no hay fallas en la aplicación al tener parámetros erróneos, también nos devolverá un mensaje en el caso de que no estén los campos llenos, así como se muestra en la **figura 5.2.11**.

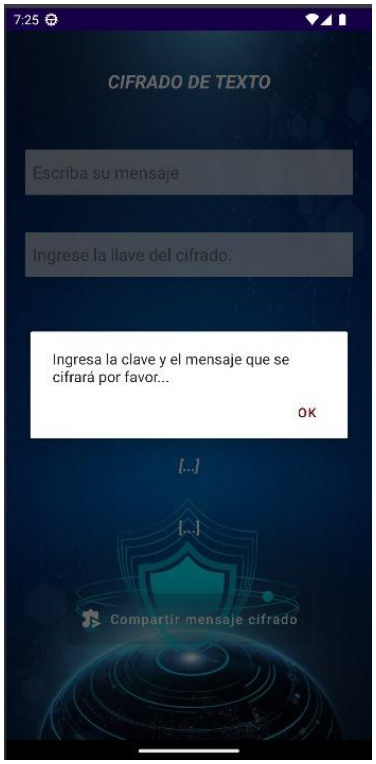


Figura 5.2.11: Mensaje de advertencia.

5.3 Ejemplo de un caso real

La aplicación móvil se utilizó para un caso real, demostrando que es funcional y estable.

Para este caso en particular, el usuario necesitaba mandar la contraseña de su internet de forma segura y encriptada, esto para que no se haga mal uso de cambio de información por cualquier red de comunicación.

Figura 5.3.1: Cifrado de contraseña privada.

Como se puede observar en la anterior **figura 5.3.1**, ingresaron la contraseña en la primera caja de texto, al igual que la llave de cifrado en el segundo espacio. Teniendo los campos llenos, se pudo crear el mensaje cifrado que se necesitaba.

El usuario prefirió mandar el texto de dos formas:

1. Copiándolo y enviándolo por su red social de confianza.

2. Mandarlo por correo electrónico.

Por medio de la aplicación, se solicitó con el usuario la visualización de envío por correo electrónico. Como se puede observar en la **figura 5.3.2**, se anotó el correo electrónico del destinatario y el asunto en cuestión.

En la **figura 5.3.3** demuestra los detalles del envío de correo electrónico, solo enviándose el asunto y el código de cifrado.

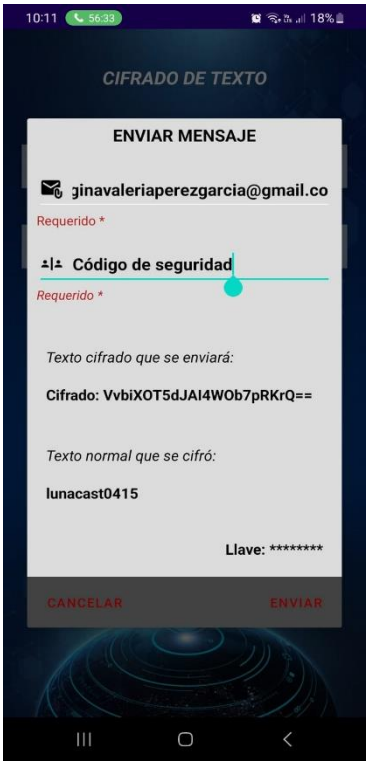


Figura 5.3.2: Envío de la llave.

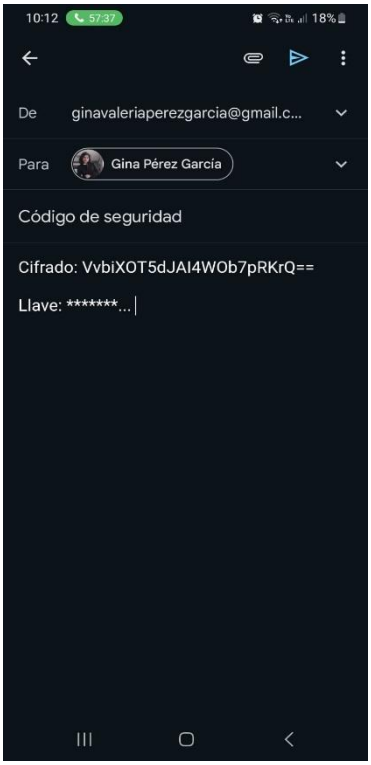


Figura 5.3.3: Envío de correo.

Al momento de enviar él cifrado de la contraseña privada, el usuario quiso verificar que el mensaje que estaba mandando era correcto, al igual que la llave ingresada.

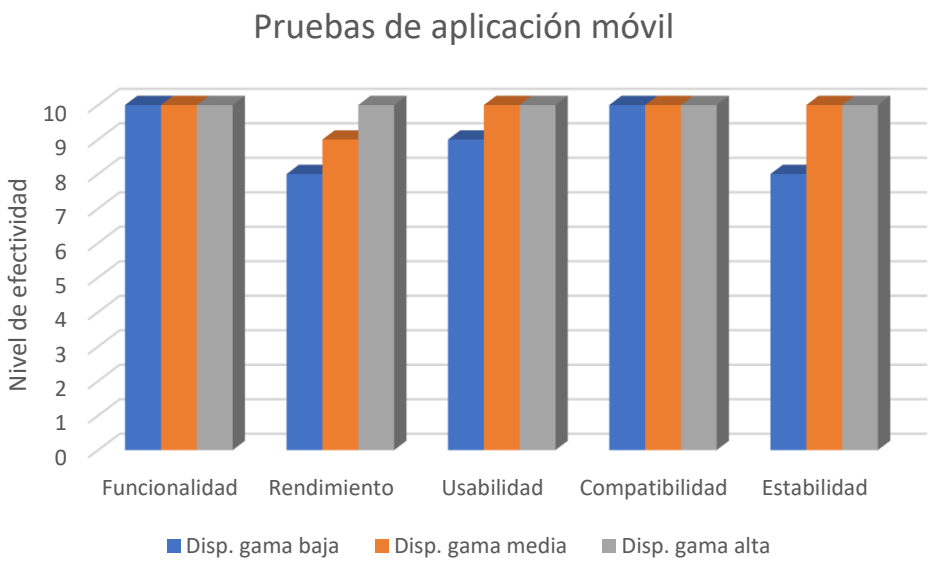
Así que ingreso al segundo apartado, este caso el de descifrado… Como ya tenía la clave cifrada copiada, fue cuestión de pegarla en el primer campo, y en el segundo campo agregar la llave que él o ella ya sabía.

Figura 5.3.4: Descifrado y verificación.

Como se puede observar en la **figura 5.3.4**, nos muestra ya descifrado la referencia que teníamos, pero solo con la contraseña correcta.

Podemos observar que la conversión del algoritmo AES funcionó de manera correcta y eficiente.

Así el usuario ya se sentía más seguro de las claves que él o ella haya enviado, ya sea por cualquier red social o de comunicación.



Gráfica 5.3.5: Resultados de efectividad de la aplicación móvil

De acuerdo con las pruebas hechas en los dispositivos móviles y la usabilidad que se vio al usar la aplicación móvil, se concluyó que tiene muy buena compatibilidad en todos los parámetros y objetivos que se querían cumplir.

En la **gráfica 5.3.5** se puede observar los resultados de usar la aplicación móvil en diferentes dispositivos Android.

5.4 Conclusión de pruebas

De acuerdo con los dispositivos que se usaron, se obtuvieron resultados satisfactorios que cumplen con los objetivos planeados, se tenían dudas por el manejo de información privada de los usuarios, pero se demostró la confiabilidad y seguridad del algoritmo que se utiliza.

Como se puede observar, dependiendo del dispositivo y de lo actual que sea, se verá reflejado el rendimiento y estabilidad, no mostrando problemas, pero si mostrando una diferencia al lado de dispositivos de media y alta gama.

CONCLUSIÓN GENERAL

Las pruebas realizadas sobre la aplicación móvil han permitido evaluar de manera absoluta los aspectos; clave de seguridad, funcionalidad, rendimiento, usabilidad, compatibilidad y estabilidad.

- **Algoritmo de Seguridad (AES):**

  El algoritmo implementado en la aplicación, ha demostrado ser robusto y eficaz en la protección de los datos sensibles. Las pruebas de cifrado y descifrado que se realizaron confirmaron la integridad y confidencialidad de la información, cumpliendo con los estándares de seguridad esperados.

- **Funcionalidad:**

  Se ha cumplido con todos los requisitos funcionales establecidos, al igual que todas las funciones principales y secundarias operaron según lo previsto, sin fallas ni errores significativos durante las pruebas.

- **Rendimiento:**

  La aplicación ha sido satisfactoria en cuanto a rendimiento, con tiempos de respuesta óptimos bajo diferentes condiciones de carga. La aplicación mantuvo un rendimiento consistente y no experimentó caídas de velocidad que pudieran afectar la experiencia del usuario.

- **Usabilidad:**

  Las pruebas indicaron que la aplicación es intuitiva y fácil de usar, ya que los usuarios pudieron navegar y utilizar las funciones sin dificultades, lo que demuestra un diseño de interfaz adecuado y accesible.

- **Compatibilidad:**

  De acuerdo con este objetivo se mostró una alta compatibilidad en las plataformas y dispositivos probados. No se identificaron problemas importantes al ejecutarla en diferentes sistemas operativos o versiones de hardware.

- **Estabilidad:**

  Durante las pruebas, la aplicación mantuvo un alto nivel de estabilidad, en la cual no se registraron fallos críticos, cierres inesperados ni problemas que pudieran comprometer la experiencia del usuario.

En conclusión, la aplicación móvil ha superado satisfactoriamente todas las pruebas, mostrando un alto grado de seguridad, rendimiento y usabilidad, mientras garantiza la compatibilidad y estabilidad en los entornos de prueba.

Se considera que la aplicación está lista para su despliegue en producción con las expectativas de ofrecer una experiencia segura y confiable a los usuarios que deseen utilizarla.

BIBLIOGRAFÍA

[1]. Palencia, E. B. (2012). Implementación del algoritmo AES sobre arquitectura ARM con mejoras en rendimiento y seguridad.

[2]. Ciberseguridad.com. (n.d.). Cifrado de bloque. Ciberseguridad.com. Retrieved November 18, 2024, URL: https://ciberseguridad.com/guias/prevencion-proteccion/criptografia/cifrado-bloque/

[3]. Casas García, O. (2010). Implementación de los cifradores de bloque Rijndael, Serpent, MARS, Twofish y RC6 para su uso en sistemas embebidos. Universidad de San Buenaventura.

[4]. Usuario de Studocu. (2021/2022). Proyecto de criptografía. Studocu. URL: https://www.studocu.com/ec/document/universidad-de-guayaquil/telecomunicaciones/proyecto-criptografia/30318179.

[5]. Currás Paz, B. (2013). Sistema de medida de consumo para aplicaciones criptográficas en microcontroladores ARM Cortex-M3.

[6]. García Ocón, M. (2013). Implementación del algoritmo de cifrado AES para bajo consumo sobre FPGA. Universidad Carlos III De Madrid Escuela Politécnica Superior.

[7]. Aranda Melo, L. R. (2006). Seguridad en sistemas de información: Criptografía [PDF]. Ptolomeo UNAM. URL: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/728/arandamelo.pdf?sequence=16&isAllowed=y.

[8]. Godínez Rodríguez, E. (2015). Cifrado de imágenes utilizando advanced encryption standard (AES) con permutación variable.

[9]. Neira, B. S. H., & Pedraza, L. F. (2013). Implementación del algoritmo criptográfico AES para un controlador de tráfico vehicular. Tecnura, 17(1), 35-48.

[10]. G. Hoyos A. (2012). Cifrado por bloques [Diapositivas]. SlideShare. URL: https://es.slideshare.net/slideshow/cifrado-por-bloques/13461314

[11]. LÓPEZ, M. J. L. (2010). Criptografía y Seguridad. Universidad de Jaén. v4-0.8.1.

[12]. Tarqui Triguero, E. D. (2015). Protocolos de criptografía para la privacidad y seguridad de información almacenada en aplicaciones libres de la computación en la nube (Doctoral dissertation).

[13]. Google. (2024). Introducción a Android Studio. Android Developers. URL: https://developer.android.com/studio/intro?hl=es-419**.**