Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Diseño e implementación de un esquema de cifrado híbrido basado en DHIES

Tesis que presenta

Jesús Barrón Vidales

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Computación

Director de la Tesis

Dr. Debrup Chakraborty

México, D. F. **Noviembre 2008**

Resumen

Un esquema de cifrado híbrido es un mecanismo para cifrar que puede ser construido a partir de dos esquemas distintos: un mecanismo de encapsulado de llave (KEM) basado en un esquema de cifrado de llave pública, y un mecanismo de encapsulado de datos (DEM) basado en un esquema de cifrado de llave privada. La mejor aportación del cifrado híbrido radica en que resuelve en gran medida el problema de distribución de llaves para esquemas simétricos y el problema de eficiencia para esquemas asimétricos. En años recientes el área de la criptografía híbrida ha tenido grandes avances. A partir de 1997 con el surgimiento de DLAES, han aparecido distintos esquemas de cifrado que usan esta idea para construir esquemas seguros y eficientes siendo los de Cramer-Shoup, de Kurosawa-Desmedt y DHIES (de Abdalla, Bellare y Rogaway) los más significativos. En este proyecto se buscaba implementar un esquema de cifrado híbrido eficiente y seguro para probar su desempeño y añadirlo a una aplicación de uso común. Se realizó un análisis comparativo de los esquemas híbridos reportados a la fecha y de sus modificaciones. En base a ese análisis, se tomó el esquema DHIES como punto de partida para realizarle algunas modificaciones buscando incrementar su eficiencia y conservar el mismo nivel de seguridad. Específicamente se agregó un cifrador simétrico con autenticación para reemplazar los dos algoritmos originales de DHIES. Se implementaron todas las operaciones y funciones necesarias para la construcción del esquema, desde las más básicas como las operaciones aritméticas y el generador de números primos hasta las más complejas como el cifrador AES y la función SHA. Las implementaciones del cifrador final arrojaron datos favorables, ya que los tiempos de ejecución logrados son competitivos con otras implementaciones reportadas del mismo cifrador simétrico. También se añadió el uso del cifrador al sistema de correo electrónico Pine con buenos resultados en cuanto a su funcionalidad y los beneficios ofrecidos. Así pues, gracias al análisis realizado y a los resultados obtenidos de las implementaciones, podemos afirmar que los esquemas de cifrado híbrido representan una alternativa viable en aplicaciones comunes, y con un buen desarrollo podrían incluirse en ambientes más demandantes.

Abstract

A hybrid encryption scheme is a mechanism for encryption that can be constructed from two different schemes: a key encapsulation mechanism (KEM) based on a public key encryption scheme and a data encapsulation mechanism (DEM) based on a symmetric key encryption scheme. The advantage of hybrid encryption is that it largely solves the problem of key distribution for symmetric encryption schemes and the problem of efficiency for asymmetric encryption schemes. In recent years the field of hybrid cryptography has progressed in many ways. Beginning in 1997 with the publication of DLAES, different encryption schemes that uses this idea to build safe and efficient schemes have been proposed, the Cramer-Shoup's, the Kurosawa-Desmedt's and DHIES (from Abdalla, Bellare and Rogaway) being the most significant ones. In this project, we implemented an efficient and secure hybrid encryption scheme, tested its performance and added it to an application of common usage. A comparative analysis was made of the hybrid schemes reported up to date and based on this analysis, the scheme DHIES was taken as a starting point. We made some changes to the basic DHIES scheme to enhance its efficiency but maintaining its security properties. Specifically, we added a symmetric cipher with implicit authentication (authenticated encryption scheme) to replace the two original algorithms of encryption and authentication in DHIES. All the necessary operations and functions for the construction of the scheme were implemented, from the most basic arithmetic operations and the prime numbers generator to the more complex functions as the AES cipher and the hash function SHA. Implementations of final encryption scheme showed favorable results, since the execution times achieved are competitive with other implementations reported using the same symmetric cipher. Also, the scheme was added to the e-mail system Pine with good results in terms of its functionality and the offered benefits. So, based on the analysis and the obtained results of the implementations, we can state that the hybrid encryption schemes represent a viable alternative to use in common applications.

Agradecimientos

Gracias a mi amada esposa Yazmin por haberme apoyado tanto siempre, gracias por tu comprensión en las largas esperas y por estar siempre ahí cuando te he necesitado. Gracias por compartir tu vida conmigo.

Gracias a mi hija Naomi por darme un nuevo motivo para vivir y para seguir esforzándome en los momentos de desánimo. Y gracias por acompañarme también en las noches de desvelo.

Gracias a mi *ama* y a mi *apa* por todo el amor, por la educación y por la ayuda brindados durante mis 26 años de vida. Gracias por ser mi ejemplo de vida y mi modelo a seguir.

Un agradecimiento especial a todos mis hermanos por los ánimos y por enseñarme distintas maneras de ver y vivir la vida.

A toda la familia por sus buenos deseos y su apoyo incondicional.

Agradezco a mis compañeros de generación y amigos, quienes a lo largo de estos años me acompañaron en las largas jornadas de desesperación estudiantil, por los momentos felices que pase junto a ustedes y por haber compartido conmigo esta gratificante experiencia.

A mi director de tesis, el Dr. Debrup Chakraborty, por haberme guiado y ayudado en este proyecto, por haberme confiado parte importante de sus conocimentos y principalmente por brindarme su amistad.

A los doctores del Departamento de Computación por los conocimientos impartidos durante sus clases y fuera de ellas.

Especialmente agradezco a los doctores Debrup Chakraborty, Francisco Rodríguez y Guillermo Morales por sus comentarios, por sus consejos, por haber despertado en mi el gusto por este bello arte (la criptografía) y por la cultura en general.

Quiero agradecer a ese Ser que nos ha permitido a cada uno de nosotros tener la astucia, la inteligencia, la determinación y el ímpetu para que nada ni nadie nos detenga en el camino que hemos elegimos.

Y finalmente agradezco al CINVESTAV y al CONACyT por darme la oportunidad y el apoyo para realizar estos estudios de alta calidad y por seguir apoyando la investigación en México.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

Una máquina puede hacer el trabajo de 50 hombres corrientes.
Pero no existe ninguna máquina que pueda hacer el trabajo
de un hombre extraordinario.
Elbert Hubbard

Mantener segura la información que deseamos comunicar a otra u otras personas ha sido desde siempre una tarea en la que el hombre ha invertido gran parte de su tiempo y de su ingenio. Desde hace aproximadamente 4000 años hasta la época actual, los gobiernos y sus ejércitos han sido quienes más han concentrado sus esfuerzos en mantener secreta la información que intercambian entre sí pero, no son los únicos; los niños buscan enviarse mensajes tratando de que los adultos no los entiendan, las "pandillas" usan códigos de símbolos e imágenes que únicamente sus miembros comprenden, las parejas crean señales para tratar de comunicarse sin que los demás sepan lo que dicen, etc.

En este primer capítulo se presenta una introducción a los conceptos concernientes a la seguridad de la información y a la criptografía. Se presenta información básica de las distintas filosofías de cifrado de datos y algunas definiciones básicas necesarias para entender el concepto de seguridad sobre el que se basa este trabajo. Para conocer a fondo cada uno de los temas aquí expuestos, puede dirigirse a [26], [27] y [20], de donde fue basado gran parte del material presentado en este capítulo. Adicionalmente se presenta una breve descripción del contenido de cada uno de los capítulos de este manuscrito.

1.1. Seguridad y Criptografía

Desde la Antigüedad se ha buscado la manera de comunicarse de forma segura. El uso de códigos, de tintas especiales, de sobres sellados, además de confiar en un servicio postal certificado fueron algunos de los medios tradicionales para ocultar información y en realidad estas técnicas no evolucionaron mucho con el paso de los siglos hasta el surgimiento

y auge de la información digital en las últimas décadas.

Pero lograr transmitir información de forma segura a través de cualquier medio no sólo se refiere a evitar que personas no autorizadas tengan conocimiento del contenido de dicha información, sino que la seguridad debe cumplir con una serie de requisitos o servicios que se deben cumplir durante el envío y la recepción de la información. Una definición de algunos de estos servicios de seguridad se da en la tabla 1.1.

Estos servicios definen el nivel de seguridad que tendrá cierta información o comunicación. Dependiendo de la aplicación que se requiera para esa información y del medio en el que se entable dicha comunicación serán los servicios con los que deberemos cumplir. Así pues, podemos definir que la *seguridad de la información* es el conjunto de mecanismos o técnicas aplicadas a la transmisión de información que pretenden cubrir algunos o todos los servicios de seguridad.

Servicio	**Objetivo**
Confidencialidad	Mantener secreta la información para todos excepto para quienes tengan autorización de acceso.
Integridad	Asegurar que los datos nos han sido alterados en ninguna forma.
Autenticación de mensaje	Comprobar la fuente del mensaje.
Identificación	Comprobar la identidad de alguna entidad participante.
Firma digital	Poder relacionar un mensaje con alguna entidad.
Certificación	Aprobación de cierta información por parte de una entidad confiable.
Anonimato	Ocultar la identidad de una entidad involucrada en algún proceso.
Revocación	Retractarse de alguna certificación o autorización.
No repudio	Prevenir la negación de acuerdos o acciones previas.

Tabla 1.1: Algunos servicios de seguridad de la información.

Por otro lado la **criptografía** es un área dentro de la seguridad que se encarga de proveer especialmente algunos de estos servicios. Desde sus inicios, la criptografía fue considerada como un arte, incluso en el diccionario de la Real Academia de la Lengua Española la criptografía se define como el arte de escribir con clave secreta o de un modo enigmático. Sin embargo, con el paso de los siglos y los grandes avances en las técnicas de cifrado, la criptografía se apega más al concepto de ciencia que al de arte, aunque aún conserva su belleza.

Desde el punto de vista etimológico, criptografía proviene del griego *krypto*, "ocultar", y de *graphos*, "escribir". Dentro del área de la seguridad, una de las definiciones aceptadas

es la que se presenta en [26], que define a la criptografía como "el estudio de técnicas matemáticas relacionadas a aspectos de seguridad de la información como son la confidencialidad, la integridad de datos, la autenticación de entidades, y la autenticación del origen de los datos". Como se especifica en esta definición, la criptografía tiene como objetivo brindar cuatro de los servicios de seguridad definidos en la tabla 1.1: confidencialidad o privacidad, integridad de datos, autenticación y no repudio. Cabe señalar que la autenticación se refiere tanto a la autenticación de entidades (identificación) como a la autenticación del origen del mensaje, la cual provee implícitamente la integridad de datos [26].

Así como existe la criptografía, existe también su contraparte. Tenemos un área que se encarga de estudiar las técnicas matemáticas que permitan "romper" las técnicas propuestas por la criptografía y en general de cualquier servicio de seguridad, a la cual se denomina como **criptoanálisis**. A estas dos áreas de la seguridad se les suele englobar con el término de **criptología** que se entiende como el estudio de los criptosistemas. Entiéndase por criptosistema un conjunto de operaciones y técnicas criptográficas básicas que son usadas para brindar servicios de seguridad a la información. Actualmente al término de criptología también se le asocian más áreas de estudio como son la *esteganografía* y la *firma digital*.

Los primeros vestigios de criptografía se remontan a los antiguos egipcios, hace unos 4 mil años, quienes "cifraban" sus jeroglíficos en algunas de sus escrituras en monumentos. También algunas escrituras hebreas mostraban algo parecido a un cifrado. Hace unos 2 mil años, los romanos, específicamente Julio César, utilizaba un cifrador por sustitución bastante sencillo que hasta la actualidad recibe el nombre de "cifrador César". Por su parte, los espartanos utilizaban uno de los primeros aparatos de cifrado por transposición, la escítala, con fines militares principalmente. Todos los cifradores utilizados durante la antigüedad eran de tipo monoalfabético y eran considerados suficientemente seguros hasta que, en el siglo IX, el matemático árabe Abu Yusuf Yaqub ibn Ishaq al-Sabbah Al-Kindi publicó un método, conocido desde entonces como *análisis de frecuencias*, útil para poder "romper" cualquier cifrador de este tipo [47].

Hablando ya de tiempos un poco menos antiguos, en el siglo XV, Leon Battista Alberti fue el primero en proponer el concepto de cifradores polialfabéticos y además creó un disco de cifrado que lleva su nombre, *disco de Alberti*. También podemos destacar el cifrador polialfabético de Blaise de Vigenère que data del siglo XVI, que marco un gran avance para el cifrado de la época debido a lo complicado de su estructura. Tal era la fortaleza del cifrador, que no fue sino hasta el siglo XIX cuando lograron romperlo, e incluso llegó a ser nombrado como "la cifra indescifrable".

Dentro de los hechos que han quedado grabados en la historia de la criptografía están también el hallazgo de la piedra de la Rosetta a finales del siglo XVII, el surgimiento de máquinas cifradoras del siglo XX, especialmente Enigma que fue usada durante la segunda

guerra mundial, y posteriormente los inicios de la "criptografía moderna" en las últimas décadas del siglo pasado.

1.2. Criptografía moderna

Como se mencionó en la sección anterior, antiguamente la criptografía se apegaba más al concepto de arte ya que los métodos de cifrado eran diseñados "a la medida" dependiendo de lo que la persona creía seguro y del uso que necesitaba dar a ese método, en cambio la criptografía moderna tiene bases más científicas, paradigmas que la diferencían de la criptografía clásica. Dentro de estos paradigmas podemos resaltar la importancia de 3 principios básicos pero fundamentales para la criptografía de nuestra era:

1. Formulación de definiciones exactas: es un requisito esencial el definir formalmente la seguridad para una tarea criptográfica, ya sea una simple primitiva o un protocolo completo, ya que esto nos ayudará a diseñar y estudiar de la mejor forma posible dicha tarea.

2. Dependencia en suposiciones concisas: la seguridad reside en alguna o algunas suposiciones las cuales deben estar bien definidas para que sea más sencillo realizar las pruebas de seguridad.

3. Pruebas detalladas de seguridad: son necesarias debido a que problemas en la seguridad podrían provocar graves daños si se usara la función sin asegurarnos de que es confiable.

Conforme el tiempo fue transcurriendo, la criptografía se dividió principalmente en dos filosofías distintas, diferenciadas esencialmente por las técnicas de cifrado que utilizan, aunque en general mantienen la misma estructura. A continuación se definen los conceptos básicos estructurales de ambas filosofías. Estos conceptos se utilizaron a lo largo de este documento y además son necesarios para comprender el tema central de este trabajo.

El objetivo principal de la criptografía es transmitir cierta información de forma segura, es decir, establecer una comunicación segura. Esta comunicación debe mantener un medio por el cual viaje la información, al cual llamaremos **canal**. Un canal puede ser físicamente seguro o físicamente inseguro, es decir, accesible o no para cualquiera, y por este motivo es que necesitamos a la criptografía, ésta se encargará de dar seguridad a la información que viaja por ese canal físicamente inseguro. En lo siguiente se puede entender como **participantes** o **entidades** dentro de esta comunicación a algo o alguien que envía, recibe o manipula la información. Las entidades pueden actuar como *remitente*, quien envía información, o como *destinatario*, quien recibe dicha información. Y debido a que cualquiera tiene acceso al canal de comunicación, definimos a una tercera entidad llamada *adversario*, que no es ni el remitente ni el destinatario y que trata de violar los servicios de seguridad de la información, obtener la información o incluso tratar de usurpar el lugar de cualquiera de las otras entidades. Al adversario sólo es útil mencionarlo

para las definiciones y las pruebas de seguridad.

Como ya se mencionó, la criptografía se centra en el estudio de técnicas que permitan brindar ciertos servicios de seguridad, principalmente confidencialidad. Esto lo hace mediante el diseño de **criptosistemas**. Un criptosistema puede verse formalmente como una quíntupla $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ compuesta de 3 espacios o conjuntos finitos y de dos conjuntos de funciones:

- $\mathcal{M}$ es el conjunto de posibles mensajes o textos en claro,
- $\mathcal{C}$ es el conjunto de posibles cifras o textos cifrados,
- $\mathcal{K}$ es el conjunto de posibles llaves,
- $\mathcal{E}$ es una familia de funciones de cifrado y
- $\mathcal{D}$ es una familia de funciones de descifrado

El *texto en claro* se refiere al mensaje (de cualquier tipo) que el remitente desea enviar y el *texto cifrado* es el mensaje resultante del proceso de cifrado que es recibido por el destinatario. La única restricción es que para cualquier texto en claro $m \in \mathcal{M}$ y cualquier función de cifrado $Enc \in \mathcal{E}$, donde $Enc(ek, m) = c$ con $c \in \mathcal{C}$, debe existir una función de descifrado $Dec \in \mathcal{D}$ tal que $Dec(dk, Enc(ek, m)) = m$ y $Dec(dk, \cdot) = Enc^{-1}(ek, \cdot)$, donde $ek, dk \in \mathcal{K}$ son llaves que indexan o definen las funciones de cifrado y de descifrado. Si adicionalmente al par de funciones $(Enc(ek, \cdot), Dec(dk, \cdot))$ para valores $ek, dk \in \mathcal{K}$ tomamos en cuenta el método para obtener ambas llaves, obtendremos un **esquema de cifrado**. Todo esquema de cifrado se compone principalmente de estos tres algoritmos: un generador de llaves KG, un cifrador $Enc(ek, \cdot)$ y un descifrador $Dec(dk, \cdot)$. Un punto importante es tomar en cuenta que todo excepto la llave o llaves del sistema es información públicamente conocida, a esto se le conoce como el "principio de Kerckhoffs" [20].

Como se dijo al principio de esta sección, existen dos filosofía de cifrado en las que se han centrado las investigaciones, estas dos ramas de la criptografía son conocidas como **criptografía simétrica** y **criptografía asimétrica**, o más comúnmente como de llave secreta y de llave pública, respectivamente. Como el nombre nos indica, la principal diferencia entre éstas radica en las llaves, en la criptografía simétrica la llave de cifrado ek es idéntica a la llave de descifrado dk mientras que en la criptografía asimétrica ambas llaves son distintas sólo que ek es públicamente conocida pero dk es reservada únicamente para conocimiento y acceso del propietario.

1.2.1. Cifrado simétrico

La criptografía simétrica es aquella que utiliza criptosistemas en los cuales la llave de cifrado ek y la llave de descifrado dk son idénticas, por lo cual cuando hablemos de cifrado simétrico únicamente nos referiremos a la llave K. Este método de cifrado es comúnmente

llamado criptografía de *llave simétrica* o *llave secreta*. En esta filosofía, cuando dos entidades se desean comunicar deben establecer los parámetros del criptosistema a utilizar, es decir, establecer los algoritmos de cifrado E_K y descifrado D_K que utilizaron y establecer cuál será la llave K que van a compartir, este último punto es el problema principal del paradigma, cómo establecer el valor de la llave compartida de forma segura.

Este tipo de cifrado ha sido el más conocido y utilizado. Desde los cifradores antiguos como el del Cesar hasta los cifradores clásicos como el de Vigenère e incluso el realizado por la máquina Enigma entran dentro de esta categoría de cifrado, ya que todos utilizaban alguna especie de clave compartida necesaria tanto para cifrar como para descifrar. La llave podía ser desde una tabla de sustitución, una palabra clave o hasta el número de rotor a utilizar en cada parte del proceso. Con el auge de las computadoras la criptografía tuvo un pequeño cambio de dirección y todo se comenzó a representar como números o cadenas de bits, así pues, para los algoritmos de cifrado moderno, la llave K es un número, o su respectiva representación binaria, que se encuentra dentro de un conjunto delimitado principalmente por una longitud específica de bits.

Dentro del cifrado simétrico existen dos categorías de cifradores, las cuales se dividen de acuerdo a la forma en la que leen el texto en claro para convertirlo en el texto cifrado. La primera de estas categorías es el **cifrado por flujo** donde el mensaje de entrada es leído y cifrado un símbolo (bit o byte) a la vez de forma independiente lo que evita la propagación de errores. Es necesario mencionar que la salida del cifrado no sólo depende de la llave y del texto en claro sino también de un estado que define qué transformación se aplicará, por lo cual también se les conoce como *cifradores de estado*. Los cifradores por flujo son los más eficientes ya que las operaciones que utilizan sus transformaciones son muy simples, además no requieren de grandes cantidades de memoria o espacio de almacenamiento lo que las hace muy útiles en ambientes de recursos limitados como lo son la telefonía móvil. Los cifradores por flujo mas utilizados han sido RC4 y A5.

La segunda categoría es el **cifrado por bloques** donde el cifrador lee consecutivamente bloques de símbolos (bits) de una longitud fija n para cifrarlos. Este tipo de cifradores, a diferencia de los anteriores, requieren más recursos computacionales pues necesitan almacenar ciertos datos en memoria durante el procesamiento además de que las operaciones de transformación suelen ser menos sencillas. Un esquema de cifrado por bloques puede verse como una función que mapea elementos $m \in \mathcal{M}$ de longitud n a elementos $c \in \mathcal{C}$ también de longitud n. Los ejemplos de cifrado por bloques más notables son **D**ata **E**ncryption **S**tandard (DES, 1977) y su sucesor **A**dvanced **E**ncryption **S**tandard (AES, 2001).

A diferencia del cifrado por flujo, aquí la transformación de mensaje a cifra depende únicamente del valor de la llave y del mensaje, así pues, si ciframos dos bloques idénticos del texto cifrado el resultado serán dos bloques idénticos del texto cifrado, esto equivale a una perdida en el nivel de seguridad del esquema por lo que entre otros motivos, para evitar este problema, los cifradores por bloques requieren ser aplicados mediante un pro-

ceso denominado *modo de operación* que no es más que un conjunto de reglas que nos dicen la forma de cifrar cada uno de los bloques del mensaje.

Los modos de operación pueden incluir el uso de algún valor especial o de algunas técnicas para mejorar su procesamiento. Pueden usar un vector de inicialización para agregar aleatoriedad al proceso, un nonce cuya función es la misma que el vector de inicialización pero que debe ser único y no debe repetirse su uso en futuras acciones, retroalimentación con valores anteriores del cifrado, entre otras opciones. Los modos de operación más antiguos y más conocidos son *Electronic Codebook* (ECB), *Cipher Block Chaining* (CBC) y *Output Feedback* (OFB) aunque estos han dejado de ser seguros [46], por tal motivo fue necesario crear nuevos modos de operación como *Counter Mode* (CTR) que utiliza un nonce y un contador, *Galois/Counter Mode* (GCM) [25] que añade autenticación, y *Xor Encrypt Xor* (XEX) [38] que está especialmente diseñado para el cifrado de discos.

1.2.2. Cifrado asimétrico

La criptografía asimétrica es aquella que utiliza criptosistemas en los cuales tenemos un par de llaves distintas ek y dk tal que para cualquier valor $m \in \mathcal{M}$ se cumple que $Dec(dk, \cdot) = Enc^{-1}(ek, \cdot)$. A nuestro par de llaves se les denomina como llave pública y llave privada, respectivamente, y como su nombre lo indica, la llave pública ek debe ser accesible para cualquier entidad que quiera establecer una comunicación segura con el dueño de dicha llave mientras que el acceso a la llave privada se restringe únicamente a su propietario.

El concepto de cifrado de llave pública surgió por primera vez en 1976 en un trabajo publicado por Whitfield Diffie y Martin Hellman el cual llevaba el nombre de "New directions in Cryptography" [9]. En este artículo los autores dan una descripción detallada de este nuevo paradigma de cifrado y aunque aún no habían realizado nada de forma práctica, comenzó una nueva era de investigación para los criptógrafos. Fue hasta 1978 cuando se creó el primer esquema práctico de cifrado de llave pública, el cual lleva el nombre de *RSA* [36] haciendo referencia a sus autores Ronald Rivest, Adi Shamir y Leonard Adleman. Junto a RSA los esquemas de cifrado de llave pública más conocidos son el criptosistema de Rabin presentado en 1979 [33] y el de ElGamal creado en 1985 [10].

La principal ventaja que tenemos aquí es la facilidad de comunicación entre las entidades. Cada quien genera su par de llaves o las obtiene de una tercera entidad "de confianza" y distribuye su llave pública sin afectar la seguridad del sistema. El principal problema que existe con este paradigma es el hecho de que las operaciones de cifrado son muy costosas computacionalmente ya que se basan principalmente en la exponenciación. Así que el interés principal de estudiar este tipo de cifradores ha sido para cifrar valores pequeños como son números de identificación o códigos de tarjetas de crédito, para usarlo en autenticación, en firmas digitales y en nuestro caso como medio de intercambio de llaves.

1.2.3. Cifrado híbrido

En el área de estudio de la criptografía, el cifrado híbrido ha tenido gran atención en los últimos años. Desde la aparición del esquema de cifrado diseñado por T. ElGamal, donde se propone una implementación eficiente para un esquema de cifrado de llave pública, muchos trabajos han sido realizados para buscar formas de mejorar las características del cifrado tanto para criptografía de llave pública como privada. Un esquema de cifrado híbrido es un mecanismo de cifrado que puede ser construido a partir de dos esquemas distintos: un esquema de cifrado de llave o KEM (*Key Encapsulation Mechanism*) basado en un esquema de llave pública, y un esquema de cifrado de datos o DEM (*Data Encapsulation Mechanism*) basado en un esquema de llave privada.

Anque los esquemas de cifrado híbrido aún son pocos, y la mayoría de las investigaciones se centran en tratar de probar la seguridad de los ya existentes o de modicarlos de alguna manera. Los trabajos más notables son los realizados por K. Kurosawa e Y. Desmedt [22], V. Shoup [44], y principalmente para esta investigación el trabajo desarrollado por M. Abdalla, M. Bellare y P. Rogaway [1].

1.3. Descripción de la tesis

El punto central del problema a ser abordado en esta tesis radica en la implementación eficiente en software de un esquema de cifrado híbrido basado en el esquema conocido como DHIES. Una vez implementada la propuesta original, se buscaron posibles mejoras que pudieran ser hechas al esquema para incrementar la eficiencia del mismo, y al concluir los cambios, se integró dicha implementación a Pine, una aplicación para envío y recepción de correos, para darle uso real. Específicamente con las mejoras que se realizaron a DHIES, además de hacer más viable su uso real, se buscó simplificarlo para que en un futuro, cuando los parámetros de entrada deban aumentar de tamaño, como ha sucedido con la mayoria de los algoritmos existentes, no se requieran hacer modificaciones significativas y que tampoco afecte el rendimiento del esquema.

Por otro lado, el punto central del proyecto es la seguridad por lo cual se presenta también una prueba, sencilla pero muy significativa, sobre el nivel de seguridad que ofrece el esquema implementado.

En el capítulo 2 se presentan algunas definiciones sobre seguridad demostrable, algunas de las supociciones en las que se fundamenta y los ataques criptográficos a los que se enfretan los esquemas. También se definen algunas de las funciones básicas usadas en la construcción de distintos esquemas criptográficos. Después, en el capítulo 3, se presenta una pequeña introducción acerca del surgimiento y la importancia del cifrado híbrido, junto con el estado del arte del mismo y la descripción de algunos de los esquemas híbridos más representativos reportados hasta el momento. En el capítulo 4 se da una descripción y un análisis detallados del esquema DHIES, que es la base del proyecto, así como de las

modificaciones hechas a éste para mejorarlo. También se presenta la prueba de seguridad correspondiente al esquema resultante y una comparativa de costos con otros esquemas de cifrado híbrido.

Posteriormente, se detalla toda la información importante relacionada con la implementación realizada y sobre la integración del esquema con una aplicación de correos, esto en los capítulos 5 y 6 respectívamente. En lo referente a la implementación del esquema se dan las bases teóricas que fueron necesarias antes de la implementación, información de los algoritmos elegidos y los resultados obtenidos de dichas implementaciones. En lo referente a la aplicación se presentan datos de la aplicación elegida, de cómo se añadió el esquema de cifrado, de los servicios que presta y de sus restricciones.

Por último, en el capítulo 7 se dan las conclusiones a las que se llegó al final del proyecto, además de algunos puntos sobre como ampliarlo y mejorarlo y sobre futuros trabajos que se pueden realizar en base a este proyecto.

Adicionalmente se presentan tres apéndices, el primero con las bases para entender la aritmética utilizada en las operaciones del esquema, el segundo con la descrpción del algoritmo de cifrado usado en la construcción del esquema, y el último a manera de *manual de usuario* para la aplicación.

CAPÍTULO 2

Conceptos básicos, seguridad y suposiciones

La insignificancia es siempre una garantía de seguridad.
Esopo

El término de seguridad proviene del latín, de la palabra "securitas". Comúnmente cuando hablamos de seguridad nos referimos a la ausencia de riesgos o la confianza que tenemos en algo o alguien, sin embargo, este término tiene diversos sentidos dependiendo del área o campo en el que estemos trabajando.

En este apartado vamos a ver algunos puntos importantes sobre seguridad que son necesarios para entender las pruebas realizadas posteriormente. Se definen algunos conceptos de seguridad, los tipos de ataques usados en contra de esquemas de cifrado, algunos datos sobre el modelo de prueba usado y algunas de las suposiciones bajo las cuales están basadas las pruebas.

2.1. Definiciones de seguridad

En este apartado vamos a definir algunos tipos o modelos de seguridad que pueden ser provistos por los esquemas de cifrado. Aunque existe una gran variedad, algunos de ellos son muy similares entre sí, además únicamente serán definidos aquellos que tienen relación directa con el proposito del proyecto.

2.1.1. Seguridad perfecta

En este apartado se introduce el concepto de seguridad perfecta y se muestran algunos detalles de este modelo de seguridad.

En palabras sencillas, un cifrador logra la seguridad perfecta si, dada una llave $k \in \mathcal{K}$, un adversario que obtiene un texto cifrado $c \in \mathcal{C}$ es incapaz de deducir información relevante sobre el texto en claro $m \in \mathcal{M}$ que lo produjo. Esta es la definición de seguridad más estricta y es el nivel de seguridad ideal a alcanzar por cualquier cifrador. También es conocida como *seguridad incondicional* pues no se ponen restricciones en cuanto al poder de cómputo del adversario o como *seguridad teórica* debido a que es posible comprobar la seguridad de forma matemática.

Tal y como lo define Shannon en [43], un esquema de cifrado $(KG, Enc(k), Dec(k))$ sobre un espacio de mensajes $\mathcal{M}$ tal que $|\mathcal{M}| = |\mathcal{K}| = |\mathcal{C}|$ es considerado **perfectamente seguro** si y sólo si:

- cada llave $k \in \mathcal{K}$ es elegida con igual probabilidad $1/\mathcal{K}$ por el algoritmo generador de llaves KG, y
- para cada mensaje $m \in \mathcal{M}$ y cada texto cifrado $c \in \mathcal{C}$, existe una única llave $k \in \mathcal{K}$ tal que $Enc(k, m) = c$

De acuerdo a la definición de indistinguible, un esquema de cifrado sobre un espacio de mensajes $\mathcal{M}$ es perfectamente seguro o perfectamente indistinguible si y sólo si, para $m_0, m_1 \in \mathcal{M}$ y cada $c \in \mathcal{C}$, la probabilidad de que c corresponda al cifrado de m_0 es igual a la probabilidad de que c corresponda al cifrado de m_1. Basándonos en el experimento mostrado en 2.2, podemos decir en este caso que el adversario A tiene probabilidad $1/2$ de adivinar cuando se enfrenta a un esquema perfectamente seguro.

Si este es el modelo de seguridad ideal, entonces ¿por qué definir más? La respuesta es sencilla. El crear esquemas de cifrado bajo este modelo de seguridad conlleva a dos grandes limitantes: la primera de ellas es que cada llave debe ser utilizada una única vez y, como resultado, la segunda es que el espacio de llaves $\mathcal{K}$ debe ser tan grande como el espacio de mensajes $\mathcal{M}$, o incluso mayor. Estos problemas hacen difícil el poder crear esquemas que sean prácticos y perfectamente seguros por lo cual es necesario buscar otras alternativas, la seguridad computacional es una de ellas.

2.1.2. Seguridad computacional

Este tipo de seguridad surgió para resolver los problemas inherentes a la seguridad perfecta en lo referente a la práctica; los cifradores definidos bajo este tipo de seguridad permiten cifrar mensajes "grandes" (mega o gigabytes) con llaves "pequeñas" (cientos de bits). Recibe el nombre de "computacional" debido principalmente a que, a diferencia del tipo anterior, establece ciertos límites en cuanto al poder de cómputo del adversario (tiempo escencialmente). La principal idea de la *seguridad computacional* es que un esquema de cifrado puede ser roto pero el tiempo que se necesita para lograrlo es muy grande, hablemos de años equivalentes a varias vidas humanas.

Esta seguridad es más débil que la anterior, ya que los esquemas pueden ser rotos, pero en la actualidad es "suficiente" para propositos prácticos. En 1.2 se mencionó lo que se conoce como el principio de Kerckhoffs, pero éste en realidad es parte de una serie de seis principios expuestos por el Dr. Auguste Kerckhoffs en el siglo XIX [21]. La seguridad computacional toma como base otro de esos seis principios el cual establece que *un cifrador debe ser, prácticamente, sino es que matematicamente, indecifrable*. La idea de un cifrador computacionalmente seguro, o prácticamente indecifrable, es que obtenemos un nivel de seguridad suficiente si el esquema no puede ser roto en un tiempo y con una probabilidad razonables.

Para pasar de la seguridad perfecta a la práctica es necesario suavizar dos requerimientos. Primero debemos considerar que el esquema será seguro únicamente contra adversarios "eficientes" durante una cantidad de tiempo razonable; y segundo que los adversarios pueden romper la seguridad con probabilidad "muy pequeña" (tan **insignificante** que en realidad no nos preocupa que ocurra).

La seguridad computacional puede ser definida bajo dos enfoques distintos. El enfoque *concreto* especifica límites exactos sobre la máxima probabilidad de éxito y el tiempo máximo de ejecución del adversario. Específicamente se definen esquemas con seguridad (t, ϵ), es decir, un adversario ejecutado a lo más en tiempo t tiene a lo más una probabilidad ϵ de éxito. Estos dos parámetros de seguridad suelen ser dados en potencias de 2, haciendo referencia a los ciclos del procesador en el caso del tiempo. El problema con este enfoque es que es demasiado específico, no sabemos nada acerca de qué pasa si el adversario "ejecuta" durante el doble o la mitad del tiempo especificado; tampoco sabemos sobre que tipo de hardware se supone seguro, una computadora personal, una super-computadora o un cluster de cientos de procesadores. Lo correcto sería definir el poder de cómputo del adversario y dar un rango de valores para t y ϵ para los cuales el esquema permanece seguro.

El segundo es el enfoque asintótico, que es un poco más flexible. Aquí tanto el tiempo como la probabilidad de éxito son vistos como *funciones* de un parámetro de seguridad n, un entero, establecido por los participantes y que además se asume será conocido por el adversario. En este enfoque, con adversarios eficientes nos referimos a la ejecución de algoritmos probabilísticos en tiempo polinomial en n (PPT por sus siglas en ingles), es decir, dadas un par de constantes a, c el algoritmo se ejecuta en tiempo $a * n^c$; suponemos que los adversarios tienen mucho más poder de cómputo que los participantes. Y al hablar de una probabilidad de éxito "insignificante", nos referimos a que la probabilidad es menor a cualquier inverso polinomial en n, es decir, dada una constante c la probabilidad de éxito del adversario es menor que n^{-c}.

Cabe señalar que en la gran mayoría de los casos el parámetro de seguridad es el que determina el tamaño de la llave usada, y es por eso que es esos casos a mayor tamaño de llave mayor será la seguridad. Por otro lado, debemos tener cuidado con el tamaño de n

pues éste garantiza la seguridad sólo para valores lo suficientemente grandes; el valor de "grande" depende de las características propias del esquema.

2.1.3. Seguridad demostrable

Un esquema de cifrado cuenta con *seguridad demostrable* si es posible mostrar de alguna forma que romperlo es tan difícil como resolver un problema conocido que se supone difícil; decimos que se "supone" difícil porque aunque no existe un método de resolverlo en un tiempo razonable, tampoco existe una prueba de que en realidad lo sea. Ejemplo de estos problemas difíciles son la factorización de números enteros grandes y la resolución de logaritmos discretos. En realidad "demostrable" se refiere a igualar con una **suposición**.

Este tipo de seguridad puede ser considerada como parte de la anterior pues muchos de los métodos de cifrado definidos bajo la seguridad práctica se equiparan a resolver problemas considerados difíciles sólo que, a diferencia de la seguridad demostrable, no existe una **prueba** de dicha equivalencia. Sumado a esto, los enfoques concreto y asintótico también son aplicables a la seguridad demostrable ya que lo que probamos es que el esquema de cifrado es seguro computacionalmente (bajo las restricciones y definiciones dadas en la sección anterior).

Las pruebas realizadas para demostrar la seguridad son conocidas como **reducciones**, esto por el hecho de que reducimos el problema de analizar un esquema de cifrado completo a analizar una (o raramente más de una) primitiva criptográfica básica. Las pruebas por reducción *suponen* que un problema o primitiva básica es difícil de resolver y prueban, en base a esa suposición, que el esquema en cuestión es seguro; en realidad no demostramos la seguridad del esquema sino que lo reducimos a uno que se considera seguro. Este tipo de pruebas muestran cómo convertir un adversario eficiente A, que tiene una probabilidad no despreciable de éxito de romper el cifrador, en un adversario eficiente A', el cual tiene éxito en resolver el problema que se supone difícil.

Un punto importante es que las supocsiciones usadas como base puede ser débiles o fuertes. Una suposición débil es aquella que supone pocas cuestiones relacionadas al problema por lo cual ofrece un mayor nivel de seguridad. Por el contrario, una suposición fuerte indica que se dan por hecho muchas condiciones sin conocimientos reales por lo cual reduce el nivel de seguridad esperado.

Los pasos generales a seguir para realizar una prueba por reducción son:

1. Establecer un adversario eficiente A que ataca al cifrador con una probabilidad $\epsilon(n)$ de éxito.

2. Construir un algoritmo eficiente A' que trata de resolver el problema X que suponemos difícil usando a A como una sub-rutina. A' no sabe como funciona A. Así que,

dada una instancia x del problema X, el algoritmo A' simula una instancia del cifrador tal que:

   a) la vista de A cuando actua como sub-rutina de A' es lo más cercana posible a la vista de A cuando interactua con el cifrador, y
   b) si A tiene éxito rompiendo la instancia del cifrador simulado por A', esto le permitirá a A' resolver la instancia x del problema con probabilidad no despreciable.

3. Tomar las dos instancias del punto anterior implica que si $\epsilon(n)$ no es despreciable, entonces A' resuelve el problema X con probabilidad no despreciable; esto contradice la supocición inicial sobre X.

4. Concluimos que, dada la suposición sobre el problema X, no existe adversario eficiente que pueda romper el cifrador con probabilidad no despreciable.

Hay que mencionar también que las pruebas pueden diseñarse bajo distintos modelos. El primero de ellos es el **modelo estándar** en el cual se limita al adversario sólo en la cantidad de tiempo y el poder de cómputo disponible. Este modelo es el que hemos estado definiendo hasta este momento, y son los esquemas de cifrado que basan su seguridad en alguna suposición difícil de resolver en tiempo polinomial. El problema es que nuevamente es difícil en algunos casos demostrar la seguridad en base a estas suposiciones así que es necesario definir otros modelos en los que se reemplaza a estas funciones básicas por versiones "ideales". El modelo de **oráculo aleatorio** y el modelo de **conocimiento-cero** son ejemplos de otros modelos de pruebas.

Las pruebas presentadas posteriormente se ubican dentro del modelo de *oráculo aleatorio*. A grandes rasgos, las pruebas dentro del modelo de oráculo aleatorio consideran que los participantes e incluso el adversario tienen acceso a una función aleatoria que funciona como oráculo, es decir, responde a cada consulta con una respuesta tomada de forma uniformemente aleatoria dentro del espacio de posibles valores, excepto en casos especiales donde se requiere que a la misma consulta siempre devuelva la misma respuesta. Algunos investigadores no aceptan por completo este tipo de pruebas debido a que en la realidad no existe una función que implemente un verdadero oráculo aleatorio pero aún así siguen siendo las pruebas más apegadas a la seguridad real.

Al principio, la seguridad demostrable era conocida únicamente por personas enfocadas a la parte puramente teórica pero en años recientes esto ha cambiado. Hemos visto como en la práctica se prefiere el uso de esquemas con seguridad demostrable sobre aquellos que no la tienen, e incluso ahora se ve a la seguridad demostrable como una característica deseable para los esquemas propuestos.

Existen distintas desventajas y limitantes que podemos encontrar en las pruebas de seguridad demostrable. Como principal limitante está el hecho de que no abarca la seguridad contra ataques usados en la práctica como son ataques de tiempo y análisis diferencial

de potencia o de fallas. Esto no quiere decir que los esquemas demostrados seguros son débiles contra este tipo de ataques. Algunos sí lo son otros no, simplemente no podemos asegurar que sean seguros contra ellos. Dentro de las desventajas tenemos que podemos demostrar la seguridad de cierto esquema pero lo hacemos para un problema equivocado o en el modelo incorrecto. También hay problemas en la práctica, podemos probar que un esquema es seguro contra una clase específica de ataques pero lo usamos en un medio donde es necesario un esquema seguro contra ataques de otra clase, tal vez más efectivos. Debido a estos problemas es que debemos ser muy cuidadosos en definir los requerimientos prácticos del diseño y uso del esquema antes de demostrar su seguridad.

2.2. Conceptos básicos de la seguridad

La busqueda de la seguridad inicia desde tiempos muy antiguos a la era digital, pero es la seguridad dentro del ámbito computacional la que nos interesa para nuestra investigación. En este ámbito, se consideraba que un criptosistema podía alcanzar dos niveles de seguridad; el primer nivel era llamado "seguridad semántica" y el segundo como "maleabilidad" (en el sentido criptográfico). La seguridad semántica fue la primera definición formal de seguridad para esquemas de cifrado, siendo S. Goldwasser y S. Micali los primeros en establecer la idea en 1982 [13]. Los mismos autores demuestran dos años después [14] que la seguridad semántica es equivalente a atribuir la propiedad de ser **indistinguible**, una definición más sencilla y fácil de probar.

Al afirmar que un cifrador tiene la característica de ser *indistinguible* estamos diciendo que un adversario que obtiene un texto cifrado $c \in \mathcal{C}$ es incapaz de distinguir cual texto en claro $m \in \mathcal{M}$ lo produjo. La *maleabilidad* es una característica que nos indica que un esquema de cifrado permite que el adversario sea capaz de, dado el texto cifrado c de un mensaje m, transformar c en otro texto cifrado c' cuyo decifrado corresponda al texto en claro $f(m)$ para algún tipo de función f conocida, sin tener ningún conocimiento acerca de m.

De manera formal, las pruebas de seguridad hacen uso de "experimentos", que son una especie de juego entre un adversario que trata de romper el esquema y una entidad de pruebas, para saber si el esquema de cifrado es o no seguro. Los experimentos usados aquí son del tipo *encuentra y adivina* (*fg*) los cuales están divididos en dos etapas, en la primera el adversario "encuentra" dos mensajes que enviará al cifrador y en la segunda etapa intenta "adivinar" a cual de sus dos mensajes corresponde el texto cifrado que recibió. La finalidad de definir experimentos es para establecer qué probabilidad tiene el adversario de vencer la seguridad del esquema, a esta probabilidad le llamamos **ventaja** (Adv). A continuación se muestra con un experimento básico la forma de probar si un cifrador es indistinguible (*ind*) en presencia de un adversario. En lo siguiente, la notación $x \xleftarrow{\$} S$ significa que se selecciona a x del conjunto finito S de forma uniformemente alea-

toria.

En este experimento, se considera a $ENC = (KG(n), Enc(ek, \cdot), Dec(dk, \cdot))$ como un esquema de cifrado con parámetro de seguridad n (comúnmente la longitud de la llave) y un adversario A (fisgón), que se supone un algoritmo probabilístico eficiente. Primero A genera un par de mensajes m_0, m_1 de igual longitud. Después el algoritmo generador de llaves obtiene la llave ek para el cifrador. Se selecciona de forma aleatoria un bit $b \in \{0, 1\}$ y se cifra el mensaje m_b con el algoritmo correspondiente $c = Enc(ek, m_b)$ y se entrega c, el texto cifrado o **reto**, a A. Con base en el "reto" c, A elige un bit $\hat{b}$ tratando de decidir cual mensaje m fue el que se cifró. La salida del experimento decide si el adversario fue o no exitoso en su tarea.

$$
\begin{aligned}
&\texttt{experimento } Exp^{ind-fg}_{ENC,\ A}(n)\\
&\quad (m_0, m_1) \leftarrow A\\
&\quad ek \leftarrow KG(n)\\
&\quad b \xleftarrow{\$} \{0, 1\}\\
&\quad c \leftarrow Enc(ek, m_b)\\
&\quad \hat{b} \leftarrow A(c)\\
&\quad \texttt{salida: si } \hat{b} = b \text{ regresa 1 si no 0}
\end{aligned}
$$

Actualmente la seguridad de un esquema de cifrado se define en base a los ataques que es capaz de soportar. Existen distintos tipos de ataques que un adversario puede emplear y dependiendo de cuáles de ellos es capaz de resistir el esquema, podemos asegurar que tenemos un cifrador semánticamente seguro y/o no-maleable. En la siguiente sección se presentan los tipos de ataques que pueden ser efectuados por un adversario para tratar de romper algún cifrador.

2.3. Ataques criptográficos

Como se mencionó en el capítulo 1, necesitamos definir la seguridad de un esquema de cifrado en base al tipo de adversario al que nos vamos a enfrentar. De forma general podemos clasificar a todo el conjunto de posibles adversarios en dos grandes grupos dependiendo de las actividades que realizan: los que realizan ataques **pasivos** y los que realizan ataques **activos**.

Un atacante pasivo, también conocido como "fisgón", es aquel que únicamente monitorea el canal por el que se está llevando a cabo la comunicación, es decir, trata de atacar la confidencialidad del esquema. Un atacante activo o "intruso" es aquel que además de "escuchar" la comunicación, trata de borrar, introducir y/o alterar de alguna forma las transmisiónes hechas por los participantes. El intruso se enfrenta al problema de la confidencialidad, la integridad de los datos e incluso la autenticación.

Dentro de los ataques pasivos existen cuatro tipos específicos de ataques que pueden ser usados para violar la seguridad de un cifrador. Lo que diferencía a cada uno de estos ataques es principalmente la cantidad de información que tiene disponible el adversario cuando tratar de romper el esquema de cifrado. Dentro de esa información disponible, en algunos casos suponemos que el adversario cuenta con una herramienta muy poderosa conocida como **oráculo**. Un oráculo es una especie de "máquina" vista a manera de una caja negra que se encarga de cifrar o descifrar mensajes elegidos por el adversario, sin revelar ningún otro tipo de información. A continuación se da una breve explicación de dichos ataques; el orden en que se describen es del menos al más poderoso:

1. **Sólo texto cifrado**: aquí el adversario únicamente tiene acceso a una copia del texto cifrado y su finalidad es tratar de encontrar la llave o algo de información sobre el texto en claro.

2. **Texto en claro conocido**: aquí el adversario conoce el texto cifrado y su correspondiente texto en claro, trata de encontrar la llave o ciertos patrones que le permitan obtener información de mensajes futuros.

3. **Texto en claro elegido** (CPA): en este caso, el adversario obtiene acceso a un "oráculo" de cifrado durante cierto tiempo, con la restricción de que él no puede obtener la llave de dicho oráculo de ninguna forma. El adversario puede usarla para cifrar los mensajes de su elección durante ese tiempo y trata de deducir la llave a partir de los textos cifrados que obtuvo.

4. **Texto cifrado elegido** (CCA): este es el ataque más poderoso. Aquí el atacante obtiene adicionalmente acceso temporal a un "oráculo" de descifrado. Ahora el adversario elige las cadenas que desea descifrar y en base a los resultados obtenidos trata de deducir la llave utilizada.

De los ataques definidos arriba, los dos últimos son los más importantes por ser los más poderosos y es en base a ellos dos que se realizan las pruebas de seguridad para esquemas de cifrado. Adicionalmente, se definen extensiones para los ataques CPA y CCA conocidas como *adaptables*; los ataques *texto en claro elegido adaptable* y *texto cifrado elegido adaptable* (CCA2) se distinguen ya que el adversario tiene la posibilidad de elegir los mensajes que dará al oráculo a partir de resultados obtenidos con anterioridad.

2.3.1. Privacidad contra CPA

A continuación se muestra una forma para determinar si un cifrador es indistinguible contra ataques de tipo CPA, esto mediante el uso del experimento o juego mostrado aquí. Un criptosistema será considerado indistinguible contra CPA (*ind-cpa*) si cualquier adversario probabilístico ejecutado en tiempo polinomial tiene una ventaja despreciable sobre el evento de adivnar de forma aleatoria, es decir, la probabilidad de que el adversario gane es $1/2 + \epsilon(n)$ donde $\epsilon(n)$ es una función despreciable para el parámetro de seguridad n

(vease 2.1.2 para más detalles).

El juego es, dado un esquema de cifrado $ENC = (KG(n), Enc(ek, \cdot), Dec(dk, \cdot))$ y un adversario A, se considera el experimento anterior, donde $A^{\mathcal{E}(ek,\cdot)}$ indica que, antes de generar el dato solicitado, el adversario tiene acceso a un oráculo de cifrado al puede hacer las consultas que le parezcan convenientes. Este acceso al oráculo es lo único que diferencía a este experimento del experimento genérico mostrado en 2.2. Cabe mencionar que este juego es la versión adaptable del ataque, en la versión no adaptable el acceso al oráculo sólo se permite en la primera etapa, antes de recibir el "reto".

$$
\begin{array}{l}
\texttt{experimento } Exp\ {}^{ind-cpa-fg}_{ENC,\ A}\ (n) \\
\quad ek \leftarrow KG(n) \\
\quad (m_0, m_1) \leftarrow A^{\mathcal{E}(ek,\cdot)}(n) \\
\quad b \xleftarrow{\$} \{0,1\} \\
\quad c \leftarrow Enc(ek, m_b) \\
\quad \hat{b} \leftarrow A^{\mathcal{E}(ek,\cdot)}(c) \\
\quad \texttt{salida: si } \hat{b} = b \text{ regresa 1 si no 0}
\end{array}
$$

La ventaja de A en el sentido indistinguible contra CPA (*ind-cpa*) para un experimento *fg* es:

$$\mathbf{Adv}\ {}^{ind-cpa-fg}_{ENC,\ A} = \Pr[Exp\ {}^{ind-cpa-fg}_{ENC,\ A} = 1] - 1/2$$

En el caso de cifradores simétricos ek y dk son una misma y son secretas, por eso es necesario el oráculo, mientras que para cifradores asimétricos ek es pública y se entrega al adversario en lugar del oráculo para que él mismo realize el cifrado. Por definición, un esquema semanticamente seguro o indistinguible debe ser probabilístico ya que de lo contrario el adversario siempre acertaría su respuesta; simplemente consulta al oráculo con ambos mensajes m_0, m_1 y los compararía con c para obtener el resultado. Una alternativa para cifradores determinísticos es prohibir al adversario consultar al oráculo con sus mensajes m_0, m_1.

Esquemas de cifrado como ElGamal son semánticamente seguros y además cuentan con seguridad demostrable. Otros esquemas que no son semanticamente seguros como RSA pueden ser transformados en esquemas semanticamente seguros con la ayuda de ciertas primitivas, pero su seguridad es más débil debido al uso de suposiciones fuertes.

2.3.2. Privacidad contra CCA

La definición para la seguridad contra este tipo de ataques es muy parecida a la anterior, la notable diferencia está en que aquí el adversario obtiene acceso tanto a un oráculo de cifrado como a uno de descifrado. Una vez más la diferencia entre las versiones es que para la versión adaptable (CCA2) el acceso al oráculo de descifrado es en ambas etapas mientras que en la versión no adaptable se prohibe el acceso a éste después de recibir el

"reto". Aquí se hablará de la versión adaptable.

Para este experimento vamos a considerar también a un adversario A y a un esquema de cifrado $ENC = (KG(n), Enc(ek, \cdot), Dec(dk, \cdot))$, donde $A^{\mathcal{D}(dk,\cdot)}$ indica que el adversario tiene acceso a un oráculo de descifrado al que puede hacer las consultas que le parezcan convenientes. La única restricción es que el adversario no puede consultar al oráculo con el texto cifrado del reto.

$$
\begin{aligned}
&\texttt{experimento } Exp\ {}^{ind-cca2-fg}_{ENC,\ A}\ (n)\\
&\qquad ek \leftarrow KG(n)\\
&\qquad (m_0, m_1) \leftarrow A^{\mathcal{E}(ek,\cdot),\mathcal{D}(dk,\cdot)}(n)\\
&\qquad b \overset{\$}{\leftarrow} \{0,1\}\\
&\qquad c \leftarrow Enc(ek, m_b)\\
&\qquad \hat{b} \leftarrow A^{\mathcal{E}(ek,\cdot),\mathcal{D}(dk,\cdot)}(c)\\
&\qquad \texttt{salida: si } \hat{b} = b \text{ regresa 1 si no 0}
\end{aligned}
$$

La ventaja de A en el sentido indistinguible contra CCA2 (*ind-cca2*) para un experimento *fg* es:

$$\mathbf{Adv}\ {}^{ind-cca2-fg}_{ENC,\ A} = \Pr[Exp\ {}^{ind-cca2-fg}_{ENC,\ A} = 1] - 1/2$$

Un esquema de cifrado puede ser seguro contra ataques de texto en claro elegido CPA (adaptable o no) e incluso seguro contra ataques de texto cifrado elegido no adaptable CCA y aún seguir siendo maleable, sin embargo si éste es seguro contra ataques de texto cifrado elegido adaptable $CCA2$ es equivalente a ser no maleable [14].

2.4. Suposiciones sobre Diffie-Hellman

Como ya se mencionó en la sección anterior, para demostrar la seguridad de un esquema de cifrado es necesario reducir el problema a uno que se supone o se considera difícil. En esta sección se presenta una serie de suposiciones sobre un problema bastante conocido, el problema de Diffie-Hellman. Estas suposiciones son importantes ya que son la base para la demostración de seguridad del esquema de cifrado híbrido propuesto en está investigación.

Diffie y Hellman propusieron en su artículo [9] un protocolo de comunicación que permite compratir un valor secreto entre dos entidades que no tienen ni han tenido contacto directo y las cuales se comunican a través de un canal inseguro. Comúnmente este valor secreto es utilizado como llave secreta para un cifrador simétrico. Este esquema de comunicación se considera seguro únicamente en presencia de un adversario pasivo y su seguridad radica en la dificultad atribuida al problema de calcular logaritmos discretos.

Formalmente podemos formular al problema del *logaritmo discreto* (PLD) como: dado un grupo cíclico G de orden n, cuyo generador está denotado por α, y dado un elemento $\beta \in G$, PLD se refiere a la dificultad de encontrar el exponente único x, donde

$0 \leq x \leq n-1$, tal que $\alpha^x = \beta$.

En general, para el buen funcionamiento del protocolo se establece un grupo cíclico y finito G de elementos relacionados mediante la multiplicación, y la exponenciación como consecuencia de multiplicaciones repetitivas, del cual tomamos un elemento generador g (para mayores referencias sobre los conceptos de teoría de grupos dirijirse al apéndice A). Se define a los dos participantes en la comunicación A y B, y se supone un adversario fisgón. El primer paso es que A elige un elemento aleatorio $u \in G$, calcula g^u y envía este valor a B. Por su parte B elige un elemento aleatorio $v \in G$, calcula g^v y lo envía a A. El secreto o llave compartida se establece como g^{uv}, el cual es calculado de forma sencilla mediante $(g^u)^v$ y $(g^v)^u$ por cada uno de los participantes.

En los apartados siguientes se definen cuatro versiones acerca del problema de DH. Las primeras dos son suposiciones básicas que son utilizadas comúnmente y las últimas dos son dadas como introducción para la prueba final del esquema de este proyecto.

2.4.1. DH computacional (CDH)

La primera de estas suposiciones es conocida como la suposición *estándar de Diffie-Hellman* o suposición de *Diffie-Hellman computacional*. La seguridad del protocolo se basa en la suposición de que para un adversario que obtenga g^u y g^v será difícil calcular g^{uv}. Para esto se define el siguiente experimento contra un adversario A:

$$
\begin{aligned}
&\texttt{experimento } Exp\,{}^{cdh}_{G,\,A}\\
&\qquad u, v \xleftarrow{\$} \{1, \ldots, |G|\}\\
&\qquad U \leftarrow g^u\\
&\qquad V \leftarrow g^v\\
&\qquad Z \leftarrow A(U, V)\\
&\qquad \texttt{salida: si } Z = g^{uv} \text{ regresa 1 si no 0}
\end{aligned}
$$

donde la ventaja de dicho adversario está dada por

$$\mathbf{Adv}\,{}^{cdh}_{G,\,A} = \Pr\left[Exp\,{}^{cdh}_{G,\,A} = 1\right]$$

Se considera que la suposición CDH se mantiene en un grupo G si para todo adversario eficiente A, la ventaja $\mathbf{Adv}\,{}^{cdh}_{G,\,A}$ es insignificante.

2.4.2. DH de decisión (DDH)

Esta suposición es más fuerte que la anterior y establece que dos tripletas (g^u, g^v, g^{uv}) y (g^u, g^v, g^w) con valores u, v, w tomados aleatoriamente de $\{1, \ldots, |G|\}$ son computacionalmente indistinguibles para el adversario A. Para lo cual definimos el par de experimentos

mostrados mas adelante, donde A elige $b = 1$ si es capaz de distingir con que experimento está interactuando, real o aleatorio. La ventaja del adversario está dada por

$$\mathbf{Adv}\,{}^{ddh}_{G,\,A} = \Pr\left[Exp\,{}^{ddh-real}_{G,\,A} = 1\right] - \Pr\left[Exp\,{}^{ddh-rand}_{G,\,A} = 1\right]$$

`experimento` $Exp\,{}^{ddh-real}_{G,\,A}$	`experimento` $Exp\,{}^{ddh-rand}_{G,\,A}$
$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$	$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$
$U \leftarrow g^u$	$U \leftarrow g^u$
$V \leftarrow g^v$	$V \leftarrow g^v$
$Z \leftarrow g^{uv}$	$Z \leftarrow g^z$
$b \leftarrow A(U, V, Z)$	$b \leftarrow A(U, V, Z)$
`salida:` b	`salida:` b

2.4.3. DH con hash (HDH)

Esta suposición es también más fuerte que CDH pero es más débil que DDH. El motivo por el cual surgió HDH es porque Diffie-Hellman no cumple por completo con la noción de seguridad semántica, es decir, es posible deducir ciertos bits de g^{uv} conociendo g^u y g^v. La idea de HDH es que si aplicamos una función picadillo (*hash*) $H : \{0,1\}^* \to \{0,1\}^{hl}$ al valor secreto g^{uv} éste no podrá distinguirse de una cadena de bits aleatorios, de esta forma se ofrece mayor seguridad. El experimento se define así:

`experimento` $Exp\,{}^{hdh-real}_{G,\,H,\,A}$	`experimento` $Exp\,{}^{hdh-rand}_{G,\,H,\,A}$
$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$	$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$
$U \leftarrow g^u$	$U \leftarrow g^u$
$V \leftarrow g^v$	$V \leftarrow g^v$
$Z \leftarrow H(g^{uv})$	$Z \xleftarrow{\$} \{0,1\}^{hl}$
$b \leftarrow A(U, V, Z)$	$b \leftarrow A(U, V, Z)$
`salida:` b	`salida:` b

donde A elige $b = 1$ si es capaz de distinguir con que experimento está interactuando, real o aleatorio. La ventaja del adversario está dada por

$$\mathbf{Adv}\,{}^{hdh}_{G,\,H,\,A} = \Pr\left[Exp\,{}^{hdh-real}_{G,\,H,\,A} = 1\right] - \Pr\left[Exp\,{}^{hdh-rand}_{G,\,H,\,A} = 1\right]$$

Mientras que en DDH suponemos que g^{uv} aparenta ser un elemento del grupo G tomado de forma aleatoria, en HDH suponemos que $H(g^{uv})$ aparenta ser una cadena de bits aleatorios, ambos incluso conociendo g^u y g^v. Mientras mejor elijamos la función picadillo (*hash*) dentro del conjunto de funciones criptográficas, más débil será nuestra suposición.

2.4.4. Oráculo de DH (ODH)

Esta es una nueva suposición acerca del problema de DH. Esta suposición es muy parecida a HDH sólo que aquí se define un oráculo $\mathcal{H}_v(X)$ el cual recibe cierto valor X y retorna el valor de $H(X^v)$, siendo $H : \{0,1\}^* \to \{0,1\}^{hl}$ una función hash y $v \in \mathbb{Z}_q$ un valor preestablecido.

Lo que suponemos con ODH es que incluso si otorgamos al adversario acceso a este oráculo $\mathcal{H}_v(\cdot)$, con la única restricción de no consultar g^u, la seguridad del esquema híbrido se mantiene. El experimento se define así:

`experimento` $Exp^{odh-real}_{G,\ H,\ A}$	`experimento` $Exp^{odh-rand}_{G,\ H,\ A}$
$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$	$u, v \xleftarrow{\$} \{1, \ldots, \lvert G\rvert\}$
$U \leftarrow g^u$	$U \leftarrow g^u$
$V \leftarrow g^v$	$V \leftarrow g^v$
$W \leftarrow H(g^{uv})$	$W \xleftarrow{\$} \{0,1\}^{hl}$
$\mathcal{H}_v(X) \stackrel{def}{=} H(X^v)$	$\mathcal{H}_v(X) \stackrel{def}{=} H(X^v)$
$b \leftarrow A^{\mathcal{H}_v(\cdot)}(U, V, W)$	$b \leftarrow A^{\mathcal{H}_v(\cdot)}(U, V, W)$
`salida:` b	`salida:` b

donde A elige $b = 1$ si es capaz de distingir con que experimento está interactuando, real o aleatorio. La ventaja del adversario en ODH está dada por

$$\mathbf{Adv}^{odh}_{G,\ H,\ A} = \Pr\left[Exp^{odh-real}_{G,\ H,\ A} = 1\right] - \Pr\left[Exp^{odh-rand}_{G,\ H,\ A} = 1\right]$$

2.5. Algunas primitivas básicas

A continuación se definen tres de las primitivas (operaciones básicas) criptográficas que son necesarias para comprender el desarrollo del proyecto. La primera de ellas es una breve introducción al criptosistema ElGamal, que ha servido de base para desarrollar algunos esquemas de cifrado y para probar su seguridad, incluyendo el nuestro. Posteriormente se define el concepto de función hash y se hablará un poco sobre código de autenticación de mensajes (MAC).

2.5.1. ElGamal

Como se mencionó en 1.2.2, ElGamal es un esquema de cifrado de llave pública creado por Taher ElGamal que fue publicado en 1985 [10]. Este criptosistema puede ser visto como el esquema de acuerdos Diffie-Hellman, de ahí que su seguridad recaiga directamente tanto en el problema del logaritmo discreto como en el problema de Diffie-Hellman.

Suponga ahora que cierta entidad B desea comunicarse con otra entidad A de forma segura. Como primer paso en el esquema, la entidad A debió haber "ejecutado" el algoritmo generador de llaves KG, el cual se compone de tres pasos:

1. generar un número primo "grande" p y un generador α del grupo multiplicativo $\mathbb{Z}_p^\star$ de enteros módulo p,
2. seleccionar un número a de forma aleatoria tal que $1 \leq a < p-1$ y calcular $\alpha^a \ (mod\ p)$,
3. la llave pública es (p, α, α^a) y la llave privada es a.

Una vez que el participante A generó sus llaves, puede comenzar a recibir mensajes cifrados de cualquier entidad incluyendo a B. Cada vez que B desee cifrar un mensaje, antes de enviárselo a A, lo que debe hacer es seguir cinco sencillos pasos:

1. obtener la llave pública de A, (p, α, α^a),
2. convertir el texto en claro en un número entero m en el rango $0, 1, \ldots, p-1$,
3. elegir aleatoriamente un número b tal que $1 \leq b < p-1$,
4. calcular $\beta \equiv \alpha^b \ (mod\ p)$ y $\gamma \equiv m(\alpha^a)^b \ (mod\ p)$ y
5. enviar el texto cifrado $c = (\beta, \gamma)$ a A.

Hecho esto, cuando A reciba el mensaje y desee descifrar el contenido, únicamente deberá:

1. usar su llave privada para calcular $\beta^{p-1-a} (mod\ p)$ y
2. descifrar m mediante $\beta^{-a}\gamma (mod\ p)$.

De esta descripción podemos identificar el mayor problema que tiene este criptosistema, y este es que los mensajes están severamente limitados por la longitud y el valor del número primo elegido. En cuanto a la eficiencia, puede verse que en el cifrado se requieren de 2 exponenciaciones modulares y en el descifrado únicamente se necesita una, lo que es costoso. Además tiene una importante expansión en la longitud del mensaje, en teoría el texto cifrado tendrá el doble de la longitud del texto en claro.

2.5.2. Funciones picadillo

Las funciones picadillo son de las primitivas más comunes en la criptografía moderna y actualmente tienen usos variados dentro de ella. En esencia una función hash debe, en base a un mensaje de entrada, obtener su "resumen" que es una cadena de longitud fija. De manera formal una función picadillo $H : \{0,1\}^* \rightarrow \{0,1\}^h$ debe cumplir esencialmente con dos requisitos: el primero es la *compresión*, es decir, debe tomar como entrada elementos o cadenas de longitud arbitraria y finita n y transformarlas en cadenas de longitud

fija h, en la mayoría de los casos $h < n$, el segundo requisito es la *facilidad de cálculo*, es decir, dada una función H y una cadena m, $H(m)$ es fácil de obtener. El resumen o valor hash puede ser usado con distintos fines, como una huella digital que identifique al mensaje la cual puede ser firmada con fines de autenticación o puede servir para validar la integridad del mensaje, incluso el valor hash puede servir como llave para algún esquema de cifrado.

Desde el punto de vista criptográfico y de seguridad, para que una función hash sea segura deberá de contar con tres propiedades básicas que se definen a continuación:

- **resistencia a pre-imagen**: dada una cadena y (la imagen), debe ser computacionalmente difícil encontrar una cadena x (pre-imagen) tal que $H(x) = y$,
- **resistencia a segunda pre-imagen**: dada una pre-imagen x, es computacionalmente difícil encontrar una segunda pre-imagen $x' \neq x$ tal que $H(x) = H(x')$, y
- **resistencia a colisiones**: es computacionalmente imposible encontrar un par de cadenas distintas x_1 y x_2 tal que $H(x_1) = H(x_2)$.

Dentro de las funciones picadillo más utilizadas se encuentran las familias *Secure Hash Algorithm* (SHA [29]) y *Message Digest* (MD [34, 35]). Cabe señalar que existen distintos tipos de funciones picadillo que podemos catalogar por estructura o por funcionamiento entre otras (ver capítulo 9 de [26]). Por estructura principalmente podemos identificar funciones hash con llave y funciones hash sin llave, por funcionamiento identificamos dos categorías que son de gran importancia criptográfica, las funciones para código de detección de modificaciones (MDC) para asegurar integridad de datos y las funciones para código de autenticación de mensajes (MAC) que permiten tanto autenticar la fuente del mensaje como su integridad.

2.5.3. Código de autenticación de mensajes

Algoritmos de código de autenticación de mensajes es el nombre con el que se le denomina a un tipo de funciones hash que requieren una llave secreta para su funcionamiento y que, como su nombre lo dice, sirven para proveer el servicio de autenticación de mensajes. Estas funciones o algoritmos fueron propuestos inicialmente basándose en cifradores por bloques pero en la actualidad hay también algoritmos basados en cifradores por flujo y en funciones hash sin llave. Éstas últimas son construidas usando una función hash (SHA-1, MD5, etc) como si fuera una caja negra y agregando ciertas concatenaciones a los valores y aplicando algunas funciones *xor*, y se les denomina como funciones HMAC.

Comúnmente las funciones MAC se componen de dos algoritmos (T, V): el primer algoritmo $T(mK, M) = tag$ recibe un mensaje M a autenticar y una llave mK y es usado para crear la etiqueta de autenticación tag, mientras que el segundo algoritmo $V(mK, M, tag)$ recibe un mensaje M, una llave mK y una etiqueta tag y se encarga de verificar si dicha etiqueta corresponde al mensaje.

CAPÍTULO 3

Esquemas de cifrado híbrido

A través de los años, la criptografía ha ido avanzando de forma muy acelerada y por distintos caminos, siendo la criptografía simétrica y asimétrica las dos ramas principales. Específicamente hablando del cifrado de llave pública, a partir del surgimiento del esquema ElGamal en 1985, se ha intentado acondicionar los esquemas de este tipo para que su eficiencia sea comparable a la de los esquemas de llave privada, siendo la creación de cifradores híbridos una de las posibles alternativas.

Un esquema de cifrado híbrido es un mecanismo de cifrado que puede ser construido a partir de dos esquemas distintos: un mecanismo de encapsulado de llave o KEM (*Key Encapsulation Mechanism*) basado en un cifrador de llave pública, y un mecanismo de encapsulado de datos o DEM (*Data Encapsulation Mechanism*) basado en un cifrador de llave privada.

Aquí se hace una breve introducción a lo que es cifrado híbrido, su surgimiento e importancia. Además, a manera de estado del arte, se describen cinco de los esquemas híbridos más importantes publicados al momento de la realización de este trabajo. También se hace mención de un tipo de esquemas distinto a los anteriores que, aunque no pueden ser comparados con los demás debido a su estructura, es importante estudiar, pues también se ha desarrollado el equivalente de una versión híbrida.

3.1. Importancia del cifrado híbrido

Históricamente los esquemas de cifrado simétrico o de llave secreta fueron los primeros en usarse y estudiarse. Con el avance del tiempo y haciendo uso de la tecnología, algunos cifradores simétricos son los más eficientes gracias a su diseño. Actualmente tenemos algoritmos para cifrado simétrico que actúan en tiempos imperceptibles para el hombre y algunos de ellos ocupando pocos recursos de procesamiento y/o almacenamiento.

El gran problema que siempre ha tenido esta filosofía de cifrado es la comunicación. El problema de intercambiar la llave secreta que se va a utilizar por los participantes es muy complejo y pasaron largos siglos sin poder encontrar una solución. Fue hasta la aparición del trabajo de Whitfield Diffie y Martin Hellman en la década de los 80 que surgió una nueva filosofía de cifrado, llamada asimétrica o de llave pública, que resolvía el problema del intercambio. Con los esquemas asimétricos cada quien utiliza distintas llaves y la comunicación de ellas no necesita de secretos.

Pero surge un nuevo gran problema, la eficiencia. Ninguno de los esquema de llave pública propuestos es eficiente, más aún son bastante costosos comparados con los esquemas simétricos, computacionalmente hablando. Dicha ineficiencia impide el uso de estos esquemas en aplicaciones reales y cotidianas. Así pues, tanto la criptografía de llave pública como de llave privada tienen distintas dificultades de implementación. A pesar de los grandes avances logrados en ambas áreas, aún siguen siendo dificiles de usar, principalmente los de llave pública por su gran costo computacional.

Por otra parte, nótese que un esquema híbrido es en sí mismo un esquema de llave pública, cuyas llaves pública y privada son las mismas que en el esquema de encapsulado de llave, y nótese también que para cifrar mensajes, la mayor parte del trabajo en el cifrado/descifrado es hecha por el esquema de llave secreta, debido a su mayor eficiencia, mientras que el ineficiente esquema de llave pública es usado sólo para cifrar/descifrar el valor pequeño de la llave compartida.

La mejor aportación del cifrado híbrido radica en que resuelve en gran medida los dos grandes problemas de la criptografía de los que hablamos, a saber, la distribución de llaves para esquemas simétricos y la eficiencia para esquemas asimétricos. Al combinar los principios básicos de estas dos filosofías de cifrado se soluciona mutuamente el principal problema de cada una. Al crear nuevos esquemas híbridos que nos proveen la facilidad de comunicación de los esquemas asimétricos y con una eficiencia comparable con la de los esquemas simétricos, estamos avanzando en materia de seguridad y tendremos mayor probabilidad de proteger la información, sobre todo con el gran incremento en capacidad de procesamiento con que cuentan los medios digitales.

3.2. Esquemas existentes

Durante los años 90'surgieron algunos esquemas de cifrado que establecieron las bases para lo que ahora conocemos como cifrado híbrido, pero fue hasta el año 2001 cuando Victor Shoup [44] estableció formalmente los conceptos de KEM y DEM en los que se basa esta filosofía.

Durante la investigación realizada en el área de la criptografía híbrida se ha encontrado una cantidad considerable de trabajos publicados, pero en realidad son pocas las propues-

tas para nuevos esquemas de cifrado híbrido, comparadas con las propuestas de cifradores simétricos y asimétricos. A continuación se van a presentar algunas propuestas de esquemas de cifrado distintas al esquema base DHIES, el cual será ampliamente detallado en el siguiente capítulo. Los primeros cinco son esquemas muy importantes que permiten la comparación con nuestro trabajo, los últimos son cifradores basados en identidad que tienen una estructura distinta y no podemos hacer una comparación "justa".

3.2.1. Esquemas basados en RSA

Ambos esquemas presentados en esta sección necesitan de los parámetros del algoritmo **RSA**. El primer paso es elegir dos números primos p y q que cubran las necesidades de seguridad (longitud). Después construimos la llave pública con un número entero compuesto n tal que $n = p \times q$ y con un exponente público e, el cual de cumplir con $MCD(e, \phi(n)) = 1$. La llave privada será el exponente secreto d donde $d = e^{-1} \bmod \phi(n)$.

El primer esquema de cifrado híbrido fue presentado por Shoup [44] en el 2001, aunque su descripción completa se presenta en [15] con el nombre de **RSA-KEM+DEM1**, el cual se muestra claramente en la tabla 3.1. Es un esquema basado en RSA que se compone de un mecanismo de cifrado de llave *KEM* y que fue convertido en un esquema de cifrado híbrido mediante la incorporación del mecanismo de cifrado de datos *DEM1*. Las principales ventajas que presenta esta propuesta de cifrador son su simplicidad y su eficiencia.

En los algoritmos presentados aquí, KDF (función de derivación de llaves) es una función diseñada para la obtención de la llave simétrica; $MAC_{K'}$ es una función para autenticar el mensaje; n, e y d son respectivamente el módulo, el exponente público y el exponente privado de RSA, generados previamente; SE_K y SD_K corresponden respectivamente al cifrador y descifrador del esquema simétrico usando la llave K.

Cifrado	**Descifrado**
`entrada` (m)	`entrada` (y, c, t)
$r \xleftarrow{\$} \{0..n-1\}$	$r \leftarrow y^d \bmod n$
$y \leftarrow r^e \bmod n$	$K \| K' \leftarrow \mathrm{KDF}(r)$
$K \| K' \leftarrow \mathrm{KDF}(r)$	rechazar si $t \neq MAC_{K'}(c)$
$c \leftarrow SE_K(m)$	$m \leftarrow SD_K(c)$
$t \leftarrow MAC_{K'}(c)$	`salida` (m)
`salida` (y, c, t)	

Tabla 3.1: Esquema de cifrado híbrido **RSA-KEM+DEM1**.

El segundo esquema, conocido como **RSA-REACT**, propuesto en el año 2001 por Okamoto y Pointcheval [31], surgió gracias al criptosistema de llave pública llamado RSA-OAEP [4]. Debido a las inconsistencias en la demostraciones de seguridad del esquema OAEP (*Optimal Asymmetric Encryption Padding*), principalmente a que estaban incompletas, los autores decidieron complementar RSA con un esquema distinto, **REACT**

(*Rapid Enhanced-Security Asymmetric Cryptosystem Transform*) [30]. REACT es una transformación creada por los mismos autores que puede ser aplicada a algún criptosistema débilmente seguro, ésta es óptima desde el punto de vista computacional y de seguridad, y fue elegida por sus ventajas sobre OAEP. En la tabla 3.2 podemos observar los métodos de cifrado/descifrado de RSA-REACT, donde KDF es la función para la obtención de la llave simétrica; H es una función hash para autenticar el mensaje; n, e y d son el módulo, el exponente público y el exponente privado de RSA; SE_K y SD_K corresponden respectivamente al cifrador y descifrador del esquema simétrico.

Se sabe que RSA-OAEP ofrece un nivel de seguridad de apenas 2^{40} cuando se utiliza un módulo de 1024 bits. Muchas alternativas fueron propuestas pero debido a la eficiencia que presenta, RSA-REACT parece ser la mejor pues aumenta el nivel de seguridad a 2^{80} con el mismo tamaño del módulo [31]. Además, ha sido comprobado que una construcción completa del esquema junto con algún cifrador simétrico es segura, lo que garantiza la seguridad de la comunicación completa.

Cifrado	**Descifrado**
`entrada` (m)	`entrada` (y, c, t)
$r \xleftarrow{\$} \{0..n-1\}$	$r \leftarrow y^d \bmod n$
$y \leftarrow r^e \bmod n$	$K \leftarrow \mathrm{KDF}(r)$
$K \leftarrow \mathrm{KDF}(r)$	$m \leftarrow SD_K(c)$
$c \leftarrow SE_K(m)$	rechazar si $t \neq H(r, y, m, c)$
$t \leftarrow H(r, y, m, c)$	`salida` (m)
`salida` (y, c, t)	

Tabla 3.2: Esquema de cifrado híbrido **RSA-REACT**.

Una gran ventaja con la que cuentan ambos esquemas de cifrado anteriores, a diferencia del próximo, es que ofrecen un código de autenticación de mensaje **MAC**. Ambos esquemas incorporan funciones para verificar la integridad del mensaje y aunque esto significa cierto sacrificio de tiempo de cómputo y de recursos, ya que el mensaje cifrado es más extenso, los beneficios son importantes.

Desde el punto de vista de la eficiencia, el esquema RSA-KEM+DEM1 tiene ventaja sobre RSA-REACT, mientras que desde el punto de vista de la seguridad RSA-REACT está por delante debido a que este sí cuenta con una prueba de seguridad completa mientras que RSA-KEM+DEM1 además de no tenerla, se han mostrado deficiencias en su seguridad [15]. Así vemos que dependiendo de las necesidades de la aplicación debe elegirse sacrificar la seguridad o la eficiencia.

3.2.2. Cramer-Shoup

Este es un esquema de cifrado híbrido presentado por R. Cramer y V. Shoup en [8]. En dicho trabajo, los autores presenta un par de esquemas de cifrado, uno de llave pública

llamado CS1 y otro de tipo híbrido llamado CS3, y algunas variantes de los mismos junto con las demostraciones de seguridad contra ataques CCA2. En relación a este proyecto, el artículo [8] tiene dos grandes aportaciones: primero el concepto del *mecanismo de encapsulado de llave* (KEM) referente a la construcción de cifradores híbridos, y segundo la formulación de un teorema para demostrar la seguridad de un cifrador híbrido, en el cual basamos la prueba de seguridad de nuestro esquema. Otra característica importante del trabajo de Cramer y Shoup es que, a diferencia de los esquemas publicados anteriormente, los esquemas presentados además de ser de los más prácticos cuentan también con una demostración de seguridad.

En este proyecto únicamente nos interesa analizar el esquema híbrido propuesto. En [8] se presenta un capítulo especificamente para tratar las bases y definiciones sobre cifrado híbrido, y es donde se establece el concepto de KEM. Se presenta un KEM llamado **CS3**, el cual se basa en la construcción del esquema de llave pública del mismo artículo. Adicionalmente se presentan dos variaciones, *CS3a* y *CS3b*, con sus respectivas demostraciones de seguridad, bajo suposiciones estándar y bajo el modelo de oráculo aleatorio.

En las tablas 3.3 y 3.4 presentamos los algoritmos correspondientes al KEM **CS3b**, el cual es considerado por sus creadores como el mejor de los tres esquemas propuestos. Dentro de las ventajas que ofrece sobre los demás esquemas está el hecho de ser el KEM más eficiente y que permite cifrar mensajes de longitud arbitraria. Desde el punto de vista de la seguridad, se demostró que al menos tiene el mismo nivel de seguridad que el esquema conocido como *ElGamal con hash*, presentado en el mismo artículo, además de que su seguridad contra ataques de tipo CCA2 puede ser demostrada en el modelo de oráculo aleatorio bajo la suposición CDH.

Generación de llaves
$hk \xleftarrow{\$} HF.KeySpace$
$dk \xleftarrow{\$} KDF.KeySpace$
$w \xleftarrow{\$} \mathbb{Z}_q^*$
$x, y, z \xleftarrow{\$} \mathbb{Z}_q$
$\hat{g} \leftarrow g^w$
$e \leftarrow g^x$
$f \leftarrow g^y$
$h \leftarrow g^z$
`llave pública PK`$= (\Gamma, hk, dk, \hat{g}, e, f, h)$
`llave privada SK`$= (\Gamma, hk, dk, x, y, z)$

Tabla 3.3: Generación de llaves para **CS3b**.

Para los algoritmos del encapsulado de llave por medio de CS3b, necesitamos tomar en cuenta un grupo descrito por $\Gamma(\hat{G}, G, g, q)$, donde $\hat{G}$ es un grupo abeliano, G es un

Cifrado	**Descifrado**
`entrada` (PK)	`entrada` (SK, ψ)
$u \xleftarrow{\$} \mathbb{Z}_q$	separar ψ en $(a, \hat{a}, d)$; $\perp$ en caso de error
$a \leftarrow g^u$	probar si $a \in G$; $\perp$ sino
$\hat{a} \leftarrow \hat{g}^u$	$v \leftarrow HF_{hk}(a, \hat{a})$
$b \leftarrow h^u$	probar si $\hat{a} = a^w$ y si $d = a^{x+yv}$; $\perp$ sino
$K \leftarrow KDF_{dk}(a, b)$	$b \leftarrow a^z$
$v \leftarrow HF_{hk}(a, \hat{a})$	$K \leftarrow KDF_{dk}(a, b)$
$d \leftarrow e^u f^{uv}$	`salida` (K)
`salida` $(K,\ \psi = (a, \hat{a}, d))$	

Tabla 3.4: Mecanismo de encapsulado de llave **CS3b**.

subgrupo de $\hat{G}$ de orden primo q, y g es un elemento generador de G. También se especifican dos espacios de llaves, $HF.KeySpace$ para la función hash y $KDF.KeySpace$ para la función de derivación de llaves. El algoritmo de descifrado hace uso del símbolo $\perp$ para identificar los eventos de rechazo y paro de la ejecución.

La construcción del esquema de cifrado híbrido especificada en el documento, parte del hecho que necesitamos un cifrador simétrico con una única restricción, la longitud de su llave debe ser compatible con la longitud de la llave producida por el KEM. Primero suponemos que ese esquema simétrico es inseguro contra CCA2, entonces el esquema híbrido también lo será. Para evitar este problema, se debe añadir un código de autenticación de mensajes (MAC), que sepamos sea seguro, al esquema de cifrado simétrico, esto provoca que el nuevo esquema, cifrado simérico y MAC, sea seguro contra ataques CCA2.

Una vez que tengamos nuestro esquema de cifrado de llave secreta (DEM), se une con el KEM propuesto CS3b para construir un esquema de cifrado híbrido seguro contra ataques de texto cifrado elegido.

3.2.3. Kurosawa-Desmedt

Como ya mencionamos, Victor Shoup junto con Ronald Cramer fueron quienes formalizaron la definición de KEM en [8], pero además establecieron un teorema que indica que un KEM debe ser resistente contra ataques de texto cifrado elegido adaptable para que al construir un cifrador híbrido, este nivel de seguridad permaneciera. En el trabajo de Kaoru Kurosawa e Yvo Desmedt [22] los autores muestran que esta regla no es necesaria. Ellos proponen un nuevo paradigma para el cifrado híbrido, el cual se supone más eficiente que el propuesto por Shoup, descrito en el próximo apartado, donde el KEM usado no necesariamente debe ser seguro en este sentido y sin embargo el esquema completo lo será.

La principal contribución hecha en el área del cifrado híbrido por este esquema es de gran importancia. Primero vemos que el número de exponenciaciones requeridas tanto

en el cifrado como el descifrado es menor al de los esquemas anteriores, lo que lo hace más eficiente, principalmente debido a que podemos utilizar un KEM *débil* y corregirlo con el cifrador híbrido. Con ésto, se abre una puerta para crear más esquemas de este tipo que sean más eficientes y por lo tanto su viabilidad de uso se incremente notablemente.

Para los algoritmos presentados en las tablas 3.5 y 3.6, todos los elementos se consideran dentro del grupo G de orden Q, donde Q es un número primo grande y el grupo que forma es de tipo abeliano. Dentro de los algoritmos de cifrado y descifrado, TCR es una función hash resistente a colisiones indexada por el valor k de longitud máxima γ; $H : G \to \{0,1\}^\lambda$ es una función hash donde λ es la longitud de llave del esquema simétrico; $SKE.C$ y $SKE.D$ corresponden respectivamente al cifrador y descifrador del esquema simétrico; K será la llave simétrica.

Generación de llaves
$g_1,\ g_2 \leftarrow$ generadores de G
$x_1,\ x_2,\ y_1,\ y_2 \xleftarrow{\$} \mathbb{Z}_Q$
$c \leftarrow g_1^{x_1} g_2^{x_2}$
$d \leftarrow g_1^{y_1} g_2^{y_2}$
$k \xleftarrow{\$} \{0,1\}^\gamma$
`llave pública` (g_1, g_2, c, d, k)
`llave privada` (x_1, x_2, y_1, y_2)

Tabla 3.5: Generación de llaves para **Kurosawa-Desmedt**.

Cifrado	**Descifrado**
`entrada` (m)	
$r \xleftarrow{\$} \mathbb{Z}_Q$	`entrada` (u_1, u_2, c)
$u_1 \leftarrow g_1^r$	$\alpha \leftarrow \mathrm{TCR}(k; u_1, u_2)$
$u_2 \leftarrow g_2^r$	$v \leftarrow u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha}$
$\alpha \leftarrow \mathrm{TCR}(k; u_1, u_2)$	$K \leftarrow H(v)$
$v \leftarrow c^r d^{r\alpha}$	$m \leftarrow SKE.D(1^\lambda, K, c)$
$K \leftarrow H(v)$	`salida` (m)
$c \leftarrow SKE.C(1^\lambda, K, m)$	
`salida` (u_1, u_2, c)	

Tabla 3.6: Esquema de cifrado híbrido **Kurosawa-Desmedt**.

A este esquema, presentado en las tablas 3.5 y 3.6, al igual que ocurre con otros, se le han hecho distintas modificaciones para tratar de buscar fallas de seguridad o de incrementar su eficiencia. Algunos de los trabajos más relevantes sobre este modelo son los realizados por R. Gennaro y V. Shoup [12] donde gracias a nuevas pruebas de seguridad se muestra que puede ser incorporado algún código de autenticación de mensajes seguro, o el trabajo

de L.T. Phong y W. Ogata [32] donde se optimizan las exponenciaciones requeridas en el descifrado además de añadir MAC y de aumentar la velocidad de la generación de llaves.

También fue presentado en 2007 el trabajo de D. Hofheinz y E. Kiltz [17], ésta es la más reciente propuesta de cifrado híbrido y está basada en el esquema de Kurosawa-Desmedt, también es conocido como *Dual Kurosawa-Desmedt* (Dual-KD). Este nuevo paradigma propone la noción de seguridad contra *ataques de texto cifrado elegigo restringido* para algoritmos KEM. El nivel de seguridad que se ofrece contra este nuevo tipo de ataque es menor al del ofrecido contra el ataque mencionado en los esquemas anteriores, aunque aún no ha sido comprobado que este nivel sea suficiente para que el cifrado híbrido sea seguro.

En la tabla 3.7 se presentan los algoritmos de este cifrador. El encapsulado de llave $KEM = (G, C, D)$ se compone de un generador de llaves, un cifrador y un descifrador asimétricos. El esquema de cifrado simétrico con autenticación $AE = (C, D)$ se compone de un cifrador $AE.C$ y un descifrador $AE.D$. pk y sk son las llaves pública y privada del KEM. K y C son la llave simétrica y su correspondiente cifra. El parámetro k se refiere a la longitud de llave, que debe ser compatible entre los esquemas utilizados.

Generación de llaves	**Cifrado**	**Descifrado**
`entrada` (1^k)	`entrada` (pk, M)	`entrada` (C, χ)
$(pk, sk) \leftarrow KEM.G(1^k)$	$(K, C) \leftarrow KEM.C(pk)$	$K \leftarrow KEM.D(sk, C)$
`salida` (pk, sk)	$\chi \leftarrow AE.C(K, M)$	$M \leftarrow AE.D(K, \chi)$
	`salida` (C, χ)	`salida` (M) o "invalido"

Tabla 3.7: Esquema de cifrado híbrido **Dual-KD**.

Dentro del mismo artículo se incluyen algunos algoritmos que presentan variantes para la construcción del KEM, por tal motivo aquí únicamente se hace referencia a la generación de llaves $KEM.G$ y será decisión del usuario que variante utilizar. Dentro de esas variantes están una que incluye rechazo implícito y una que no requiere funciones hash.

Las tres principales aportaciones de este nuevo paradigma son la de proponer un nivel de seguridad distinto a los anteriores lo que al parecer mejora el desempeño, también la de basar su seguridad en un problema distinto al de Diffie-Hellman y más débil, y por último, la de proponer un cifrador simétrico con autenticación.

A continuación se describe un esquema muy distinto a los anteriores.

3.2.4. Cifrado basado en identidad

En esta sección se define un esquema de cifrado híbrido muy distinto a los anteriores, el cual se basa en una filisofía conocida como **cifrado basado en identidad** (IBE) surgida a partir de 1984 con el trabajo de A. Shamir [42]. En ese trabajo, Shamir describe un

nuevo tipo de esquemas criptográficos, usados tanto cifrado como para firmas, que permite una comunicación entre pares de usuarios sin la necesidad de intercambiar ninguna llave, de tener repositorios de llaves, ni necesitar servicios de una tercera entidad.

Los esquemas IBE, como se muestra en la figura 3.1, se basan en un criptosistema de llave pública pero, en lugar de generar un par aleatorio de llaves pública-privada y publicar la primera de ellas, el usuario elige un nombre y dirección en la red como la llave pública, por ejemplo, una cadena (con datos personales) que identifique al usuario de forma única y que pueda ser obtenida de forma pública por cualquiera. Por otra parte, la correspondiente llave privada será calculada por un **centro de generación de llaves** (PKG) y puede ser entregada al usuario en forma de una **tarjeta inteligente**, la cual contiene toda la información y programas necesarios para que el usuario pueda cifrar/descifrar y firmar/verificar todos los mensajes que son enviados y recibidos.

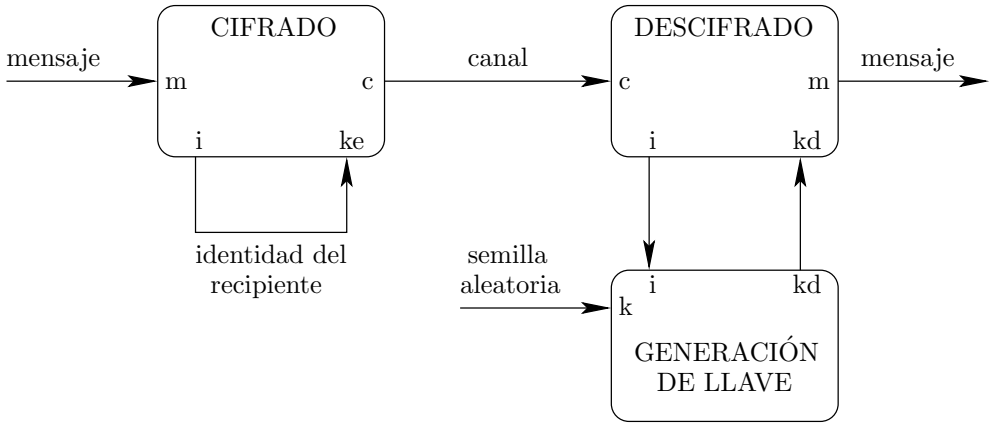

Figura 3.1: Esquema de cifrado basado en identidad (IBE).

Posteriormente en el año 2002, aparece un nuevo concepto de cifrado basado en identidad. En [18] se define el concepto de **cifrado jerárquico basado en identidad** (HIBE), cuya principal aportación fue la de establecer niveles para los participantes de la red y así reducir la carga de trabajo de los centros generadores de llaves. El PKG delega la responsabilidad a participantes de niveles altos para que estos generen las nuevas llaves privadas para los niveles inferiores.

En este apartado vamos e definir únicamente el primero de los esquemas HIBE, el cual utiliza las mismas ideas de los cifradores híbridos descritos anteriormente en este capítulo. El cifrador HIBE híbrido propuesto por Palash Sarkar y Sanjit Chatterjee en [40], es una extensión del esquema propuesto por los mismos autores en [41], y la principal modificación es que la versión híbrida utiliza un cifrador con autenticación simétrico para verificar la validez del texto cifrado, idea tomada del esquema de cifrado híbrido de Kurosawa-Desmedt (3.2.3).

Las principales características de los esquemas de cifrado jerárquico basados en identidad son que basan su trabajo en una operación conocida como *mapeo criptográfico bilineal* y que se componen de cuatro algoritmos: configuración, generación de llaves, cifrado y descifrado. La definición del mapeo bilineal e nos permite efectuar una de las operaciónes más complejas, y a su vez de las más llamativas, de la criptografía, los **emparejamientos**. En este proyecto no ahondaremos más en el tema de (H)IBE, principalmente debido a la gran cantidad de conceptos que habría que definir y a que no tienen gran relación con las características del tema central del proyecto, si el lector desea saber más sobre el tema puede consultar los artículos mencionados en esta sección. En lo posterior, definiremos el funcionamiento del esquema HIBE híbrido presentado en [40], mencionando únicamente los conceptos escenciales para comprender sus algoritmos.

Los dos parámetros necesarios para cualquier HIBE son: la profundidad máxima h del esquema y la identidad de los usuarios definida por $v = (v_1, \ldots, v_j)$, donde $1 \leq j \leq h$. Además necesitamos el mapeo bilineal $e : G_1 \times G_1 \to G_2$, donde G_1, G_2 son respectivamente un grupo cíclico aditivo y un grupo cíclico multiplicativo, ambos de orden primo p y $G_1 = \langle P \rangle$.

Adicionalmente, este esquema hace uso de tres componentes más, un cifrador simétrico con autenticación AE, una función de derivación de llaves KDF y una función hash universal H_s. AE se compone de su respectivo algoritmo de cifrado $AE.Enc$, que recibe un mensaje M y produce el texto cifrado C y la etiqueta de verificación tag, y su respectivo algoritmo de descifrado $AE.Dec$, que recibe un C y un tag para recuperar M; ambos algoritmos hacen uso de una llave privada dk y de un vector de inicialización IV. A su vez, la función KDF recibe una entrada K y retorna el par de valores (IV, dk) usados por el esquema AE.

En la tabla 3.8 se pueden observar los algoritmos de **configuración** y de **generación de llaves** de este cifrador. En el algoritmo de configuración se generan los *parámetros públicos* que serán utilizados por los otros tres algoritmos, la *llave secreta maestra* (del PKG) y se selecciona la función H_s que se va a utilizar en el cifrado y descifrado. En el generador de llaves se recibe una identidad v y se genera la *llave secreta* correspondiente a ella, definida por $d_v = (d_0, d_1, \ldots, d_j)$.

Otra de las ideas novedosas de este esquema es la incorporación de la longitud de la identidad v en el texto cifrado. Para tal efecto, se utiliza la función V_k. Considere $v = (v_1, \ldots, v_j)$ donde cada v_i es una cadena de n/j bits (j divide a n), n es la longitud en bits de la identidad y j el número de bloques en los que se divide, y $v_i \in \mathbb{Z}_{2^{n/j}}$. Para $1 \leq k \leq h$ definimos la función

$$V_k(v) = U'_k + \sum_{i=1}^{j} v_i U_i$$

Configuración	**Generación de llaves**
$\alpha \xleftarrow{\$} \mathbb{Z}_p$	`entrada` (v)
$P_1 = \alpha P$	$r_1, \ldots, r_j \xleftarrow{\$} \mathbb{Z}_p$
$P_2, U'_1, \ldots, U'_h, U_1, \ldots, U_l \xleftarrow{\$} G_1$	$d_0 = \alpha P_2 + \sum_{k=1}^{j} r_k V_k(v_k)$
$W \xleftarrow{\$} G_1$	$d_k = r_k P$ para $k = 1, \ldots, j$
`Parámetros públicos`	`salida` (d_v)
$(P, P_1, P_2, U'_1, \ldots, U'_h, U_1, \ldots, U_l, W)$	
`Llave maestra secreta` (αP_2)	

Tabla 3.8: Configuración y generación de llaves para **HIBE híbrido**.

Cifrado	**Descifrado**
`entrada` (v, M)	`entrada` (v, φ, d_v)
$t \xleftarrow{\$} \mathbb{Z}_p$	$\gamma = H_s(j, C_1)$
$C_1 = tP, B_1 = tV_1(v_1), \ldots, B_j = tV_j(v_j)$	$W_\gamma = W + \gamma P_1$
$K = e(P_1, P_2)^t$	Si $e(C_1, W_\gamma) \neq e(P, C_2)$ `salida` $(\perp)$
$(IV, dk) = KDF(K)$	$K = e(d_0, C_1) \times \prod_{k=1}^{j} e(B_k, -d_k)$
$(cpr, tag) = AE.Enc_{dk}(IV, M)$	$(IV, dk) = KDF(K)$
$\gamma = H_s(j, C_1)$	$M = AE.Dec_{dk}(IV, C, tag)$
$W_\gamma = W + \gamma P_1$	`salida` M o $\perp$
$C_2 = tW_\gamma$	
`salida` $\varphi = (C_1, C_2, B_1, \ldots, B_j, cpr, tag)$	

Tabla 3.9: Esquema de cifrado **HIBE híbrido**.

En los algoritmos presentados en la tabla 3.9 podemos observar como obtenemos un KEM con la combinación entre un esquema de cifrado basado en identidad y un esquema de cifrado de llave pública, y con él generamos la llave secreta para el cifrado de datos (DEM).

CAPÍTULO 4

DHIES

En este capítulo se presenta una descripción detallada del funcionamiento del esquema de cifrado DHIES, el esquema base para el proyecto, y se establece el nivel de seguridad que ofrece. También se presentan las modificaciones realizadas al esquema para el mejoramiento del mismo, así como una prueba de seguridad para el esquema propuesto.

4.1. Descripción del esquema

DHIES (*Diffie-Hellman Integrated Encryption Scheme*) es un esquema de cifrado basado en Diffie-Hellman, conocido anteriormente como DHES o DHAES. Este esquema fue presentado por M. Abdalla, M. Bellare, y P. Rogaway [1] en el 2001, junto con un análisis completo de seguridad, pero apareció por primera vez en 1997 con el nombre de DLAES [5]. DHIES es un esquema híbrido bastante atractivo que cuenta con una eficiencia comparable a la del esquema ElGamal (2.5.1) pero con un mejor nivel de seguridad. Debido principalmente a estas características, es que se decidió elegirlo como el esquema base para el proyecto. Además si se compara con los esquemas presentados en el capítulo 3, puede verse la sencillez de éste.

En las figuras 4.1 y 4.2 se puede observar respectivamente los procesos de cifrado y descifrado de este esquema, el cual se compone de tres grandes bloques. El primero de ellos es un bloque KEM que necesita un par de llaves, debido a que se basa en un esquema de llave pública, y sirve para generar de forma segura una cadena de bits de la cual se obtendran dos llaves secretas compartidas, $macKey$ y $encKey$, cada una de las cuales será utilizada en alguno de los dos bloques siguientes. Este bloque esta basado en la idea de Diffie y Hellman [9] y adicionalmente se incorpora una función hash. En segundo lugar se encuentra un bloque DEM, basado en un esquema de cifrado simétrico, usado para procesar los mensajes que se desean proteger. Y por último tenemos un algoritmo de autenticación de mensajes (MAC) para brindar protección contra pérdida de datos o modificación del mensaje original.

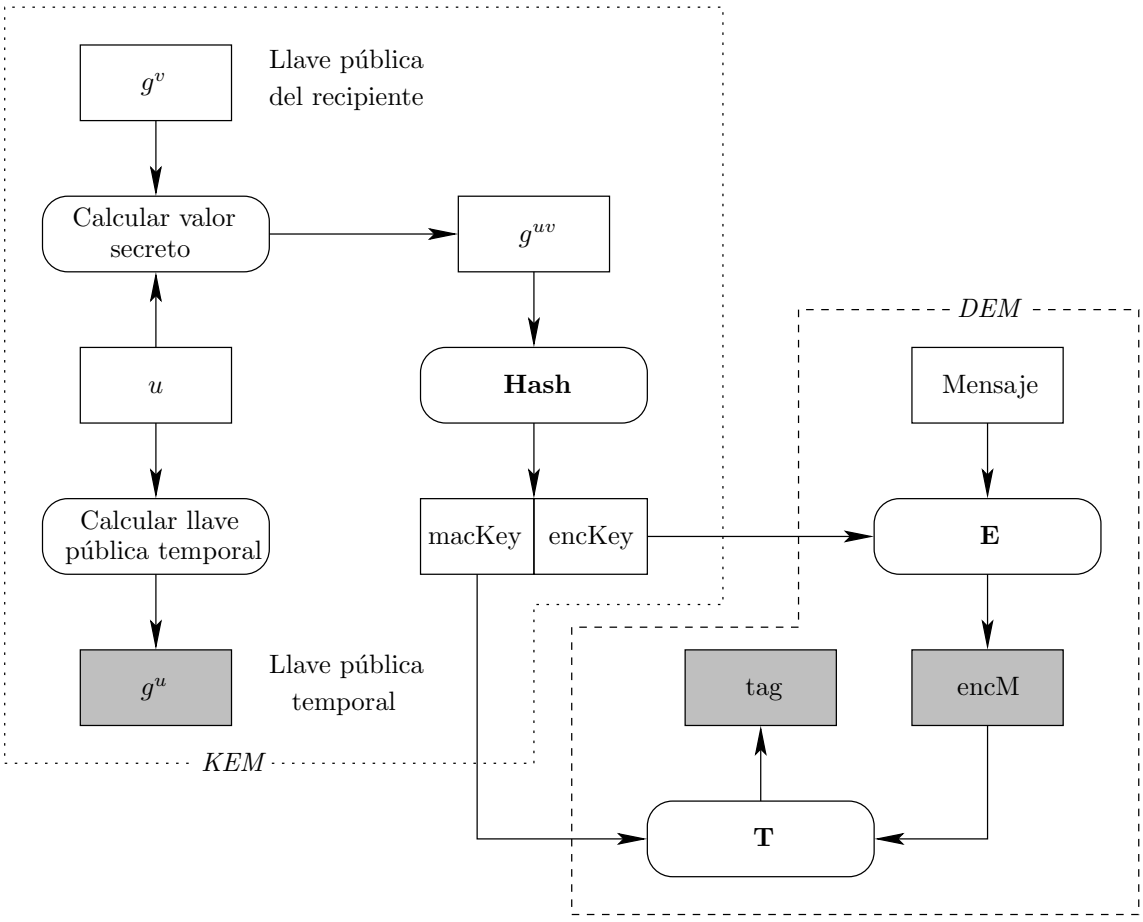

Figura 4.1: Esquema de cifrado **DHIES**.

Como podemos ver en los algoritmos presentados en las tablas 4.1 y 4.2, este esquema está formado por distintas funciones básicas como son las operaciones aritméticas en el grupo de trabajo G, las ya mencionadas funciones hash (2.5.2) y códigos de autenticación de mensajes (2.5.3), y las primitivas de algún esquema de cifrado simétrico.

Para las operaciones necesarias tomamos un grupo G de orden q, donde sus elementos tienen una longitud de $gLen$ bits. Tomamos también un esquema de cifrado simétrico $SYM = [E, D]$ cuya longitud de llave es $eLen$ bits, y un código de autenticación de mensajes $MAC = [T, V]$ con longitud de llave $mLen$ bits y longitud de etiqueta $tLen$. Es necesario considerar una función hash $H : \{0,1\}^{gLen} \rightarrow \{0,1\}^{mLen+eLen}$ que toma elementos del grupo G y produce un valor que será partido en dos para tener una llave de cifrado y una llave de MAC. Con las primitivas anteriores definimos el esquema $DHIES = [G, SYM, MAC, H]$.

Para la generación de llaves de la tabla 4.1 se considera un grupo de elementos G del

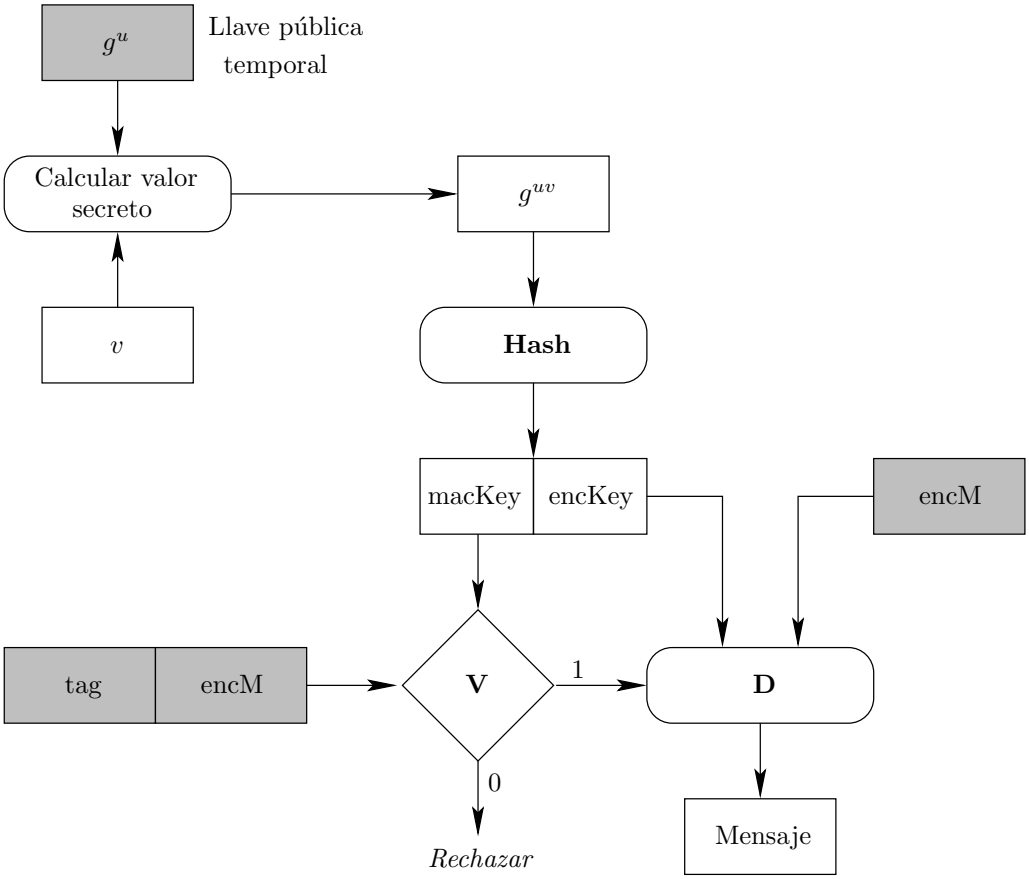

Figura 4.2: Esquema de cifrado **DHIES**.

cual se selecciona un generador g, el cual será del dominio público. Los valores pk y sk son respectivamente las llaves pública y privada dentro del esquema Diffie-Hellman.

Dentro de los algoritmos de cifrado y descifrado, $H : \{0,1\}^{gLen} \rightarrow \{0,1\}^{mLen+eLen}$ es una función hash donde $gLen$ es la longitud de los elementos del grupo G, $mLen$ es la longitud de la llave de MAC, y $eLen$ es la longitud de la llave simétrica. (T, V) son los componentes MAC para creación y verificación de la etiqueta de autenticación respectivamente, donde el algoritmo V regresa 1 en caso de que la etiqueta se válida o 0 en caso contrario. Y (E, D) son los componentes de cifrado y descifrado para el esquema simétrico.

Generación de llaves
$v \xleftarrow{\$} \{1, ..., \|G\|\}$
$pk \leftarrow g^v$
$sk \leftarrow v$
`salida` (pk, sk)

Tabla 4.1: Generación de llaves para **DHIES**.

Al analizar el esquema, podemos observar como se trata de una simple, pero ventajosa, extensión del esquema ElGamal. Como ya vimos, ElGamal es un esquema sencillo y eficiente que introdujo una idea novedosa para el cifrado pero tiene dos grandes desventajas. Por un lado, la llave debe tener al menos la misma longitud del mensaje a cifrar, y por otra parte, no provee ni seguridad semántica ni no-maleabilidad. Con DHIES se corrigen estos problemas. DHIES provee seguridad semántica además de ser indistinguible en el sentido CCA2, es decir, es no maleable bajo la suposición ODH; también con DHIES es posible cifrar mensajes sin restricción de tamaño haciendo uso de llaves de unos cientos de bits y manteniendo una eficiencia comparable a la de ElGamal.

Cifrado	**Descifrado**
`entrada` (pk, M)	
$u \leftarrow \{1, ..., \|G\|\}$	`entrada` (sk, EM)
$\chi \leftarrow pk^u$	$U \| encM \| tag \leftarrow EM$
$U \leftarrow g^u$	$\chi \leftarrow U^{sk}$
$hash \leftarrow H(\chi)$	$hash \leftarrow H(\chi)$
$macKey \leftarrow hash\,[1...mLen]$	$macKey \leftarrow hash\,[1...mLen]$
$encKey \leftarrow hash\,[mLen + 1...mLen + eLen]$	$encKey \leftarrow hash\,[mLen + 1...mLen + eLen]$
$encM \leftarrow E(encKey, M)$	si $V(macKey, encM, tag) = 0$ rechazar
$tag \leftarrow T(macKey, M)$	$M \leftarrow D(encKey, encM)$
$EM \leftarrow U \| encM \| tag$	`salida` (M)
`salida` (EM)	

Tabla 4.2: Esquema de cifrado híbrido **DHIES**.

4.2. Descripción de la seguridad

DHIES es un esquema basado en el problema de Diffie-Hellman que combina operaciones de grupos, un cifrador simétrico, una función hash y un código de autenticación de mensajes, todo de una forma sencilla que intuitivamente parece ser seguro pero a la vez parece difícil de probar, sobre todo bajo las suposiciones estándar definidas acerca de Diffie-Hellman.

Las pruebas presentadas en [1] sobre la seguridad de DHIES en el sentido IND-CPA e IND-CCA2 están realizadas sobre el modelo estándar y se define como adversario a un algoritmo A que ataca a DHIES. La idea general es construir nuevos algoritmos que funjan como adversarios y ataquen por separado a los bloques que componen DHIES para después obtener la ventaja de A en base a la ventaja de esos nuevos algoritmos. Los teoremas 4.1 y 4.2 presentados a continuación establecen la seguridad de DHIES en ambos sentidos.

Para la prueba IND-CPA se establece el teorema 4.1 y se definen dos posibles casos: uno donde la salida de la función hash H parece aleatoria en cuyo caso se construye un adversario que ataca al cifrador simétrico SYM, y el segundo donde la salida de H no

luce como una cadena aleatoria en cuyo caso se construye al adversario para atacar a la función hash; en la prueba no se considera a ambos como casos separados.

Teorema 4.1 ([1, Theorem 1])**.** *Considere a G como un grupo, a SYM como un esquema de cifrado simétrico, a MAC como un esquema de autenticación de mensajes, y a H como una función hash. Definimos a $DHIES$ como un esquema de cifrado asimétrico asociado a las primitivas mencionadas tal y como se definió en la sección anterior (4.1). Entonces para cualquier valor de complejidad de tiempo del adversario y cualquier longitud de bits del reto,*

$$\mathbf{Adv}_{DHIES}^{ind-cpa-fg} \leq 2 \cdot \mathbf{Adv}_{G,\ H}^{hdh} + \mathbf{Adv}_{SYM}^{ind-cpa-fg}$$

Para esta prueba se asume que SYM es seguro y que H es segura bajo las condiciones del problema de Diffie-Hellman.

Para la prueba de seguridad IND-CCA2 se consideran los mismos dos casos de la seguridad CPA para probar el teorema 4.2; en caso de que la salida de H no luzca una cadena aleatoria se considera un adversario que ataca la función hash, y en caso contrario se consideran dos adversarios que atacan al cifrador SYM y al código de autenticación de mensajes MAC. Se toma en cuenta que el atacante de la función hash actúa bajo el experimento sobre la suposición de ODH. En este teorema, $\mathbf{Adv}_{MAC}^{suf-cma}$ se refiere a la ventaja del adversario para falsificar las etiquetas generadas por el algoritmo MAC.

Teorema 4.2 ([1, Theorem 2])**.** *Considere a G como un grupo, a SYM como un esquema de cifrado simétrico y a MAC como un esquema de autenticación de mensajes. Definimos a $DHIES$ como un esquema de cifrado asimétrico asociado a las primitivas mencionadas tal y como se definió en la sección (4.1). Entonces para cualquier valor de complejidad de tiempo del adversario, cualquier número de consultas, cualquiera suma de las longitudes en bits de las consultas y cualquier longitud de bits del reto,*

$$\mathbf{Adv}_{DHIES}^{ind-cca2-fg} \leq \mathbf{Adv}_{SYM}^{ind-cpa-fg} + 2 \cdot \mathbf{Adv}_{G,\ H}^{odh} + 2 \cdot \mathbf{Adv}_{MAC}^{suf-cma}$$

Es muy importante remarcar una característica esencial con la cual debe contar el grupo G elegido. Para poder establecer la seguridad de DHIES es necesario que ese grupo G sea de orden primo denotado por $|G| = q$, es decir, que contenga un número primo de elementos, de lo contrario el esquema sería maleable. Esto se debe a que en caso de que $|G| = q$ no sea primo (como es el caso de $\mathbb{Z}_p^*$), la pareja g^{uv} y g^v no determina de manera única el valor g^u. En caso de usar un grupo de orden no-primo, pueden existir dos elementos distintos $u \neq u'$ tales que $g^{uv} = g^{u'v}$, por lo que ambos valores producirían distintos textos cifrados válidos para un mismo mensaje. Para corregir este problema, en caso de que q no sea primo es necesario concatenar g^u al valor obtenido de g^{uv} antes de procesar este último con la función hash H.

4.3. Discusión

Dentro de las razones por las cuales DHIES se eligió como base para este proyecto está el hecho de que resuelve los problemas que presentaba ElGamal –espacio limitado

de mensajes y no provee buena privacidad–. Con DHIES, podemos cifrar mensajes de cualquier tamaño y además provee privacidad y seguridad contra ataques de texto cifrado elegido adaptable, con prueba formal bajo el modelo estándar, y a pesar de los cambios realizados se mantiene la eficiencia de ElGamal.

DHIES es un esquema que se popularizó después de su primera publicación en 1998. Fue tal la importancia que se incluyo en propuestas de estándares como el P1363a para la IEEE y el ANSI X9.63. Lo que buscamos con este proyecto es mejorar el esquema DHIES de forma tal que podamos simplificarlo aún más, mejorar en lo posible su eficiencia y mantener el nivel de seguridad que se logra con él.

4.4. Cifrado con autenticación

Este es un tipo especial de esquemas de cifrado que se encargan al mismo tiempo de crear el texto cifrado y una etiqueta para la autenticación. Si lo vemos desde el punto de vista de la seguridad, son esquemas que proveen a la vez confidencialidad e integridad en las comunicaciones, que son los dos servicios con que esperamos que cuente el esquema. El interés en el estudio de estos esquemas surge a partir de las dificultades tanto de implementación como de seguridad que conlleva el utilizar dos algoritmos independientes dentro de una misma aplicación.

Algo característico de los esquemas de cifrado con autenticación es que ofrecen mayor resistencia contra ataques de texto cifrado. En un cifrado no autenticado el adversario puede consultar al oráculo de descifrado con cualquier cadena que él desee, sin embargo cuando se enfrenta a un esquema de cifrado autenticado el oráculo verifica la estructura de la consulta para cerciorarse de la validez de los textos cifrados, pudiendo retornar el texto en claro correspondiente o un mensaje que indique el rechazo de la consulta. Con el diseño y la implementación correctos se podría eliminar la utilidad del oráculo de descifrado para el adversario.

En la última década se ha desarrollado un buen número de esquemas de cifrado autenticado, generalmente se contruyen combinando un esquema de cifrado que al menos sea semánticamente seguro y un código de autenticación de mensajes que impida falsificaciones frente a ataques de mensaje elegido (suf-cma). Los autores Bellare y Namprempre definen en [3] tres tipos de esquemas, clasificados de acuerdo a su composición, y analizan la seguridad de cada composición de forma genérica. Los tres tipos de composición son “Cifrado y MAC”, “MAC luego cifrado” y “Cifrado luego MAC”; como resultado obtuvieron que *Cifrado luego MAC* es la mejor opción, puesto que provee seguridad del tipo IND-CCA además de garantizar la integridad de los textos cifrados, esto bajo la suposición de que el esquema MAC sea altamente resistente a falsificaciones.

Trabajos recientes se enfocaron en crear una clase de “modos de operación” que fueran

capaces de realizar cifrado con autenticación, algunos basados en modos de operación comunes y otros totalmente distintos. Dentro de los modos de operación diseñados para este tipo de cifrado se encuentran *OCB* (Offset Codebook), *CCM* (Counter-CBC-MAC), *IAPM* (Integrity Aware Parallelizable Mode), **GCM** (Galois/Counter Mode), **EAX**, entre otros.

4.5. Mejoras al esquema

Las modificaciones que se le hacen al esquema son sencillas pero muy significativas. Principalmente el trabajo se centra en sustituir los bloques del cifrado simétrico y del código de autenticación de mensajes por un bloque de **cifrado con autenticación**. En este proyecto se optó por agregar al esquema un modo de operación para el cifrador simétrico que permite el cifrado con autenticación en una sola ejecución. La ventaja principal de esta idea es que únicamente se necesita un algoritmo para ambos propósitos, lo que supone menor tiempo de implementación y de ejecución, comparado con el tiempo de cifrar más el tiempo de autenticar, además de que facilita la comprensión del procesamiento y la relación entre el texto cifrado y la etiqueta.

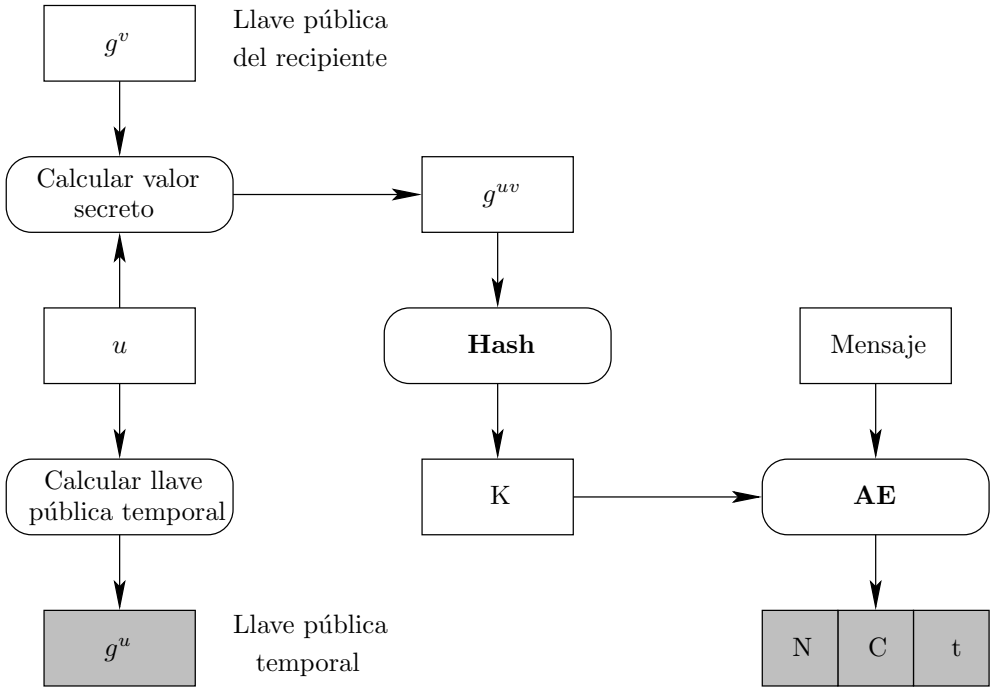

Figura 4.3: Esquema de cifrado híbrido propuesto **DHIES-M**.

En la figura 4.3 se presenta el diseño final del esquema de cifrado híbrido del proyecto. El resto de la sección presenta la construcción de este diseño final en sus dos secciones principales: encapsulado de llave y encapsulado de datos.

4.5.1. Diseño del KEM

El diseño del KEM se mantiene prácticamente igual a la versión original de esquema con la excepción de que la cadena de bits resultante de la función hash H no se dividirá sino que será tomada como el valor de una única llave secreta.

Generación de llaves $KeyG$	Cifrado $EncK$	Descifrado $DecK$
`entrada`(λ)	`entrada`(pk)	`entrada`(ϑ, sk)
$u \xleftarrow{\$} \mathbb{Z}_q^\star$	$v \xleftarrow{\$} \mathbb{Z}_q^*$	$\chi \leftarrow V^u$
$U \leftarrow g^u$	$V \leftarrow g^v$	$K \leftarrow H(\chi)$
$pk \Leftrightarrow U$	$\chi \leftarrow U^v$	`salida`(K)
$sk \Leftrightarrow u$	$K \leftarrow H(\chi)$	
`salida`(pk, sk)	$\vartheta \Leftrightarrow V$	
	`salida`(K, ϑ)	

Tabla 4.3: Mecanismo de encapsulado de llave para **DHIES-M**.

El esquema general del KEM se muestra en la tabla 4.3 de forma tal que permita entender las etapas de cifrado y ayude a describir una prueba de seguridad. Aquí se considera a G como un grupo cíclico de orden primo q, cuyos elementos son generados por un elemento g. También, tomamos a H como una función hash criptográfica. Podemos ver gracias a estos algoritmos como en realidad no se está cifrando un mensaje sino que únicamente se genera la llave y se "encapsula" mediante la función picadillo, de ahí el nombre de KEM.

4.5.2. Diseño del DEM

El DEM del esquema DHIES contiene dos algoritmos distintos, un algoritmo de cifrado simétrico **E** y un algoritmo MAC **T** (ver tabla 4.2).

Cifrado	**Descifrado**
`entrada` (pk, M)	`entrada` (sk, U, C, t)
1. $u \xleftarrow{\$} \mathbb{Z}_q^*$	1. $\chi \leftarrow U^{sk}$
2. $U \leftarrow g^u$	2. $K \leftarrow H(\chi)$
3. $\chi \leftarrow pk^u$	3. $(M, t^*) \leftarrow AE^{-1}(K, C)$
4. $K \leftarrow H(\chi)$	4. si $t \neq t^*$ `salida` ("MAL")
5. $(C, t) \leftarrow AE(K, M)$	si no `salida` (M)
`salida` (U, C, t)	

Tabla 4.4: Esquema de cifrado híbrido **DHIES-M**.

En la práctica estos dos algoritmos pueden ser fusionados para crear un único algoritmo llamado *cifrado con autenticación* (ver 4.4). Si AE denota un algoritmo de cifrado con autenticación entonces este recibe como entrada una llave K y un mensaje en claro M para

enregar un mensaje cifrado C y una etiqueta de verificación t. El algoritmo de descifrado, el cual denotaremos como AE^{-1}, toma como entrada un texto cifrado y una etiqueta y regresa un texto en claro o "MAL". Usando un esquema de cifrado con autenticación AE podemos escribir los algoritmos de cifrado y descifrado para el esquema DHIES-M como se muestra en la tabla 4.4.

4.6. Seguridad del esquema DHIES-M

En este apartado se discutirá acerca de la seguridad del esquema propuesto DHIES-M. En [8] fue mostrado que la seguridad de un esquema de cifrado híbrido **HPKE** puede ser dividida en la seguridad del KEM y la del DEM. Más específicamente, si $\mathbf{Adv}^{ind-cca}_{HPKE,\ A}$ denota la ventaja en el sentido CCA de un adversario A cuando trata de romper el HPKE, entonces

$$\mathbf{Adv}^{ind-cca}_{HPKE,\ A} \leq \mathbf{Adv}^{ind-cca}_{KEM,\ B} + \mathbf{Adv}^{ind-cca}_{DEM,\ C} \tag{4.1}$$

donde $\mathbf{Adv}^{ind-cca}_{KEM,\ B}$ y $\mathbf{Adv}^{ind-cca}_{DEM,\ C}$ corresponden a la ventaja en el sentido CCA de los adversarios B y C que tratan de romper el KEM y el DEM respectivamente.

Entonces en DHIES-M podemos claramente separar los componentes KEM y DEM. De la tabla 4.4, las lineas 1 a 4 del algoritmo de cifrado forman el componente KEM y el resto forman el DEM. Por consiguiente, de acuerdo a la ecuación 4.1 podemos decir que

$$\mathbf{Adv}^{ind-cca}_{DHIES-M,\ A} \leq \mathbf{Adv}^{ind-cca}_{DHIES-M-KEM} + \mathbf{Adv}^{ind-cca}_{DHIES-M-DEM} \tag{4.2}$$

Hay que hacer notar que el KEM del esquema DHIES-M es igual al esquema ElGamal donde el cifrado es procesado a través de una función hash H. Este es un esquema bastante conocido, llamado esquema "Hash-ElGamal" (ElGamal con Hash). También es sabido que el esquema de ElGamal con Hash es seguro contra ataques CCA si la suposición de Oráculo de Diffie-Hellman (ODH) se cumple para el grupo G y la función hash H es considerada como un oráculo aleatorio [1, 23, 7]. Por lo tanto $\mathbf{Adv}^{ind-cca}_{DHIES-M-KEM}$ es insignificante bajo el modelo de oráculo aleatorio. También en la literatura existen muchos esquemas de cifrado con autenticación que son seguros en el sentido CCA, lo que sugiere que también $\mathbf{Adv}^{ind-cca}_{DHIES-M-DEM}$ es insignificante.

Estos argumentos nos indican que $\mathbf{Adv}^{ind-cca}_{DHIES-M,\ A}$ es insignificante y de ahí que podamos considerar que DHIES-M es seguro contra ataques de tipo CCA si la suposición del oráculo de Diffie-Hellman se cumple en el grupo G y si la función hash H puede ser considerada como un oráculo aleatorio.

4.7. Comparativa

Como se ha establecido a lo largo de este capítulo, DHIES es un esquema que combina eficiencia y seguridad. Debido a la sencillez de las operaciones que forman al esquema es

que cuenta con tal eficiencia y además ha sido probada su seguridad en el sentido CCA2. Asumimos que el esquema modificado cuenta con una eficiencia incluso mayor a la del esquema original y, como ya se comprobó, cuenta con el mismo nivel de seguridad.

Comparando estos esquemas con los que fueron definidos en el capítulo 3 podemos darnos cuenta que son una excelente opción, tal vez la mejor. En la tabla 4.5 podemos comparar el costo en cuanto a las operaciones realizadas por cada esquema. Suponemos el mismo tipo de cifrado simétrico para todos los esquemas, de lo contrario la comparación no tendría mucho sentido. La comparación se realiza en lo referente a cuatro categorías: variables de estado, generación de llaves, cálculo de la llave simétrica y código de autenticación de mensajes. En esta comparación el costo de obtener un valor aleatorio se considera insignificante. Cabe señalar que esta tabla no contempla al esquema HIBE debido a que su estructura es muy distinta a los demás esquemas y por lo tanto no podemos compararlos bajo los mismo parámetros.

Esquema	**Estado**	**Llaves**	**Llave secreta**	**MAC**
DHIES	g	$sk = (u)$ $pk = (g^u)$	$mK \| eK = H(g^{uv})$	$MAC(mK, C)$
RSA - KEM+DEM1	r, n	$sk = (d)$ $pk = (e)$	$K \| K' = KDF(r)$	$MAC(K', C)$
RSA - REACT	r, n	$sk = (d)$ $pk = (e)$	$K = KDF(r)$	$H(r, y, M, C)$
Cramer - Shoup	Γ	$\hat{g} = g^w$; $e = g^x$ $f = g^y$; $h = g^z$ $sk = (\Gamma, x, y, z)$ $pk = (\Gamma, \hat{g}, e, f, h)$	$a = g^u$ $b = h^u$ $K = KDF_{dk}(a, b)$	$MAC(mk, C)$
Kurosawa - Desmedt	g_1, g_2	$c = g_1^{x_1} g_2^{x_2}$ $d = g_1^{y_1} g_2^{y_2}$ $sk = (x_1, x_2, y_1, y_2)$ $pk = (g_1, g_2, c, d, k)$	$\alpha = TCR(k; g_1^r, g_2^r)$ $K = H(c^r d^{r\alpha})$	sin
Dual - KD	g	$sk = (x, y, w)$ $pk = (g^x, g^y, g^w)$	$K = (g^w)^r$	implicito
DHIES-M (nuestro)	g	$sk = (u)$ $pk = (g^u)$	$K = H(g^{uv})$	implicito

Tabla 4.5: Comparativa de KEMs para cifradores híbridos.

Primero tomamos en cuenta las *variables de estado*, que son variables necesarias tanto para la generación de llaves como para las operaciones posteriores. En este caso el esquema está a la par con el de Cramer-Shoup (CS) pero tiene ventaja sobre Kurosawa-Desmedt (KD) que necesita un generador extra, es decir, una exponenciación más. Obtener un generador requiere el costo extra de encontrar un número primo de ciertas características, en cambio los esquemas basados en RSA no necesitan un generador pero requieren encontrar dos números primos, de mayor longitud que para los otros esquemas, y realizar

una multiplicación de los mismos.

En segundo lugar tenemos la *composición de las llaves* donde se nota una gran ventaja. El costo para el esquema modificado es de una exponenciación, al igual que DHIES, mientras que KD requiere cuatro exponenciaciones y dos multiplicaciones, CS y Dual-KD requieren de cuatro y tres exponenciaciones respectivamente, y para RSA se requiere una inversión, una operación que dependiendo del algoritmo utilizado puede ser muy costosa o más barata que la exponenciación.

En cuanto al *cálculo de la llave secreta* K se tiene ventaja sobre DHIES ya que la longitud de la cadena necesaria para este último es de aproximadamente el doble. Mientras que nuestro esquema requiere de una exponenciación y una ejecución de H, KD requiere 4 exponenciaciones, 2 multiplicaciones, y una ejecución de H y TCR. Dual-KD requiere de una sola exponenciación para obtener la llave K pero adicionalmente requiere de 3 exponenciaciones, 2 multiplicaciones y una ejecución de TCR para ocultar el valor. Claramente DHIES-M es la opción.

Por último tenemos el *código de autenticación*. Clara ventaja sobre KD que no cuenta con este servicio y, más eficiente que DHIES, RSA y CS pues el costo de cifrar y después autenticar se supone mayor al del cifrado con autenticación. Así que, como podemos observar, DHIES-M tiene ventajas de eficiencia sobre cada esquema en al menos una categoría, incluso sobre DHIES.

CAPÍTULO 5

Detalles de implementación

¿Por qué esta magnífica tecnología científica, que ahorra trabajo
y nos hace la vida más fácil, nos aporta tan poca felicidad?
La respuesta es ésta simplemente:
porque aún no hemos aprendido a usarla con tino.
Albert Einstein

Este capítulo pretende mostrar los detalles de la implementación realizada para el esquema definido anteriormente. En la primer sección se presentan datos acerca de las características de la implementación. Después se presenta la construcción final de los dos bloques principales del cifrador y posteriormente algunos detalles sobre la implementación de las funciones y operaciones básicas necesarias para la constitución de cada bloque. En la tercera sección se presentan detalles de implementación de otras operaciones necesarias para lograr el esquema de cifrado completo. Y por último se presentan datos sobre las pruebas realizadas y los resultados obtenidos con ellas.

5.1. Sobre la implementación

Comencemos por hablar de las características generales de la implementación. El código para esta aplicación está por completo escrito en lenguaje C++; realmente la implementación hubiera podido estar en C ya que no es programación orientada a objetos, pero la elección fue hecha por simplicidad de algunas cuestiones propias del lenguaje, como el manejo de ciclos, y por costumbre de este programador. El trabajo fue realizado sobre el sistema operativo Linux, debido a que estaba enfocado a ser usado en ambientes de código libre y con miras únicamente académicas.

El código es compilado al momento de la instalación, y aunque se probó para el compilador integrado **g++** en sus versiones 3.3.5, 4.1.1 y 4.1.3, podría causar algunos conflictos

con algunas versiones. En realidad g++ no es más que un script que manda llamar a gcc con las opciones necesarias para que reconozca el código de C++. El compilador *gcc* procesa el código fuente en una o hasta cuatro etapas, dependiendo de la opciones de ejecución, primero está la etapa de *preprocesamiento*, luego la de *compilación*, la de *ensamblador* y por último la de *enlace*; cada una de estas etapas procesará el archivo de entrada a distintos niveles y con algún nivel de precisión.

Dentro de las posibles opciones de ejecución se encuentra una que se encarga de optimizar el código fuente introducido, esta opción es -O. Este es un optimizador incluido para ser usado durante la compilación y cuenta con distintos niveles a escoger por el usuario. Primero tenemos el nivel -O0 que es la opción por defecto y equivale a no ponerla, en este nivel el código no será optimizado. El primer nivel de optimización es -O o -O1, en este se busca reducir el tiempo de compilación y de ejecución del código generado sin incrementar el tamaño del código; con esta opción obtenemos la mejor relación entre velocidad de compilación y tiempo de ejecución, según los desarrolladores del compilador. El segundo nivel de optimización -O2 es mayor, en este caso el tiempo de compilación aumenta para producir un código más veloz sin comprometer demasiado el tamaño del código. Por último tenemos -O3, el nivel más completo y agresivo de optimización que se sugiere cuando el desempeño de la ejecución está por encima de la necesidad de disminuir recursos. Con O3 crecerá el tiempo y la cantidad de recursos usados en la compilación pero mejorará el tiempo de ejecución, debido a que además de realizar la optimización de O2, sin limitar tiempo y memoria, realiza cambios incluso en la semántica del código fuente, incluso para funciones inline.

Incrementar el nivel de optimización no es garantía de mejoras en el desempeño ya que esto depende de la estructura misma del código. El uso de O3 es riesgoso y no se recomienda para versiones de gcc posteriores a 4.x pero durante las pruebas realizadas no se presentó ningún problema y si hubo mejoras sobre O2. Además debido a que nuestro esquema está diseñado para utilizarse en computadoras personales, los recursos de espacio y tiempo de compilación no son tan esenciales como la necesidad de velocidad en la ejecución.

Otra característica importante de implementación es que el código está organizado en macros y no en funciones. La razón principal es que ahorramos los tiempos de llamadas a funciones y comprobación de tipos inherentes a una función, mejorando el desempeño. El concepto de macro fue largamente usado con el código C para preservar un nivel de eficiencia ya que aunque no funcionan de la misma forma que una función son bastante parecidas con la ventaja de que la macro ahorra el tiempo del llamado (su código se expande) y el tiempo de comprobación de tipos para los argumentos y el valor de retorno. Las macros son expandidas al momento del preprocesamiento y no durante la compilación, pero hay algunos problemas con ellas; complican la depuración de errores y no permiten recursividad, entre otras desventajas. C++ trata de solucionar los problemas de uso de macros mediante la implentación de un tipo especial de funciones llamadas “inline”. Las funciones

inline son identicas en su estructura y manejo a una función normal sólo que su código es expandido, al igual que el de una macro, aunque sigue realizando la comprobación de tipos.

El uso de funciones inline presenta grandes ventajas cuando se programa con objetos, que no es nuestro caso, pero tiene un par de enormes desventajas sobre las macros. La primera es que mientras que una macro puede ser expandida sin importar cual se su contenido, una función inline no puede ser ni muy extensa ni muy "complicada". Esto se debe a que el compilador revisa la estructura de la función inline y se ésta contiene "código complicado" (para el compilador el uso de ciclos es *muy complicado*) simplemente ignora la definición y trata a la función como a cualquier otra no-inline. La segunda desventaja está muy ligada a la primera, el problema es que cualquier definición *inline* es tomada únicamente como una *recomendación* por el compilador, es decir, éste no está obligado a expandir su código, mientras que es una regla el expandir el código de una macro.

En los apartados siguientes se presenta la implementación de las funciones más importantes para el correcto funcionamiento del esquema.

5.2. Construcción del cifrador

En este apartado se muestra la construcción realizada para el cifrador propuesto. Presentaremos la información general que sirvió de base para la elección y construcción cada una de las funciones, además de algunos detalles sobre la implementación de las operaciones y funciones usadas en la construcción del esquema final. Otros datos sobre implementaciones complementarias pueden encontrarse en la siguiente sección.

5.2.1. Cifrador DHIES-M

Para comenzar veamos la construcción del bloque encargado del cifrado y descifrado, el cual pude ser considerado como el más importante principalmente por dos razones: porque es el que realizará realmente el trabajo completo de cifrar-descifrar cualquier mensaje y porque es la parte para la cual se proponen las modificaciones al esquema DHIES. Al igual que como se ha venido presentando a lo largo de esta tesis, se describe por separado la construcción de los dos bloques del esquema híbrido, primero el KEM y luego el DEM.

Construcción del KEM

El mecanismo de encapsulado de llave está basado en el intercambio de Diffie-Hellman, el mismo que en DHIES, por lo tanto es bastante sencillo de construir. Para esto suponemos que los participantes ya generaron su par de llaves e intercambiaron o publicaron de alguna forma su llave pública, para ver detalles sobre la generación y estructura de llaves leer la sección 5.3.1. En la figura 5.1 podemos observar de forma general como se comporta el KEM para cifrar un mensaje.

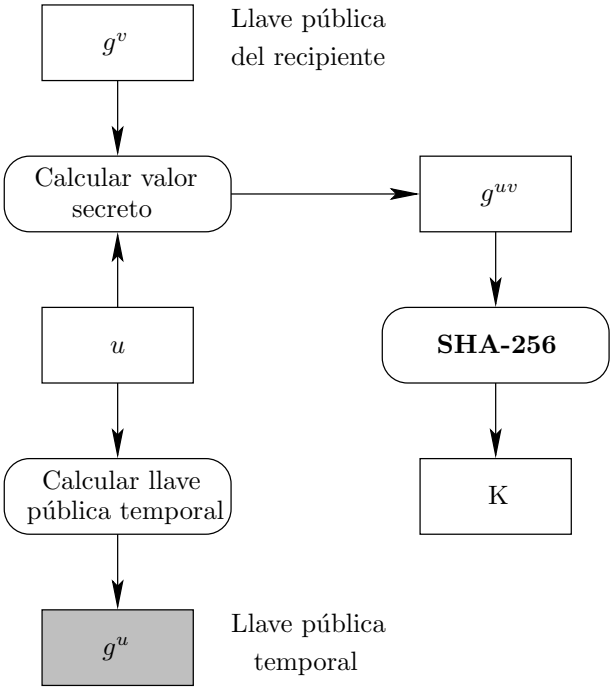

Figura 5.1: Mecanismo de encapsulado de llave implementado.

Una vez que ya tenemos el archivo con la llave pública del recipiente para quien vamos a cifrar el texto en claro, el primer paso es generar un nuevo par de llaves "temporales" que únicamente será usado para cifrar un mensaje y después será desechado. Como paso inicial se lee del archivo de la llave pública el valor del primo p y el valor del generador g, con los cuales se generarán las llaves temporales u y $U = g^u \bmod p$, privada y pública respectivamente. La llave pública temporal U se añade como un primer bloque al mensaje cifrado. La llave privada sirve para calcular $V^u = g^{uv} \bmod p$, el valor secreto de Diffie-Hellman, donde V es leído del archivo de la llave pública del recipiente.

Como último paso del KEM, el valor de DH ($g^{uv} \bmod p$), que también es un elemento del grupo G, sirve de entrada para la función picadillo del esquema. La función picadillo elegida es SHA-256, la cual procesa bloques de 512 bits y arroja un resultado de 256 bits. Para obtener un buen nivel de seguridad y por compatibilidad, las llaves generadas para DH que sirven de entrada para la función picadillo son de 512 bits, es decir, esa es la longitud de los elementos del grupo G. La cadena resultante de SHA-256 será el valor que tomaremos como la llave secreta SK para el DEM.

Para descifrar un mensaje recibido sólo se requieren unos pequeños cambios. Primero se lee el valor $U = g^u$ del archivo recibido, se pide la contraseña para descifrar el archivo de la llave privada y se lee dicho valor v junto con p, calculamos DH $U^v = g^{uv} \bmod p$ y por último le aplicamos SHA-256 para obtener la misma llave secreta SK con la que fue cifrado el mensaje.

Construcción del DEM

El mecanismo de encapsulado de datos es la parte principal del cifrado y descifrado, ya que en ella recae la eficiencia y gran parte de la seguridad del esquema completo. En el esquema original se propone aplicar un algoritmo de cifrado simétrico y despúes un algoritmo MAC, nuestra propuesta es sustituir ambos bloques por un sólo bloque de cifrado con autenticación, con esto esperamos reducir tiempos de ejecución y además permite que la prueba de seguridad del DEM sea una sola en lugar de requerir probar por separado cada bloque.

También éste es el bloque más simple pues una vez implementado el cifrador únicamente se efectúan dos pasos sencillos, primero se espera a que el KEM obtenga la llave secreta SK y segundo está se introduce junto con el texto en claro al esquema de cifrado simétrico con autenticación. El cifrador simétrico implementado fue AES, por ser el estándar actual y por su nivel de seguridad, para llaves de 256 bits que obtenemos de SHA-256. El modo de operación del cifrador es OCB, el cual provee además del cifrado la etiqueta MAC al mismo tiempo que va cifrando el mensaje. Para mayores detalles sobre el cifrador puede dirijirse a 5.2.4. Como podemos ver en la figura 5.2, el texto cifrado que obtenemos al aplicar el DEM se compone de tres bloques. El primer bloque contiene el valor en claro del nonce N que requiere OCB, el segundo es el texto cifrado que tiene la misma longitud en bytes del texto en claro, y el tercero contiene la etiqueta T para el código de autenticación de mensajes.

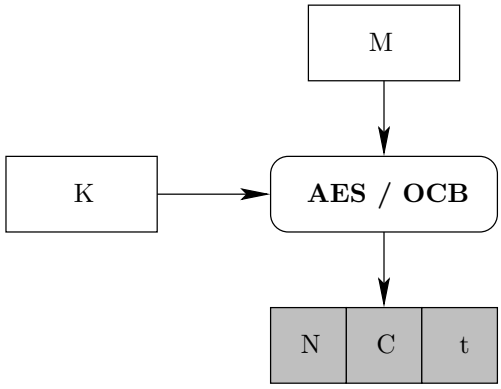

Figura 5.2: Mecanismo de encapsulado de datos implementado.

Algoritmo de cifrado DHIES-M

Basados en los diagramas de KEM y DEM mostrados, podemos construir el algoritmo final que será usado en el esquema. La tabla 5.1 muestra los pasos a seguir necesarios para cifrar y para descifrar un mensaje, esta tabla es idéntica a la tabla 4.4 pero se presenta aquí para comodidad del lector.

Cifrado	**Descifrado**
`entrada` (pk, M)	`entrada` (sk, U, C, t)
$u \xleftarrow{\$} \mathbb{Z}_q^*$	$\chi \leftarrow U^{sk}$
$U \leftarrow g^u$	$K \leftarrow H(\chi)$
$\chi \leftarrow pk^u$	$(M, t^*) \leftarrow AE^{-1}(K, C)$
$K \leftarrow H(\chi)$	si $t \neq t^*$ `salida` ("MAL")
$(C, t) \leftarrow AE(K, M)$	si no `salida` (M)
`salida` (U, C, t)	

Tabla 5.1: Esquema de cifrado híbrido **DHIES-M**.

Al terminar el cifrado, el mensaje completo es enviado al recipiente. La estructura general de los mensajes cifrados se muestra a continuación. XL se refiere a la longitud del texto en claro.

Bloques de un archivo cifrado $(XL + 96)$			
Llave pública temporal (64)	Nonce (16)	Texto cifrado (XL)	Etiqueta (16)

Analizando este algoritmo, podemos observar que la construcción de nuestro esquema requiere de tres tareas principales que deben ser definidas desde el inicio, la cuales deben ser bien elegidas para asegurar la compatibilidad tanto de las operaciones como de los algoritmos principales y procurar el nivel de seguridad deseado. Dichas tareas son:

- Seleccionar un grupo apropiado, específicamente necesitamos uno donde se cumpla la suposición DDH.
- Seleccionar la función hash que servirá para derivar la llave simétrica.
- Elegir el esquema de cifrado simétrico con autenticación que vamos a utilizar.

En el apartado 5.2.2 detallamos lo relacionado a la selección del grupo G, en 5.2.3 se presenta la selección de la función hash H y se describe su funcionamiento, y por último en 5.2.4 se describe al cifrador con autenticación AE.

5.2.2. Selección del grupo

Del capítulo 1, sabemos que al conjunto de las posibles llaves se le conoce como el espacio de llaves $\mathcal{K}$, para cuestiones prácticas este espacio de llaves se establece como un grupo finito de elementos. Un grupo es una especie de construcción matemática especial con ciertas características y para el cual se establece una operación específica (ver apéndice A); en nuestro caso deseamos un grupo cíclico finito G establecido bajo la multiplicación que, como se dijo anteriormente, cumpla con la suposición DDH.

El grupo conocido más común y con el que es más fácil de trabajar es el grupo $\mathbb{Z}_{\mathbf{p}}^*$, donde p es un número primo que nos ayuda a definir los elementos del grupo. El grupo $\mathbb{Z}_p^*$ está compuesto por los elementos $\{1, 2, \cdots, p-1\}$ y la operación definida para este grupo

es la multiplicación módulo p (mód p). Debido a que los elementos con los que tratamos son cadenas de bits, aprovechamos que pueden ser representados como números enteros y tomamos de base a $\mathbb{Z}_p^*$ para establecer a nuestro grupo G. Desafortunadamente es sabido que dentro de este grupo, DDH no se cumple. De hecho es sabido que cuando se desea aplicar las suposiciones DH es mejor hacerlo dentro de un grupo de orden primo.

El orden del grupo $\circ(\mathbb{Z}_p^*)$ (el número de elementos que lo componen) es $p-1$, un número par, y como buscamos que nuestro grupo G sea de orden primo, lo que hacemos es seleccionar un subgrupo de $\mathbb{Z}_p^*$, el cual cumpla con las características deseadas, que sea cíclico y de orden primo. Antes de hablar más sobre la elección del subgrupo a utilizar necesitamos definir un nuevo concepto, el de **residuo cuadrático**, presentado en la definición 5.1.

Definición 5.1. Dado un grupo G y un entero $r \in G$, se dice que r es un *residuo cuadrático* si existe un $x \in G$ tal que $x^2 = r$.

El conjunto de residuos cuadráticos(RQ) tienen algunas propiedades especiales, algunas muy sencillas de mostrar. Para nuestros propósitos, son esenciales dos las propiedades del conjunto RQ para el caso de $\mathbb{Z}_p^*$, que son establecidas en el lemma 5.2.

Lemma 5.2. *Tomando a G como un grupo cíclico finito y a $G' = \{r \in G : \exists x \in G$ donde $x^2 = r\}$ como un conjunto, tal que los elementos de G' son residuos cuadráticos. Entonces se afirma que G' es un subgrupo de G y que $\circ(G') = \circ(G)/2$*

Con este lemma se establecen dos cosas importantes, primero que el conjunto RQ es un subgrupo y segundo que este subgrupo contiene la mitad de los elementos del grupo en el que se encuentra. Esta segunda propiedad es muy facil de mostrar en $\mathbb{Z}_p^*$. Dado el módulo p, la lista de residuos se obtiene elevando al cuadrado (mód p) los enteros $\{1, \ldots, p-1\}$. Algo curioso es que, dado que $x^2 \equiv (p-x)^2$ mód p, la lista que se obtiene es simétrica alrededor de $p/2$ por lo que sólo necesitamos generar la mitad de la lista. Como ejemplo tomemos a $p = 7$, entonces la lista de cuadrados perfectos mód 7 es $\{1^2 = 1, 2^2 = 4, 3^2 = 2, 4^2 = 2, 5^2 = 4, 6^2 = 1\}$ donde $1, 2, 4$ son residuos cuadráticos y $3, 5, 6$ no lo son.

Por otro lado, las opciones de posibles números primos son amplias, pero debemos considerar sólo aquellas que son consideradas como "seguras" en el área de la criptografía. Dentro de esas opciones tenemos al conjunto de números conocidos como **primos de Sophie Germain**. Estos números primos tienen la particularidad de que dado un número primo q, es *primo de Sophie Germain*(SG) si $p = 2q + 1$ también es un número primo, ejemplos de primos SG son los tres primeros números primos: $(q = 2,\ p = 5)$, $(q = 3,\ p = 7)$ y $(q = 5,\ p = 11)$.

Por último establezcamos un número primo de Sophie Germain $p = 2q + 1$ y definamos a G como el conjunto de residuos cuadráticos de $\mathbb{Z}_p^*$. Dados ambos y por todo lo mencionado anteriormente, obtenemos que G es un subgrupo cíclico de $\mathbb{Z}_p^*$, que G tendrá $q = \frac{p-1}{2}$

elementos ($\circ(G)$ es un primo), y que todos los elementos de G son generadores.

Esto último es una característica muy importante del grupo de RQ, puesto que podremos encontrar de forma muy sencilla un generador. De la definición sabemos que un residuo cuadrático es un cuadrado perfecto, por lo tanto, seleccionando de forma aleatoria un elemento $x \in \mathbb{Z}_p^*$ y elevandolo al cuadrado produciremos un residuo que será generador de G.

Ahora bien, el primer paso para comenzar a trabajar es tener un número primo de Sophie Germain que nos permita generar el grupo y realizar las operaciones modulares, por lo cual fué necesario implementar un algoritmo de *prueba de primalidad*. Un algoritmo de este tipo se encarga de probar si un entero n es un número primo o uno compuesto. Para este proyecto se implementó el algoritmo probabilístico de tipo Monte Carlo más eficiente y más usado en la práctica, el de Miller-Rabin (MRT) [46]. MRT es un algoritmo aleatorio *predispuesto al sí*, es decir, cuando el algoritmo responde de forma afirmativa (contesta **sí**) entonces la respuesta es siempre correcta pero, si responde **no** entonces existe cierta probabilidad de error. Es importante saber que MRT está diseñado para resolver el problema para *números compuestos*, esto es que dado un número n, responde sí o no a la pregunta ¿es n un número compuesto?

Algoritmo de Miller-Rabin

`entrada:` (n, t) $[n \geq 3,\ t \geq 1]$
Calcular valores (s, r) que cumplan con $n - 1 = 2^s r$ para r impar.
`Para` $i \leftarrow 1$ `hasta` t {
$a \xleftarrow{\$} \{2, \cdots, n-2\}$
$y \leftarrow a^r$ mód n
`Si` $(y \neq 1)$ y $(y \neq n-1)$ `entonces` {
$j \leftarrow 1$
`Mientras` $(j \leq s-1)$ {
$y \leftarrow y^2$ mód n
`Si` $y = 1$ `entonces salida:` COMPUESTO
`Si` $y = n-1$ `entonces` termina `mientras`
`Si no` $j \leftarrow j+1$ }
`Si` $y \neq n-1$ `entonces salida:` COMPUESTO } }
`salida:` PRIMO

Tabla 5.2: Algoritmo probabilístico de Miller-Rabin para la prueba de primalidad.

El algoritmo MRT basa su prueba en el hecho de que dado un número primo impar n tal que $n - 1 = 2^s r$ donde r es impar, si un entero a es primo relativo con n, entonces a cumple con alguna de las siguientes propiedades: o $a^r \equiv 1$ mód n o $a^{2^j r} \equiv -1$ mód n para $0 \leq j \leq s-1$. Como puede verse en la tabla 5.2, lo que hace el algoritmo es seleccionar un número aleatorio $a \in \mathbb{Z}_n$ y comprobar si éste cumple con alguna de las propiedades

mencionadas. Si a no cumple con ninguna se le llama *testigo* y por lo tanto podemos estar seguros que n es compuesto. Si a cumple con alguna de las propiedades, entonces n es llamado *pseudo-primo fuerte* en base al número a. Hay tres características importantes de esta prueba:

- si MRT siempre declara "compuesto" entonces n es compuesto,
- si n es primo, MRT siempre declara "primo", y
- la probabilidad de error, que MRT declare que n es "primo" cuando es compuesto, depende del número de pruebas realizadas.

La prueba se realiza para un conjunto de t valores aleatorios, siendo t el parámetro que determina la probabilidad de que el algoritmo dé un falso positivo, es decir, declare que un número es primo cuando en realidad es compuesto. Es un hecho que para un número compuesto impar n, a lo sumo $1/4$ de los posibles valores de $a \in \mathbb{Z}_n$ son "mentirosos", es decir, al aplicar MRT diran que n es primo. Con esto obtenemos que la probabilidad de error para el algoritmo de primalidad de Miller-Rabin es de a lo más $(1/4)^t$. Las únicas restricciones son que n debe ser mayor a 2 y t mayor a 0, en este caso se utilizó $t = 20$.

Lo que hacemos en la implementación es aplicar el algoritmo MRT sobre un número aleatorio q de 511 bits hasta encontrar uno que sea primo. Encontrado q primo, aplicamos la misma prueba a $p = 2q + 1$ para verificar si este también es primo, en caso de que si lo sea lo usamos para la generación del grupo, en caso contrario regresamos a buscar un nuevo q. El proceso se repite hasta encontrar un primo SG. El único valor que almacenamos es p ya que con él es fácil y rápido obtener q y g.

En esta parte del proceso se sugiere generar una tabla auxiliar que contenga los primeros números primos para que antes de aplicar el algoritmo MRT se verifique si el número a evaluar tiene como factor a alguno de los elementos en la tabla. Con esto se mejoraría la eficiencia de la búsqueda y por lo tanto del algoritmo en general.

5.2.3. Hash

Ya hemos dicho que la función hash implementada es SHA-256. Los *Algoritmos Hash Seguros*(SHA) son algoritmos que permiten una representación condensada de datos electrónicos. Estos algoritmos reciben un mensaje de entrada y arrojan una cadena llamada *resumen*. En la especificación [29] se describen 4 algoritmos, SHA-1, SHA-256, SHA-384 y SHA-512, que indican el tamaño del resumen resultante, pero en este apartado únicamente se hablará de SHA-256, que fue la implementación realizada para el proyecto.

SHA-256 se desarrolla en 2 etapas principales: primero está el *preprocesamiento* que incluye rellenar el mensaje, dividirlo en bloques de 512 bit e inicializar algunos valores; después está el cálculo del resumen, que genera una planeación del mensaje relleno para

que con ayuda de las funciones y constantes pueda obtener el valor del resumen.

Las propiedades básica de SHA-256 son: 1) recibe mensajes de tamaño variable que sean menores a 2^{64} bits; 2) procesa bloques de 512 bits; 3) utiliza palabras de 32 bits; 4) produce un resumen de 256 bits; y 5) ofrece un nivel de seguridad equivalente a 128 bits.

Funciones y constantes

SHA-256 utiliza seis funciones lógicas, todas ellas operan sobre palabras de $w = 32$ bits, representadas aquí por x, y y z. Los resultados de dichas funciones es una nueva palabra también de w bits. Las funciones definidas en el algoritmo hacen uso de distintos operadores sobre esas palabras, esos operadores son: AND($\wedge$), OR($\vee$), NOT($\neg$), XOR($\oplus$), suma módulo 2^w ($+$), desplazamiento(SHR^n) y rotación($ROTR^n$) de n bits a la derecha. Las funciones lógicas son:

$$\begin{aligned} Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \end{aligned}$$

$$\begin{aligned} \textstyle\sum_0^{\{256\}}(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\ \textstyle\sum_1^{\{256\}}(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \\ \sigma_0^{\{256\}}(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \\ \sigma_1^{\{256\}}(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \end{aligned}$$

SHA-256 hace uso también de una serie de 64 constantes, palabras de 32 bits, que identificaremos como $K_0^{\{256\}}, \ldots, K_{63}^{\{256\}}$. Estas constantes tienen los siguientes valores hexadecimales, de izquierda a derecha:

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90beffa	a4506ceb	bef9a3f7	c67178f2.

Preprocesamiento

Se divide en tres etapas, las cuales se definen a continuación.

Rellenar mensaje: El propósito de esta etapa es asegurar que el mensaje M a procesar sea un multiplo de 512 bits. Primero suponemos que el mensaje M es de longitud l bits. Se le agrega un bit "1" al final de M seguido de k bits en 0, donde k es la solución no-negativa más pequeña a la ecuación $l + 1 + k \equiv 448$ mód 512. Por último se agrega un bloque de 64 bits que corresponde a la representación binaria de l.

Separar en bloques: Ahora lo que se debe hacer es dividir el mensaje rellenado en N bloques de 512 bits, $M^{(1)}, \ldots, M^{(N)}$. Cada bloque de 512 bits es representado con 16 palabras de 32 bits. Para el bloque i del mensaje, sus 16 palabras se denotan por $M_0^{(i)}, \ldots, M_{15}^{(i)}$.

Inicializar valor *hash*: Antes de que comience el cómputo del *hash*, es necesario inicializar el valor de este en $H^{(0)}$. Para SHA-256, este valor inicial consiste de 8 palabras de 32 bits. Los valores hexadecimales son:

$$\begin{aligned} H_0^{(0)} &= 6a09e667 \\ H_1^{(0)} &= bb67ae85 \\ H_2^{(0)} &= 3c6ef372 \\ H_3^{(0)} &= a54ff53a \\ H_4^{(0)} &= 510e527f \\ H_5^{(0)} &= 9b05688c \\ H_6^{(0)} &= 1f83d9ab \\ H_7^{(0)} &= 5be0cd19. \end{aligned}$$

Algoritmo hash

Acabado el preprocesamiento sólo queda por definir el algoritmo de SHA-256 para calcular el valor de resumen. Este algoritmo esta formado por un ciclo principal, con cuatro pasos en cada iteración, que procesa en orden cada bloque del mensaje desde $M^{(0)}$ hasta $M^{(N)}$.

Para i desde 1 hasta N hacer {

1. Preparar programa del mensaje, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-2} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases}$$

2. Inicializar 8 variables de trabajo, $a,\ b,\ c,\ d,\ e,\ f,\ g$, y h, con el valor hash $(i-1)$:

$$\begin{aligned} a &= H_0^{(i-1)} \\ b &= H_1^{(i-1)} \\ c &= H_2^{(i-1)} \\ d &= H_3^{(i-1)} \\ e &= H_4^{(i-1)} \\ f &= H_5^{(i-1)} \\ g &= H_6^{(i-1)} \\ h &= H_7^{(i-1)} \end{aligned}$$

3. Para t desde 0 hasta 63 hacer {

$$
\begin{aligned}
T_1 &= h + \textstyle\sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t \\
T_2 &= \textstyle\sum_0^{\{256\}}(a) + Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

}

4. Calcular $i-$ésimo valor intermedio del hash $H^{(i)}$:

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}
$$

}

Después de repetir estos pasos hasta llegar a N repeticiones, el resumen resultante de 256 bits para el mensaje M es:

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}.$$

La implementación se dividió en dos partes. La primera de ellas es una biblioteca que contiene macros para las seis funciones básicas del algoritmo: Ch, Maj, Σ_0, Σ_1, σ_0 y σ_1. Estas operaciones actuan sobre palabras de 32 bits (definidas como enteros sin signo) y hacen uso de las operaciones sobre bits propias del lenguaje. Además contiene una macro que tiene la función de preparar las "palabras" a ser procesadas, al inicio de cada ronda. La segunda parte contiene la definición de las constantes usadas en el proceso, en forma de arreglos, y la función "SHA256" que realiza los pasos descritos arriba como "algoritmo hash" que podrá ser llamada cuando sea necesario y que actua directamente sobre archivos para obtener su hash. Esta macro recibe como parámetros dos cadenas, la primera con el nombre del archivo de entrada y la segunda con el nombre del archivo donde se almacenará el resultado.

5.2.4. Cifrado con autenticación

El primer paso de este bloque es elegir el esquema de cifrado simétrico, la elección fue AES y en cuanto al calendario de trabajo, fue lo primero en implementarse. El algoritmo AES es el mecanismo de cifrado simétrico por excelencia, tomado como estándar en la gran mayoría de las aplicaciones por lo tanto no había mucho que pensar. Una vez decidido el esquema simétrico se tuvo que elegir cuales serían sus características que, por cuestiones de compatibilidad y por el nivel de seguridad que es bien conocido, tampoco presentó gran dificultad, se implementó AES-256 (usa llaves de 256 bits) que ofrece un alto nivel de seguridad gracias al propio algoritmo y a la longitud de la llave. Las características específicas del algoritmo AES para llaves de 256 son tres: el procesamiento se realiza en bloques de 128 bits al igual que con la versión más común (AES-128), aunque la llave es de 256 bits únicamente se procesan 128 bits en cada ronda, y el número de rondas para esta versión es de 14, cuatro más que la versión común. Debido a estas tres características AES-256 no representa un gran aumento en el tiempo de ejecución del algoritmo y si un gran salto en el nivel de seguridad.

La implementación de AES-256 realizada está basada en el documento FIPS 197 [28], con las versiones originales no con la propuesta de descifrado equivalente. Para esto se creo una biblioteca en la cual se definen todas las macros y las tablas de constantes necesarias para el algoritmo. Las variables usadas son arreglos bidimensionales de 16 elementos (matrices de 4x4) que contienen caracteres sin signo (bytes). Se declaran las cinco tablas necesarias, dos para la sustitución de bytes, dos para la mezcla de columnas y una para la generación de las llaves de ronda.

Lo primero fue definir una macro que calculara todas las llaves de ronda y almacenar los valores resultantes en un arreglo bidimensional con capacidad para 15 matrices de 4x4, necesarias para una ronda inicial y 14 rondas completas. Después se definieron las cuatro macros que realizan las funciones de ronda del algoritmo: sustitución de bytes, desplazamiento de filas, mezcla de columnas y suma de llave de ronda. Las primeras tres operaciones reciben la matriz sobre la que van a operar y el modo de trabajo, cifrado o descifrado, para realizar lo adecuado, y la última recibe además, el arreglo de llaves de ronda y el número de ronda en la que fue llamada.

Además de las operaciones ya mencionadas, fue necesario implementar una macro para realizar una multiplicación en el campo $GF(2^8)$ que es una de las operaciones básicas de este algoritmo junto con la operación *xor* y los corrimientos de bits. El campo utilizado en AES está definido sobre el polinomio irreducible $x^8 + x^4 + x^3 + x + 1$.

Una vez elegido e implementado el algoritmo de cifrado, se tuvo que elegir el modo de operación a utilizar. Para ese momento ya se tenía idea de los posibles cambios que se deseaban aplicar al esquema DHIES y de qué es lo que buscabamos obtener con esos cambios: mantener el mismo nivel de seguridad, prestar los mismos servicios y mejorar

la eficiencia. Al estudiar las posibles opciones se llegó a la conclusión de que lo mejor sería añadir un cifrado con autenticación, de esta manera sustituimos dos algoritmos con tiempos independientes por uno que realiza lo mismo en una sola ejecución. De esta forma se simplificó el esquema y se mejoró la eficiencia de procesamiento.

La opción fue implementar un modo de operación llamado *Offset Codebook* (OCB), el cual ofrece un cifrado con autenticación en una sola ejecución, con un costo bajo y que además tiene una estructura sencilla y bastante útil. OCB fue diseñado por P. Rogaway en 2000 [37], y al año siguiente fue publicado [39] con ayuda de M. Bellare, J. Black y T Krovetz. Este modo de operación se diseño en base al esquema de cifrado autenticado IAPM [19], propuesto por Jutla. Además de esta versión original del 2000 existe una segunda versión, comúnmente llamada OCB1 y presentada por el mismo autor, cuyo principal cambio es el método para calcular la serie de "compensaciones". Tal fue la importancia de este modo de cifrado que OCB apareció como propuesta de estándar para redes locales inalámbricas en el IEEE 802.11.

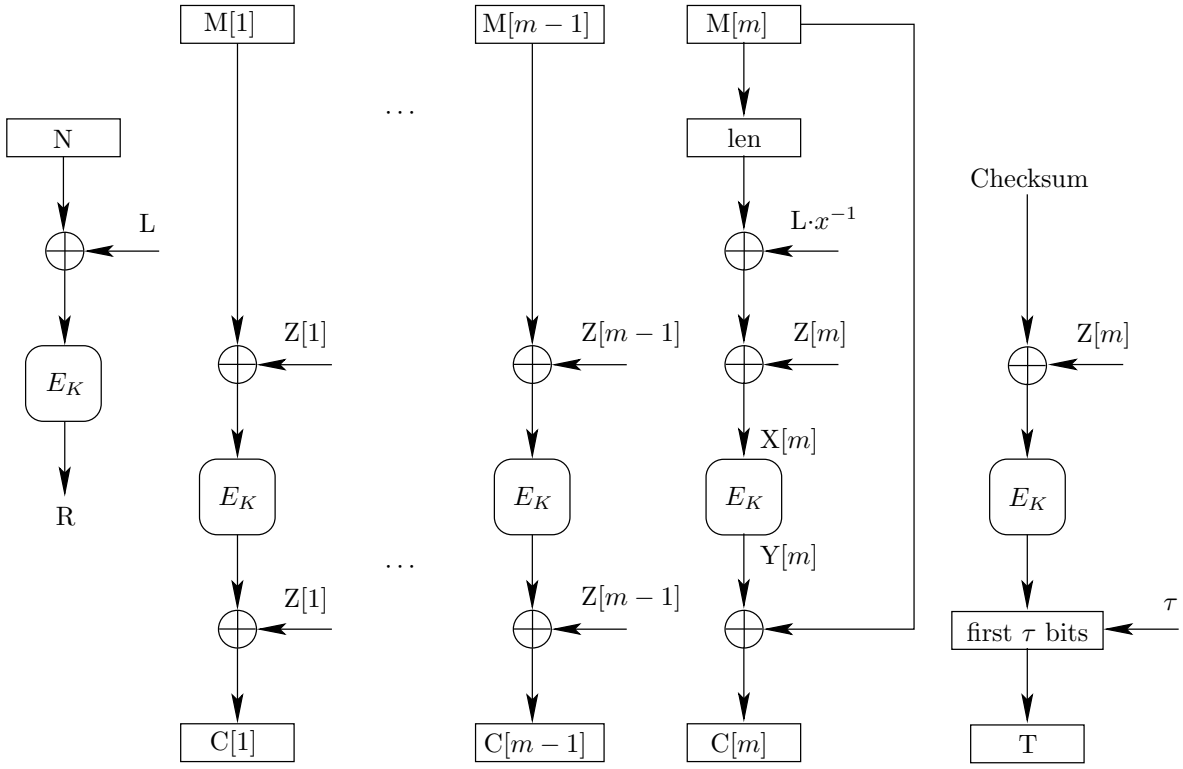

Figura 5.3: Modo de operación **OCB**.

Para describir el funcionamiento del cifrado con autenticación de OCB debemos antes definir algunos conceptos y funciones necesarios. Para empezar es necesario establecer los dos parámetros iniciales de OCB: un cifrador por bloques $SYM = (E, D)$ donde $E(K, \cdot) : \{0,1\}^n \to \{0,1\}^n$ es la función de cifrado para bloques de longitud n, $K \in \mathcal{K}$ es un elemento del espacio de posibles llaves y $D_K = E_K^{-1}$ es la función de descifrado, y en

segundo lugar la longitud de la etiqueta $\tau \in [0..n]$. También será necesario el uso de la función $ntz(i)$ para enteros $i > 0$, la cual retorna el número entero más grande z tal que 2^z divide a i, es decir, el número de bits contiguos en cero, a partir del menos significativo, que contiene la representación binaria de i. Además debemos tomar en cuenta dos valores más, L y R, que junto con la definición de los códigos de Gray dados en el artículo [39] sirven para calcular la serie de "compensaciones" Z. En la figura 5.3 se muestra el esquema a seguir para el cifrado con OCB.

La tabla 5.3 presenta las reglas completas de cifrado y descifrado que establece el modo de operación OCB. Para estos algoritmos se toman las siguentes consideraciones: M será el mensaje a cifrar y se escribe como $M[1] \dots M[m]$ para representar cada uno de los m bloques, sucede lo mismo con la cifra C; K es la llave secreta del cifrador por bloques; N será el nonce, el cual será elegido por quien cifra, no debe repetirse y debe ser enviado junto con el texto cifrado; en las sumas $Y[m] \oplus M[m]$ y $Y[m] \oplus C[m]$ se truncará $Y[m]$ al tamaño del otro operando; y $C[m]0^*$ se refiere a añadir a la derecha tantos bits en 0 como sean necesarios para lograr la longitud n.

Cifrado	**Descifrado**
`entrada` (N, M)	`entrada` $(N, \mathcal{C})$
Dividir M en $M[1] \cdots M[m]$	Dividir $\mathcal{C}$ en $C[1] \cdots C[m]\,T$
$L \leftarrow E_K(0^n)$	$L \leftarrow E_K(0^n)$
$R \leftarrow E_K(N \oplus L)$	$R \leftarrow E_K(N \oplus L)$
para $i \leftarrow 1$ hasta m hacer	para $i \leftarrow 1$ hasta m hacer
$\quad Z[i] = \gamma_i \cdot L \oplus R$	$\quad Z[i] = \gamma_i \cdot L \oplus R$
para $i \leftarrow 1$ hasta $m-1$ hacer	para $i \leftarrow 1$ hasta $m-1$ hacer
$\quad C[i] = E_K(M[i] \oplus Z[i]) \oplus Z[i]$	$\quad M[i] = D_K(C[i] \oplus Z[i]) \oplus Z[i]$
$X[m] = len(M[m]) \oplus L \cdot x^{-1} \oplus Z[m]$	$X[m] = len(C[m]) \oplus L \cdot x^{-1} \oplus Z[m]$
$Y[m] = E_K(X[m])$	$Y[m] = E_K(X[m])$
$C[m] = Y[m] \oplus M[m]$	$M[m] = Y[m] \oplus C[m]$
$C = C[1] \cdots C[m]$	$M = M[1] \cdots M[m]$
Checksum$\leftarrow M[1] \oplus \cdots$	Checksum$\leftarrow M[1] \oplus \cdots$
$\quad \cdots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$	$\quad \cdots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$
$T \leftarrow E_K(\text{Checksum} \oplus Z[m])$ [τ bits]	$T' \leftarrow E_K(\text{Checksum} \oplus Z[m])$ [τ bits]
`salida` $(\mathcal{C} \leftarrow C \| T)$	`salida` (M si $T = T'$, sino MAL)

Tabla 5.3: Modo de operación **OCB** (cifrado con autenticación).

Aunque a simple vista parece un poco complicado, podemos darnos cuenta de que las operaciones son sencillas. Nótese que OCB mantiene las dos características principales de IAPM, es completamente paralelizable y tiene costos indirectos mínimos en comparación con los modos de operación clásicos sin autenticación. Además combina cinco importantes características: puede cifrar cadenas de cualquier longitud, el número de llamadas al cifrador es la más adecuada, pocos requerimientos sobre el nonce, mejora los cálculos de compensación y utiliza una única llave.

También podemos ver que este modo de operación tiene un par de requerimientos especiales para su funcionamiento, la necesidad de un nonce y el cálculo de códigos Gray. La definición del nonce tiene pocos requerimientos, no se requiere que el nonce sea aleatorio ni secreto y la única restricción es que no debe repetirse para una misma llave. La elección del valor usado como nonce se deja a elección de quien cifra y no es necesario usar métodos criptográficos ni para su creación ni para su comunicación. Una buena opción es la de establecerlo como un contador, el cual deberá incrementarse con cada mensaje enviado. En nuestro caso, el nonce es llenado como una cadena aleatoria al momento de cifrar el mensaje y debido a que la llave de cifrado es prácticamente un valor aleatorio, la probabilidad de que se repita la misma pareja de llave-nonce es insignificante.

En lo que respecta a los codigos Gray, éstos son necesarios para realizar los cálculos de *compensación* y se necesitaron dos funciones básicas. Primero se necesitó una macro muy sencilla que calculará el número de ceros contiguos en los bits menos significativos de un arreglo (NTZ). Y en segudo lugar se necesitaron definir algunas operaciones en un campo binario; para AES se necesitan operaciones sobre $GF(2^8)$, en este caso las operaciones son de 128 bits para lo cual definimos el campo $GF(2^{128})$ para lo cual se hace uso del polinomio irreducible $x^{128} + x^7 + x^2 + x + 1$. A diferencia de las especificaciones originales del algoritmo, el conjunto de valores de compensación Z no son calculados antes de cifrar, en nuestro caso únicamente se calculan los valores de $Li = L \cdot x^i$ para los códigos Gray y son usados para calcular Z conforme se van necesitando durante el cifrado de cada mensaje.

A continuación se presenta una serie de propiedades de OCB que pueden ser vistas como ventajas, dentro de las cuales se encuentran las 5 características ya mencionadas.

- Cifrado de mensajes de cualquier tamaño sin expandir la longitud del texto cifrado, sólo se añade la longitud de la etiqueta,
- uso de una sola llave de cifrado,
- no se requiere que el nonce sea aleatorio, impredecible ni secreto, sólo que no se repita para una misma llave,
- la elección y comunicación del nonce se deja a elección del usuario ya que no necesita de métodos criptográficos,
- no es necesario conocer el tamaño del mensaje para poder procesarlo,
- eficiencia, llamadas al cifrador
- el preprocesamiento es opcional,
- es simple en el sentido de que todos los bloques de procesan de la misma forma y el relleno de bits es el básico (agregar ceros),
- evita la colisión de etiquetas,

- es completamente paralelizable, a excepción del último bloque y un cifrado inicial,
- es eficiente, el número de llamadas al cifrador $\lceil |M| / n \rceil + 2$ es una más que el óptimo, a diferencia de modos como CCM que requieren cerca del doble,
- cuenta con seguridad comprobable.

Algo importante es que a pesar de existir una patente sobre la aplicación de OCB, existe una exención que permite utilizarlo sin ningún costo en implementaciones hechas bajo la licencia pública general(GPL) de GNU y en aquellas que no sean ni desarrolladas ni vendidas en los Estados Unidos.

5.3. Otras implementaciones

En esta sección se presentan algunos datos sobre la implementación algunas otras funciones esenciales para el esquema de cifrado final.

5.3.1. Generación de llaves

Una vez establecidos los protocolos de comunicación y los algoritmos a utilizar, la generación de las llaves es la parte inicial de cualquier cifrador. Como se mencionó en los capítulos iniciales, el esquema híbrido puede ser visto como un esquema de cifrado de llave pública, por lo tanto, esta etapa inicial debe encargarse de generar un par de llaves, llave pública y llave privada, para cada uno de los participantes.

Teniendo el generador g para nuestro grupo G, lo que nos resta por hacer ahora es generar el par de llaves. Como se mostró a lo largo del capítulo anterior, ese par de llaves son las usadas en el esquema de Diffie-Hellman y dado que nuestro grupo tiene q elementos, procedemos de la siguiente forma: seleccionamos de forma aleatoria un elemento v de $\mathbb{Z}_q^*$ y lo establecemos como la llave privada. Dados el generador g y la llave privada v calculamos la llave pública V como g^v mód p.

Así termina la generación de llaves pero eso es de forma teórica, en la práctica hay distintas cuestiones que debemos tomar en cuenta, por ejemplo la forma de almacenar las llaves y como encontrar el número primo, entre otras. A continuación se presenta una serie de pasos, los cuales son los realizados por la aplicación implementada. Como datos adicionales, las llaves pública y privada se almacenan como una cadena de caracteres para su fácil manejo, y por seguridad, el archivo de la llave privada se almacena cifrado bajo el mismo esquema que implementa PGP donde se pide una contraseña para mantener segura la llave privada segura, ya que de no ser así alguien con acceso al sistema podría robarla o modificarla.

1. Solicitar contraseña al usuario.

2. Seleccionar un número primo de Sophie Germain.
3. Seleccionar un número aleatorio y multiplicarlo por sí mismo (módulo p) para obtener g.
4. Seleccionar de forma aleatoria un número módulo q y lo establecemos como la llave privada v.
5. Con el generador g y la llave privada v se calcula la llave pública V como g^v mód p.
6. Guardar las llaves pública y privada en los archivos "owner.pk" y "owner.sk", respectivamente.
7. Cifrar el archivo de la llave privada.

De lo anterior, en el paso número 2, el número primo de Sophie Germain p se selecciona de un conjunto de números almacenados en un archivo, los cuales fueron encontrados previamente usando el algoritmo de Miller-Rabin. También por razones de seguridad, ese archivo es cifrado con la misma contraseña con la que se cifró el archivo de la llave privada, sino se cifrara, alguien con acceso al sistema podría modificar los valores de los números primos para poner valor no seguros(números compuestos o primos débiles) y en la próxima generación de llaves las llaves serían inseguras. Por su parte, los archivos de las llaves tienen una estructura específica que se muestra en la tabla 5.4, donde el valor que se da entre paréntesis es la longitud del bloque en bytes.

Como puede verse en la estructura de los archivos de llave, el tamaño del número primo, del generador y de ambas llaves ocupa el doble de sus bytes, 128 en lugar de 64, esto debido a que se almacenan la cadena hexadecimal de su valor, lo que requiere, por cada byte, 2 bytes(2 caracteres) para su representación.

Bloques del archivo de llave pública *pk* (384)		
Primo SG (128)	Generador (128)	Llave pública (128)

Bloques del archivo llave privada *sk* (288)			
Nonce (16)	Primo SG cifrado (128)	Llave privada cifrada (128)	Etiqueta (16)

Tabla 5.4: Composición de los archivo de llave pública y privada.

5.3.2. Aritmetica modular

La implementación de las macros que realizarán las operaciones de la aritmética modular fue el paso inicial en la construcción del KEM, y lo primero era definir cómo se manejarían los valores para cada número que debía ser almacenado. Después de analizar el procedimiento de cada una de la operaciones a ser implementadas y teniendo en cuenta

que las cadenas a manejar tenían una longitud de 512 bits, la elección fue la de trabajar con una estructura de datos que nos ofrece varias ventajas.

Primero se decidió que lo mejor era trabajar con arreglos que almacenaran bytes, por lo que el tipo básico usado es el de *unsigned char*. Y después de analizar las operaciones se definió una estructura con el nombre de **StructBytes** que contiene tres variables: un puntero al arreglo de bytes, un entero que nos indica el tamaño total de localidades del arreglo y un entero que nos indica el número de casillas del arreglo que ocupa el valor almacenado. El primer entero se decidió agregarlo para verificar el correcto uso de los arreglos en las operaciones aritméticas cuando se trabaje con distintas longitudes, por ejemplo en caso de sumar dos arreglos de 64 bytes y trate de almacenar el resultado en uno de 32. El segundo entero está planeado para agilizar las operaciones, esto es por ejemplo si se realiza un desplazamieto a un arreglo de 64 bytes pero del cual sólo las primeras tres posiciones están en uso, el desplazamiento se realiza para tres bytes y no para 64 que sería un desperdicio de tiempo; y esto aplica para todas la demás operaciones. Hay que señalar que debido a que para propósitos de nuestro esquema todas las estructuras son de 64 bytes por lo que no se aprovechan todas las ventajas que éstas nos ofrecen, pero sin embargo quedarán como base y serviran de referencia para futuros trabajos de implementación.

Una vez definido el tipo de dato a manejar, se comenzó con la implementación de las operaciones, las cuales están divididas en dos bibliotecas. La biblioteca "Aritmética" contiene 5 operaciones aritméticas, todas ellas, excepto la resta, reciben el número primo p como argumento. Las operaciones son:

- Módulo: recibe una estructura (x) y calcula x mód p, almacenando el resultado en el mismo x.
- Resta: recibe tres estructuras (a, b, r), calcula $a - b$ y almacena el resultado en r.
- Suma: recibe tres estructuras (a, b, r), calcula $a + b$ mód p y lo almacena en r.
- Multiplicación: recibe tres estructuras (a, b, r), calcula $a \cdot b$ mód p y almacena el resultado en r. El algoritmo de multiplicación es el de Blakley, basado en doblados y sumas sucesivas interclados con reducciones. Debido a que este método es bastante lento, se implementó la multiplicación eficiente de Karatsuba-Ofman pero junto con la reducción de división con restauración se volvía tan ineficiente como la primera y mucho más complicada.
- Exponenciación: recibe tres estructuras (a, b, r), calcula a^b mód p, almacenando el resultado en r. El algoritmo implementado es el de la exponenciación binaria.

La otra biblioteca es "Bitwise" y contiene una serie de operaciones sobre bits muy útiles que son utilizadas por las operaciones aritméticas y en la busqueda de números primos. Las operaciones incluidas aquí son: desplazamientos hacia la izquierda y derecha, copia de

una estructura a otra, determinar si una estructura es igual a otra o si es mayor (con estas dos podemos realizar cualquier comparación $=, \neq, <, >, \leq, \geq$), determinar el tamaño de la estructura, borrarlas o inicializarlas en cero. Por otra parte, algunas de las operaciones mencionadas operan también sobre las matrices usadas en AES. La implementación de todas estas operaciones hace uso de las operaciones de bits definidas por el lenguaje.

5.3.3. Interfaz

A pesar de que el código implementado está en su totalidad en C++, la interfaz con la que interactúa el usuario es una serie de **scripts**. Esto se debe a dos razones: la primera es que la aplicación está planeada para usarse primariamente en los servidores del Departamento de Computación del CINVESTAV Zacatenco, institución donde se realizó el proyecto, los cuales trabajan sobre el S.O. GNU/Linux y a los cuales se tiene acceso para trabajar mediante una terminal, y la segunda es que estos scripts nos permiten manejar de forma más sencilla los archivos y los comandos del sistema, lo que facilita el uso.

Los scripts están diseñados específicamente para terminales tipo *bash* pero debido a que no usan comandos ni funciones especializadas, pueden trabajar en algunas otras consolas. Todos estos scripts funcionan como cualquier comando y se encargan de llamar a los códigos generados en la compilación de los archivos fuente de C++. Los comandos (scripts) definidos permiten administrar las llaves (generar, agregar y eliminar), cifrar y descifrar.

Otro punto importante es que todos los comandos pueden ser ejecutados por el usuario desde cualquier ubicación/ruta dentro del equipo.

5.4. Resultados obtenidos

Ahora bien, una vez terminada la implementación de todas las operaciones necesarias para cada bloque sólo queda mostrar los resultados obtenidos. En esta sección se presentan algunas tablas referentes a los tiempos de ejecución de los bloques principales.

Comencemos por decir que los tiempos reportados fueron obtenidos utilizando la función **clock()**, propia del lenguaje. Las cantidades están en ciclos de reloj que fueron necesarios para realizar los cálculos. Aunque las pruebas fueron realizadas en distintas computadoras, incluyendo el servidor de Computación, los tiempos reportados fueron tomados de una laptop cuyas características son:

- Gateway® Portátil MX6937m
- Procesador Intel® Centrino Duo a 1.60GHz
- Memoria RAM DDR2 de 1024MB

- Sistema Operativo Linux Mint 4 Daryna

En lo referente al DEM, la primera implementación fue la de AES, para la cual se midieron los tiempos de cifrado y descifrado. El tiempo que tarda en cifrar un bloque (16 bytes) es de ≈ 42.52 ciclos mientras que el algoritmo de descifrado tarda ≈ 100.82 ciclos, pero esto es para un sólo bloque. Ya incluido en el modo de operación estos tiempos aumentan, especialmente para mensajes pequeños. En la tabla 5.5 y en la tabla 5.6 se presentan respectivamente los tiempos de cifrado y descifrado con OCB para mensajes muestra de distintos tamaños: de 16 bytes para comparar con AES independiente, 128 y 512 bytes para mensajes pequeños, 1 Kbyte para mensaje de tamaño medio y 1 Mbyte como muestra de mensajes grandes.

Entrada	**Total de ciclos**	**Ciclos por bloque**
16 bytes	246	246
128 bytes	577	72.125
512 bytes	$1,875$	58.594
1 Kbytes	$3,334$	52.094
1 Mytes	3.0012×10^6	45.795

Tabla 5.5: Tiempos de ejecución de cifrado con OCB.

Observando los resultados, vemos que la cantidad de ciclos que requiere nuestra implementación es razonable y competitiva con otras implementaciones reportadas, las cuales pueden ser vistas en [6, 24]. Dentro de las últimas publicaciones tenemos implementaciones para el mismo tipo de procesador que reportan desde 9.2 hasta 18.9 ciclos/byte, mientras que la nuestra está por debajo de los 15 ciclos/byte.

Entrada	**Total de ciclos**	**Ciclos por bloque**
16 bytes	245	245
128 bytes	$1,099$	137.375
512 bytes	$3,551$	110.969
1 Kbytes	$6,887$	107.609
1 Mytes	6.7445×10^6	102.913

Tabla 5.6: Tiempos de ejecución de descifrado con OCB.

En lo que respecta al KEM se tenian dos funcionalidades principales por implementar, la función SHA-256 y la aritmética modular, ésta última necesaria para la busqueda de números primos y para la generación de llaves. La función hash es bastante eficiente y dado que se ocupa una sola vez en el esquema y para procesar un solo bloque (dos contando el *padding*) no influye realmente en el tiempo de ejecución.

Por otro lado la aritmética modular implementada no es nada eficiente, hablando exclusivamente del caso de la multiplicación (y por consecuencia la exponenciación), esto debido al algoritmo seleccionado. La implementación reporta un tiempo promedio de 443 μ-segundos por cada multiplicación de 512 bits. Recordemos que la multiplicación y la exponenciación son las operaciones básicas en la busqueda de números primos mediante el algoritmo de Miller-Rabin y aunado a este problema se encuentra el hecho de que los números primos de Sophie Germain son escasos. La conjetura de los primos de Sophie Germain indica que para un número n, la cantidad de números primos SG menores a n se aproxima a $(2 \cdot C \cdot n)/\ln(n)^2$ donde $C \approx 0.66016$ es la constante de los *primos gemelos* [2]. Hablando de los números de 512 bits, la probabilidad de encontrar un número primo SG es de aproximadamente 1/95415.

Las pruebas sobre la implementación realizada, mostraron que en algunos casos se encontraba un primo SG en cuestión de 2 a 5 minutos mientras que en otros casos pasaba de los 60 minutos. Dada la situación y tomando en cuenta que ese primo SG únicamente es necesario en la generación de llaves fue que se decidió crear un archivos con los números primos generados, y en lugar de calcularlo al momento de la ejecución, únicamente se selecciona un número de la lista para reducir los tiempos. Esto no reduce la seguridad del esquema pues cada número primo es capaz de ser útil para un conjunto grande de usuarios debido a la forma en la que se generan las llaves y el nonce.

CAPÍTULO 6

Sobre una aplicación

En el presente capítulo se presentan detalles de la aplicación en la cual se implantó el cifrador híbrido implementado. Se describen los objetivos que se deaseaban cumplir con la aplicación y cuales eran los requisitos que se debían cumplir para ello. Se presentan las funciones especiales que se implementaron para el correcto funcionamiento de la aplicación además de las funcionalidades de la aplicación y el uso de los mismos.

6.1. Confidencialidad e integridad

Desde el inicio de este trabajo se mencionó que dos de los grandes objetivos de la criptografía son el de brindar confidencialidad e integridad de datos. También hemos establecido y comprobado que, haciendo uso del esquema de cifrado híbrido presentado aquí, podemos garantizar ambos objetivos. La confidencialidad (mantener la información secreta) se logra mediante el cifrado de datos y la integridad de éstos (asegurar que no han sido alterados) mediante el código de autenticación. Recuerde también que en nuestro caso se logran ambos en una sola ejecución.

Una vez que se tiene un esquema de cifrado funcionando de forma correcta y que su nivel de seguridad ha sido comprobado el paso a seguir es desarrollar o elegir una aplicación en la cual pueda ser probado, obviamente dicha aplicación debe requerir de los servicios de seguridad que se ofrecen por el esquema. Pensando en las características de la implementación y en la gran variedad de posibles aplicaciones que podian servir para la implantación, se llegó a una lista de tres aplicaciones candidatas, de las cuales se eligió por eliminación: en un servidor FTP, en servicios web y en correo electrónico. Cabe señalar que todas las opciones fueron analizadas a la par y el orden en que se presentan aquí no tiene ninguna relación con la importancia sino se debe a la estructura del escrito.

La primera de la lista en eliminarse fue la opción del servidor FTP, la cual constituía en una implementación bastante sencilla. En esta opción se deseaba instalar un servidor de

archivos que mantuviera un conjunto de archivos, los cuales serían puestos allí (subidos) por los usuarios después de haberlos cifrado. Esos archivos cifrados podrían ser descargados únicamente por aquellas personas que tuvieran la llave correcta para descifrar el archivo. El problema con esta aplicación es que la finalidad principal de un servidor FTP es la de permitir la descarga de contenidos a un gran número de personas y nuestro esquema de cifrado, tal y como está, únicamente permite el cifrado para un sólo destinatario debido al esquema de Diffie-Hellman. Si deseáramos modificarlo para cifrar hacia varios destinatarios caeriamos en el problema del **cifrado broadcast** que es un problema muy extenso por sí sólo y este fue el motivo de su eliminación.

Después llegamos a la opción de implementar algún servicio web que requiriera de seguridad. Ésta proponía la de implementar alguno de los servicios básicos para archivos, como lo es comprimir, en un servidor que permitiera a los usuarios acceder al servicio por medio de internet. Las ventajas aquí eran dos principalmente, una que los usuarios de servicios web normalmente buscan resultados para sí mismos o para entregarlos a alguien más por lo cual cumpliamos con la restricción de un sólo destinatario y la segunda era que a futuro es posible ir agregando nuevos servicios al servidor para hacerlo más completo y atractivo. El problema surge por las limitaciones en cuanto a las habilidades de programación del autor, pues los servicios que podía implementar en el tiempo disponible no eran nada atractivas así que pasamos a la siguiente y ultima opción de la lista.

Y por último llegamos a la opción del correo electrónico que en realidad fue la primera en orden de aparición en la lista de opciones. Aquí se propone añadir los servicios del cifrador a una aplicación de envío y recepción de correos electrónicos para mantener segura la información transmitida. Esta aplicación presenta dos puntos importantes a su favor: el primero es que representa una aplicación bastante útil en la actualidad y su uso es cada vez mayor, y en segundo lugar está que cuando deseamos enviar por este medio información verdaderamente importante como para que necesite mantenerse confidencial, normalmente es enviada a una sóla persona o a un conjunto muy reducido por lo que no habría problemas con nuestro esquema cifrado. Por otro lado esta opción no suponía problemas o desventajas significativos en cuanto a la implementación. Por todos estos motivos es que la aplicación en la cual decidimos implantar el cifrador para probarlo fue la de correos.

A continuación se describen algunos de los requisitos que deben cumplirse par poder tener una buena aplicación para el envío y recepción de correos electrónicos de forma segura.

6.1.1. Correos electrónicos seguros

Comencemos por decir que el objetivo principal es el de cifrar toda la información, si así lo desea el usuario, que se envie por el correo, al igual que descifrar aquellos mensajes recibidos si es el caso, esto de una forma eficiente, sencilla y lo más transparente posible

para el usuario. Para ello se requiere tomar en cuenta algunos puntos escenciales.

Las dos primeras características que debemos tomar en cuenta están relacionadas con la etapa de generación de llaves y con su administración. Debemos establecer la forma en la que se van a establecer esas llaves de usuario, quién las genera y cómo serán distribuidas. Algo importante aquí es que debido a esa transparencia que deseamos, debemos buscar la mejor forma de relacionar cada dirección de correo electrónico con el respectivo archivo que contiene la llave de cifrado o descifrado. También es importante definir en que forma se mantendrán seguras las llaves del usuario.

Como segundo punto se debe buscar la mejor forma y el mejor momento para realizar el cifrado de los mensajes, tanto del texto del mensaje como de los posibles archivos adjuntos. Esto dependerá principalmente de la forma de trabajo de la aplicación de correo en la que se implante el cifrador.

Por último, buscamos que al momento de recibir un mensaje cifrado, éste sea descifrado de forma automática si es que se tiene la llave correcta. Esto se refiere a que la aplicación debe ser capaz de reconocer cuáles de los mensajes recibidos fueron cifrados y cuáles no, y tratarlos de acuerdo a esa clasificación.

6.2. Detalles de la aplicación

Ahora pasemos a describir concretamente las características de la aplicación completa que se generó con el proyecto. Como ya dijimos era necesario tener una aplicación de correo en la cual fuera posible inplantar nuestro esquema de cifrado. La opción surgida de forma directa fue la de utilizar **Pine**, que es el cliente de correo utilizado dentro del área de trabajo en la que se desarrolló este proyecto de tesis.

Pine® (Program for Internet News & Email) es una aplicación basada en texto útil para el manejo y administración de mensajes de correo electrónico. Esta es una herramienta libre desarrollada por la universidad de Washington (http://www.washington.edu/pine/) principalmente para sistemas operativos tipo Unix pero existe la versión *PC-Pine* para el sistema operativo MS Windows®. Aunque comenzó siendo una herramienta diseñada para usuarios sin experiencia, ha ido evolucionando y actualmente soporta una gran cantidad de operaciones y tiene un gran número de opciones que permiten configurarlo y personalizarlo en gran medida.

Dentro de las opciones de configuración se encuentran dos que fueron esenciales para el proyecto, dos tipos filtros. La configuración de Pine se realiza mediante la declaración de un gran número de variables de distintos tipos, unas de ellas se conocen como filtros, las cuales permiten declarar acciones que se pueden aplicar a los mensajes al enviarlos o recibirlos. La primera de esas variables es **sending-filters**, ésta nos permite declarar

uno o más filtros (programas o scripts) que pueden ser aplicados a los mensaje salientes. Cuando esa lista de filtros no está vacia, al momento de mandar un mensaje se le permitirá al usuario seleccionar entre los filtros declarados. Además existen algunos valores predeterminados que podemos usar como argumentos de entrada en línea de comandos para los filtros, los tres valores que nos interesan son:

- _RECIPIENTS_: es reemplazado por la lista de recipientes del mensaje.
- _TMPFILE_: es reemplazado por la ruta y nombre del archivo temporal que contiene el texto del mensaje, el que va a ser procesado por el filtro.
- _RESULTFILE_: es reemplazado por la ruta y nombre del archivo temporal que contiene el texto que se presenta en los mensajes de notificación.

La segunda de las variables de configuración que nos interesan es **display-filters** y nos permite declarar una lista de filtros que pueden ser aplicados a los mensajes entrantes. El formato para declarar la lista de filtros es <*desencadenador*><*filtro*><*argumentos*>. El *filtro* es el programa o script que deseamos ejecutar. Los *argumentos* son los mismos valores predefinidos que en el caso de los *sending-filters*, excepto los recipientes:

- _TMPFILE_: es reemplazado por la ruta y nombre del archivo temporal que contiene el texto del mensaje, el que va a ser procesado por el filtro.
- _RESULTFILE_: es reemplazado por la ruta y nombre del archivo temporal que contiene el texto que se presenta en los mensajes de notificación.

Y el *desencadenador* es una cadena de texto que, en caso de ser encontrada en el mensaje, ejecutará el filtro. El desencadenador puede ser definido de cuatro formas:

- _CHARSET(*cadena*)_: el filtro se ejecuta si el texto está en el conjunto de caracteres especificado por *cadena*.
- _LEADING(*cadena*)_: el filtro se ejecuta si *cadena* está en el primer no espacio en blanco del texto.
- _BEGINNING(*cadena*)_: el filtro se ejecuta si *cadena* está al inicio de cualquier línea del texto.
- _*cadena*_: el filtro se ejecuta si *cadena* es encontrada en cualquier parte dentro del texto.

Gracias a la posibilidad de configurar estos filtros, y de forma muy sencilla, es que Pine ofrece grandes ventajas para este proyecto. No fue necesario conocer a fondo el funcionamiento de la herramienta y mucho menos introducirse a nivel del código para poder agregar el cifrador.

6.3. Pine seguro

Por las características y funcionalidades que ofrece Pine fue bastante sencilla la integración del esquema de cifrado para poder crear una aplicación de correo segura. Las pruebas se realizaron con la versión 4.64 (la última) pero no debería existir problemas con versiones anteriores debido al manejo externo para el cifrado. Una vez que tiene instalada la herramienta de correo debe conseguir los archivos de instalación de nuestro cifrador e instalarlo dentro de la cuenta de usuario con la que utiliza Pine.

La instalación crea la estructura de carpetas y archivos necesaria para el funcionamiento del cifrador. Dicha instalación pone en funcionamiento distintos comandos que le brindan al usuario los tres servicios principales, a saber, la administración de llaves, el cifrado de archivos y el cifrado de correos electrónicos. Como ya hemos visto el primer paso es la creación de las llaves para la cuenta ya que sin el par de llaves no podremos hacer nada. Además se tienen las funciones para agregar y eliminar llaves del anillo de llaves públicas así como volver a generar las llaves tantas veces como el usuario desee o requiera.

En cuanto al cifrado de archivos se permite al usuario cifrar cualquier archivo local siempre y cuando tenga la llave pública del destinatario, que puede ser él mismo. Para el descifrado de archivos únicamente requiere de poseer la llave privada correcta, es decir, la correspondiente llave par de la llave pública con la que se cifró el archivo. Estos archivos cifrados pueden ser enviados o entregados al recipiente por cualquier medio, ya sea correo electrónico, por descarga de un servidor, o directamente en algún medio de almacenamiento.

Por último tenemos la funcionalidad que más nos interesa para la aplicación, el cifrado de mensajes de correo electrónico. Lo primero que debemos hacer es, una vez instalada la aplicación, configurar los filtros que mencionamos anteriormente para que Pine reconozca la aplicación. La forma más sencilla es ir directamente al archivo de configuración de Pine, el cual se llama **.pinerc** y se encuentra en la carpeta *home* del usuario, y localizar en él las líneas donde están las variables *sending-filters* y *display-filters*; la otra forma es ejecutar Pine y entrar al *setup*, ir al apartado de configuración y desplazarse en la lista que se presenta hasta localizar ambas variables.

El filtro para envíos se configura únicamente escribiendo la ruta y el comando **EncFilter** junto con los tres parámetros mencionados en la sección anterior. Debido a que será el único filtro declarado en la lista (a menos que el usuario tenga más filtros) al momento de enviar un correo, el usuario deberá elegir entre enviar el correo sin filtrar, la opción por defecto, o filtrar a través de EncFilter. Hay algo que se debe tener en cuenta, el hecho de que el comando para cifrar archivos y el comando para cifrar correos son distintos, esto debido a los requerimiento del formato del correo electrónico. El problema es que el contenido del mensaje de texto (únicamente el texto del mensaje) debe contener sólo caracteres ASCII, no valores binarios que es lo que regresa un texto cifrado. Por tal mo-

tivo es necesario agregar una transformación extra al cifrado para correos, a este tipo de funciones se les llama MIME.

MIME (Multipurpose Internet Mail Extensions) es un conjunto de especificaciones, definidas en documentos RFC, cuyo principal objetivo es el de normalizar el intercambio de información vía red en sus distintos formatos (texto, imagen, audio, etc.) para de estar forma eliminar el problema de compatibilidad con el protocolo SMTP (Simple Mail Transport Protocol) que soporta sólo caracteres ASCII de 7 bits. Actualmente se ha trasladado el uso de MIME a la web con HTTP. El uso de MIME se basa en encabezados que, entre otros datos, indican al protocolo y a las aplicaciones el tipo de contenido del mensaje transmitido, la versión de MIME y el método de codificación utilizados. Dentro de los métodos de codificación existentes que sirven para usarse con SMTP están tres:

- **7bit**: es el más antiguo y usado por defecto, los caracteres usados están en el rango entre 1..127 con retorno de carro y nueva línea que solo pueden aparecer como parte de un fin de línea.
- **Quoted printable**: usado para codificar secuencias arbitrarias, aunque en su mayoria caracteres ASCII de 7 bits con algunos caracteres de 8 bits y donde cada carácter de 8-bits es codificado en tres caracteres de 7 bits (el signo igual y el valor hexadecimal del carácter). Este es el caso de idiomas occidentales de la categoría $ISO - 8859 - n$ como el español.
- **base64**: tiene el propósito de ser usado con datos que no sean de texto o textos que contengan pocos valores dentro del rango de ASCII. Este formato hace uso de un conjunto de 64 caracteres (10 son dígitos, 52 para el alfabeto en mayúsculas y minúsculas, y los signos + y /) para representar cualquier mensaje de entrada.

Debido a que Pine agrega ese tipo de encabezados para los archivos adjuntos pero no para el texto del mensaje se tuvo que implementar y agregarlo como un paso final en el cifrado de correos. El tipo de codificación que se realiza es *base64*.

Base64 [11] permite codificar cualquier mensaje de entrada en uno representado sólo por caracteres de 7 bits, el cual tendrá un tamaño aproximadamente 33 % mayor al original. El proceso que sigue es tomar un grupo de 24 bits (3 grupos de 8 bits) del mensaje original, reacomodarlo como 4 grupos de 6 bits (4 enteros de 0 a 63) y codificarlos en una secuencia de 4 caracteres escogidos de un alfabeto de 64. La tabla 6.1 muestra cada uno de los 64 números con su respectivo caracter de codificación. Además de los caracteres definidos, se utiliza el caracter "=" para los casos especiales cuando la longitud del mensaje de entrada no es multiplo de 24 bits. En ese caso se debe realizar un *padding*, es decir, se deben concatenar los signos necesarios para que la cantidad de caracteres del mensaje codificado final sea multiplo de 4; para formar el último grupo de 4 caracteres se concatenan los dos caracteres correspondientes y dos "=" en caso de que el mensaje de entrada tenga 8 bits extras, o se concatenan los tres caracteres correspondientes y un "=" en caso de que el mensaje de entrada tenga 16 bits extras.

Valor	**Código**	**Valor**	**Código**	**Valor**	**Código**	**Valor**	**Código**
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Tabla 6.1: Alfabeto de codificación para **base64**

Para la decodificación de base64 se realiza el proceso inverso. Se lee un conjunto de 4 caracteres del mensaje codificado, se obtienen los correspondientes números o índices de la tabla (4 enteros de 6 bits) y se concatenan para formar una cadena de 24 bits (3 grupos de 8 bits) que serán escritos en el mensaje decodificado.

Regresando a la configuración de los filtros para Pine, para configurar *display-filters*, damos la ruta y el comando **DecFilter**, y los parámetros serán los dos que se definieron ya para esta variable. En lo que respecta al desencadenador, usamos la cuarta opción y el valor de la cadena es *DHIES*. Aunque el uso de los primeros tres formatos reduce el tiempo de búsqueda de la cadena, la forma en la que fue implementado el filtro de cifrado elimina el problema de retardos. Durante el cifrado del mensaje se agrega la cadena *_DHIES_* al inicio del texto, por lo cual cuando se recibe el mensaje la localización de dicha cadena es inmediata y se manda ejecutar el filtro en ese momento, aplicando la decodificación de base64 y después el descifrado. Si quien recibe tiene la llave privada correcta el mensaje será descifrado.

Para ver detalles sobre la instalación, configuración y el uso de los servicios disponibles vease el apéndice C.

Dentro de las ventajas que ofrece el uso de Pine junto con el esquema de cifrado están el hecho de que obtenemos una comunicación segura para transmitir cualquier información, que estamos usando un cifrador cuyo nivel de seguridad está comprobado, y que por el tamaño común de un correo electrónico, del orden de algunos Kbytes, y por la eficiencia

del cifrador, el uso de éste es transparente al usuario.

En cuanto a las desventajas tenemos principalmente que los archivos que deseen adjuntarse al mensaje deben ser cifrados con anterioridad y que cada mensaje cifrado únicamente puede enviarse a un sólo destinatario. Esto último no representa un gran problema ya que como habiamos mencionado si deseamos compartir información que requiera de ser cifrada, no la distribuimos a un gran número de personas.

CAPÍTULO 7

Conclusiones

El trabajo del pensamiento se parece a la perforación de un pozo.
El agua es turbia al principio pero con el tiempo se clarifica.
Proverbio

En este capítulo se presentan las conclusiones ha las que se llegó con este trabajo de investigación. Primero se presentan las conclusiones en base al conocimiento adquirido y a los resultados obtenidos, y posteriormente se mencionan una serie de líneas posibles para el trabajo futuro.

7.1. Conclusiones

Durante el periodo de realización de este trabajo de investigación se llevaron a cabo una serie de actividades distintas que permitieron alcanzar los dos objetivos principales, implementar un esquema de cifrado híbrido que fuera seguro e implantarlo en alguna aplicación de uso común. Dentro de dichas actividades se encuentran una serie de implementaciones que permitieron construir el esquema completo y algunas de ellas quedan como base para futuros proyectos.

Como resultado quedan dos biblioteca básicas. Una con operaciones de aritmética modular y otra para la generación de números primos del tipo conocido como Sophie-Germain, esta última que usa las operaciones de la primera. Aunque estas implementaciones no son eficientes, sirvieron para tener experiencia en el área y quedan como base para realizar nuevas implementaciones que resuelvan el problema, debido principalmente a los algoritmos utilizados. También queda la implementación del cifrador simétrico AES-256, usando el modo de operación OCB para añadir autenticación, la cual reporta tiempos competitivos con implementaciones reportadas anteriormente, y queda la implementación de la función SHA-256.

Por otro lado, se dejan disponibles la implementación del cifrador híbrido basado en DHIES, el cual cuenta con la demostración de seguridad contra ataques de texto cifrado elegido adaptable bajo el modelo de oráculo aleatorio. Dicha implementación puede ser usada de forma independiente para el cifrado de archivos y puede ser usada conjuntamente con la aplicación Pine para el envío y recepción de correos electrónicos de forma segura. Debido a los tiempos reportados, no se recomienda el uso de este cifrador en aplicaciones críticas, es decir, en aquellas donde los tiempos de ejecución son muy estrictos o en las que se manejan cantidades de información demasiado grandes.

Como se mencionó al inicio, en los últimos años se ha incrementado considerablemente el número de estudios en el área del cifrado híbrido pero hasta donde se sabe, al momento de realizar este trabajo, no existen reportes de implementación sobre esquemas híbridos, por lo que este trabajo abre el camino a nuevas implementaciones. Además de las implementaciones, en esta investigación se realizó un análisis comparativo de los esquemas de cifrado híbrido existentes, con respecto a su construcción, lo que permitió mejorar el esquema final.

En base al análisis realizado y a las implementaciones hechas, podemos decir que un cifrador híbrido observa ciertas ventajas sobre un cifrador de llave pública pues, aunque sigue necesitando de una autoridad certificadora, su eficiencia es mucho mayor. También ofrece ventajas sobre el cifrado simétrico pues cuenta con una mayor facilidad de comunicación y con un nivel de eficiencia bastante competitivo, en nuestro caso sólo agrega el cálculo de dos exponenciaciones modulares.

Otras conclusiones sobre las ventajas del esquema propuesto son que los esquemas que basan su construcción en el intercambio de Diffie-Hellman cuentan con una mayor sencillez, además de ser eficientes para el establecimiento de la llave secreta. Estos esquemas tienen una ventaja extra, ofrecen el mismo nivel de seguridad a pesar de su sencillez, y las demostraciones son más directas y basadas en suposiciones bien conocidas y aceptadas. También el uso de un *nonce* representa ventajas, por su fácil generación, manipulación y comunicación. Y no olvidemos el uso del cifrado con autenticación, que se considera más eficiente que tener dos algoritmos separados.

7.2. Trabajo futuro

El cifrado híbrido tiene dos grandes retos, tener una eficiencia igual o mejor a la de los esquemás simétricos comúnmente usados y que los esquemas tengan seguridad demostrable. El segundo punto depende directamente de las funciones y estructuras usadas en su diseño y construcción. El primero depende además de la habilidad que tengamos para implementar dichas funciones. En este sentido este trabajo necesita mejorar la implementación de la biblioteca de operaciones aritméticas, lo que mejoraría los tiempos de generación de los números primos.

En cuanto a las mejoras directas de la aplicación hay dos puntos que pueden desarrollarse, que aunque no son de interés científico, mejorarían algunos aspectos prácticos: primero está encontrar la forma de cifrar automáticamente los archivos que se adjunten al mensaje y en segundo lugar está el hecho de hacerlo compatible con otras aplicaciones o servidores de correo electrónico, específicamente hablamos de los usados por la mayoría de las personas (hotmail, yahoo, gmail, etc).

Por último está una línea de investigación que parece estar tomando fuerza actualmente, nos referimos al cifrado basado en identidad. Se podrían crear nuevos esquemas híbridos basado ya no en esquemas de llave pública sino en esquemas HIBE. La principal ventaja que observamos en esta nueva área es que podriamos realizar cifrado broadcast, es decir, cifrar un mismo mensaje a más de un destinatario. Como consecuencia directa, nos permitiría utilizar este tipo de esquemas de cifrado en aplicaciones de cómputo distribuido, un área con grandes demandas en cuanto a seguridad. De igual forma se pueden investigar otro tipo de esquema que permita cifrado broadcast para tomarlos como base.

APÉNDICE A

Conceptos preliminares

En este apéndice encontrará algunas definiciones necesarias para comprender mejor algunas secciones de este manuscrito. La información y conceptos presentados aquí fueron tomados principalmente de [16] y [45], puede dirigise a ellos para mayores referencias.

A.1. Grupo multiplicativo

Dado un conjunto G no vacio de elementos y una operación binaria definida sobre los elementos del conjunto G denotada por $*$ y llamada producto, se considera a este objeto algebraico como un **grupo multiplicativo** $< G, * >$ si cumple con las siguiente propiedades:

- *Cerradura*: $\forall\ a, b \in G: \quad a * b \in G$
- *Asociatividad*: $\forall\ a, b, c \in G: \quad (a * b) * c = a * (b * c)$
- *Elemento identidad*: existe un $e \in G$ tal que $\forall\ a \in G: \quad a * e = e * a = a$
- *Inversos*: $\forall\ a \in G$ existe un $a' \in G$ tal que $a * a' = a' * a = e$

A.2. Grupo finito

Una característica importante de cualquier grupo es el número de elementos que lo compone, al cual se le conoce como el *orden del grupo* y está denotado por $\circ(G)$. Cuando el orden $\circ(G)$ es un número finito, se dice que tenemos un **grupo finito**.

A.3. Grupo abeliano

Un **grupo abeliano** esta formado por un conjunto no vacio G y una operación binaria $*$, que además de cumplir con las propiedades de un *grupo* debe cumplir con

✯ *Conmutatividad*: $\forall\ a, b \in G: \quad a * b = b * a$

A.4. Grupo cíclico

Definamos un grupo G. Dado un $a \in G$ definimos $\langle a \rangle = \{az : z \in \mathbb{Z}\}$. Afirmamos que $\langle a \rangle$ es el *subgrupo* más pequeño de G que contiene al elemento a, es decir, cualquier subgrupo Hde G que contenga a deberá contener $\langle a \rangle$. El subgrupo $\langle a \rangle$ se dice que es generado por a.

El grupo G se considera un **grupo cíclico** si $G = \langle a \rangle$ para algún $a \in G$, en cuyo caso a es un generador de G. Es fácil ver que $\circ(G) = \circ(\langle a \rangle)$. Adicionalmente podemos afirmar que si $\circ(G)$ es un número primo, entonces G siempre será un grupo cíclico.

A.5. Subgrupo

Dado un subconjunto no vacio H de cierto grupo G, se dice que H es un **subgrupo de G** si bajo la operación $(*)$ definida en G, H forma también un grupo. En otras palabras, H es un subgrupo si cumple con las propiedades de *cerradura* e *inversos* bajo el producto $(*)$ de G.

APÉNDICE B

Algoritmos

B.1. AES-256

El *Estándar de Cifrado Avanzado*(AES por sus siglas en ingles) especifíca un algoritmo criptográfico que puede ser usado para proteger información electrónica. AES es un cifrador simétrico por bloques que puede cifrar los datos (hacerlos ininteligibles) y descifrarlos (regresarlos a su forma original).

El algoritmo es capaz de procesar bloques de datos de 128 bits haciendo uso de llaves de 128, 192 o 256 bits. Aquí únicamente se presenta la versión para llaves de 256 bits tomada directamente de [28].

Para los algoritmos presentados aquí, la longitud de 128 bits del bloque de entrada, del bloque de salida y del estado intermedio del mismo está representado por **Nb=4**. La longitud de la llave de cifrado **K** es de 256 bits y es representada por **Nk=8**, que es el número de columnas o de palabras de 4 bytes. El número de rondas ejecutadas es de 14, representada por **Nr=14**.

B.1.1. Expansion de llave

Esta es una rutina que recibe la llave K y sirve para generar un total de $Nb(Nr+1)$ palabras denotadas por $[w_i]$. Tanto en el cifrado como en el descifrado cada ronda requiere un conjunto de Nb palaras para usarlas como *llave de ronda*. En la tabla B.1 se presenta el algoritmo de expansión de llave.

La función **SubWord** se encarga de sustituir los bytes en base a la tabla de sustitución de la tabla B.1. La función **RotWord** realiza una permutación cíclica a la palabra que recibe, es decir, si recibe la palabra $[a_0, a_1, a_2, a_3]$ regresa la palabra $[a_1, a_2, a_3, a_0]$. *tmp* es una palabra que almacena valores temporales.

```
entrada: K[4 * Nk]
i = 0
Mientras (i < 8) {
    W[i] = (K[4 * i], K[4 * i + 1], K[4 * i + 2], K[4 * i + 3])
    i = i + 1 }
i = 8
Mientras (i < Nb * (Nr + 1)) {
    tmp = W[i − 1]
    Si (i mód Nk = 0) entonces {
        tmp = SubWord(RotWord(tmp))XORRcon[i/Nk] }
    Sino, si (Nk > 6) y (i mód Nk = 4) entonces {
        tmp = SubWord(tmp) }
    W[i] = W[i − Nk]XORtmp
    i = i + 1 }
salida: W[Nb * (Nr + 1)]
```

Tabla B.1: Algoritmo de generación de llaves de ronda para **AES**.

B.1.2. Cifrado

Este algoritmo de cifrado, presentado en la tabla B.2 se compone de una serie de $Nr+1$ rondas de transformaciones. En la ronda inicial se aplica una sola transformación llamada **AddRoundKey** mientras que la ronda iterativa se aplica $Nr-1$ veces y se compone de 4 transformaciones: **SubBytes**, **ShiftRows**, **MixColumns** y **AddRoundKey**. La ronda final esta compuesta casi como la ronda iterativa, a excepción de que no incluye **MixColumns**. Cada una de estas transformaciones de se describen más adelante.

La variable *sta* dentro del algoritmo de cifrado almacena los valores intermedios de cada operacion. M y TC corresponden respectivamente al texto en claro que se recibe de entrada y al texto cifrado que se produce. W contiene las palabras generadas con el algoritmo de la tabla B.1.

A continuación, se presenta la descripción de cada una de las transformaciones.

SubBytes()

Esta es una transformación que opera independientemente en cada byte de *sta*. Realiza una sustitución no lineal de los bytes en base a una tabla de sustitución(S-box), presentada en la figura B.1 con los valores hexadecimales correspondientes. Como ejemplo suponga que el algún elemento de *sta* contiene el valor (53), entonces al aplicar SubBytes buscamos la fila $x = 5$ y después la columna $y = 3$ y dicho elemento será reemplazado por el valor (ed).

entrada: $\mathbf{M}[4 * Nb]$, $\mathbf{W}[Nb * (Nr + 1)]$
$sta = M$
AddRoundKey$(sta, W[0, Nb - 1])$
Para $r = 1$ por 1 hasta $Nr - 1$ hacer {
    SubBytes(sta)
    ShiftRows(sta)
    MixColumns(sta)
    AddRoundKey$(sta, W[r * Nb, (r + 1) * Nb - 1])$ }
SubBytes(sta)
ShiftRows(sta)
AddRoundKey$(sta, W[Nr * Nb, (Nr + 1) * Nb - 1])$
$TC = sta$
salida: $\mathbf{TC}[4 * Nb]$

Tabla B.2: Algoritmo de cifrado por **AES**.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura B.1: Tabla de sustitución (S-box) en formato hexadecimal.

ShiftRows()

En esta transformación las tres últimas filas de *sta* son desplazadas con distinto número de bytes, mientras que la fila inicial permanece igual. El desplazamiento es cíclico en dirección de derecha a izquierda, así pues suponga que *sta* está compuesto por las filas

$$\begin{bmatrix} s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3} \end{bmatrix}$$
$$\begin{bmatrix} s_{1,0}, s_{1,1}, s_{1,2}, s_{1,3} \end{bmatrix}$$
$$\begin{bmatrix} s_{2,0}, s_{2,1}, s_{2,2}, s_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} s_{3,0}, s_{3,1}, s_{3,2}, s_{3,3} \end{bmatrix}$$

donde para cada $s_{r,c}$, r indica el número de la fila y c la posición dentro de la fila (la columna). Al aplicar ShiftRows, se desplazan los elementos tantas posiciones como indique el número r de la fila, entonces nuestro *sta* quedará formado por

$$[s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}]$$
$$[s_{1,1}, s_{1,2}, s_{1,3}, s_{1,0}]$$
$$[s_{2,2}, s_{2,3}, s_{2,0}, s_{2,1}]$$
$$[s_{3,3}, s_{3,0}, s_{3,1}, s_{3,2}].$$

MixColumns

Esta transformación opera sobre *sta* columna a columna y cada una es vista como un polinomio de 4 términos dentro del campo finito $GF(2^8)$. La operación puede ser vista ya sea como una multiplicación de matrices o como una multiplicación de polinomios módulo $x^4 + 1$ dentro del campo. La matriz por la que se debe multiplicar es

$$\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

obteniendo una nueva columna por la que será reemplazada.

AddRoundKey()

En esta transformación simplemente se "suma" *sta* y la correspondiente llave de ronda mediante una función XOR. Esto es, a cada columna de *sta* compuesta por $[s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}]$ se suma la palabra W_{r*Nb+c} donde $0 \leq c < 4$ y r idica el número de la ronda.

B.1.3. Descifrado

Para el descifrado o *cifrado inverso* se debe invertir el orden de las transformaciones e invertir el proceso individual de cada una de ellas. La funciones que se van a utilizar aquí son: **InvShiftRows**, **InvSubBytes**, **InvMixColumns** y **AddRoundKey**.

El algoritmo de descifrado se presenta en la tabla B.3. Nuevamente *sta* contiene los datos intermedios entre cada transformación y W contiene las llaves de ronda generadas en B.1.1. La diferencia está en que ahora se recibe un texto cifrado TC y se retorna el texto en claro M.

InvShiftRows()

Como transformación inversa de ShiftRows, también las tres últimas filas de *sta* son desplazadas de forma cíclica cierto número de bytes y la fila inicial permanece igual, pero ahora en dirección de izquierda a derecha. Suponga que *sta* está compuesto por las filas

$$[s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}]$$
$$[s_{1,0}, s_{1,1}, s_{1,2}, s_{1,3}]$$
$$[s_{2,0}, s_{2,1}, s_{2,2}, s_{2,3}]$$
$$[s_{3,0}, s_{3,1}, s_{3,2}, s_{3,3}]$$

`entrada:` $\mathbf{TC}[4 * Nb]$, $\mathbf{W}[Nb * (Nr + 1)]$
$sta = TC$
AddRoundKey$(sta, W[Nr * Nb, (Nr + 1) * Nb - 1])$
Para $r = Nr - 1$ por -1 hasta 1 hacer {
InvShiftRows(sta)
InvSubBytes(sta)
AddRoundKey$(sta, W[r * Nb, (r + 1) * Nb - 1])$
MixColumns(sta) }
ShiftRows(sta)
SubBytes(sta)
AddRoundKey$(sta, W[0, Nb - 1])$
$TC = sta$
`salida:` $\mathbf{M}[4 * Nb]$

Tabla B.3: Algoritmo de descifrado por **AES**.

donde para cada $s_{r,c}$, r indica el número de la fila y c la posición dentro de la fila (la columna). Al aplicar la transformación, se desplazan los elementos tantas posiciones como indique el número r de la fila, entonces nuestro *sta* quedará formado por

$$\begin{bmatrix} s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3} \\ s_{1,3}, s_{1,0}, s_{1,1}, s_{1,2} \\ s_{2,2}, s_{2,3}, s_{2,0}, s_{2,1} \\ s_{3,1}, s_{3,2}, s_{3,3}, s_{3,0} \end{bmatrix}.$$

InvSubBytes()

Esta transformación inversa opera sobre cada byte de *sta* de la misma forma que SubBytes solo que utiliza la tabla de sustitución inversa(Inv S-box), que se presenta en la figura B.2 con los valores hexadecimales correspondientes.

InvMixColumns()

Al igual que con MixColumns, aquí también se opera sobre *sta* columna a columna y cada una es vista como un polinomio de 4 términos dentro del campo finito $GF(2^8)$. También la operación puede ser vista ya sea como una multiplicación de matrices o como una multiplicación de polinomios módulo $x^4 + 1$ dentro del campo. La diferencia es que ahora matriz por la que se debe multiplicar es

$$\begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

obteniendo una nueva columna por la que será reemplazada.

		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	e9	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	18	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

Figura B.2: Tabla de sustitución inversa (Inv S-box) en formato hexadecimal.

AddRoundKey()

Por último, la transformación AddRoundKey es su propia inversa ya que sólo implica una operación XOR.

APÉNDICE C

Manuales

C.1. Manual de instalación

- Primero debe obtener el paquete "Paq_DHIES-Mv1.2.xpkg" que contiene la aplicación y el archivo ejecutable "unxpack" para desempaquetarlo. Coloque ambos en una misma carpeta dentro de su sistema.
- En una terminal (shell) dirijase a la carpeta donde colocó los archivos y ejecute el desempaquetador:

```
@$ ./unxpack
```

- Esto creará un folder oculto llamado "sec_dhies", dentro de su *home* (su carpeta de usuario), la cual contiene todos los archivos necesarios para realizar la instalación. También se creará un archivo ejecutable "install"dentro de la carpeta en la que se encuentra.
- Ejecute el instalador:

```
@$ ./install
```

- Cuando la instalación haya finalizado, la guía o manual de usuario se abrirá de forma automática en la ventana activa de su terminal. El comando con el que se visualiza es *more*.
- Nota:
  - Esta aplicación puede descargarse desde `http://computacion.cs.cinvestav.mx/~jebarron/secure_pine.html`.

C.2. Manual de usuario

C.2.1. Configuración y uso de Pine

- Abra el archivo ".pinerc", que se encuentra en su directorio de usuario (*home*), con algún editor de texto. Ejemplo:

```
@$ vim ~/.pinerc
```

- Encuentre la linea que contiene la variable "display-filters=" y escriba lo siguiente frente a ella:

```
_DHIES_ HOME/.sec_dhies/bin/DecFilter _TMPFILE_ _RESULTFILE_
```

- Encuentre la linea que contiene la variable "sending-filters=" y escriba lo siguiente frente a ella:

```
HOME/.sec_dhies/bin/EncFilter _RECIPIENTS_ _TMPFILE_ _RESULTFILE_
```

- Cuando desee cifrar algún mensaje de los que va a enviar, después de presionar $Ctrl + X$ para enviarlo, Pine le va a preguntar si desea enviar el mensaje sin filtrar (*unfiltered*). Presione $Ctrl + N$ para cambiar de filtro y cuando aparezca el texto "filter thru EncFilter" (filtrar con EncFilter) presione Y para enviarlo.

- El descifrado de los mensajes es automático.

- Notas:

  - "HOME" debe ser reemplazado por la ruta completa su directorio de usuario.
  - Todas las veces que se descifre un mensaje recibido, la aplicación le va a solicitar una *contraseña secreta* que debe ser creada al momento de generar el par de llaves del usuario.

C.2.2. Administración de llaves

- Para generar su propio par de llaves, abra una ventana de su terminal (shell) y ejecute el siguiente comando:

```
@$ GenKeys USER_MAIL
```

  esto generará sus llaves privada y pública, y va a almacenar una copia de su llave pública dentro de su carpeta de usuario (home) con el nombre "USER.pk".

- Para agregar una nueva llave pública a su anillo de llaves, usted necesita un archivo "FILE"que contenga la información de la llave. Si ya tiene dicho "FILE'én su sistema, dentro de su terminal (shell) ejecute el comando:

```
@$ Addpk RECIPIENT FILE
```

- Para eliminar llaves públicas de su anillo, en su terminal ejecute el comando:

```
@$ Delpk RECIPIENT
```

- Notas:
  - Cuando cree o renueve sus llaves, la aplicación le solicitará una *contraseña secreta* que servirá para mantener segura su llave privada.
  - "USER_MAIL" debe ser reemplazado por la dirección de correo electrónico que usará.
  - "RECIPIENT" es la dirección de correo electrónico del dueño de la llave pública que va a agregar o eliminar del anillo.
  - "FILE‘" es el archivo (con la ruta y nombre completos) que contiene la información de la llave pública.
  - "USER.pk" es el archivo que deberá distribuir a aquellas personas con quienes desea establecer una comunicación segura.

C.2.3. Cifrado de archivos

- Primero debe dirigirse al folder donde se encuentra el archivo que desea cifrar o descifrar.

- Para cifrar un archivo, abra una terminal y ejecute el comando:

```
@$ EncryptF RECIPIENT FILE
```

  esto va a producir el archivo cifrado "FILE.dhies" dentro del mismo directorio donde se encuentra el archivo "FILE" original.

- Para descifrar, en una terminal tiene que ejecutar el comando:

```
@$ DecryptF FILE
```

- Notas:
  - "RECIPIENT" es la dirección de correo electrónico del usuario a quien desea enviar el archivo.

- ➢ "FILE" es el archivo (con nombre completo) que desea cifrar o descifrar. Para el cifrado puede selecionar cualquier archivo pero para el descifrado necesita que el archivo tenga la extensión ".dhies" (sólo por compatibilidad de nombres).
- ➢ Todas las veces que desee descifrar la aplicación le va a solicitar que introduzca su contraseña (creada durante la generación de llaves).

C.2.4. Notas importantes

- ✯ Con esta aplicación podrá enviar mensajes a sólo un destinatario.
- ✯ Los archivos adjuntos a un correo de Pine no son cifrados con los filtros así que used tendrá que cifarlos primero en una terminal y después adjuntar los archivos ya cifrados al mensaje.
- ✯ También los archivos cifrados que reciba con un correo tendrán que ser descifrados manualmente.
- ✯ Al instalar la aplicación, este manual o guía de usuario puede encontrarla en el directorio "HOME/.sec_dhies/docs".

Bibliografía

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Topics in Cryptology CT-RSA 2001*, pages 143–158, USA, 2001.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 2:781–793, 2002.

[3] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology - ASIACRYPT 2000*, pages 531–545, Japan, 2000.

[4] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption - How to encrypt with RSA. In *Advances in Cryptology - EUROCRYPT 94*, pages 92–112, Berlin, 1995.

[5] Mihir Bellare and Phillip Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *International Conference on Information and Communications Security (ICIS)*, pages 1–16, China, 1997.

[6] Daniel J. Bernstein. AES speed. http://cr.yp.to/aes-speed.html.

[7] David Cash, Eike Kiltz, and Victor Shoup. The twin diffie-hellman problem and applications. In *Advances in Cryptology - EUROCRYPT 08*, pages 127–145, Berlin, 2008.

[8] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33:167–226, 2003.

[9] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[10] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology - CRYPTO 84*, pages 10–18, USA, 1985.

[11] Ned Freed and Nathaniel S. Borenstein. Multipurpose Internet Mail Extensions (MIME) part one: Format of internet message bodies. Request for Comments 1321, 1996.

[12] Rosario Gennaro and Victor Shoup. A note on an encryption scheme of Kurosawa and Desmedt. Cryptology ePrint Archive, Report 2004/194, 2004.

[13] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *ACM symposium on theory of computing - STOC 82*, pages 365–377, New York, USA, 1982.

[14] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system science*, 28:270–299, 1984.

[15] Louis Granboulan. RSA hybrid encryption schemes. Reporte NESSIE: NES/DOC/ENS/WP5/012/3, 2001.

[16] Israel N. Herstein. *Topics in Algebra*. John Wiley & Sons Inc, segunda edition, 1975.

[17] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *Advances in Cryptology - CRYPTO 2007*, pages 553–571, USA, 2007.

[18] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *Advances in Cryptology - EUROCRYPT 2002*, pages 466–481, London, UK, 2002.

[19] Charanjit S. Jutla. Encryption modes with almost free message integrity. *Lecture Notes in Computer Science*, 2045:529–544, 2001.

[20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2008.

[21] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, 1883.

[22] Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In *Advances in Cryptology - CRYPTO 2004*, pages 426–442, Germany, 2004.

[23] Kaoru Kurosawa and Toshihiko Matsuo. How to remove MAC from DHIES. In *ACISP 2004*, volume 3108 of *LNCS*, pages 236–247, Australia, 2004.

[24] Mitsuru Matsui. On the power of bitslice implementation on Intel Core2 processor. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 121–134, Alemania, 2007.

[25] David A. Mcgrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In *Advances in Cryptology - Indocrypt 2004*, pages 343–355, 2004.

[26] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Versión electrónica disponible en http://cacr.math.uwaterloo.ca/hac.

[27] Richard A. Mollin. *An introduction to criptography.* CRC Press, 2001.

[28] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards(FIPS) Publication 197, 2001.

[29] National Institute of Standards and Technology. Secure Hash Standard. Federal Information Processing Standards(FIPS) Publication 180-2, 2002.

[30] Tatsuaki Okamoto and David Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In *Topics in Cryptology CT-RSA 2001*, pages 159–175, USA, 2001.

[31] Tatsuaki Okamoto and David Pointcheval. RSA-REACT: An alternative to RSA-OAEP. Proc. second open NESSIE workshop, 2001.

[32] Le Trieu Phong and Wakaha Ogata. On a variation of Kurosawa-Desmedt encryption scheme. Cryptology ePrint Archive, Report 2006/031, 2006.

[33] Michael O. Rabin. Digital signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS, No. TR 212, Massachusetts Institute of Technology (MIT), 1978.

[34] Ronald Rivest. The MD4 Message Digest Algorithm. Request for Comments 1320, 1992.

[35] Ronald Rivest. The MD5 Message Digest Algorithm. Request for Comments 1321, 1992.

[36] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[37] Phillip Rogaway. OCB mode: Parallelizable authenticated encryption. Contributions to NIST, 2000.

[38] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In *Advances in Cryptology - ASIACRYPT 2004*, pages 16–31, 2004.

[39] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 196–205, USA, 2001.

[40] Palash Sarkar and Sanjit Chatterjee. Construction of a hybrid (hierarchical) identity-based encryption protocol secure against adaptive attacks. Cryptology ePrint Archive, Report 2006/362, 2006.

[41] Palash Sarkar and Sanjit Chatterjee. HIBE with short public parameters without random oracle. In *Advances in Cryptology - ASIACRYPT 2006*, pages 145–160, China, 2006.

[42] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology - CRYPTO 84*, pages 47–53, New York, USA, 1985.

[43] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28:656–715, 1949.

[44] Victor Shoup. A proposal for an ISO standard for public key encryption. Input for Committee ISO/IEC JTC 1/SC 27, 2001.

[45] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[46] Douglas R. Stinson. *Cryptography: theory and practice*. CRC Press, segunda edition, 2002.

[47] Wikipedia. Al-kindi's biography. `http://en.wikipedia.org/wiki/Al-Kindi`.