eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Encriptación de comunicaciones en tiempo real mediante las extensiones criptográficas de ARM

Autor/a

Gorka Gómez Etxaniz

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Encriptación de comunicaciones en tiempo real mediante las extensiones criptográficas de ARM

Autor/a

Gorka Gómez Etxaniz

Directore/a(s)

Jose A. Pascual Saiz

Resumen

El objetivo de este trabajo es el desarrollo de una librería que permita la utilización de las extensiones criptográficas proporcionadas por ARM de manera sencilla. En particular, las funciones que implementan el estándar de cifrado AES y las funciones de resumen SHA-1 y SHA-2. La librería permitirá cifrar/descifrar cualquier flujo de información como pueden ser mensajes enviados a través de una canal de comunicación, o incluso, el cifrado/descifrado de ficheros de una manera eficiente gracias al hardware criptográfico que implementa la arquitectura ARMv8. ARM ya proporciona una librería de bajo nivel (AArch64cryptolib) para utilizar las funciones hardware pero su uso es complicado y solo permite el uso de dichas funciones a muy bajo nivel. Por ello, la librería que se desarrollará será un *wrapper* de la de ARM que permita utilizar todos los algoritmos para cifra/descifrar información de cualquier tamaño.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

En los últimos años, las comunicaciones se han convertido en algo indispensable para la sociedad. Debido en gran medida a la expansión de Internet, y más recientemente a la expansión de las comunicaciones móviles, hoy en día tenemos la oportunidad de comunicarnos y transmitir la información de forma rápida y eficaz. El tipo de información que se envía, incluye desde mensajes de tipo personal en forma de correos electrónicos o mensajes instantáneos, a información muy sensible como pueden ser transacciones bancarias. Todo esto ha llevado a que la seguridad y la privacidad en las comunicaciones sea un punto de mucho interés, no solo para las empresas o gobiernos, sino también para la ciudadanía en general.

La seguridad en las comunicaciones se consigue mediante el uso de técnicas criptográficas. La criptografía se puede definir como el estudio de los algoritmos, protocolos, y sistemas que se utilizan para proteger la información y dotar de seguridad a las comunicaciones mediante el uso de técnicas matemáticas. El concepto de seguridad en las comunicaciones hace referencia al conjunto de propiedades que se deben garantizar para proteger la información manteniendo, entre otras, la confidencialidad, la integridad y la autenticación de los datos enviados. Estas propiedades son garantizadas mediante el uso de técnicas criptográficas.

La confidencialidad es la propiedad que impide la divulgación no autorizada de la información mediante el uso de técnicas de cifrado. Por otra parte, la integridad asegura que los datos sean correctos, es decir, que no se modifiquen de forma no autorizada y para ello se utilizan funciones de resumen (*hash*). Por último, la autenticación es la propiedad que

garantiza que un mensaje puede asociarse con un emisor, es decir, verificar su identidad. La presencia de estas tres propiedades garantizan la seguridad de las comunicaciones.

Las técnicas criptográficas para el cifrado/descifrado de información se clasifican en dos grupos: criptografía simétrica y criptografía asimétrica. La criptografía simétrica se basa en la utilización de una sola clave para cifrar/descifrar la información. Esto plantea un problema que es el envío de esa clave debido a que debe ser compartida por el emisor y el receptor del mensaje. Por el contrario, en la criptografía asimétrica no se utiliza una única clave, siendo las claves de cifrado y descifrado diferentes. En este caso, tanto el emisor como el receptor disponen de dos claves, una pública que se puede compartir sin problema y una privada que no se comparte. Ambos tipos de criptografía son computacionalmente costosos, pero la asimétrica lo es mucho más, por lo que en la práctica solo se utiliza para intercambiar una clave que será utilizada para cifrar las comunicaciones de forma simétrica.

Otro tipo de funciones criptográficas son las funciones resumen, hash o de un solo sentido. Estas funciones utilizan una función matemática para transformar un conjunto de datos en un código alfanumérico de longitud fija. Las funciones hash son unidireccionales, es decir, una vez cifrada la información no se puede volver a descifrar y es por esto que se suelen usar para almacenar contraseñas o garantizar la integridad de los datos.

Tradicionalmente, estos algoritmos criptográficos han sido implementados en software. Sin embargo, en la actualidad, debido a que se utilizan en multitud de situaciones los fabricantes de procesadores han empezado a incluir implementaciones hardware de ellos, con lo que se descarga al procesador de realizar este trabajo. Aunque el cifrado/descifrado por software resulta bastante más económico, ya que no requiere de ningún hardware específico para su uso, el cifrado por hardware permite acelerar estas tareas y además, es más seguro, ya que el proceso de cifrado/descifrado se mantiene separado del resto de funciones de la máquina.

El objetivo de este trabajo es el desarrollo de una librería que permita la utilización de las extensiones criptográficas proporcionadas por ARM de manera sencilla. En particular, las funciones que implementan el estándar de cifrado AES y las funciones de resumen SHA-1 y SHA-2. La librería permitirá cifrar/descifrar cualquier flujo de información e, incluso, el cifrado/descifrado de ficheros de una manera eficiente gracias al hardware criptográfico que implementa la arquitectura ARMv8.

2. CAPÍTULO

Documento de objetivos del proyecto

Después de haber puesto en contexto el proyecto, en este capítulo vamos a detallar los objetivos que queremos alcanzar en este trabajo de fin de grado. El objetivo principal es la creación de una librería de alto nivel que permita el uso de una manera sencilla de las extensiones criptográficas presentes en algunos procesadores ARM. Mediante el uso de esta librería se podrán cifrar cualquier flujo de datos, como pueden ser mensajes enviados a través de una canal de comunicación o ficheros almacenados en el sistema de archivos local, utilizando para ello la aceleración hardware disponible en esos procesadores.

La primera tarea de este proyecto será el estudio y descripción de los algoritmos criptográficos que proporcionan las extensiones hardware. En particular, ARM implementa dos tipos de algoritmos de criptografía simétrica: (1) el estándar AES (Advanced Encryption standard)incluyendo los modos de operación CBC y GCM y (2) la familia de funciones criptográficas SHA (Secure Hash Algorithm) en las variantes SHA-1 y SHA-256.

Debido a que uno de los objetivos es el desarrollo de una librería, a la que llamaré *arm-commlib*, el siguiente paso consistirá en investigar el modo de implementarla en un sistema Linux. ARM ya proporciona una librería de bajo nivel (AArch64cryptolib) para utilizar las funciones hardware pero su uso es complicado y solo permite el uso de dichas funciones a muy bajo nivel. Por ello, la librería que se desarrollara será un *wrapper* de la de ARM que permita utilizar todos los algoritmos para cifra/descifrar información de cualquier tamaño.

Después de realizar las tareas de documentación descritas se procederá a la fase de desarrollo. Para ello, se preparará un entorno de desarrollo y pruebas que estará compuesto

por el *toolchain* de GNU y la mencionada librería de ARM. A continuación, se detalla cada una de las tareas que han sido identificadas para el desarrollo del proyecto:

1. Identificación de las actividades y etapas del desarrollo, planificación y seguimiento del trabajo de fin de grado.
2. Estudio del estándar de cifrado AES y de la familia de funciones criptográficas SHA.
3. Estudio del formato que utiliza Linux para la creación de librerías.
4. Estudio del API proporcionado por la librería criptográfica *AArch64cryptolib*.
5. Identificación de las herramientas a utilizar y preparación del entorno de desarrollo y pruebas.
6. Desarrollo y evaluación de la librería *armcommlib* desarrollada durante el proyecto.
7. Análisis de los resultados obtenidos añadiendo las conclusiones y lecciones aprendidas durante el desarrollo del proyecto.

El proyecto se desarrollará sobre una NanoPi R2S con el sistema operativo Ubuntu Core. NanoPi R2S es una plataforma hardware basada en ARM desarrollada por la empresa *FriendlyElec*. Dispone de 1GB de memoria RAM y un procesador Rockchip RK3328 (procesador quad-core Cortex-A53 de ARM que implementa las extensiones criptográficas).

3. CAPÍTULO

Advanced Encryption Standard (AES)

En este capítulo se analiza uno de los estándares de criptografía simétrica más utilizado. Se comienza haciendo una breve introducción a dicho estándar, seguido de un análisis exhaustivo de su funcionamiento. En particular, se detalla cada paso realizado en el proceso de cifrado/descifrado. Entre los conceptos que se describen se pueden destacar los diferentes modos de operación de AES, los vectores de inicialización y el rellenado.

3.1. Introducción

AES, también conocido como Rijndael, es un estándar de cifrado por bloques usado para el cifrado de datos electrónicos establecido en 2001 por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST). En particular, consiste en una serie de cifrados con diferentes modos de operación y tamaños de bloque desarrollado por los criptógrafos belgas Vincent Rijmen y Joan Daemen. Durante el proceso de selección del futuro estándar, dichos criptógrafos presentaron distintas propuestas y el NIST seleccionó tres miembros de la serie Rijndael con un tamaño de bloque de 128 bits pero diferentes longitudes de clave: 128, 192 y 256 bits.

Desde entonces, el uso de AES se ha extendido por todo el mundo y ha sustituido al Estándar de Cifrado de Datos (DES) publicado y utilizado desde 1977. El algoritmo empleado en AES es un algoritmo de clave simétrica, es decir, que la clave usada para el cifrado de datos es la misma que se usa para su descifrado.

El 26 de mayo de 2002, tras la aprobación del Secretario de Comercio de los Estados Unidos, AES, que está incluido en el estándar ISO/IEC 18033-3, entró en vigor como estándar del gobierno de EE.UU. AES se usa en muchos paquetes de encriptación y es el primer y único cifrado de acceso público utilizado en módulos criptográficos aprobados por la Agencia de Seguridad Nacional de los EE.UU. (NSA) para el cifrado de información de alto secreto.

3.2. Descripción del proceso de cifrado

AES está basado en el principio de red de sustitución-permutación y todas las operaciones las realiza sobre una matriz de 4 x 4 columnas de orden mayor de bytes a la que se denomina estado. La gran mayoría de los cálculos que se realizan en AES se hacen sobre lo que se conoce en álgebra abstracta como cuerpo finito, campo finito o campo de Galois.[1] [Nechvatal et al., 2000]

El cifrado en AES se realiza aplicando iterativamente (rondas de transformación) una serie de operaciones que convierten la entrada, denominada *texto plano*, en la salida, denominada *texto cifrado*. El número de rondas de transformación viene especificado por el tamaño de la clave utilizada como se puede ver a continuación:

- 128 bits = 10 rondas de transformación
- 192 bits = 12 rondas de transformación
- 256 bits = 14 rondas de transformación

El proceso de descifrado es similar, pero en este caso se aplica un conjunto de rondas de transformación inversas para convertir el texto cifrado en el texto plano original usando la misma clave. En cada ronda de transformación se aplican 4 tipos de operación, las cuales se detallan a continuación.

- Expansión de la clave: AES necesita una clave diferente para cada ronda que son derivadas de la clave inicial. Para ello se utiliza el planificador de claves de AES que expande la clave original en un número de claves de ronda diferentes. En total, AES utiliza un numero de claves de 128 bits igual al número de rondas más una más.

[1] Cuerpo definido sobre un conjunto finito de elementos.

- Ronda 1: Se le conoce como *AddRoundKey* y cada byte del estado se combina con un byte de la clave de la ronda usando la operación XOR.

- Rondas hasta la 9, 11 o 13 (dependiendo del tamaño de clave): En cada ronda se realizan las 4 operaciones siguientes:

    1. SubBytes: Paso de sustitución no lineal en el que cada byte se sustituye por otro basándose en una tabla predefinida (Rijndael S-box).
    2. ShiftRows: Paso de permutación donde todas las filas del estado, excepto la primera, se desplazan cíclicamente un cierto número de veces.
    3. MixColumns: Operación de transformación lineal que combina, en las columnas del estado, los cuatro bytes de cada columna. Multiplica cada una de las columnas de la matriz de estado por un polinomio fijo de manera similar a como se realiza en el cifrado Hill[2].
    4. AddRoundKey: Se realiza la misma operación realizada en la Ronda 1.

- Etapa final (Ronda 10, 12 o 14): Se aplican las operaciones SubBytes, ShiftRows y AddRoundKey.

A continuación, se describirán en detalle cada una de las operaciones realizadas en cada ronda de cifrado.

3.2.1. El paso SubBytes

En el paso SubBytes, representado en la Figura 3.1, cada byte $a_{i,j}$ de la matriz de estado se reemplaza con un SubByte $S(a_{i,j})$ utilizando una caja de sustitución Rijndael de 8 bits (S-box). El uso de la caja S proporciona no linealidad al cifrado y se construye utilizando el inverso multiplicativo sobre GF(2^8), el cual posee propiedades no lineales. La caja de sustitución Rijndael se crea combinando la función inversa con una transformación afín invertible con el fin de evitar ataques basado en propiedades algebraicas. A la hora del descifrado, se utiliza el paso InvSubBytes, que viene siendo el inverso del paso SubBytes, el cual requiere tomar el inverso de la transformación afín y luego encontrar el inverso multiplicativo.

[2]Cifrado de sustitución poligráfica basado en el álgebra lineal

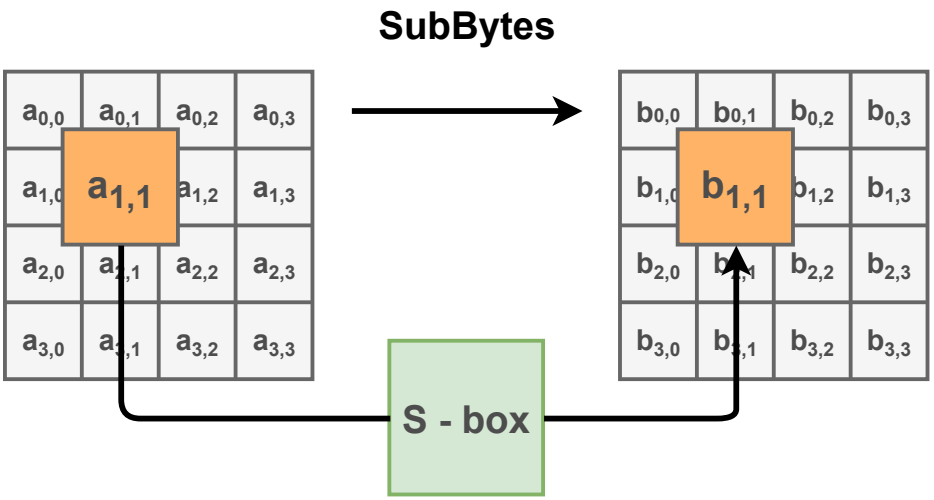

Figura 3.1: Representación del paso SubBytes.

3.2.2. El paso ShiftRows

El paso ShiftRows, representado en la Figura 3.2, se realiza en las filas del estado y los bytes de cada fila son desplazados un cierto número de posiciones. En particular, la primera fila se mantiene sin cambios y en la segunda fila cada byte se desplaza una posición hacia la izquierda. De esta manera, en la tercera y cuarta fila, cada byte se desplaza dos y tres espacios a la izquierda respectivamente. Una vez realizados los desplazamientos, cada columna del estado de salida quedará compuesta por bytes de cada columna del estado de entrada. Este paso sirve para evitar que las columnas se cifren independientemente, lo cual provocaría que AES degenerara en cuatro cifrados de bloque independientes.

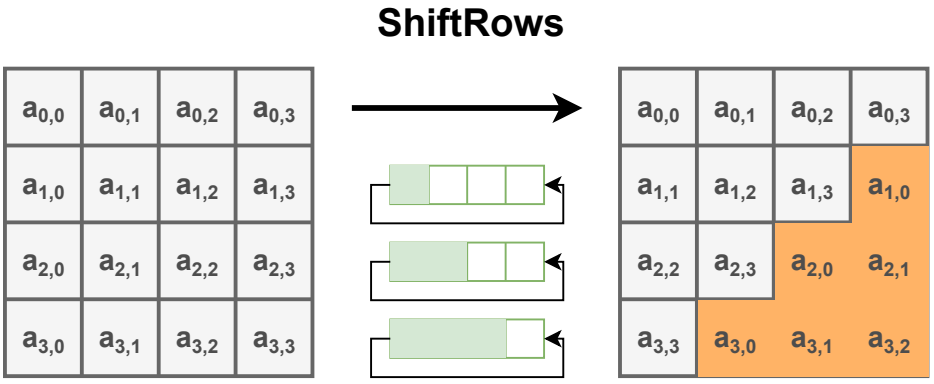

Figura 3.2: Representación del paso ShiftRows.

3.2.3. El paso MixColumns

En este paso, los cuatro bytes que componen cada columna del estado se combinan usando una transformación lineal invertible. Esta función usa cuatro bytes como entrada y cuatro bytes como salida, donde cada byte de entrada afecta a los cuatro bytes de salida. Este paso, junto a MixColumns y ShifRows, se utiliza para proporcionar difusión al cifrado, esto es, protección frente a ataques estadísticos y otros métodos de criptoanálisis. Durante esta operación cada columna se transforma usando una matriz de valores fijos para conseguir nuevos valores en la columna del estado (ver Figura 3.3).

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

Figura 3.3: Representación del paso MixColumns.

La multiplicación de la matriz está compuesta por la multiplicación y adición de las entradas. Las entradas son bytes tratados como coeficientes de un polinomio de orden x^7 y la adición es simplemente la operación XOR. Es decir, cada columna es tratada como un polinomio sobre GF(2^8) y luego se multiplica con un polinomio fijo. El paso MixColumns también puede verse como una multiplicación por la matriz MDS[3] de la mostrada en el campo finito GF(2^8). En la Figura 3.4 se representa el resultado de aplicar el paso MixColumns.

3.2.4. El paso AddRoundKey

En este paso, la subclave de ronda se combina con la matriz de estado. Esta subclave es derivada en cada ronda utilizando el planificador de claves AES y la clave utilizada en la ronda anterior. Todas las subclaves tienen el mismo tamaño que el estado y cada byte del estado se combina con un byte de la subclave usando la operación XOR. En la Figura 3.5 se ha representado el funcionamiento del paso AddRoundKey.

[3]La matriz Maximum Distance Seaparable (MDS) representa una función con ciertas propiedades de difusión que tienen aplicaciones útiles en criptografía.

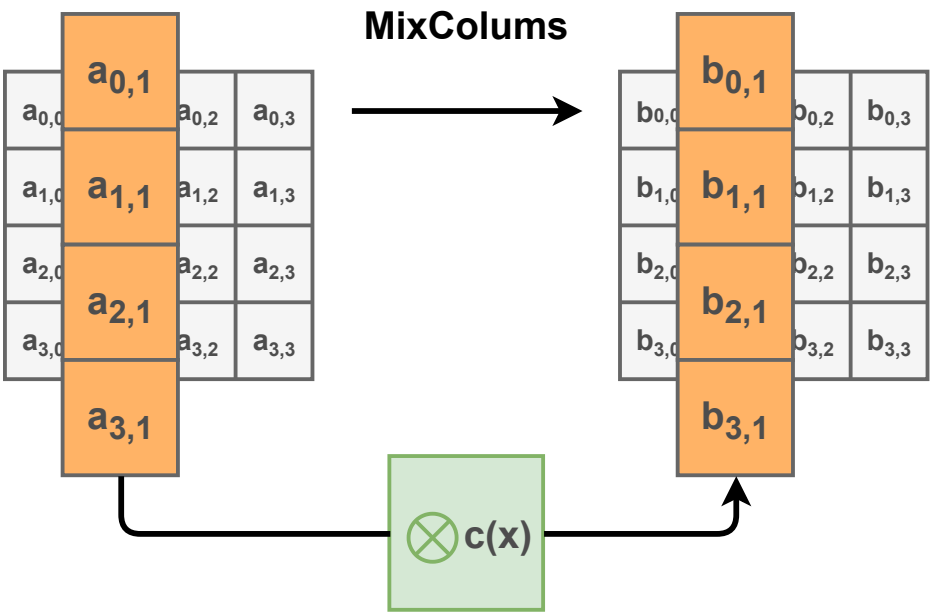

Figura 3.4: Representación del paso MixColumns.

3.2.5. Optimización del cifrado

En sistemas de 32 bits o más, existe la posibilidad de acelerar la ejecución del cifrado combinando los pasos de SubBytes y ShiftRows con el de MixColumns, transformándolos en una secuencia de búsquedas en tablas. Para combinar los pasos mencionados se necesitan cuatro tablas de 32 bits de 256 entradas (que en conjunto ocupan 4096 bytes). Una ronda se puede realizar con 16 operaciones de búsqueda en las tablas, 12 operaciones XOR de 32 bits y cuatro operaciones XOR de 32 bits en el paso AddRoundKey.

Alternativamente, es posible realizar las operaciones de búsqueda con una única tabla de 256 entradas de 32 bits (ocupando 1024 bytes) utilizando operaciones de rotación circular, o combinar los pasos SubBytes, ShiftRows y MixColumns en una única operación circular utilizando un enfoque orientado a bytes.

3.3. Modos de operación

El modo de operación describe cómo aplicar repetidamente una operación de cifrado en un bloque simple para la transformación segura de cantidades de datos mayores que un bloque y requieren de una secuencia binaria conocida como vector de inicialización (IV).

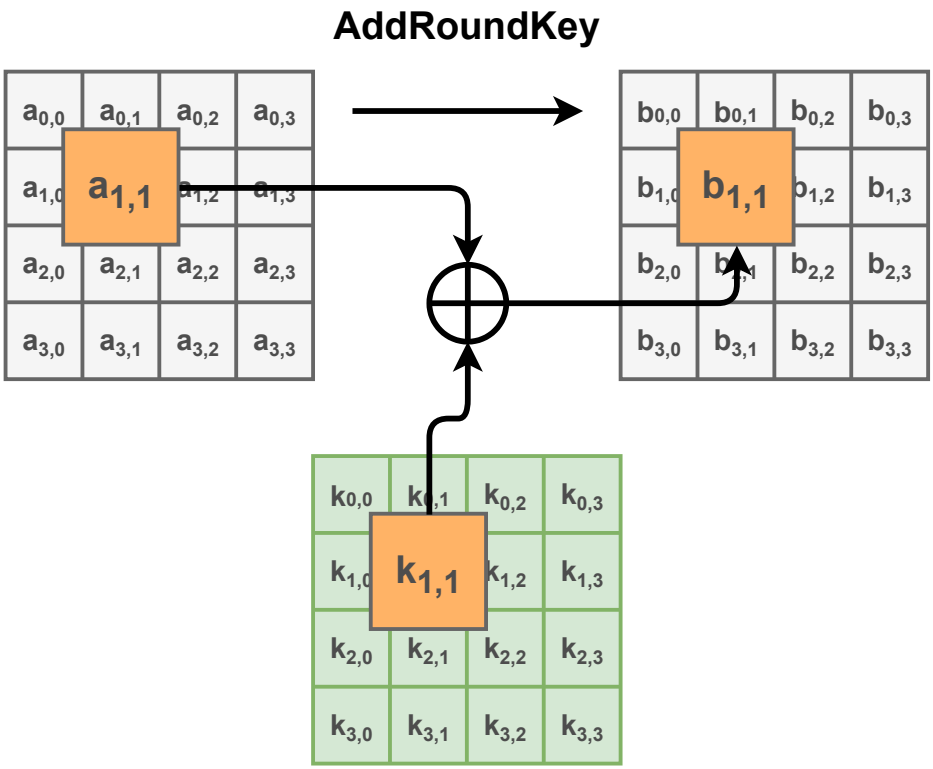

Figura 3.5: Representación del paso AddRoundKey.

El vector IV debería ser aleatorio para garantizar que, aun usando la misma secuencia de bytes, al cifrarla se produzcan secuencias cifradas distintas aunque la clave utilizada sea la misma. Los modos de cifrado por bloques hacen sus operaciones con bloques completos, por lo tanto, si el tamaño de bloque fuese menor a sus anteriores, la última parte de dicho bloque tendría que ser completada, proceso que se conoce como rellenado. Esto no se aplica a todos los modos de operación dado que los cifrados de flujo no requieren de un rellenado.

Los modos de operación ECB, CBC, OFB y CFB proporcionaban confidencialidad, pero no la integridad. Es por ello que se diseñaron otros modos como CCM, EAX y OCB para asegurar ambas características.

3.3.1. Cipher block chaining (CBC)

El modo de operación Cipher Block Chaining, más conocido como CBC, fue inventado en 1976 por Ehrsam, Meyer, Smith y Tuchman. En este modo de operación CBC, se aplica una operación XOR a cada bloque de texto en claro con el anterior bloque de texto

cifrado. De esta forma se consigue que cada bloque cifrado dependa de todos los bloques de texto en claro previos a ese punto. Además, para hacer cada mensaje único hay que partir de un vector IV. En las Figuras 3.6 y 3.7 están representados el funcionamiento del cifrado/descifrado utilizando el modo CBC.

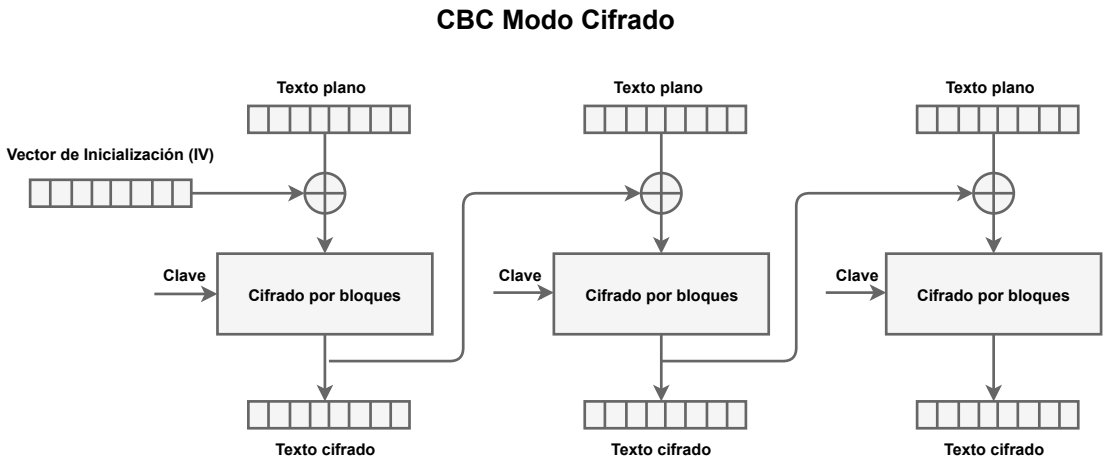

Figura 3.6: Representación del proceso de cifrado en el modo CBC.

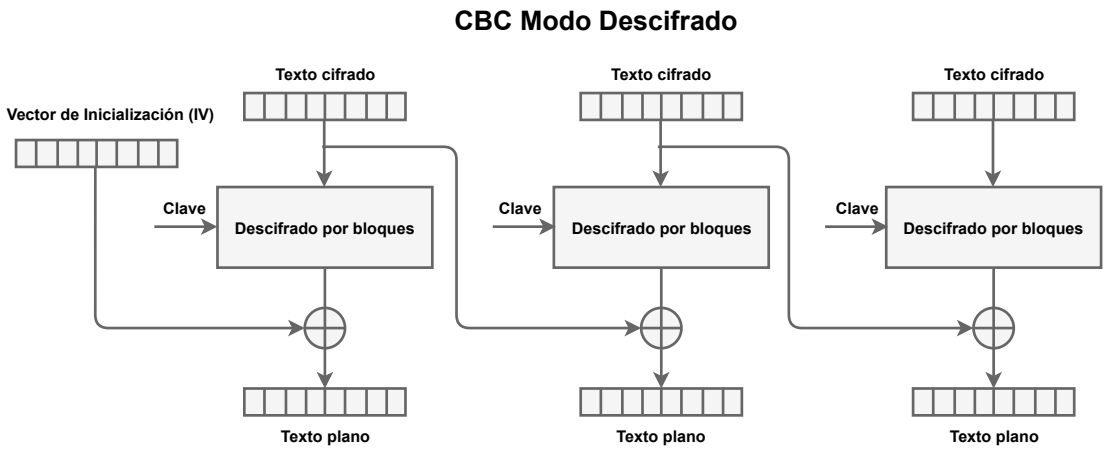

Figura 3.7: Representación del proceso de descifrado en el modo CBC.

Aunque CBC haya sido el modo de operación más usado, tiene dos grandes inconvenientes: (1) al ser un cifrado secuencial no se puede paralelizar y (2) el mensaje debe ser rellenado en caso de no ser un múltiplo de la longitud del bloque de cifrado. En el descifrado, no es necesario descifrar el bloque anterior antes de usarlo como IV para descifrar el bloque actual. Esto es debido a que en cada bloque se aplica una operación XOR con el texto cifrado del bloque anterior, no con el texto en claro. Por ello se puede recuperar

un bloque de texto en claro con sus bloques adyacentes de texto cifrado. En cambio, si el IV es incorrecto a la hora de descifrar el texto en claro, el primer bloque será corrompido, pero los siguientes bloques no. En la Figura 3.8 está representado el funcionamiento del descifrado CBC. [McGrew, 2014]

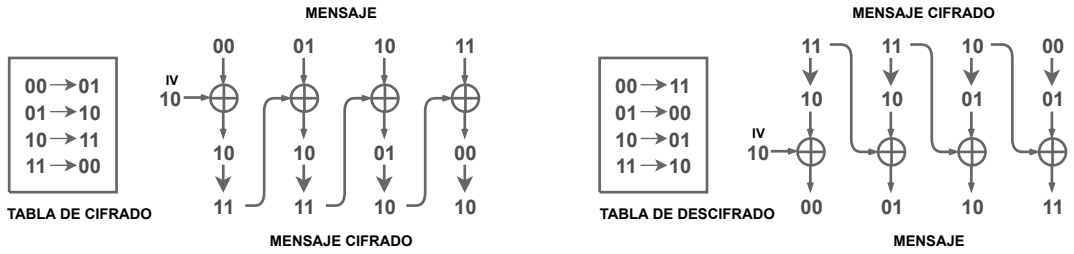

Figura 3.8: Representación del cifrado/descifrado en el modo CBC.

3.3.2. Galois/Counter (GCM)

GCM combina el modo de cifrado counter con el modo de autenticación de Galois. El modo counter convierte un bloque de cifrado en un cifrado de flujo, es decir, genera el siguiente bloque de flujo de claves mediante el cifrado de los valores sucesivos de un "contador". Como contador puede utilizarse cualquier función que produzca una secuencia que se repetirá durante mucho tiempo, aunque un contador de incremento de uno es el más simple y usado.

A diferencia del modo de operación CBC, GCM tiene la ventaja de que la multiplicación del Campo de Galois usada para la autenticación puede ser paralelizada, por lo que puede obtenerse un mayor rendimiento. Los bloques de cifrado en este modo de operación tienen un tamaño definido de 128 bits. [McGrew, 2008]

Para la autenticación, GCM dispone de una variante llamada GMAC (Galois message authentication code), la cual sirve para crear un código de autenticación incremental. Tanto GCM como GMAC pueden utilizar vectores IV de longitud arbitraria. En la figura 3.9 se ha representado el funcionamiento de GCM.

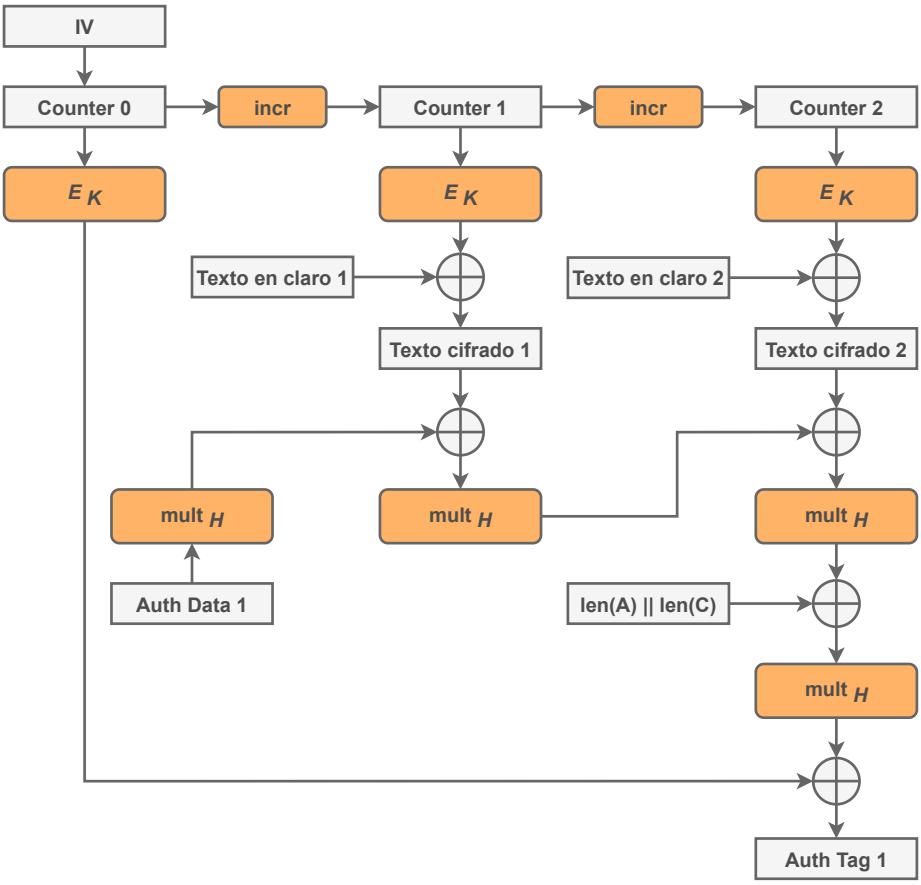

Figura 3.9: Representación del cifrado/descifrado en el modo GCM.

4. CAPÍTULO

Secure Hash Algorithms (SHA)

En este capítulo se analizan las funciones criptográficas de un solo sentido o hash, la cuales se utilizan para garantizar la integridad y confidencialidad de los datos. En particular, se estudia la familia SHA en la cual ha habido varias versiones a lo largo del tiempo que se han ido descatalogando por no ser seguras. Entre otras cosas, se analizarán sus características, el funcionamiento y en que ámbitos se utilizan.

4.1. Introducción

Secure Hash Algorithms (SHA) son una familia de funciones criptográficas publicadas por el NIST diseñadas para mantener los datos seguros. Para ello, lo que hacen básicamente es transformar los datos de entrada en un valor de salida de tamaño fijo mediante el uso de una función hash. Este tipo de funciones basan su funcionamiento en operaciones de bits, adiciones modulares y funciones de compresión.

Independientemente del tamaño de la entrada, las funciones hash siempre devuelven un código alfanumérico de tamaño fijo, el cual está determinado por el tipo de función que se le aplique. Para obtener el valor hash o resumen, el primer paso consiste en dividir la información de entrada en bloques de tamaño fijo denominados bloques de datos. Esto se debe a que estas funciones operan con bloques de tamaño fijo y el tamaño del bloque de datos varía dependiendo del algoritmo que se use. Debido a que normalmente el tamaño de los datos de entrada no corresponde con el tamaño del bloque, habrá que rellenar el

último bloque hasta que el tamaño sea múltiplo del tamaño de bloque. Este proceso se conoce como rellenado y consiste en llenar el último bloque con ceros.

Una de las propiedades más interesantes de las funciones hash es el efecto conocido como avalancha, que consiste en que cualquier modificación en los datos de entrada, por mínima que sea, provocará un gran cambio en los valores de la salida o que entradas de datos muy distintas pueden crear valores hash muy similares. Gracias a esta propiedad, los valores hash no revelan ninguna información acerca de los datos de entrada, ni de los valores ni de la longitud de los datos originales.

Este tipo de funciones proporcionan cifrados unidireccionales, es decir, una vez obtenido el valor cifrado ya no se pueden recuperar los datos originales. Es por ello que la aplicación más común de este tipo de algoritmos es el cifrado de contraseñas para su almacenaje. Por ejemplo, en servidores, en vez de guardar la contraseña del usuario, se almacena el valor hash de ella. De esta forma, los datos permanecen seguros ante un robo de información, ya que el atacante solo encontrará los valores hash en la base de datos y no podrá conocer el contenido original de la contraseña. Otra aplicación de las funciones hash es la detección de manipulaciones externas de los datos. Cualquier modificación en los datos causará que el valor hash sea distinto al de los datos originales permitiendo su detección. [Chaves et al., 2016]

4.2. Tipos de funciones SHA

La familia de funciones criptográficas SHA está compuesta por: SHA-0, SHA-1, SHA-2 y SHA-3, cada una diseñada para proporcionar un cifrado más fuerte y seguro que la anterior. En la Figura 4.1 SHA-0 y SHa-1, por ejemplo, actualmente no se utilizan debido a varias vulnerabilidades expuestas.

- SHA-0: Publicada en 1993 bajo el nombre de SHA. Es una función con una longitud de 160 bits. Al poco tiempo de su publicación fue retirada por un fallo significativo no revelado.
- SHA-1: Esta versión de 160 bits es similar al algoritmo MD5. La NSA la creó para formar parte del Algoritmo de Firma Digital. A partir de 2010 dejó de ser un estándar debido a algunas debilidades criptográficas.
- SHA-2: Esta versión está compuesta por dos funciones hash similares, pero con distinto tamaños de bloque: SHA-256 y SHA-512. La única diferencia entre ambas

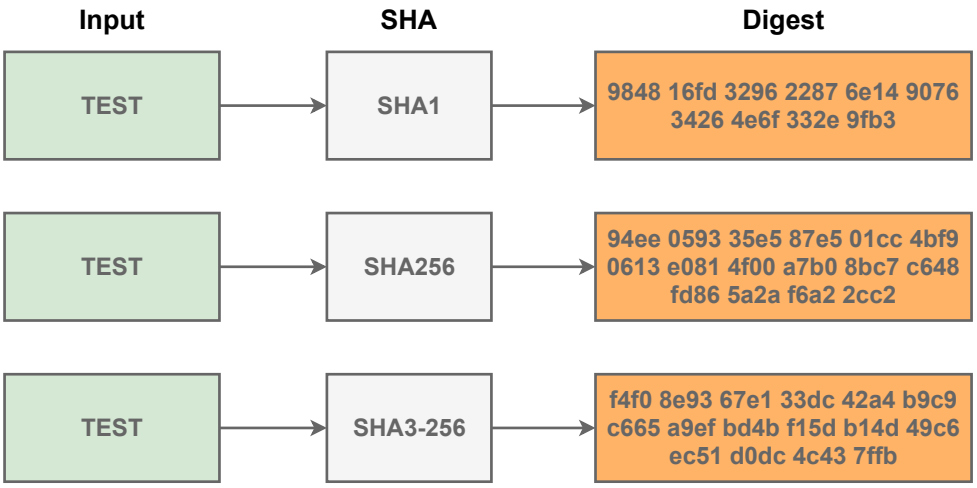

Figura 4.1: Tipos de SHA.

es el tamaño de bloque que utilizan: SHA-256 utiliza bloques de 32 bytes, mientras que SHA-512 utiliza bloques de 64 bytes.

- SHA-3: Antes conocida como Keccak, fue elegida en 2012 tras un concurso organizado por el NIST. Las longitudes pueden ser las mismas que las de SHA-2 pero su estructura es significativamente distinta a la del resto de la familia SHA.

4.3. Características de SHA

Las funciones hash sirven para proporcionar seguridad a los datos basándose en tres características de la seguridad de la información: resistencia de la preimagen, segunda resistencia de la preimagen y resistencia a la colisión.

La primera característica, resistencia de la preimagen, trata de, para una salida especificada, y, hacer computacionalmente inviable encontrar una entrada, x, cuyo valor hash sea esa salida. Es decir, encontrar una x tal que $h(x) = y$. Esta característica es proporcionada por cualquier de las funciones criptográficas unidireccionales y evita la realización de ataques de fuerza bruta.

La segunda característica proporcionada por SHA se llama resistencia de la segunda preimagen. Consiste en hacer computacionalmente inviable encontrar cualquier segunda entrada, x', que tenga la misma salida que una entrada específica, x. Es decir, dada x, encontrar una segunda preimagen, donde $x' \neq x$, resulta $h(x) = h(x')$. Sin esta carac-

terística dos entradas distintas (por ejemplo, contraseñas), podrían tener el mismo valor hash.

La última característica de seguridad es la resistencia a las colisiones (no confundir con la resistencia de la segunda preimagen). En esta característica es computacionalmente inviable encontrar dos entradas distintas x, y x' que se obtengan la misma salida; es decir, $h(x) = h(x')$. Esta característica se proporciona cuando el número de entradas posibles y de salidas posibles es similar y hay más entradas que salidas. Por esta razón, la resistencia a las colisiones es necesaria, ya que implica que es extremadamente difícil encontrar dos entradas que obtengan en el mismo valor hash.

La diferencia entre la característica de resistencia de la segunda preimagen y la resistencia a las colisiones consiste en que, en el primer caso, el atacante conoce la entrada, x, y debe encontrar una segunda entrada diferente, x', cuyos valores hash sean el mismo. En la resistencia a las colisiones, en cambio, el atacante trata de encontrar dos entradas diferentes, x y x', cuyos valores hash sean el mismo. Claramente, la resistencia a las colisiones implica una resistencia de la segunda preimagen.

En general, el comportamiento de una función hash debe parecer lo más aleatorio que sea posible, pero siendo proceso determinista y rápido de realizar.

5. CAPÍTULO

Librerías en el Kernel Linux

En este capítulo se analiza qué son y cómo se utilizan las librerías compartidas (*shared*) en entornos Linux. Asimismo, se estudiará cuál es su estructura interna y los dos tipos que se utilizan: estáticas y dinámicas.

5.1. Introducción y utilización

Una librería es un conjunto de códigos precompilados que ofrece cierta funcionalidad a través de una interfaz definida y que puede ser usada por varios programas. Las librerías suelen implementar funciones de bajo nivel que son usadas habitualmente en programas de más alto nivel evitando tener que implementar dichas funciones repetidamente. En particular, la característica distintiva de una librería es que permite a los programadores utilizar código sin necesidad de conocer los detalles internos de implementación.

Básicamente, cuando un programa utiliza una librería, este obtiene las funciones y el comportamiento implementado dentro de esa librería sin tener que implementar esas funciones o ese comportamiento por sí mismo. Las librerías facilitan la distribución del código. Actualmente, la mayoría de programas compilados están basados en una librería estándar, como son la librería estándar *libc* o la implementación de GNU, *glibc*, aunque también se pueden crear librerías propias.

El proceso para incorporar o utilizar el código de una librería en un programa ejecutable se denomina proceso de enlace (*linkage*). Este proceso, realizado por el enlazador *linker*,

básicamente se encarga de resolver las referencias de enlace a los módulos de librerías y se realiza en tiempo de compilación o en tiempo de ejecución. Este programa utiliza los objetos generados en los primeros pasos del proceso de compilación, la información de todos los recursos necesarios (librería), quita aquellos recursos que no necesita, y enlaza el código objeto con su librería para producir finalmente un fichero ejecutable o una librería.

Las referencias en un programa se almacenan de forma que no pueden ser resueltas hasta asignar de forma estática todas las direcciones al código y a la librería. Para hacer frente a este problema, existe un proceso de ajuste de referencias denominado proceso de reubicación. El encargado de este proceso puede ser tanto el enlazador como el cargador. En general, la reubicación no puede hacerse a las librerías individuales, porque las direcciones en la memoria pueden variar dependiendo del programa que las utiliza y de otras librerías con las que se combinan. Si el código es independiente a la posición, no requerirá de un proceso de reubicación ya que evita las referencias a direcciones absolutas.

5.2. Tipos de librería

Existen dos tipos de librerías dependiendo de dónde se ubica el código utilizado. Si el código de la librería es "utilizado" durante la fase de compilación del programa entonces es llamada librería estática. En cambio, si se construye el ejecutable del programa independientemente de la implementación de la librería, y la librería se utiliza después de que el programa se ejecute, ya sea al inicio de la ejecución, o durante ella, se denomina librería dinámica. Es decir, las librerías dinámicas se cargan en el tiempo de ejecución.

5.2.1. Librería estática

Una librería se denomina estática cuando la vinculación se realiza durante la creación de un ejecutable, es decir, en tiempo de compilación. El encargado de realizar la vinculación suele ser el enlazador, aunque también puede realizarlo el compilador. Una biblioteca estática, como indica su nombre, se enlaza estáticamente. Originalmente, sólo existían bibliotecas estáticas. El enlace estático debe realizarse cuando se recompila cualquier módulo.

A veces, los módulos requeridos por un programa se enlazan estáticamente y se copian en el archivo ejecutable como puede verse en la Figura 5.1. A este proceso se le conoce

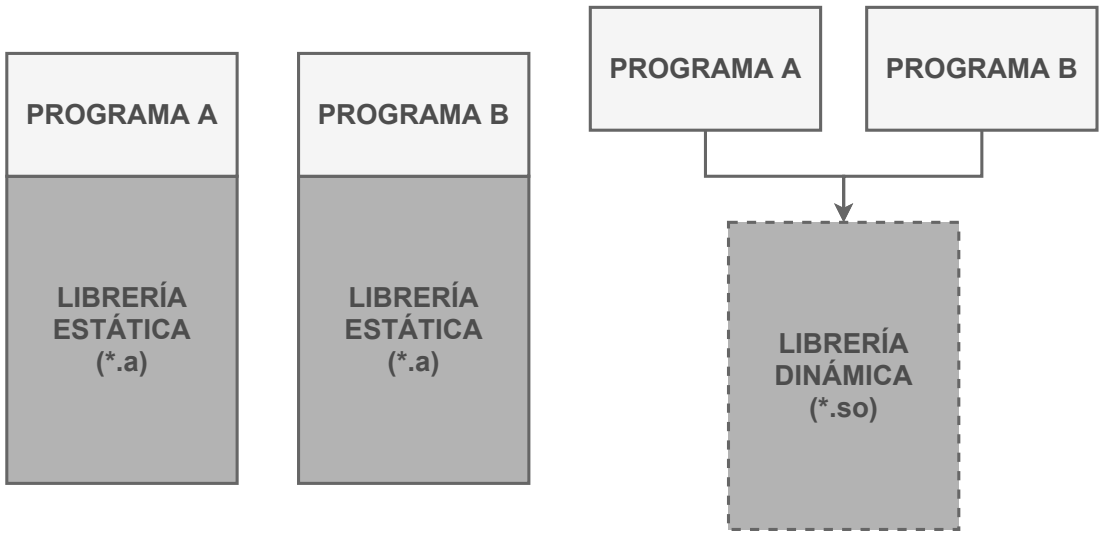

Figura 5.1: Diferencias entre librerías estáticas y dinámicas

como construcción estática del programa. Ésta puede no necesitar ninguna reubicación si se utiliza la memoria virtual ya que la direcciones serán traducidas en tiempo de ejecución.

Para compilar una librería estáticamente hay que crear los ficheros objeto y después enlazarlos. El proceso se describe a continuación:

```
/*
 * *.c: ficheros fuente
 * libname.a: fichero de la librería
/*
gcc -c *.c
ar -rc libname.a *.o
```

Listing 5.1: Compilación de una librería estática.

5.2.2. Librería dinámica

Las librerías dinámicas son archivos compartidos con archivos ejecutables u otros archivos de objetos compartidos. A diferencia de las librerías estáticas, los módulos que se utilizan se cargan desde los objetos compartidos en la memoria justo en el momento de la carga o el tiempo de ejecución. Las librerías dinámicas también pueden enlazarse estáticamente en el momento de la compilación. A efectos prácticos, significa que las referencias a los módulos de la librería se resuelven y se asignan a la memoria en la creación del ejecutable.

Tanto los ejecutables como las librerías dinámicas utilizan el mismo formato. Gracias a ello, un solo cargador es suficiente para resolver las referencias y, además, permite a los ejecutables, en caso de tener una tabla de símbolos, comportarse como una librería dinámica. Los formatos habituales para la combinación de ejecutables y librerías dinámicas son ELF y Mach-O para UNIX y PE para Windows.

Para compilar una librería dinámica primero hay que crear los ficheros de objeto. Después, hay que crear la librería y por último, hay que añadir la librería a un *path* predeterminado del sistema (por ejemplo */usr/lib*). El proceso se describe a continuación:

```
/*
 * *.c: ficheros fuente
 * libname.so: fichero de la librería
/*
gcc -c -fpic *.c
gcc -shared -o libname.so *.o
export LD\_LIBRARY\_PATH=$HOME/usr/lib:$LD\_LIBRARY\_PATH
```

Listing 5.2: Compilación de una librería dinámica.

6. CAPÍTULO

Extensiones criptográficas de ARM

En este capítulo se describen la arquitectura hardware ARMv8-A y las extensiones criptográficas de ARM. En particular se va a describir la arquitectura del procesador Cortex-A53 así como las diferentes funciones criptográficas que se implementan en hardware. Este procesador se analizará en profundidad dado que es el que se utilizará en la realización de este proyecto.

6.1. Procesador Cortex-A53

El Cortex-A53 es un procesador superescalar de 64 bits; que soporta la ejecución de código tanto de 32 como de 64 bits, este procesador pertenece a la familia de arquitecturas ARM (Advanced RISC Machines), la cual está basada en un conjunto de instrucciones RISC. Las características principales de los procesadores ARM son el bajo coste y el bajo consumo de energía. Debido a ello, son una de las opciones más utilizadas para construir sistemas empotrados, tablets, dispositivos ligeros, etc.

Sin embargo, ARM también dispone de procesadores de alto rendimiento como son el Cortex-A53, el cual dispone de uno a cuatro núcleos, cada uno de ellos con una memoria cache L1 y una caché L2 compartida entre todos. Una configuración habitual, denominada *big.LITTLE*, consiste en la combinación de núcleos de alto rendimiento con otros menos potentes pero con menor consumo de energía. [ARM, 2021b]

En la Figura 6.1 se han representado los principales sistemas que componen este pro-

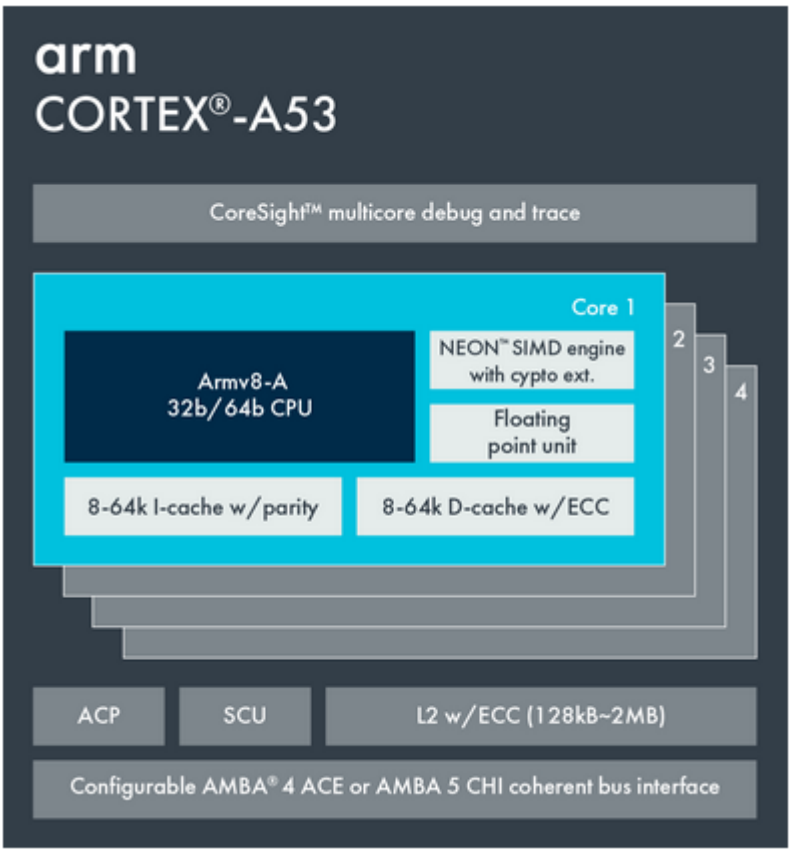

Figura 6.1: Procesador ARM Cortex-A53.

cesador, entre los que podemos destacar las extensiones vectoriales (SIMD) NEON que incorporan las extensiones criptográficas. A continuación, se describen las principales características de procesador Cortex-A53:

- Arquitectura ARMv8-A.
- Pipeline de ocho etapas y ejecución en orden.
- Menor consumo de energía utilizando técnicas como la regulación jerárquica del reloj, los dominios de energía y los modos de retención avanzados.
- Recursos de ejecución duplicados y decodificadores de instrucciones duales.
- El diseño de la caché L2 con optimización de energía ofrece una latencia más baja y equilibra el rendimiento con la eficiencia.
- Compatible con la arquitectura AArch32 para una compatibilidad total con ARMv7.

- Implementa la arquitectura AArch64 de 64 bits.
- Tecnología TrustZone que permite aislar los recursos críticos del sistema.
- SIMD (Single Instruction, Multiple Data) avanzada NEON proporciona instrucciones que operan sobre varios elementos simultáneamente mediante el uso de registros vectoriales.
- Extensiones DSP (Procesado de señales digitales) que proporcionan un conjunto de instrucciones optimizadas para aplicaciones que requieran operaciones numéricas a muy alta velocidad.
- Soporte de virtualización en hardware que permite la abstracción de los recursos.

6.2. Arquitectura ARMv8-A

El procesador Cortex-A53. perteneciente a la familia Cortex-A, utiliza una microarquitectura que implementa el conjunto de instrucciones de 64 bits ARMv8-A. Dentro de las arquitecturas desarrolladas por ARM, ARMv8-A es la última generación orientada a aplicaciones. En la Figura 6.2 se puede ver la evolución de las arquitecturas de ARM desde la v5. Esta arquitectura es compatible con la ejecución de aplicaciones de 32 y de 64 bits. AArch64 (ARM64 en la documentación de Linux) es el nombre utilizado para describir la versión de 64 bits de la arquitectura ARMv8.

El conjunto de instrucciones proporcionado por esta arquitectura proporciona un rendimiento mayor al incluir un conjunto de registros más grande. Estos registros adicionales junto con la ARM Architecture Procedure Call Standard (AAPCS) proporcionan un aumento de rendimiento cuando hay que pasar más de cuatro registros en la llamada a una función, mientras que en ARMv7, esto requería usar la pila. En esta nueva versión (AArch64) se puede pasar hasta ocho parámetros a las funciones utilizando los registros.

Los registros de enteros son más grandes y permiten que el código que opera con datos de 64 bits funcione de manera más eficiente. También es necesario mencionar que las operaciones de 64 bits permiten a las aplicaciones utilizar un espacio de direcciones virtuales más amplio. [ARM,]

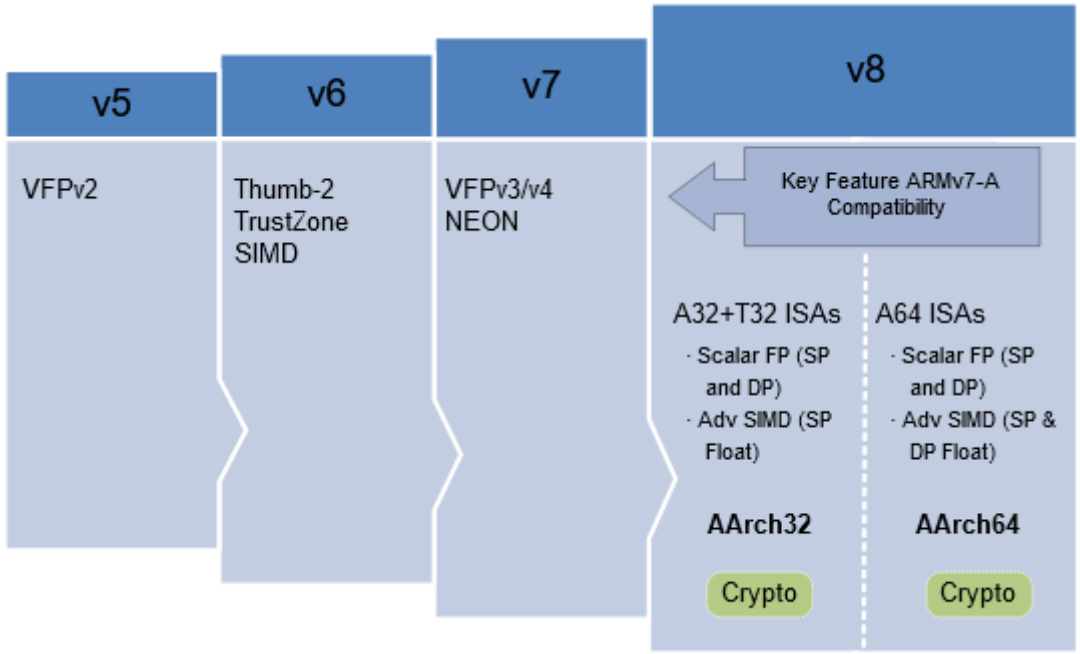

Figura 6.2: Evolucion de las ARM hasta la ARMv8.

6.3. Extensiones criptográficas

ARMv8 proporciona extensiones hardware para la aceleración de funciones criptográficas simétricas (cifrado/descifrado). En particular, incluye implementaciones del estándar de cifrado AES (con las instrucciones AESD y AESE) y la familia de funciones criptográficas SHA. La extensiones criptográficas también proporcionan instrucciones de multiplicación que operan en polinomios largos en los modos AArch64 y AArch32. En particular, se añaden nuevas instrucciones A64, A32 y T32 a las instrucciones vectoriales NEON (SIMD) que aceleran el cifrado/descifrado del estándar de cifrado avanzado (AES) y las funciones criptográficas hash (SHA) SHA-1, SHA-224 y SHA-256.

Debido a que la implementación de las extensiones criptográficas es opcional (muchos fabricantes no las implementan debido a que hay que pagar por ellas) es necesario comprobar su presencia antes de utilizarlas. Para ello es necesario acceder a los campos que componen el registro ID_AA64ISAR0_EL1 en el modo de ejecución AArch64. Como se puede ver en la Figura 6.3, existen varios campos en el registro que nos indican si las funciones AES, SHA1 y SHA2 están implementadas.

En la Tabla 6.1 se han representado los intervalos de bits del registro que representan si cada función se encuentra implementada (AES, SHA1 y SHA2). La presencia de un 1

Figura 6.3: Campos del registro ID_AA64ISAR0_EL1.

indica que la función está implementada en el hardware, en caso de que su valor sea 0 no lo está (o en el caso de AES, la presencia de un 2 indica que la función está implementada). Aun sabiendo que nuestro procesador las implementa, es imprescindible que la librería compruebe la presencia de dichas funciones en el hardware. Para ello, se ha implementado una función en C que implementa una instrucción en ensamblador que guarda el valor de ID_AA64ISAR0_EL1 en una variable de 64 bits (ver Listing 6.1).

```
/*
 * En el registro <Xt> se guarda el contenido del registro ID\_AA64ISAR0\_EL1.
 * MRS <Xt>, ID\_AA64ISAR0\_EL1;
 * En la llamada en C se sustituye el registro por una variable de 64 bits.
/*
unsigned long info;
asm("MRS \%[result], ID\_AA64ISAR0\_EL1": [result] "=r" (info));
```

Listing 6.1: Comprobación de la existencia de las extensiones criptográficas.

El resultado de llamar a esta función en nuestro procesador devuelve en la variable *info* el valor 11120 en los primeros 20 bits. Comparando este valor con los valores indicados en la Tabla 6.1, se puede comprobar (de izquierda a derecha) que el procesador implementa CRC32=1, SHA2=1, SHA1=1 y AES=2. El último 0 hace referencia al registro reservado res0.

6.4. Análisis de la librería AArch64cryptolib

En esta sección se va analizar las interfaz (funciones) que proporciona la librería criptográfica de ARM AArch64cryptolib[ARM, 2020] para el cifrado y descifrado de datos. En los *Listings* 6.2 y 6.3 se han representado todas las funciones que nos ofrece. Estas funciones se pueden clasificar en dos grupos; por un lado tenemos el grupo de funciones que implementan el AES (cifrado/descifrado) en el modo de operación CBC. En este caso, estas funciones también proporcionan la implementación SHA1 y SHA2 por lo que ambos algoritmos serán aplicados a la entrada proporcionada a las funciones. Hay que se-

Bits	Name	Function
[63:20]	-	Reserved, res0.
[19:16]	CRC32	**0x1** CRC32 instructions are implemented
[15:12]	SHA2	Indicates wheter SHA2 instructions are implemented. The possibles values are: **0x0** No SHA2 instructions are implemented. This is the values if the implementation does not include the Cryptography Extension. **0x1** SHA256H, SHA256H2, SHA256U0, and SHA256U1 implemented. This is the value if the implementation includes the Cryptography Extension.
[11:8]	SHA1	Indicates wheter SHA1 instructions are implemented. The possibles values are: **0x0** No SHA1 instructions are implemented. This is the values if the implementation does not include the Cryptography Extension. **0x1** SHA1C, SHA1P, SHA1M, SHA1SU0 and SHA1SU1 implemented. This is the value if the implementation includes the Cryptography Extension.
[7:4]	AES	Indicates wheter AES instructions are implemented. The possibles values are: **0x0** No AES instructions are implemented. This is the values if the implementation does not include the Cryptography Extension. **0x2** AESE, AESD, AESMC, and AESIMC implemented, plus PMULL and PMULL2 instructions operating on 64-bit data. This is the value if the implementation includes the Cryptography Extension.
[3:0]	-	Reserved, res0.

Tabla 6.1: Campos de bits del registro ID_AA64ISAR0_EL1.

ñalar que como las funciones están implementadas en hardware ambos algoritmos serán ejecutados en paralelo sin coste temporal extra.

En el modo de operación AES CBC encontramos dos parejas de funciones en las que se usa una *key*[1] de 128 bits, por lo que se utilizarán en 10 rondas de transformación

[1]Clave de cifrado/descifrado

(ver Sección 3.2) para el cifrado/descifrado. Una de las parejas que utilizan el modo de operación CBC usa el algoritmo de *hashing* SHA-1, mientras que la otra usa el SHA-2, concretamente el SHA-256.

El prototipo de las cuatro funciones que usan el modo de operación CBC es igual por lo que utilizan el mismo número y tipo de parámetros (ver *Listing* 6.2). Requieren de una secuencia de bytes a encriptar/desencriptar y un *output* donde se guardará la secuencia de bytes codificada/decodificada. En ambos casos hay que especificar el tamaño, el cual no puede ser mayor de 256 bits y en nuestro caso lo hemos fijado a 128 bits. Como se ha mencionado previamente, AES utiliza para el cifrado/descifrado bloques de 128 bits (16 Bytes).

```
/*
 * Prototipos de las funciones proporcionadas por la librería AArch64cryptolib.
/*
int armv8_enc_aes_cbc_sha1_128(
                reference_plaintext, output, block_byte_length,
                reference_plaintext, auth, block_byte_length,
                &arg);
int armv8_enc_aes_cbc_sha256_128(
                reference_plaintext, output, block_byte_length,
                reference_plaintext, auth, block_byte_length,
                &arg);
int armv8_dec_aes_cbc_sha1_128(
                reference_ciphertext, output, block_byte_length,
                output, auth, block_byte_length,
                &arg));
int armv8_dec_aes_cbc_sha256_128(
                reference_ciphertext, output, block_byte_length,
                output, auth, block_byte_length,
                &arg));
```

Listing 6.2: Prototipos de las funciones (CBC) expuesta por la librería AArch64cryptolib.

Como se ha comentado, las cuatro funciones también implementan los algoritmos SHA. Por ello, es necesario indicar dónde se va a almacenar el valor devuelto por el algoritmo de hashing, el cual se aplica a la secuencia de bytes de entrada y se almacena en la variable *auth*. Además, debido a que AES utiliza cifrado por bloques, es necesario inicializar el vector *IV* y, por supuesto, una clave de cifrado/descifrado (*key*). A la hora de inicializar el vector *IV* se recomienda utilizar valores aleatorios.

En resumen, el resultado de utilizar estas funciones es el siguiente: (1) cifrado/descifrado de la secuencia de bytes y (2) hashing de la secuencia de bytes, es decir, *[aes(m), sha(m)]*, donde *m* es la secuencia de bytes.

En lo que respecta al modo de operación GCM, la librería proporciona otras dos parejas de funciones: *from_state* y otra *from_constants_IPsec*. En este proyecto solo se va a utilizar la pareja de funciones *from_state*, dado que no nos interesa utilizar el protocolo seguro de comunicaciones IPsec. Sin embargo, hay que señalar que el uso de este protocolo y su incorporación a la librería serán explorados en el futuro (ver Sección 9).

```
/*
 * Prototipos de las funciones proporcionadas por la librería AArch64cryptolib.
/*
int armv8_enc_aes_gcm_from_state(
            &cs,
            aad, aad_length,
            reference_plaintext, plaintext_length,     //Inputs
            output,
            tag); //Outputs
int armv8_dec_aes_gcm_from_state(
            &cs,
            aad, aad_length,
            reference_ciphertext, plaintext_length,
            reference_tag,
            output); //Outputs
```

Listing 6.3: Prototipos de las funciones (GCM) expuesta por la librería AArch64cryptolib.

En el caso de las funciones *from_state* (ver *Listing* 6.3), es necesario utilizar un contador (*counter*) (el parámetro *&cs*), ya que como se ha explicado en la Sección 3.3.2, el modo GCM opera en base a los valores sucesivos de un contador. El modo de operación GCM también requiere de un *aad* y del tamaño de éste *aad_length* que se usa para comprobar la integridad de los datos (similar al *auth* de CBC). Como cualquier modo de cifrado, es necesario indicar la secuencia de bytes que se desea cifrar/descifrar *reference_plaintext/reference_ciphertext* junto con la longitud de esta *plaintext_length*. Además, hay que indicar donde se va a guardar el valor de la secuencia de bytes cifrada/descifrada, *output* y *tag*.

7. CAPÍTULO

Desarrollo de la librería armcommlib

En este capítulo se describe el desarrollo de la librería armcommlib. En la primera parte se explica el diseño de la librería, los pasos realizados para el desarrollo de ésta y como interactúa con la librería AArch64cryptolib. Comenzaremos explicando su estructura, seguido de un listado de las funciones que la componen junto con una explicación detallada de cada una de ellas. En la segunda parte procederemos a la evaluación y testeo de la librería. Para ello, se presenta el entorno de pruebas. El capítulo finaliza explicando los pasos necesarios para su uso en un programa.

7.1. Diseño de la librería

Como ya hemos comentado en el capítulo 2, el objetivo principal de este proyecto es crear una librería que implementa funciones para el cifrado/descifrado de información y hacer uso de ellas en comunicaciones en tiempo real. Para empezar con el desarrollo de nuestra librería armcommlib, primero hemos tenido que realizar un estudio acerca de las librerías informáticas, cómo funcionan, cómo se crean, los tipos que hay y cómo se implementan. Todo este estudio está reflejado en el estado del arte en el capítulo 5 sobre las librerías.

Después hemos tenido que analizar la librería criptográfica de ARM AArch64cryptolib con el fin de comprender las funciones que implementa y así poder crear y adaptar el código para crear nuestras propias funciones para un uso más común. Hemos decidido crear una librería dinámica, ya que hacen un uso menor de la memoria RAM y además

podemos instalarla en un path[1] predeterminado del ordenador para no tener que compilar cada vez que realicemos algún cambio en los programas que la utilicen.

7.1.1. Estructura de la librería

Nuestra librería se compone de cinco archivos distintos: *aes_decrypt.c*, *aes_encrypt.c calculate.c*, *armcommlib.h* y un *Makefile* para la compilación e instalación de la librería.

Armcommlib.h es el archivo principal de la librería, básicamente es el archivo cabecera de ésta. Aquí se encuentran declaradas todas las funciones que hemos implementado, así como las variables globales. Este archivo es el que tendremos que referenciar si queremos hacer uso de nuestra librería en un programa. La librería AArch64cryptolib está enlazada directamente a nuestra librería, ya que hacemos uso de las funciones que nos ofrece ARM.

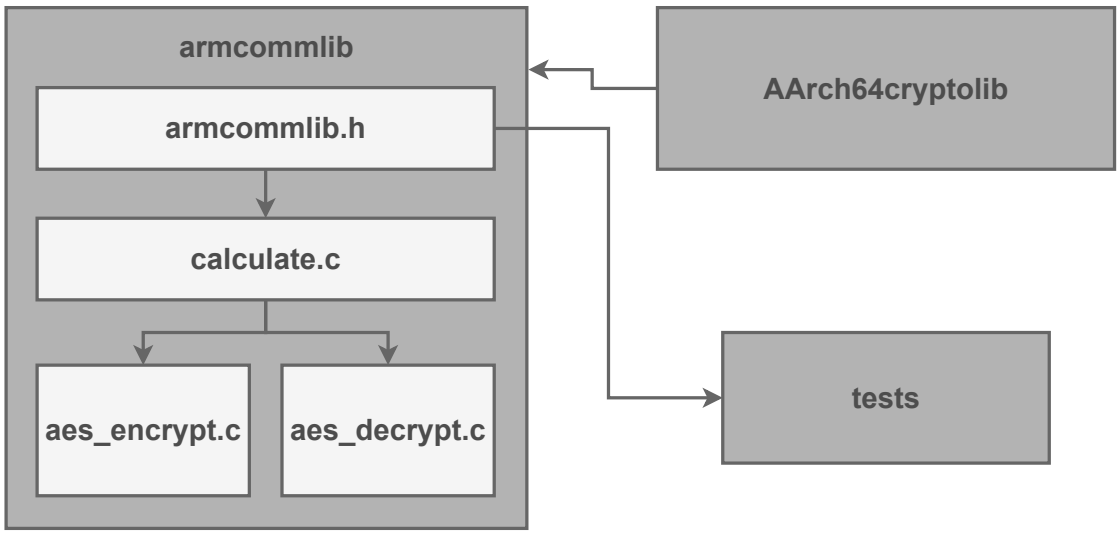

Figura 7.1: Estructura de la librería armcommlib

Calculate.c contiene las funciones que vamos a usar en los programas en los que referenciamos nuestra librería. Estas funciones son: *encrypt*, *decrypt*, *encrypt_file* y *decrypt_file*. En sí, éstas van emparejadas, una para el cifrado y otra para el descifrado, la primera pareja de *encrypt* y *decrypt* se usa para el cifrado y descifrado de mensajes y la segunda pareja, *encrypt_file* y *decrypt_file*, como bien indica el nombre, está pensada para cifrar y descifrar ficheros. Dentro de cada función se realiza un cálculo para determinar el número

[1]Path: es una variable de entorno de los sistemas operativos que especifican las rutas en las cuales, el intérprete de comandos debe buscar los programas a ejecutar.

de veces que ha de usarse la función adjunta a éstas para el cifrado y descifrado de información. Como ya hemos explicado previamente, el tamaño de la información a cifrar o descifrar no puede superar los 128 bits, es por ello que realizamos este cálculo.

Aparte, cuando hagamos uso de estas funciones tendremos que especificar los siguientes parámetros:

- **message/file:** indica el mensaje o fichero (dependiendo de la pareja de funciones que se utilice) que se desea cifrar.
- **enc_result/dec_result:** aquí es donde se va a guardar el valor del cifrado o descifrado.
- **mode:** indica el modo en el que se va a cifrar, el cual dispone de tres modos distintos: *aes_cbc_sha128*, *aes_cbc_sha256* y *aes_gcm_from_state*. Al pasarle este parámetro no hay que indicar el nombre de los modos sino un valor numérico; 0 para *aes_cbc_sha128*, 1 para *aes_cbc_sha256* y 2 para *aes_gcm_from_state*.
- **size:** indica el tamaño del mensaje a cifrar/descifrar
- **key:** la clave que vamos a usar para el cifrado/descifrado.
- **hash:** aquí es donde se va a guardar el valor para la autenticación del cifrado/descifrado.

Para *encrypt_file* y *decrypt_file* basta con pasarle solo los parámetros de *file*, *mode* y *key*, ya que, la función misma es la que se encarga de calcular el tamaño del fichero, y además crea su propio buffer para almacenar el valor del cifrado o descifrado.

En *aes_decrypt.c* encontramos las funciones para el descifrado de información, donde diferenciamos los tres modos distintos de descifrado mencionados. *Aes_encrypt.c* contiene los mismos modos que *aes_decrypt.c* pero en este caso para el cifrado de información. Los modos empleados en ambos archivos son los que hemos sacado de la librería AArch64cryptolib.

Los modos de cifrado y descifrado que disponemos en los archivos *aes_decrypt.c* y *aes_encrypt.c* son funciones adaptadas para crear nuestra librería. En las funciones que encontramos en los dos archivos mencionados, hacemos una llamada a los modos originales de ARM, que son los que implementan toda la secuencia de cifrado y descifrado.

Los tres modos son; *aes_cbc_sha_128*, *aes_cbc_sha_256* y *aes_gcm_from_state*. Todos ellos sirven para cifrar y descifrar información pero de distinta manera.

Cabe mencionar que los nombres exactos de las funciones de ARM no son esos, dado que hay una para el cifrado y otra para el descifrado de cada una de las que hemos nombrado y, por ende, contienen nombres diferentes en base a si se va a cifrar o descifrar.

El nombre original y la explicación de su funcionamiento se encuentra en el capítulo 6.

7.2. Entorno de experimentación y diseño de pruebas

Una vez que tenemos diseñadas e implementadas nuestras funciones es hora de realizar una serie de pruebas con el fin de comprobar su eficacia y funcionamiento.

Si tenemos en cuenta que nuestras funciones están pensadas para trabajar con comunicaciones en tiempo real, el tiempo dedicado para el cifrado debería ser mínimo para que no sea perceptible y no afecte a la calidad de la comunicación.

Dicho esto, hemos diseñado dos sistemas de mensajería cliente-servidor TCP[2] basados en sockets[3] para el testeo de la comunicación.

7.2.1. Sistema de mensajería cliente-servidor

Para poder realizar pruebas fiables y con una aplicación real hemos diseñado este sistema cliente-servidor. Para el diseño del sistema nos hemos basado en sockets que nos permiten establecer conexiones entre programas que se ejecuten de forma independiente. Generalmente, suele haber un servidor que se encarga de aceptar conexiones remotas de otros clientes, en el cual puede haber hasta más de un cliente.

En cuyo caso, para comprobar el funcionamiento de las funciones *encrypt* y *decrypt* hemos creado un sistema cliente-servidor para el cifrado y descifrado de mensajes y otro sistema, también cliente-servidor, para comprobar el funcionamiento de *encrypt_file* y *decrypt_file*. En él hemos implementado el mismo sistema pero en vez de cifrar y descifrar mensajes, se cifran y descifran ficheros.

[2]TCP: Transmission Control Potocol, es un protocolo de red que permite que dos anfitriones (hosts) se conecten e intercambien flujos de datos.

[3]Sockets: constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados.

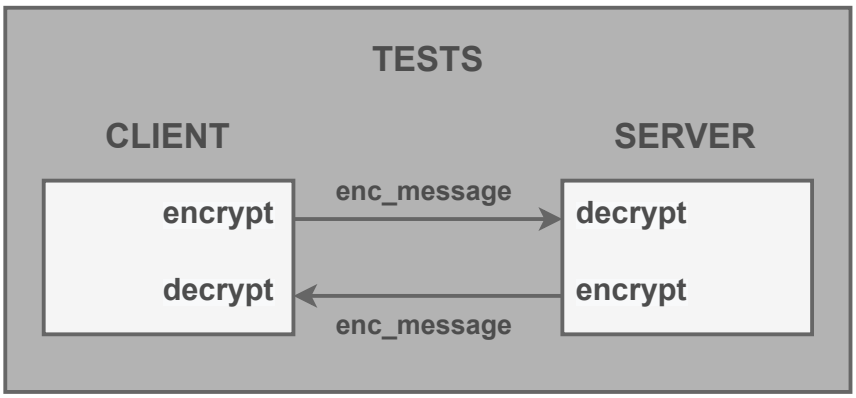

Figura 7.2: Sistema cliente-servidor basado en sockets

Estos dos sistemas que hemos implementado hacen uso de nuestra librería armcommlib para así poder realizar las pruebas necesarias para el testeo.

En el sistema de testeo de mensajes, cuando se establece la conexión cliente-servidor da comienzo el intercambio de mensajes en donde nosotros, como usuarios, podemos escribir el mensaje que deseamos enviar, empezando por el cliente. Antes de enviar el mensaje, el sistema aplica la función *encrypt* la cual hemos implementado en nuestra librería. Una vez encriptado, el mensaje es enviado y nada más recibirlo y antes de mostrárnoslo, se le aplica la función para el descifrado *decrypt*. De esta forma, el mensaje a la hora de la comunicación viaja por la red de forma segura, ya que ha sido cifrado antes del envío y en la recepción es cuando se le aplica el descifrado.

Para el testeo del cifrado de ficheros, esta vez, al ejecutar el cliente, hay que indicar el nombre del fichero que se desea cifrar y, en el servidor, hay que indicar el nombre del fichero en el que se desea descifrar y guardar la información del fichero que ha indicado el cliente. Para realizar esta tarea, este sistema implementa las funciones *encrypt_file* y *decrypt_file* de nuestra librería. Este sistema, primero abre el fichero que hemos indicado que queremos cifrar, almacena el contenido en un buffer y después lo cifra para enviarlo al servidor. El servidor, una vez recibe el buffer, lo descifra y por último lo guarda en un fichero.

7.3. Implementación de la librería en un programa

En este último apartado del capítulo procedemos a explicar los pasos a dar para usar nuestra librería en un programa informático propio.

Primero de todo, hay que remarcar que, para hacer uso de nuestra librería, es necesario que se disponga de un procesador con la arquitectura ARMv8, la cual, implementa las extensiones criptográficas AES, SHA1 y SHA2. Dentro de la misma librería hemos implementado una función que se encarga de verificar si el sistema en el que estamos implementando la librería dispone de dichas extensiones criptográficas. Sin estas extensiones sería inútil el uso de nuestra librería, ya que todas las funciones están basadas en el estándar de cifrado AES y la familia de funciones criptográficas SHA. [ARM, 2021a]

Podemos acceder a la librería mediante [Gómez, 2021] y está pensada para usar en distribuciones de GNU/Linux.

Los pasos para implementar nuestra librería en un programa son los siguientes:

1. Descargamos la librería ya sea en formato .zip o directamente usando los comandos *git clone "url"*.

2. Escribimos *make* en la terminal para compilar la librería adjunta AArch64cryptlolib

3. Escribimos *make all* en la terminal para compilar nuestra librería.

4. Una vez compiladas ambas librerías tenemos que incluir el fichero *armcommlib.h* mediante *#include* en el programa que queramos usar las funciones para cifrar y descifrar tanto información como ficheros.

5. Compilamos nuestro programa añadiendo en las opciones de compilación el path de la librería armcommlib y AArch64cryptolib. Ejemplo: *gcc -o main main.c /armcommlib/libarmcommlib.so /AArch64cryptolib/libAArch64crypto.a*

6. Por último, para ejecutar el programa podemos tanto añadir nuestra librería a los path predeterminados de Linux, como */usr/lib*, tal y como hemos indicado en la sección 5.2.2 o directamente ejecutar el programa señalando cuál es el path de la librería:

   LD_LIBRARY_PATH="home/pi/armcommlib:$LD_LIBRARY_PATH"./main

8. CAPÍTULO

Gestión y planificación del Trabajo de Fin de Grado

Tanto en el ámbito académico como laboral, la gestión y planificación son necesarias para el correcto desarrollo de un proyecto. Este apartado es de suma importancia, dado que nos ayuda a mantener la constancia a lo largo del progreso del mismo. No solo eso, sino que también nos ayuda a estar al tanto de los posibles riesgos para así poder prevenirlos o buscar soluciones.

Para una buena planificación, lo ideal es dividir el proyecto en distintas fases con tareas dentro de cada una de ellas. Como en la mayoría de proyectos, se han establecido las siguientes fases: análisis de viabilidad, planificación, ejecución, control y seguimiento y evaluación. Dentro de cada fase, se han definido paquetes de trabajo que establecen la duración del proyecto.

8.1. Diagrama EDT

Una forma visible y esquemática de representar las fases y paquetes de trabajo es mediante un EDT (estructura de desglose del trabajo). Un EDT, como indica su nombre, organiza y define el alcance de un proyecto en base a lo establecido en la documentación. Gracias a este esquema jerárquico podemos representar gráficamente las fases y paquetes de trabajo con el fin de cumplir los objetivos del trabajo de fin de grado que nos ocupa.

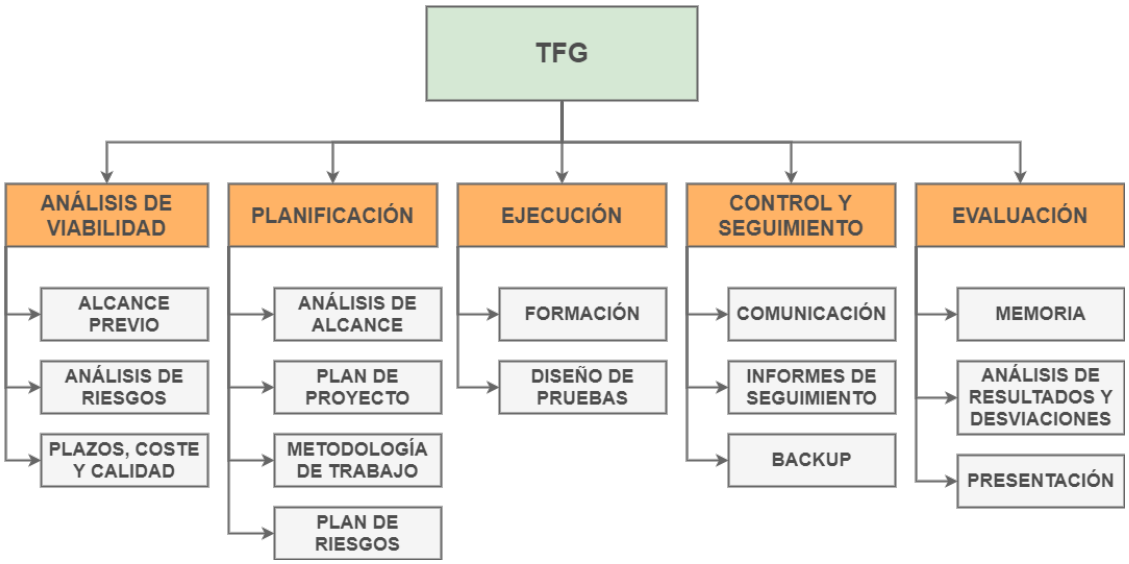

Figura 8.1: Diagrama EDT.

8.2. Descripción de las fases del EDT

En este apartado, se van a detallar los paquetes de trabajo y las tareas que completan estos paquetes. Todas y cada una de las tareas están pensadas para desarrollar de manera óptima el proyecto. Cabe mencionar que la mayoría de tareas han sido fijadas al inicio, a excepción de algunas que han ido surgiendo en el transcurso de éste.

8.2.1. Análisis de viabilidad

Esta fase abarca los inicios del proyecto donde se pretende identificar los objetivos y el alcance del proyecto. Cada paso a dar durante el ciclo de vida del proyecto está pensado para cumplir los objetivos fijados. Así pues, éste va a ser el primer paso para empezar a trabajar en el proyecto con un rumbo predeterminado. A continuación, se mostrarán los paquetes de trabajo y sus tareas identificadas para la fase de análisis de viabilidad.

Alcance previo:

- **Identificar alcance:** La primera tarea de este paquete es identificar lo que hay que hacer en el proyecto.

- **Identificar objetivos:** Después hay que identificar los objetivos a cumplir en el proyecto. Conocer y tener claros los objetivos favorece el desarrollo.

Análisis de riesgos:

En este paquete se busca identificar los posibles riesgos que supone la realización del proyecto. El mayor riesgo suele ser el retraso del proyecto y cualquier desviación en él supondría un retardo y esto podría derivar en una disminución de la calidad del producto. De haber escasez de tiempo, la realización de tareas se vería mermada, lo cual haría que ciertas tareas no necesarias, pero sí complementarias, fuesen suprimidas.

Plazos, coste y calidad:

Aquí se trata de identificar los plazos, el coste y la calidad del proyecto. En un ámbito laboral, se querría determinar si es o no viable llevar a cabo el proyecto mediante una valoración general entre este paquete y el alcance previo. En nuestro caso, los plazos del proyecto son tres, un acuerdo con el tutor para la finalización del proyecto con fecha de 31 de diciembre de 2020, la entrega oficial del TFG el 31 de enero de 2021 y la defensa del mismo alrededor de la segunda semana de febrero de 2021. El único coste, en cuanto a dinero se refiere, ha sido el del módulo NanoPi R2S y una MicroSD para la implementación de la librería. Dicho gasto corre a cuenta de la universidad. Por último, mencionar que la calidad depende de los objetivos establecidos.

8.2.2. Planificación

Esta fase quizá sea la más difícil de completar con exactitud. Se necesita realizar un plan de trabajo en base al alcance previo identificado, es decir, se trata de fijar todas las tareas necesarias para el correcto desarrollo del proyecto. Las tareas están pensadas para dividir de forma equilibrada la carga de trabajo de cada fase, por consiguiente, cuantas más tareas se determinen a lo largo de un proyecto mayor será la sensación de avance y resultará más fácil identificar la duración y riesgos de cada tarea. Todos los proyectos requieren de una planificación exhaustiva, y cuanto mayor sea la magnitud éstos, mayor será el tiempo invertido en la planificación.

Es importante dividir los paquetes de trabajo en tareas para así realizar una buena estimación del coste temporal, y poder tener en cuenta los imprevistos que puedan surgir.

Análisis de alcance:

- **Fijar alcance:** Esta tarea trata de aclarar el alcance y determinar los pasos necesarios para lograr dicho alcance.
- **Fijar objetivos:** Esta tarea trata de fijar o descartar objetivos previamente identificados. Para así hacer un plan más realista basándose en el alcance previo analizado.

Metodología de trabajo:

Para el correcto desarrollo del proyecto se han establecido unos horarios con el fin de crear una rutina de trabajo para así darle una mayor continuidad. Esta metodología de trabajo se establece para ser cumplida desde el inicio del mismo hasta su conclusión.

El proceso de elaboración del trabajo dio comienzo la primera semana de septiembre de 2020, por lo tanto, se estimó una dedicación media de 2-3 horas diarias a lo largo de 4 meses. De este modo, al finalizar el TFG se habrían completado alrededor de unas 300 horas. En mi caso particular, teniendo en cuenta que he estado realizando prácticas laborales todas las mañanas de lunes a viernes, establecí un plan de trabajo comprendido entre las cuatro y las siete horas dela tarde.

Debido al tiempo necesario que me ha ocupado el resto de las asignaturas, ha habido días en los que la metodología se ha visto desfasada en cuanto a horario y tiempo de dedicación. Por consiguiente, he tenido que invertir más horas en fines de semanas y así poder compensar las horas no invertidas.

Plan de riesgos:

Al ser un proyecto de larga duración es necesario realizar un plan de riesgos para prevenir cualquier desviación que pueda surgir. La mayoría de desviaciones suelen ser causadas por factores imprevisibles, y, es por esto, que existe el plan de riesgos para así poder afrontar de mejor manera todos los imprevistos. El plan de riesgos es conveniente para tener margen de maniobra cuando surja un retraso y nuestra planificación se vea desfasada.

La fecha de entrega del proyecto está estimada para el día 31 de enero de 2021, y teniendo en cuenta la fecha mencionada, se ha realizado un plan de riesgos. En concreto, se ha fijado con el tutor una fecha previa para la finalización del proyecto para así poder hacer frente a cualquier desviación y tener tiempo para realizar los ajustes necesarios. El día

señalado ha sido el 31 de diciembre de 2020. Ahora bien, la planificación del proyecto ha estado orientada para que éste estuviese concluido el día previsto con el tutor y no para el día de entrega final. En virtud de ésto, se ha podido hacer frente a los imprevistos con tiempo y antelación, y aun así ha sufrido algún desfase involuntario.

El plan de riesgos ha formado parte de la estimación de tiempo de cada tarea, al no disponer de la experiencia suficiente en el desarrollo de proyectos de larga duración se ha decidido añadir un tiempo extra en cada estimación, de esta manera, si hay tareas en las que surjan imprevistos y necesiten más dedicación, se verán compensadas con el tiempo sobrante de las otras tareas que se resuelvan en menor tiempo del estimado.

Plan de proyecto:

- **Identificación de paquetes de trabajo y tareas:** Este apartado es de suma importancia, dado que aquí es donde van a figurar todas las tareas a realizar para la finalización del proyecto. Cuantas más tareas y más exactas sean mejor será la ejecución. Para ello hemos decidido crear una tabla con las tareas y el coste temporal previsto, para luego ir rellenándola con el tiempo real invertido.
- **Establecer orden de realización de tareas:** Una vez consolidadas las tareas a realizar, toca establecer un orden para el cumplimiento de ellas, en base a requisitos necesarios de alguna tarea. Por ejemplo, no se puede empezar a testear el programa sin haber creado el programa.

8.2.3. Ejecución

La fase de ejecución es la fase que abarca todo el aprendizaje de las nuevas herramientas y tecnologías junto con la realización del estudio necesario para alcanzar los objetivos propuestos. Esta fase generalmente es la que más tiempo abarca del proyecto ya que es donde se realiza el mayor número de tareas y de mayor complejidad. Es por esto que es la fase que más dedicación requiere.

Tabla de desviaciones	Estimación (h)	Real (h)
Análisis de viabilidad	10	7
Alcance previo	5	3
Identificar alcance	2	1
Identificar objetivos	3	2
Análisis de riesgos	3	2
Plazos, coste y calidad	2	2
Planificación	30	25
Análisis de alcance	5	2
Fijar alcance	2	1
Fijar objetivos	3	1
Metodología de trabajo	2	1
Plan de proyecto	20	22
Identificación de paquetes de trabajo y tareas	15	20
Establecer orden de realización de tareas	5	2
Plan de riesgos	3	13
Ejecución	140	160
Formación	40	30
Estudio del estado del arte	30	20
Estudio de las herramientas	10	10
Diseño y pruebas	100	130
Desarrollo de la librería	70	110
Entorno de pruebas	30	20
Control y seguimiento	35	53
Comunicación	1	1
Informes de seguimiento	3	1
Gestionar las incidencias y desviaciones	30	50
Backup	1	1
Evaluación	100	95
Memoria	75	78
Creación de la documentación	65	70
Revisión de la documentación	10	8
Analisis de resultados y desviaciones	5	2
Presentación	20	15
Preparación de la defensa	15	10
Diseño del poster	5	5
TFG	315	340

Tabla 8.1: Tabla de desviaciones.

Formación

En este apartado se trata de aprender todo lo necesario para la ejecución del proyecto, como bien indica el nombre, se trata de formarse para así poder realizar el trabajo.

- **Estudio del estado del arte:** En esta tarea se realiza un estudio intensivo de la teoría necesaria para la realización del proyecto. En nuestro caso es sobre el estándar de cifrado AES, la familia de funciones hash criptográficas SHA y las librerías informáticas.
- **Estudio de las herramientas:** Aquí se pretende realizar un estudio sobre las herramientas en las que se va a trabajar. En este caso, un estudio avanzado sobre el lenguaje de programación C y el uso y aplicación AArch64cryptolib para la implementación de funciones de cifrado y descifrado.

Diseño y pruebas:

En el presente apartado se va crear un diseño del producto y un entorno de pruebas para verificar su funcionamiento.

- **Desarrollo de la librería:** Aquí se va a diseñar y desarrollar una librería dinámica con funciones capaces de cifrar y descifrar cualquier longitud de mensajes y ficheros.
- **Entorno de pruebas:** Crear un entorno de pruebas, el cual, va a ser un sistema de mensajería cliente-servidor basado en sockets en el que se pretende enviar mensajes cifrados y descifrarlos antes de ser visibles para el receptor.

8.2.4. Control y seguimiento

El control y seguimiento del proyecto está pensado para mantener una continuidad y estar al tanto de los avances o imprevistos del proyecto. Generalmente, esta fase está asociada a la ejecución ya que es la más propensa a tener desviaciones y complicaciones.

Comunicacion:

La comunicación es indispensable en el seguimiento y control de un proyecto, por eso hemos establecido unas reuniones periódicas. Durante los 3 primeros meses se han realizado reuniones de aproximadamente 30 minutos de duración cada dos semanas. En ellas, se comunicaba el punto del proyecto en el que nos encontrábamos junto con los pasos y tareas venideras. De esta manera, se consigue un seguimiento continuo y no se deja de

lado el proyecto en ningún periodo. Llegando al periodo de entrega del TFG, la frecuencia de reuniones aumentó a dos por semana con una duración de alrededor de una hora. Para la comunicación, debido al periodo de COVID-19 que estamos viviendo, con el fin de aminorar riesgos, se ha utilizado la plataforma online whereby.

Informes de seguimiento

Los informes de seguimiento sirven para dejar constancia del desarrollo y desviaciones surgidas para, en un futuro, poder recurrir a estos informes con el fin de evitar desviaciones o tomar decisiones fructíferas en cuanto al desarrollo. Documentar y anotar los días trabajados y los objetivos alcanzados durante el desarrollo del proyecto es de suma importancia, pues podemos ver en qué punto del desarrollo que nos encontramos y así tomar las decisiones adecuadas en todo momento.

Gestionar las incidencias y desviaciones:

Es importante hacer un seguimiento y contabilizar las desviaciones entre el tiempo estimado y el tiempo real dedicado a cada tarea. No solo eso, sino que anotar las incidencias y el cómo han ocurrido nos ayuda a realizar una mejor planificación en proyectos futuros. Por último, hay que buscar soluciones a las variantes o incidencias que han surgido, lo cual puede ser una tarea costosa.

Backup

Los backups sirven para garantizar que el proyecto siga vigente, en el caso de perder todo lo avanzado es importante haber realizado backups para así poder recuperar todo lo trabajado hasta el punto de la pérdida. En mi caso, he realizado backups semanales y en el momento de la resolución de alguna de las incidencias surgidas.

8.2.5. Evaluación

La evaluación es la fase final del proyecto, en ésta se archiva todo lo realizado y se dispone a hacer una valoración general de los objetivos y metas alcanzadas. Una vez terminada la fase de evaluación se puede dar por concluido el proyecto.

Memoria:

A lo largo del ciclo de vida del TFG el trabajo realizado se anota en este paquete denominado memoria. Para realizar la documentación se han usado dos herramientas: *draw.io* y *Overleaf*. Mediante draw.io se han creado los diagramas que aparecen a lo largo de la documentación, y, mediante Overleaf se ha ido recopilando la información y formado la memoria. Este apartado se realiza en la etapa final del proyecto, cuando ya se han realizado las distintas fases fijadas en la planificación.

La memoria, está compuesta por distintos ficheros, los cuales se agrupan mediante Overleaf para dar forma al proyecto.

- **Creación de la documentación:** En esta tarea se crea el documento el cual contiene todo lo realizado en el proyecto.
- **Revisión de la documentación:** Esta tarea trata de revisar, en busca de errores, el documento escrito previamente con el fin de mejorar lo escrito.

Análisis de resultados y desviaciones:

En este apartado se analizan los resultados obtenidos en el entorno de pruebas y las desviaciones surgidas. En cuanto a la planificación solo ha habido dos desviaciones significativas. La primera pertenece al paquete de análisis y estudio de los algoritmos de la librería criptográfica de ARM. Al realizar el estudio, se dio a entender que la gestión del cifrado la hacía automáticamente el algoritmo en caso de que se le pasase un mayor tamaño del soportado.

Por ende, en el paquete de implementación y creación de la librería hubo un aumento notable de horas dedicadas. A la hora de implementar nuestras funciones en la librería creada, se tuvo en cuenta que el tamaño no importaba a la hora del cifrado y descifrado, pero, percibimos que se trataba de un error y que no funcionaba correctamente lo que en su momento dedujimos. En consecuencia, tuvimos que reestructurar la librería de forma que se hiciese el número de llamadas necesarias a las funciones de la librería de ARM para poder cifrar/descifrar cualquier mensaje independientemente de su tamaño.

La segunda desviación notable, ha sido por el incumplimiento de la metodología de trabajo al no ser respetados los horarios, particularmente, al no haber restituido las horas de trabajo en los días en los que no se invirtió el tiempo estipulado al proyecto. Esto ha

provocado que, de haber pensado en un primer momento tener el TFG terminado para el 31 de Diciembre de 2020 se haya tenido que retrasar.

En los demás apartados las desviaciones han sido mínimas; alguna tarea ha requerido algo más de tiempo y otras menos, por lo que se han visto recompensadas. Cabe mencionar, que se han hecho retoques hasta la semana previa a la entrega oficial del trabajo de fin de grado, es decir hasta el 31 de Enero de 2021.

Presentación:

Para dar fin al proyecto ha de realizarse una presentación para la defensa del TFG, y para ello hemos identificado dos tareas.

- **Preparación de la defensa:** Esta implica realizar una presentación sobre el proyecto ya terminado y preparar la exposición del mismo.
- **Diseño del póster:** El póster contiene un resumen de todo lo que implica el proyecto.

9. CAPÍTULO

Conclusiones y trabajo futuro

En la presente memoria se ha pretendido reflejar todo el trabajo realizado durante el proceso de desarrollo del TFG. Se ha presentado un estudio del arte detallado sobre el estándar de cifrado AES, la familia de funciones criptográficas SHA y las librerías. En los distintos capítulos que componen el estado del arte se ha estudiado exhaustivamente el funcionamiento para así poder implementar nuestra librería con toda la información recopilada en dicho apartado.

Además del estado del arte, se ha investigado sobre el hardware usado para el desarrollo, en concreto, sobre el procesador de ARM Cortex-A53 y las extensiones criptográficas que implementan la arquitectura ARMv8. Sin dichas extensiones, el desarrollo del presente proyecto habría sido imposible, por lo que su presencia es imprescindible en caso de que se requiera usar la librería que se ha creado.

Aunque ha habido desajustes temporales a lo largo del ciclo de vida del proyecto, el objetivo fijado al inicio ha sido completado con éxito: desarrollar una librería basada en las funciones ofrecidas por AArch64cryptolib para poder cifrar y descifrar tanto mensajes como ficheros. La librería que hemos creado está pensada para un uso público, es decir, que cualquiera pueda utilizarla en sus proyectos. Cabe mencionar que, aun habiendo cumplido el objetivo principal del proyecto, han quedado descartadas por falta de tiempo algunas tareas y mejoras que fueron surgiendo a lo largo del desarrollo.

La tarea más costosa y compleja ha sido el estudio de las funciones implementadas en la librería criptográfica AArch64cryptolib. Esto se debe, en gran parte, a la falta de comentarios dentro del código de las funciones. Aun habiendo realizado un completo estudio

sobre el uso y funcionamiento de las funciones de cifrado AES y sus modos de operación, la interpretación del código de la librería proporcionada por ARM ha resultado un verdadero reto y ha supuesto un mayor tiempo de dedicación para el entendimiento de éste en proporción a las demás tareas. No es necesario mencionar, que para poder adaptar el código y así poder desarrollar nuestra librería, ha sido necesario entender al completo el funcionamiento de los algoritmos implementados en la librería AArch64cryptolib.

En vista a proyectos futuros, ha quedado pendiente la utilización de las funciones que permiten el uso del protocolo IPSec. Además, sería muy útil e interesante implementar dentro de la librería un sistema de transferencia de claves utilizando criptografía asimétrica. De esta forma, se evitaría que la clave usada para el cifrado y descifrado de datos tuviera que ser conocida por ambas partes.

Como conclusión al actual capítulo y por consiguiente a la memoria, cabe destacar que con el desarrollo de esta librería se pretende que sirva de ayuda para todo aquel que desee implementar un servicio o programa y quiera mantener la seguridad de la información haciendo uso de ella. Para finalizar, he de mencionar que, el desarrollo del trabajo de fin de grado ha resultado complejo, pero a la vez, enormemente gratificante.

Bibliografía

[ARM,] ARM. A-profile architectures. https://developer.arm.com/architectures/cpu-architecture/a-profile.

[ARM, 2020] ARM (2020). https://github.com/ARM-software/AArch64cryptolib.

[ARM, 2021a] ARM (2021a). Arm cortex-a53 mpcore processor cryptography extension technical reference manual. https://developer.arm.com/documentation/ddi0501/f/introduction/about-the-cortex-a53-processor-cryptography-extension.

[ARM, 2021b] ARM (2021b). Cortex-a53. https://developer.arm.com/architectures/cpu-architecture/a-profile.

[Chaves et al., 2016] Chaves, R., Sousa, L., Sklavos, N., Fournaris, A. P., Kalogeridou, G., Kitsos, P., and Sheikh, F. (2016). *Secure Hashing: SHA-1, SHA-2, and SHA-3*. CRC Press.

[Gómez, 2021] Gómez, G. (2021). armcommlib. https://github.com/ggetxaniz/armcommlib.

[McGrew, 2008] McGrew, D. (2008). An interface and algorithms for authenticated encryption. https://tools.ietf.org/html/rfc5116#section-5.1.

[McGrew, 2014] McGrew, David; Foley, J. P. K. (2014). Authenticated Encryption with AES-CBC and HMAC-SHA. https://tools.ietf.org/id/draft-mcgrew-aead-aes-cbc-hmac-sha2-03.html.

[Nechvatal et al., 2000] Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., and Roback, E. (2000). Report on the development of the advanced encryption standard (aes).