Universidad Tecnológica Nacional
Facultad Regional Santa Fe

PROTOCOLOS CRIPTOGRÁFICOS BASADOS EN LOS ALGORITMOS DE DIFFIE-HELLMAN Y RSA

Especialización en Ingeniería en Sistemas

Trabajo Final Integrador
2019

Carlos David Colliard Schneider
carloscolliard@gmail.com
Directora: Stella Maris Vaira

Resumen

Este Trabajo Final Integrador de la Especialización en Ingeniería en Sistemas se hace en el marco del proyecto *"Redefinición de algoritmos de encriptación para aprovechar los beneficios de la programación funcional"* del Laboratorio de Análisis, Procesamiento, Almacenamiento y Control de Datos "LAPACDa" - FCyT - UADER.

La importancia del conocimiento de la criptografía radica en su objeto de aplicación que son los datos, brindar las herramientas necesarias para cifrar, intercambiar claves, firmar e integridad a la información que se almacena en los equipos y se trasmite por la red.

Ya desde los inicios de la computación la criptografía tuvo un rol protagónico con la vulneración al código Enigma por parte de Alan Turing (1912 – 1954). Cabe destacar que el trabajo de Alan Turing es considerado precursor de toda la computación moderna que hoy determina nuestra sociedad, nuestras maneras de producir, las relaciones, los medios de transporte, el avance del conocimiento, e incluso nuestra manera de pensar.

Disciplinas como la Criptografía y la Seguridad en Redes disponen de herramientas y protocolos adecuados para la construcción de aplicaciones distribuidas seguras y fiables, aunque necesitan adaptarse a los escenarios exigidos por las nuevas tecnologías: tarjetas inteligentes, e-mail, sistemas biométricos, entre otros.

Actualmente la criptografía basa todos sus métodos en la teoría de números, la estadística y distintas teorías de la información, la criptografía se originó casi en simultáneo con la escritura. Siempre han existido situaciones en las que el hombre ha necesitado comunicar mensajes de vital importancia a sus semejantes, intentado que sus enemigos no los conocieran, ya que estos mensajes solían estar referidos a las estrategias militares que pudieran usar. Con el avance tecnológico, la criptografía se ha convertido en una herramienta esencial para acciones como pagar con una tarjeta bancaria, navegar por internet, realizar transmisiones militares, por sólo mencionar algunas.

Tabla de Contenidos

Tabla de Ilustraciones

Capítulo 1: Introducción

Objetivos

El objetivo principal de este trabajo es ofrecer un panorama general de los protocolos criptográficos basados en los algoritmos de Diffie-Hellman y RSA.

Para poder lograr este objetivo principal se desarrollan los siguientes particulares:

Definir los protocolos criptográficos basados en Diffie-Hellman y RSA.

Describir protocolos criptográficos que se implementan en el almacenamiento y transporte de la información haciendo énfasis en sus técnicas de criptografía de clave pública, con el fin de demostrar el impacto de estos algoritmos en la tecnología actual.

Identificar las diferentes capacidades que brindan estos algoritmos, seguridad, limitaciones y brindar ejemplos del funcionamiento.

Documentar los códigos de las técnicas y algoritmos criptográficos utilizados por los protocolos criptográficos estudiados.

Estructura del trabajo

En el capítulo 1 se expone una introducción a la criptografía, su historia, conceptos básicos y clasificación, marcando las diferencias que presenta la criptografía simétrica con respecto a la criptografía asimétrica y haciendo énfasis en la finalidad de cada una.

Luego, en el capítulo 2 se presenta la teoría numérica necesaria para comprender el funcionamiento de los algoritmos.

En el capítulo 3 se describen los algoritmos de Diffie-Hellman y RSA.

En el capítulo 4 se analizan los protocolos criptográficos, centrándose en sus características concernientes a la criptografía de clave pública.

Se concluye con un análisis integrador del trabajo y la enunciación de futuros temas de investigación relacionados.

Como anexo se desarrollan los algoritmos codificados mencionados en el trabajo.

Aplicaciones

En este trabajo, se presenta una revisión de los diferentes protocolos criptográficos actuales que implementan Diffie-Hellman o RSA. Esto permite que el lector se familiarice con estos protocolos que aparecen regularmente cuando uno trabaja con algún software de comunicación o almacenamiento de datos.

Previo a esto, la descripción del funcionamiento de estos algoritmos permite visualizar las posibles vías que puede utilizar un criptoanalista para vulnerar las implementaciones de los mismos. Se detallan los códigos necesarios para simular un entorno para probar las capacidades de Diffie-Hellman y RSA, que permite al lector realizar un seguimiento de los mismos a la par que lea el trabajo, lo cual generará un mayor entendimiento del tema.

Límites del trabajo

El presente trabajo excluye los siguientes puntos:

- Análisis en profundidad de la criptografía simétrica, solo se la desarrollará brevemente para explicar los conceptos necesarios.
- Algoritmos de generación de números aleatorios y pseudoaleatorios.
- Técnicas de criptoanálisis con el fin de vulnerar los algoritmos de RSA o Diffie-Hellman.

Criterio de selección de algoritmos DH y RSA

Para el trabajo desarrollado se seleccionaron los algoritmos de clave pública Diffie-Hellman y RSA. Si bien no son los únicos algoritmos de este tipo, han sido los que impulsaron el avance de la criptografía en los últimos 40 años con sus ideas referentes a intercambios de clave, cifrado asimétrico y firma digital.

La gran mayoría de los algoritmos asimétricos restantes fueron desarrollados por ideas a partir de estos, como son los casos de Rabin y Elgamal entre otros, que se los utiliza con el fin de cifrado y firma digital.

No obstante su selección no solamente fue debida a lo anteriormente comentado, sino que al día de hoy Diffie-Hellman y RSA siguen vigentes. No solamente se mantienen seguros a nivel teórico, sino que además con sus respectivas adaptaciones e implementaciones en los diferentes protocolos criptográficos son imprescindibles para lograr seguridad en el almacenamiento y transferencia de la información.

Criterio de selección de protocolos criptográficos a estudiar

La selección de protocolos criptográficos se acotó a los siguientes:

SSL/TLS: es un protocolo utilizado para conexiones seguras entre pares a través de los protocolos HTTPS (navegación web), DNS sobre TLS (resolución de nombres), NNTPS (gestión de noticias), SMTPS (correo electrónico), XMPP (mensajería instantánea) y herramientas como OpenVPN (redes virtuales privadas) (Oppliger, 2016) y (Razaghpanah, y otros, 2017).

SSH/SSH2: es un protocolo utilizado para acceder de manera segura a otros dispositivos de la red, enviar comandos a través de consola, transferencia de archivos (SFTP) y en herramientas como Git (control de versiones) (Ylonen T. , 2019) y (McLaren, Russell, Buchanan, & Tan, 2019).

PGP: es un protocolo utilizado para cifrar y firmar archivos y correos electrónicos, firmar paquetes de software en Linux, entre otros (Yakubov, Shbair, & State, 2018).

SMIME: es un protocolo utilizado para correo electrónico y por el protocolo XMPP (Wu & Chiu, 2017).

Protocolos como Kerberos e IPSec no fueron seleccionados por centrarse casi exclusivamente en la criptografía simétrica (Stallings, 2017).

Otros protocolos criptográficos que implementaron algoritmos asimétricos no fueron seleccionados por no ser actuales como SET (Secure Electronic Transaction) o PEM (Privacy Enhanced Mail) (Mogollon, 2008).

Estado del Arte

Hasta hace algunos años la criptografía solo resultaba interesante para agencias de seguridad, agencias gubernamentales y grandes empresas. Sin embargo, en poco tiempo, y debido al rápido crecimiento de las comunicaciones electrónicas, esta ciencia se ha convertido en un tema central que despierta el interés del público en general. Un gran cambio ha sufrido la investigación en criptografía en el último tercio del pasado siglo, ya que ha pasado del tema clásico del cifrado y su seguridad a los más actuales campos de las firmas digitales y los protocolos criptográficos. Dicha variación es una consecuencia inmediata del impacto de las nuevas tecnologías en la sociedad, que cada vez demanda más servicios seguros; claro está que la Matemática jugó y juega un rol fundamental en el avance de la criptografía y, como veremos más adelante, es parte de los protocolos criptográficos, ella ofrece herramientas que permiten analizar, evaluar y gestionarlos.

La aparición de la criptografía asimétrica o de clave pública ha permitido que se desarrollen una serie de nuevas tecnologías como firma digital, autentificación de usuarios y el cifrado de datos sin intercambio previo de secretos (clave privada), que es muy importante, en general, desde las actividades relacionadas al comercio electrónico hasta canales inseguros como Internet.

Existe mucha literatura actual referente a investigaciones realizadas de los protocolos criptográficos seleccionados para este trabajo, como así también de éstos haciendo referencia a la utilización de RSA y Diffie-Hellman y de sus aplicaciones que dan cuenta de la vigencia de estos dos protocolos, a continuación se detallan algunas de las referencias que los involucra.

Algunos de los trabajos hacen referencia a la seguridad, tales como vulnerar claves RSA utilizando Lenstra en un servidor TLS y aprovechando una débil implementación del algoritmo del Teorema Chino del Resto (Weimer, 2015); proponen un algoritmo paralelo con el fin de optimizar el cálculo de la factorización del módulo RSA utilizando la GPU durante un intercambio de claves en TLS (Pineda Vargas, Salcedo Parra, & Acosta Rodríguez , 2017); o exploran las suposiciones del problema del logaritmo discreto las cuales fundamentan la capacidad de Diffie-Hellman y las potenciales consecuencias que esto genera en la seguridad de TLS (Dorey, Chang-Fong, & Essex, 2017).

Trabajos en firma digital, tales como un nuevo esquema de firma digital para SMIME entre los algoritmos seleccionados para este nuevo esquema se encuentra RSA (Raji, Amiri, & Ahmadian, 2016); o un administrador de claves para SSH que utiliza el esquema de firma digital de RSA (Harchol, Abraham, & Pinkas, 2018).

Aplicaciones en correo electrónico como, implementaciones en PGP utilizando RSA para administrar la seguridad de los correos electrónicos dentro de aplicaciones de gobierno digital en un caso de estudio concreto (Suprihanto & Priyambodo, 2017); o ataques novedosos basados en una técnica llamada callmal-leability gadgets para revelar texto plano de correos cifrados con OpenPgp y SMIME (Poddebniak, y otros, 2018).

Trabajos en infraestructura, tales como un framework de criptografía híbrida para cloud computing utilizando una implementación de RSA sobre PGP para el intercambio de información entre el servidor y el cliente (Jothy, Sivakumar, & Delsey, 2017); o describen técnicas de blockchain para mitigar problemas de sincronización y performance en implementaciones de infraestructura de clave pública que utilizan PGP (Roh & Lee, 2018).

Revisiones de RSA con el fin de examinar fortalezas y debilidades, y proponer soluciones para mitigar estas debilidades, enunciando la utilización actual de RSA para cifrar en PGP, Googles G Suite, HTTPS y TLS y firmar digitalmente en PGP, SMIME, y TLS (Nisha & Farik, 2017).

Después de esta revisión, que resalta la importancia de ambos protocolos y el uso actual en diferentes esquemas de seguridad, firma digital y nuevas tecnologías, se presentan los conceptos asociados a la criptografía, su historia y breves descripciones de la criptografía simétrica y criptografía de clave pública. Se presentan dos posibles escenarios para un protocolo criptográfico, según el tipo de claves criptográficas que poseen los usuarios que se están comunicando, por un lado la Criptografía Simétrica, en este caso el emisor y el receptor de la comunicación (sea cifrada y/o autenticada) deben compartir alguna clave, que es secreta para el resto de los usuarios; por otro lado los protocolos asociados a la Criptografía de Clave Pública: en este escenario, cada usuario genera un par de claves: una clave secreta que solamente conoce él, y una clave pública asociada, que pone a disposición del resto de los usuarios.

Conceptos

Se considera importante como introducción establecer las diferencias entre los términos criptografía, criptoanálisis y criptología; en la ilustración 1 se visualiza la relación entre las tres y como la criptología incluye el estudio de la criptografía y el criptoanálisis.

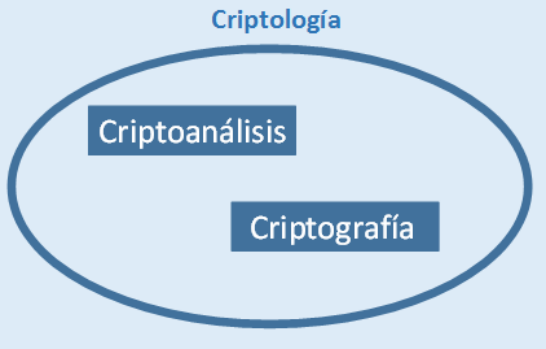

Ilustración 1 Principales componentes de la criptología

La criptografía es la ciencia que se encarga del estudio de aspectos matemáticos relacionados con la seguridad en la información y las comunicaciones. Mientras que el criptoanálisis es el estudio de las técnicas matemáticas para intentar vulnerar implementaciones criptográficas, y generalmente servicios de seguridad de información (Schoenmakers, 2019) y (Lenstra & Verheul, 2001).

En la actualidad, con el avance tecnológico, la criptografía se ha convertido en una herramienta esencial para acciones como pagar con una tarjeta bancaria, navegar por internet, transmisiones militares, voto electrónico, entre otros.

Un protocolo criptográfico es un algoritmo distribuido que describe con precisión las interacciones entre dos o más entidades para lograr ciertos objetivos de seguridad (Schoenmakers, 2019).

Un sistema criptográfico es un conjunto de elementos interrelacionados dinámicamente que tienden a lograr un objetivo de seguridad.

Los protocolos y sistemas criptográficos están presentes en todo momento permitiendo comunicaciones seguras entre entidades sobre canales inseguros, concentrándose en 3 objetivos:

- Confidencialidad: La información no debe ser leída por terceros.
- Integridad: La información debe protegerse ante modificaciones inesperadas.
- Autenticación: Debe ser claro quién es el autor de la información.

Según Shannon en su publicación (Shannon, 1949), la criptografía es una forma de establecer comunicaciones seguras sobre canales inseguros mediante el uso de una hipótesis adicional: un canal que proporcione la seguridad. Este canal se utiliza básicamente para configurar una clave simétrica autenticada y confidencial.

Otro concepto a tener en cuenta es el de criptosistema el cual (Stinson & Paterson, 2018) lo define como *una quíntupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, donde las siguientes condiciones son satisfechas:*

1. $\mathcal{P}$ *es un conjunto finito de los posibles textos planos.*
2. $\mathcal{C}$ *es un conjunto finito de los posibles textos cifrados.*
3. $\mathcal{K}$*, keyspace, es un conjunto finito de las posibles claves.*
4. *Por cada* $K \in \mathcal{K}$*, existe un regla de cifrado* $e_K \in \mathcal{E}$ *y una correspondiente regla de descifrado* $d_K \in \mathcal{D}$*. Cada* $e_K: \mathcal{P} \to \mathcal{C}$ *y* $d_K: \mathcal{C} \to \mathcal{P}$ *son funciones tales que* $d_K(e_K(x)) = x$ *por cada elemento texto plano* $x \in \mathcal{P}$*.*

Historia de la Criptografía

La criptografía inicia como un arte durante el desarrollo de la humanidad y la creación del lenguaje escrito.

Cerca del 2000 a.c. en Egipto se encuentra el primer registro de utilización de una técnica criptográfica, en la cual escribas[1] utilizaron sustitución de símbolos jeroglíficos en una pared de roca en la tumba de un noble.

En el siglo V a. c. los espartanos utilizaban para cifrar mensajes la transposición de caracteres a través de la *Scitala*. Cilindros los cuales se le envolvía un cinto, el cifrado se basaba en la escritura

[1] Escribas: lat. *scriba*. En la Antigüedad, copista, amanuense. (Real Academia Española, 2014, 23º ed.). Recuperado de https://dle.rae.es/?id=GJs2utV

del mensaje en el mismo a través del eje del cilindro y finalizaba desenvolviendo el cinto del cilindro. El descifrado requería un cilindro del mismo diámetro.

En el siglo I en Roma, Julio César utilizó otro sistema criptográfico que reemplazaba cada carácter por otro carácter el cual estaba 3 posiciones después en el alfabeto. Este algoritmo luego fue mejorado para que trabaje con desfasajes de n posiciones.

En 1585 Vigenère en su publicación *Traicte des* Chiffres se basa en el algoritmo de César y describe el cifrado de Vigenère, el cual dividía el mensaje en bloques que tenían la misma longitud que la clave y a cada uno de estos le aplicaba la clave secreta. Esto hacia que no siempre una letra sea transformada en la misma letra y aumentaba la dificultad de romper el código.

Durante la década de 1940, la criptografía fue la primera aplicación que tuvo una computadora con la intención de romper un criptosistema en tiempo real para propósitos militares (Trappe & Washington, 2005).

A principio de la década de 1970 *el Communications Electronics Security Group (CESG)* de Gran Bretaña descubrió las nociones básicas detrás de la criptografía de clave pública, aunque se mantuvo en secreto hasta finales de siglo, y se considera que Diffie y Hellman en 1976 son los pioneros de la misma. Luego en 1978 Rivest, Shamir y Adleman crearon el criptosistema RSA, la primera implementación de clave pública para cifrado y firma digital.

A continuación de una manera visual en la ilustración 2 se resumen estos hitos en la historia de la criptografía, desde las primeras técnicas de sustitución hasta las primeras implementaciones de clave pública:

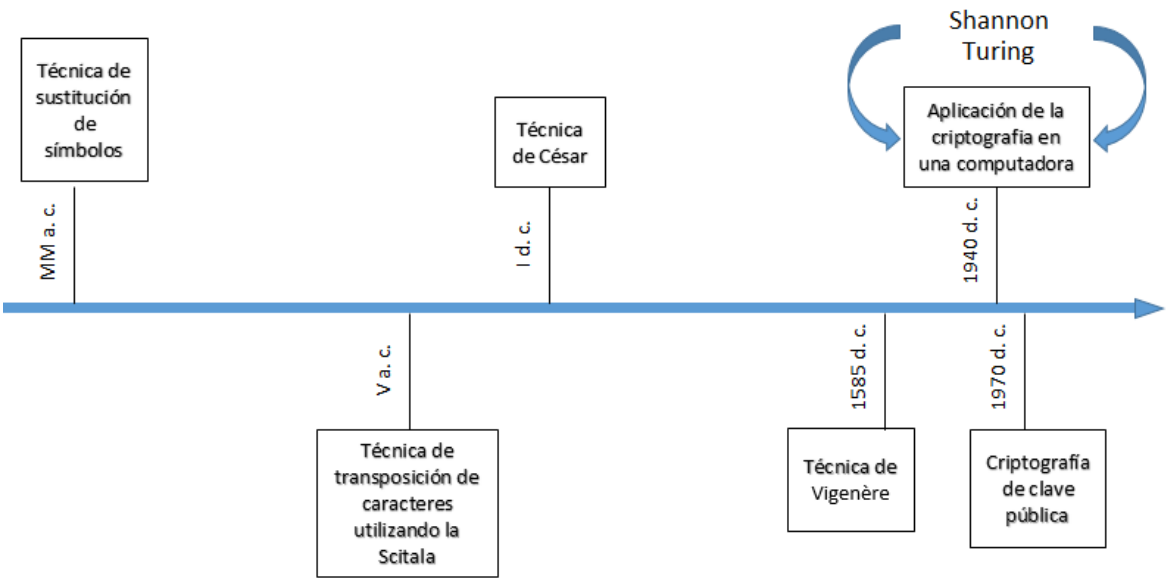

Ilustración 2 Algunas fechas claves de la historia de la evolución de la Criptografía MM a.c. -- 1970 d.c.

La evolución histórica ha mostrado como cambió el modelo de seguridad, en sus inicios se mantenía en secreto el método de cifrado, luego el método se asume que es conocido[2] y la seguridad depende de ocultar la clave secreta. Y por último con la invención de la criptografía asimétrica tanto el método de cifrado como la clave pública están a disposición y solo se mantiene en secreto la clave privada la cual se conoce como obtenerla, no obstante la seguridad es brindada por su característica de computacionalmente inviable.

Criptografía Simétrica

Un criptosistema es llamado simétrico (de clave simétrica, de una clave y convencional) si para cada par (e, d)*, la clave* d *es "computacionalmente sencilla" de determinar conociendo solamente e, y similarmente e conociendo* d (Mollin, 2007)*.*

En la criptografía simétrica se elige una clave secreta K, la cual tiene asociada una regla de cifrado e_k y una regla de descifrado d_k. En la mayoría de los criptosistemas e_k es igual a d_k o derivado de este, de aquí la denominación de criptosistemas simétricos y su fortaleza radica en mantener en secreto ambas reglas.

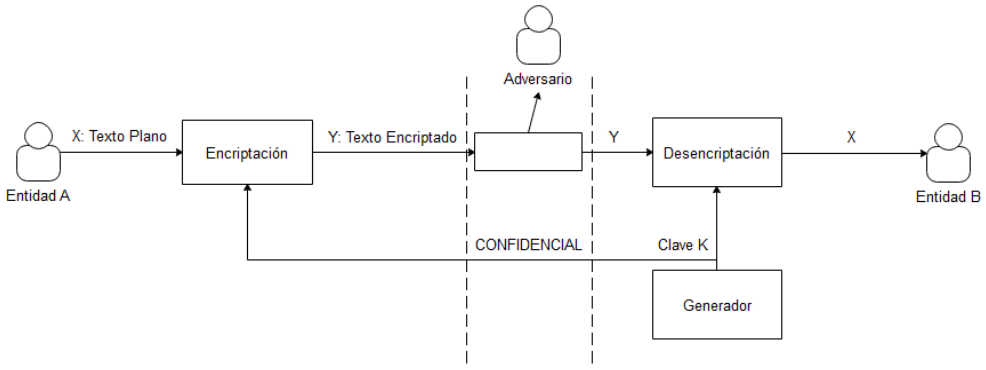

Ilustración 3 Procedimiento de la Criptografía Simétrica. Adaptado de Vaudenay, 2006.

En la criptografía simétrica la comunicación generalmente tiene un procedimiento similar al diagramado por la ilustración 3, en el cual (Vaudenay, 2006):

- Un generador determina la clave K, que es distribuida a las entidades que participarán.
- La entidad A transforma el mensaje X en el mensaje cifrado Y, y lo envía a la entidad B.
- La entidad B realiza el proceso inverso por el cual transforma el mensaje cifrado Y en el mensaje original.
- La comunicación podría ser interceptada por un intruso, el cual al no tener la clave K no podría conocer el significado del mensaje, preservando la confidencialidad del mismo.

Todos los algoritmos desarrollados hasta 1970 son simétricos, a estos algoritmos se los suele denominar criptosistemas clásicos dado que se publicaron antes de la era de la computación. A

[2] Esta suposición se conoce como el principio de Kerckhoffs.

continuación se enuncian algunos de los algoritmos simétricos más importantes que se desarrollaron para implementarse computacionalmente.

Principales Algoritmos Simétricos:

Uno de los algoritmos simétricos más antiguos es del DES (Data Encryption Standard) desarrollado con el objetivo de brindar seguridad a las transacciones bancarias y originalmente publicado por el *National Bureau of Standards* (NBS) en 1973, propuesto como estándar por IBM y adoptado por el gobierno de Estados Unidos. Para mejorar la seguridad de este algoritmo e impulsado por las aplicaciones financieras se propone una mejora en el año 1998 conocida como 3DES que se basa en aplicar tres veces el cifrado DES con claves distintas en cada iteración. Esta mejora no brindó la seguridad requerida y fue rápidamente reemplazada en 2001 por el algoritmo AES (Advanced Encryption Standard) (Boneh & Shoup, 2020).

Criptografía Asimétrica o de Clave Pública

Un criptosistema de clave pública consiste en un conjunto de transformaciones de cifrado $\{E_e\}$ y un conjunto de transformaciones de descifrado $\{D_d\}$. Si para cada par de clave (e, d), la clave de cifrado e, llamada la clave pública, es disponible de manera pública, mientras que la clave de descifrado d, llamada la clave privada se mantiene secreta y es distinta a e, de aquí la denominación como asimétrico.

Siguen el paradigma *"provably secure"*, en el cual la seguridad se basa en la dificultad de resolver un problema que se supone difícil como puede ser la factorización de números grandes o el cálculo de logaritmos discretos.

En la siguiente figura se visualiza el cifrado asimétrico con el modelo propuesto por Shannon, en el cual se transforma un canal inseguro en uno confidencial con la ayuda de la criptografía de clave pública y un canal extra para transmitir esta clave.

En la criptografía asimétrica la comunicación generalmente tiene un proceso similar al diagramado por la ilustración 4:

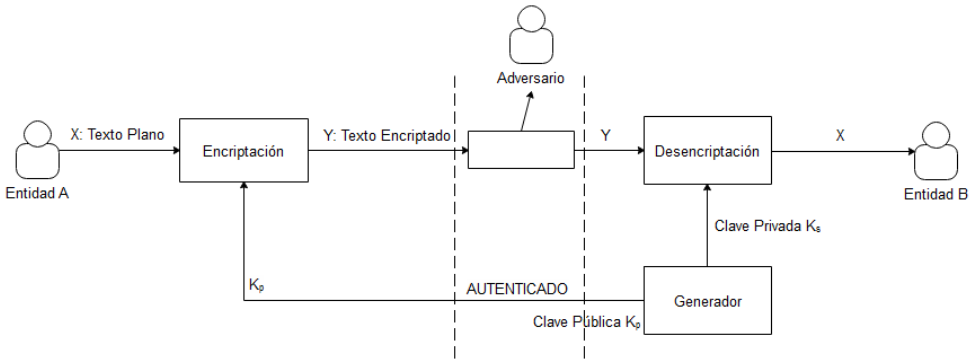

Ilustración 4 Procedimiento de la Criptografía Asimétrica. Adaptado de Vaudenay, 2006.

Formalmente un criptosistema de clave pública se define por (Vaudenay, 2006):

- Un generador Gen de claves pseudoaleatorias, un algoritmo probabilístico el cual arma un par de clave (K_p, K_s) donde K_p es la clave pública y K_s es la clave privada. Este generador debe generar claves de un tamaño considerable y las mismas deben ser aleatorias en el sentido que la probabilidad de un valor seleccionado sea lo suficientemente pequeña para que no se pueda hacer un ataque basándose en la probabilidad de este.
- Un algoritmo Enc de cifrado, el cual dado un texto plano X y una clave pública K_p devuelve un texto cifrado Y.
- Un algoritmo Dec de descifrado, este algoritmo dado el texto cifrado Y y la clave privada K_s devuelve el texto original X.

Ventajas y desventajas de la Criptografía Asimétrica

Ventajas

- Seguridad: Solamente se necesita mantener en secreto la clave privada.
- Longevidad: El par de claves suele ser utilizado sin necesidad de cambiarlo, en la práctica solo se lo cambia si se sospecha que la clave privada ha sido vulnerada, ya sea por cálculo o por acceso indebido al archivo donde estaba almacenada.
- Administración de Claves: Si se tienen n entidades que requieren comunicarse, utilizando un algoritmo simétrico el número de claves requeridas para que sea posible la comunicación entre 2 entidades es $n(n-1)/2$ y cada entidad almacena $n-1$ claves. Mientras que con la asimétrica, solo se requeriría manejar n claves públicas para cualquier comunicación entre 2 entidades
- Intercambio de claves: No es necesario el intercambio de claves en la comunicación entre las entidades.

Desventajas

- Eficiencia: El cifrado simétrico es más rápido y logra cifrar cientos de megabytes por segundo.
- Los tamaños de clave requeridos son significativamente mayores que la contraparte simétrica: Los criptosistemas simétricos usualmente utilizan 64, 128 bits o similares, mientras que los de clave pública rondan los 512, 1024, 2048 y actualmente hasta 4096 bits. Esta diferencia de tamaño se debe a que el tipo de ataque sobre la clave simétrica suele ser fuerza bruta también denominada búsqueda exhaustiva mientras que sobre la clave asimétrica los tipos de ataque utilizan técnicas más eficientes.

- Seguridad no probada: Los esquemas desarrollados basan su seguridad en la presunción de dificultad de un conjunto de problemas matemáticos teóricos, tal modelo de seguridad se conoce como “probablemente seguro” (provably secure).

Principales Algoritmos Asimétricos

- Diffie-Hellman: Publicado en el *IEEE Transactions on Information Theory* de 1976 por Whitfield Diffie y Martin Hellman en el artículo *New Directions in Criptography*, es el primer protocolo práctico de una técnica de intercambio de claves que se publicó y marcó el terreno en criptografía para el desarrollo de ideas como las funciones one-way trapdoor, criptosistemas de clave pública y firma digital.
- RSA: Es un criptosistema de clave pública que fue inventado por Ronald Rivest, Adi Shamir y Leonard Adleman, 3 investigadores cuyas iniciales conducen al nombre abreviado RSA. El cual fue publicado en 1978 en la revista *Communications of the ACM*. Su dificultad está en la factorización de números enteros y ha sido adaptado, generalizado e implementado en varios estándares.
- Rabin: Criptosistema basado en RSA publicado en el año 1979 por Michael Oser Rabin, cuya seguridad se basa en la complejidad de la factorización de números enteros y el problema de la raíz cuadrada modular (Stinson & Paterson, 2018).
- ElGamal: Criptosistema publicado en 1984 por Taher Elgamal, basado en el problema del logaritmo discreto pero a diferencia de Diffie-Hellman brinda posibilidades de cifrado y firma digital (Stinson & Paterson, 2018).

Tanto la criptografía simétrica como asimétrica necesitan en ciertos casos de alguna función que realice un resumen del mensaje o bien un código de integridad que cumpla la tarea de detectar modificaciones en el mensaje, para ello hacen uso de las funciones hash.

Funciones Hash

Una función hash es utilizada para armar una huella (código de verificación) en una cadena de datos, esto permite proteger ante alteraciones de la misma, si la cadena se ve alterada la huella ya no será válida.

Para corroborar la integridad de un mensaje recibido, se compara el hash recibido con uno generado en el momento, si son iguales el mensaje no ha sido alterado.

Se define familia hash *a una cuádrupla* $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$*, donde las siguientes condiciones son satisfechas* (Stinson & Paterson, 2018)*:*

1. $\mathcal{X}$ *es un conjunto de los posibles mensajes, puede ser finito o infinito.*
2. $\mathcal{Y}$ *es un conjunto finito de los posibles mensajes de resumen o etiquetas de autenticación.*
3. $\mathcal{K}$*, keyspace, es un conjunto finito de las posibles claves.*
4. *Por cada* $K \in \mathcal{K}$*, existe una función hash* $h_K \in \mathcal{H}$*, tal que* $h_K: \mathcal{X} \to \mathcal{Y}$*.*

Principales Algoritmos Hash

Entre los principales algoritmos hash se pueden mencionar los creados por Ronald Rivest como lo son el MD4 de 1990 y su mejora el MD5 del año 1992. Como así también la familia de algoritmos hash SHA creados por el NIST a partir de 1995, que significaron una salida de resumen de mensaje de 160 bits, la cual con el correr de los años se ha ido mejorando en SHA-256, SHA-384 y SHA-512, los números indican la cantidad de bits de la salida (Aumasson, 2017).

Esquemas de Firma Digital

Un esquema de firma digital permite a una entidad S que ha establecido una clave pública firmar un mensaje de tal manera que otra entidad que conozca esta clave pueda verificar que el mensaje se originó en S y no ha sido modificado de ninguna manera (Katz & Lindell, 2007).

La humanidad siempre utilizó distintos métodos para asociar su autoría a documentos. En la edad media la nobleza sellaba los documentos con su insignia en cera, y se suponía que la única persona que podía reproducir este sello era la persona propietaria de la insignia.

Con la llegada de las transacciones comerciales electrónicas resulta imposible verificar la propiedad de una tarjeta de crédito u otro medio de pago con solo comparar firmas tradicionales. Además estas firmas se pueden copiar y pegar en otro documento electrónico, característica que las hacen vulnerable y por lo tanto inseguras. Por esto es que se requieren firmas digitales que no puedan ser separadas de un mensaje y añadido a otro sin inconveniente.

Una de las mayores contribuciones de la criptografía de clave pública es la firma digital. En 1991 el NIST *(National Institute of Standard and Technology*) realizó el primer estándar para firmas digitales basado en el esquema de RSA denominado DSS (*Digital Signature Standard*), el cual permitía determinar la autenticidad de un mensaje como así también su integridad.

Un esquema de firma digital se compone de un algoritmo de firma digital $\mathcal{S}_A$ y un algoritmo de verificación digital $\mathcal{V}_A$ (Stinson & Paterson, 2018). Estableciendo una comparación con la criptografía de clave publica $\mathcal{S}_A$ sería la clave privada y $\mathcal{V}_A$ la clave pública.

Suponiendo que la entidad A firmará digitalmente, se cuenta con:

- El conjunto $\mathcal{M}$ denominado espacio de mensajes.
- El conjunto $\mathcal{S}$ de cadenas de bits, llamada espacio de firma.
- El algoritmo de firma digital $\mathcal{S}_A$ es una transformación desde el conjunto de mensajes $\mathcal{M}$ al de las firmas $\mathcal{S}$. Esta función es privada y será usada por A, para crear las firmas del conjunto $\mathcal{M}$.
- El algoritmo de verificación digital $\mathcal{V}_A$ es una transformación del conjunto $(m, s) \rightarrow \{false, true\}$, donde $m \in \mathcal{M}$ y $s \in \mathcal{S}$. $\mathcal{V}_A$ se mantiene público y es utilizado por otras entidades para verificar las firmas creadas por A.

Proceso de firma: Para crear la firma de un mensaje $m \in \mathcal{M}$ la entidad A debe:

1. Computa $s = \mathcal{S}_A(m)$, donde s es la firma de m.
2. Envía el par (m, s).

Proceso de verificación: La entidad B para verificar la firma s del mensaje m de la entidad A debe:

1. Obtener la transformación $\mathcal{V}_A$ de A.
2. Computar $u = \mathcal{V}_A(m, s)$.
3. Si $u = true$ aceptar la firma, en cambio si $u = false$ rechazar la firma.

PKI: Infraestructura de Clave Pública

Una infraestructura de clave pública (PKI), también llamada infraestructura de gestión de claves, es una infraestructura (que consta de hardware, software, personas, políticas y procedimientos) para un entorno distribuido que se ocupa de la gestión (creación, distribución, uso, almacenamiento, y revocación) de certificados digitales (Shirey, 2007).

Los usuarios y dispositivos que utilizan esta infraestructura se denominan entidades, y PKI permite asociar estas entidades a una clave de forma segura.

El organismo que brinda la confianza en la firma del documento y asocia la clave a la entidad se denomina autoridad de certificación (CA), y estos documentos se denominan certificados.

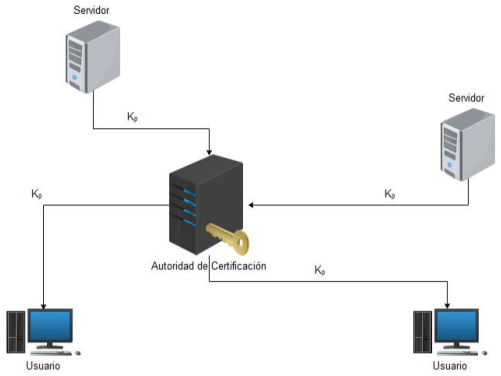

Ilustración 5 Intercambio de claves en Infraestructura de Clave Pública

En la ilustración 5 se observa como los diferentes servidores envían su clave pública a la CA para que ella gestione las claves. Es decir el alta, la baja y/o realizar modificaciones sobre las claves; como así también los clientes pueden consultar las claves de las entidades que requieran.

Certificados

Uno de los retos en la criptografía de clave pública es asegurar la autenticidad de la clave pública, los certificados son una herramienta que ayuda a lograr esta autenticación.

La sesión generalmente comienza con la autenticación entre pares, el intercambio de la clave pública y continúa con un protocolo de acuerdo de claves autenticado. Esto asegura que ambas entidades compartan una clave secreta común.

El navegador web, por ejemplo el Google Chrome, interpreta el certificado asociado a un sitio web como se visualiza en la ilustración 6. En dicho certificado se observa la versión, el tiempo de validez, al algoritmo de firma, la clave pública, la autoridad de certificación, entre otros ítems.

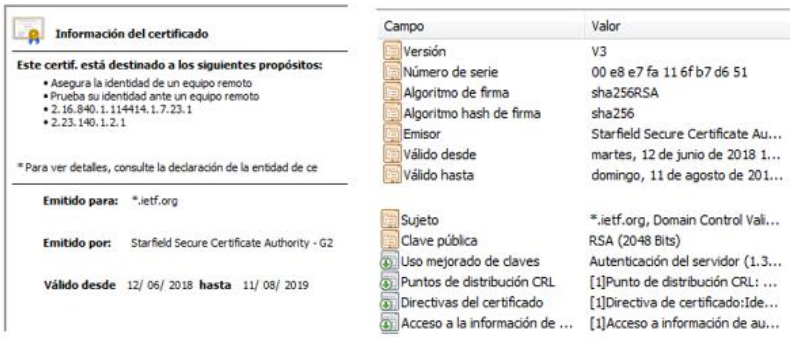

Ilustración 6 Ejemplo de Certificado de seguridad. Extraído mediante Google Chrome

Se puede mejorar un sistema de acuerdo de claves, transmitiendo una clave K usando un criptosistema de clave pública. De hecho es conveniente el intercambio con un secreto autenticado común en un protocolo cliente-servidor porque puede usarse más tarde para configurar un canal de comunicación seguro.

Los protocolos de acuerdo de clave, sin embargo, asumen por ejemplo que se transmite la clave pública K_p del servidor al cliente en un canal autenticado antes de la transmisión. Este canal previamente autenticado podría ser configurado por un tercero de confianza común entre ambas entidades, este es el objetivo de las denominadas autoridades de certificación. La CA podría emitir una firma x para K_p, y la autenticación de la clave estaría asegurada. Los únicos problemas restantes consisten en asegurar la comunicación de la clave pública K_p desde el servidor a la CA, y la comunicación de la clave pública de autorización al cliente.

Los certificados siguen el formato X.509, el cual está disponible bajo el estándar RFC 5280[3].

[3] Disponible en https://tools.ietf.org/pdf/rfc5280.pdf

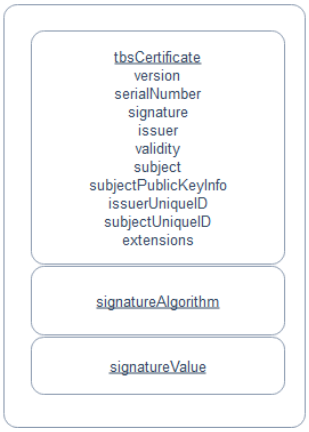

Ilustración 7 Estructura estándar de un certificado X.509

El contenido general de la estructura de un certificado puede tener diferencias pero siempre cuenta con atributos mínimos como versión, periodo de validez, clave pública, emisor y sujeto al que pertenece. Especificamente en los certificados X.509 se tiene una estructura divida en 3 campos, los cuales contienen atributos como los visualizados en la ilustración 7.En detalle:

- tbsCertificate: Contiene los nombres del propietario del certificado y del CA que lo creó y firmó, un periodo de validez, un identificador único y otra información asociada.
- signatureAlgorithm: Contiene el identificador para el algoritmo criptográfico utilizado por el CA para firmar el certificado, usualmente SHA1 y RSA.
- signatureValue: Contiene la firma digital del certificado en formato de cadena de bits.

Criptografía Híbrida

Como se ha comentado con anterioridad el cifrado con los algoritmos asimétricos es menos eficiente que el cifrado simétrico. Esto repercute a la hora de proteger grandes volúmenes de datos.

Un sistema híbrido es la unión de las ventajas de los dos sistemas anteriores. Se debe partir de la base que el problema criptográfico simétrico es inseguro y el asimétrico es lento, el proceso para usar un sistema criptográfico híbrido es el siguiente (Katz & Lindell, 2007):

1. Generar una clave pública y otra privada (en el receptor).

2. Cifrar un archivo de forma simétrica.

3. El receptor le envía al emisor su clave pública.

4. Cifrar la clave que se ha usado para encriptar el archivo con la clave pública del receptor.

5. Enviar el archivo cifrado (simétricamente) y la clave del archivo cifrada (asimétricamente y que solamente puede ver el receptor).

El cifrado hibrido es un modo de cifrado el cual incorpora una combinación de cifrado simétrico y asimétrico para beneficiarse de las fortalezas de cada una de estas formas. Es considerado un tipo de cifrado altamente seguro siempre y cuando se conserve la clave privada segura (Kuppuswamy & Al-Khalidi, 2014).

Hasta aquí se comentó las motivaciones del trabajo y los criterios de selección que se dieron sobre los algoritmos y protocolos criptográficos. Como así también se dejaron en claro algunos conceptos importantes para entender la ciencia de la criptografía y sus principales aportes a la computación.

Ahora como paso previo a entender el funcionamiento de los algoritmos criptográficos estudiados se considera importante en el siguiente capítulo hacer una revisión de conceptos matemáticos asociados al tema.

Capítulo 2: Algunos tópicos de Teoría de Números

Una de las cosas más difíciles sobre escribir cualquier libro en criptografía es decidir cuanta base matemática incluir.

Douglas R. Stinson

La criptografía en sus inicios requirió solo operaciones simples sobre el texto a cifrar como pueden ser las transposiciones de caracteres o un desfasaje como utilizaba el algoritmo de César.

Habitualmente se trabaja con números enteros, reales, complejos, polinomios y matrices y se efectúa con ellos ciertas operaciones, sin embargo no todas las operaciones se comportan de la misma manera, por ejemplo la conmutatividad en la multiplicación de polinomios no se presenta en matrices.

Es posible que dos conjuntos formados por elementos de diferente naturaleza y provistos de operaciones distintas tengan sin embargo el mismo comportamiento algebraico, se dice que las operaciones obedecen a las mismas leyes y se infiere que ambos sistemas poseen la misma estructura algebraica.

A causa del avance del tiempo y el desarrollo de la criptografía asimétrica se fue requiriendo de diferentes estructuras algebraicas como lo son los anillos, cuerpos y grupos, combinados con la utilización de la aritmética modular. Estas estructuras que se presentarán a continuación forman la base de una parte de la teoría criptográfica.

Las operaciones sobre estas estructuras requeridas tanto por Diffie-Hellman como RSA generan la necesidad de tener en claro conceptos como exponenciación modular, logaritmo discreto, teorema chino del resto, entre otros.

Estructuras Algebraicas empleadas en Teoría Criptográfica

Anillos

Un anillo es un conjunto R junto a dos operaciones binarias

$$+_R: R \times R \rightarrow R\ (suma) \ \wedge \ \cdot_R: R \times R \rightarrow R\ (producto)$$

que satisfacen las siguientes condiciones (Dixon, Kurdachenko, & Ya Su, 2014), (Adkins & Weintraub, 1992):

a) $a + b = b + a$ — Ley conmutativa de +
b) $a + (b + c) = (a + b) + c$ — Ley asociativa de +
c) Existe $z \in R \ / \ a + z = z + a = a$, para todo $a \in R$ — Existencia de una identidad para +
d) Para cada $a \in R$ existe un elemento $b \in R \ / \ a + b = b + a = z$, donde z es el elemento identidad para la + — Existencia de inversos bajo +

e) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ Ley asociativa de ·

f) $a \cdot (b + c) = a \cdot b + a \cdot c$ Leyes distributivas de · sobre +

$(b + c) \cdot a = b \cdot a + c \cdot a$

A continuación se presentan ideas básicas de la "aritmética modular" o también llamada los enteros módulo n. Primero se introduce la noción de números congruentes módulo un entero n, que dará lugar a una relación de congruencia, ideas expuestas por Gauss, cuando aparece su libro *Disquisitiones arithmeticae* sobre estos temas en 1801, teoría que ha resultado ser sumamente fecunda, ya que simplifica muchos tipos de cálculos que de otra manera serían muy tediosos, entre las aplicaciones más simples. Las clases de equivalencia que generan las congruencias dan lugar a los enteros modulares y con ellos toda una nueva aritmética, la aritmética modular.

Enteros Modulo n

Para poder definir los enteros modulo n primeramente se introduce el siguiente concepto:

Sea $n \in Z^+$, $n > 1$. Para $a, b \in Z$, se dice que $a\ es\ congruente\ con\ b\ módulo\ n$, y escribimos $a \equiv b\ (mod\ n)$, si $n\ divide\ a\ a\ menos\ b$, denotado por $n\,|(a - b)$; o, en forma equivalente, $a = b + kn$ para algún $k \in Z$ (Levin, 2018).

La congruencia módulo n establece una relación de equivalencia sobre Z. Por lo cual se puede inducir a formar particiones de éste. La congruencia módulo n divide a Z en las n clases de equivalencia:

$[0] = \{\dots, -2n, -n, 0, n, 2n, \dots\} = \{0 + nx \mid x \in Z\}$

$[1] = \{\dots, -2n + 1, -n + 1, 1, n + 1, 2n + 1, \dots\} = \{1 + nx \mid x \in Z\}$

…

…

$[n - 1] = \{\dots, -n - 1, -1, n - 1, 2n - 1, 3n - 1, \dots\} = \{(n - 1) + nx \mid x \in Z\}$

Se utiliza la notación Z_n para denotar $\{[0], [1], \dots, [n - 1]\}$ o bien de una manera más simplificada $\{0, 1, \dots, n - 1\}$.

Se definen operaciones binarias cerradas de suma y multiplicación sobre el conjunto de clases de equivalencia Z_n de modo que se obtenga un anillo. Para $[a], [b] \in Z_n$ se define $(Z_n, +, \cdot)$ como $[a] + [b] = [a + b]$ y $[a] \cdot [b] = [ab]$.

Un caso particular se da cuando n es primo, si n es primo entonces Z_n es un cuerpo, con lo cual $\forall\ a \in Z_n$ siendo $a \neq z$ (donde z es el cero de la +), permite garantizar la existencia de inverso multiplicativo: $\exists\ b \in Z_n$ tal que $ab \equiv 1\ (mod\ n)$ (Stallings, 2017).

Grupos

Otra estructura algebraica necesaria para que sea posible el funcionamiento de estos algoritmos son los grupos.

Un grupo algebraico G es un conjunto de elementos con una operación binaria, denotada por $\circ$, que satisface las siguientes condiciones (Farlow, 2019):

1. Si $a, b \in G$ entonces $a \circ b \in G$. (Propiedad de cierre)
2. Para todos $a, b, c \in G, a \circ (b \circ c) = (a \circ b) \circ c$. (Propiedad asociativa)
3. Existe $e \in G$ tal que $a \circ e = e \circ a = a$, para todo $a \in G$. (Existencia de un elemento identidad o neutro.)
4. Para cada $a \in G$ existe un elemento $b \in G$ tal que $a \circ b = b \circ a = e$. (Existencia de inversos.)

Si, además $a \circ b = b \circ a$ para todos $a, b \in G$, entonces G es un grupo conmutativo o abeliano.

Un grupo G es cíclico si existe un elemento x $\in G$ tal que para todo $a \in G, a = x^n$ para algún $n \in Z$, x es denominado generador de G, esto implica poder expresar cada elemento de G como una potencia del generador, este proceso requiere de un algoritmo. En el anexo dentro de la sección "Funciones auxiliares a los algoritmos", más específicamente en la página 63, se encuentra la implementación de un algoritmo para encontrar generadores de un grupo en la función *generador(p)*.

Un grupo de interés es el grupo multiplicativo Z_n^* en el cuál todos los elementos son invertibles y $\varphi(n)$ denota su cardinalidad u orden. El mismo cuenta con las siguientes propiedades:

1. $x \in Z_n^* \Leftrightarrow mcd(x, n) = 1$.
2. $x \in Z_n^*$, se tiene que $x^{\varphi(n)} \equiv 1 \ (mod\ n)$.

Función indicadora de Euler

Para cada $n \geq 1$, la función φ de Euler se define por $\varphi(n) = \#\{k \in N: k < n, mcd(k, n) = 1\ \})$, es decir la cantidad de enteros positivos menores que n que son relativamente primos con n (Farlow, 2019).

Exponenciación Modular

Sea $n \in Z^+, n > 1$ y $a, b, x \in Z$ se requiere calcular el valor de $x \equiv a^b \ (mod\ n)$.

La secuencia usual de operación consiste en primero resolver a^b, luego de este resultado obtener la clase de congruencia a la que pertenece en $mod\ n$.

Este algoritmo es correcto pero a los fines utilizados en la criptografía asimétrica en los cuales se tiene que a, b y n son números de un tamaño de 64 o 256 bytes[4] como mínimo este método se hace inviable computacionalmente. Es decir el esfuerzo de cómputo es demasiado alto, requiere demasiado tiempo, memoria y velocidad de procesamiento para lograr realizar la función.

La alternativa que se aplica en estos casos es la exponenciación modular también conocida como potenciación binaria o repeated-squaring que consiste en (Galbraith, 2012):

1. Convertir el número b en su expresión binaria $b = b_k b_{k-1} \ldots . b_2 b_1 b_0$.
2. Primero se analiza b_0, se asigna $p = a$:
   - Si $b_0 = 1$ almacenar el valor $c_0 \equiv p \ (mod\ n)$.
   - Si $b_0 = 0$ almacenar $c_0 = 1$.
3. Luego $\forall\ i$ tal que $0 < i \leq k$, se asigna $p = p^2$:
   - Si $b_i = 1$ almacenar el valor $c_i \equiv p \ (mod\ n)$.
   - Si $b_i = 0$ almacenar $c_i = 1$.
4. Se resolverá el valor $x \equiv \prod_0^k c_i \ (mod\ n)$.

   Otra forma equivalente de desarrollar el algoritmo es la siguiente, y es la que se presenta en el anexo página 60 como función *expModular(a,b,n)* :

1. Se asigna $c = 1$, para ir acumulando el resultado.
2. Se asigna $p = a\%n$, para asegurar que $a < n$ y ahorrar tiempo de cálculo.
3. Se itera mientras $b > 0$.
4. Se analiza el resto de b. Si $b\ \%\ 2 = 1$, entonces $c = (c * p)\ \%\ n$.
5. Se almacena el cociente entero entre b y 2. Resultando $b = b\ //\ 2$.
6. Antes de terminar el ciclo de la iteración se realiza $p = p^2$.

Teorema de Euler y "Pequeño Teorema" de Fermat

Sea Z_n^* el grupo multiplicativo de todos los elementos que poseen inverso multiplicativo en Z_n con p primo, q primo, $p \neq q$ y $n = pq$.

El teorema de Euler expresa que (Ruohonen, 2014):

- si $a \in Z_n^*$, entonces $a^{\varphi(n)} \equiv 1 \ (mod\ n)$.
- $r \in Z$ y $s \in Z$ entonces, $\forall a,\ a^r \equiv a^s \ (mod\ \varphi(n))$. La importancia del teorema radica en que en los módulos como n, los exponentes pueden reducirse en modulo $\varphi(n)$.

Un caso especial del Teorema de Euler es el denominado "Pequeño Teorema" de Fermat que se aplica sobre Z_p^* con p primo (Liben-Nowell, 2017):

- Si $mcd(a, p) = 1$, entonces $a^{p-1} \equiv 1 \ (mod\ p)$.

[4] El mayor número que se puede representar con 64 bytes (512 bits) sin signo es aproximado a 1,34078079299426e+154, mientras que con 256 bytes (2048 bits) sin signo es aproximado a 3,23170060713110073007148766886 7e+616

- $r \in Z$ y $s \in Z$, si $r \equiv s\ (mod\ p-1)$ entonces, $\forall a,\ a^r \equiv a^s\ (mod\ p)$. En este caso más específico cuando se trabaja con modulo p primo, los exponentes pueden reducirse en modulo $p-1$.
- $\forall a, a^p \equiv a\ (mod\ p)$.

Teorema Fundamental de la aritmética

El Teorema Fundamental de la Aritmética (TFA) se relaciona con el concepto de factorización única de un número entero. La factorización única se refiere a dos propiedades particulares de un número: la existencia y la unicidad. La existencia hace referencia a la posibilidad de que un número entero con valor absoluto mayor que 1 pueda ser representado como un producto finito de números primos y la unicidad significa que esta representación es única, sin importar el orden.

Sea $n \in Z$ y $n \neq 0$, n distinto de 1 y de -1, puede ser expresado como $n = \pm p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, donde $p_1, p_2, \dots, p_k$ son primos distintos y $e_1, e_2, \dots, e_k$ son enteros positivos, esta representación es única exceptuando el orden de los factores (Farlow, 2019).

En el anexo dentro de la sección "Funciones auxiliares a los algoritmos", más específicamente en la página 63, se encuentra una implementación de este algoritmo en la función *factores(n)*.

Teorema Chino del Resto

El teorema Chino del Resto es un método que resuelve ciertos sistemas de congruencia. Se supone que $n_1, \dots, n_r$ son números enteros positivos y tal que $\forall i \neq j, 1 < i \leq r, 1 < j \leq r \rightarrow mcd(n_i, n_j) = 1$, es decir primos relativos entre sí. Y que $a_1, \dots, a_r$ son enteros y se considera el siguiente sistema de congruencias (Hardy, Richman, & Walker, 2009):

$$x \equiv a_1 (mod\ n_1)$$

$$x \equiv a_2 (mod\ n_2)$$

…

$$x \equiv a_r (mod\ n_r)$$

El teorema Chino del Resto afirma que el sistema tiene una única solución modulo $n = n_1 \cdot \cdot\ n_2 \cdot \dots \cdot\ n_r$.

Funciones de un sentido o unidireccionales (One Way Functions)

Una función f uno a uno de un conjunto M a un conjunto C es llamada de un sentido si $f(m)$ es sencillo calcular para todo $m\ \varepsilon\ M$, pero para un c seleccionado de forma aleatoria en la imagen de f, encontrar un $m\ \varepsilon\ M$ talque $c = f(m)$ es computacionalmente inviable (Barakat, Eder, & Hanke, 2018).

Funciones Trampa de un sentido (Trapdoor One Way Functions)

Una función trampa de un sentido o función de encriptación de clave pública es una función de un sentido $f: M \to C$ que satisface la propiedad adicional que existe información, llamada información trampa o simplemente trampa, lo cual hace viable encontrar $m\ \varepsilon\ M$ para un $c\ \varepsilon\ img(f)$ talque $f(m) = c$, pero sin la trampa la tarea es irreversible (Barakat, Eder, & Hanke, 2018).

No se ha demostrado aún la existencia de funciones trampa de un sentido pero hay un conjunto de funciones candidatas a serlo como la factorización de números o el logaritmo discreto que no son fáciles de computar.

En este capítulo se describieron estructuras algebraicas y explicaron conceptos matemáticos vinculados a la criptografía de clave pública, y más específicamente a los algoritmos RSA y Diffie-Hellman.

La criptográfica asimétrica o de clave pública se vincula directamente con los contenidos desarrollados en este capítulo, entonces ahora se está en condiciones de describir los algoritmos criptográficos objeto de estudio de este trabajo.

Capítulo 3: Criptografía de clave pública

La criptografía lleva milenios siendo utilizada por la humanidad como método para cifrar comunicaciones secretas, en especial las de carácter militar.

El cifrado era su única aplicación hasta que en el año 1976 Diffie y Hellman propusieron un método para poder intercambiar claves de forma segura sobre un canal inseguro. En otras palabras, dos personas con este protocolo ahora podían definir una clave secreta para poder cifrar las comunicaciones entre ambas sin la necesidad de definir esta clave personalmente.

Luego en 1978 Rivest, Shamir y Adleman desarrollan el primer criptosistema de clave pública con el cual se podía tanto intercambiar claves como cifrar información, si bien el intercambio de claves no era tan eficiente como el de Diffie-Hellman y el protocolo de cifrado no era tan rápido como otras alternativas que fueron surgiendo con los años. No obstante, el principal aporte de este protocolo fue la firma digital basada en su esquema de cifrado y estandarizada a principios de la década de 1990.

Diffie-Hellman

Whitfield Diffie y Martin E. Hellman publicaron en 1976 un trabajo (Diffie & Hellman, 1976) en el cual propusieron la primera solución práctica de un protocolo de acuerdo de claves en la cual la seguridad se basaba en la intratabilidad de resolver logaritmos discretos computacionalmente (Aumasson, 2017).

El objetivo de este protocolo es generar una clave secreta común entre dos partes sobre un canal inseguro (pero autenticado) que luego puede ser utilizado para enviar un mensaje cifrado con un método simétrico.

Este tipo de protocolo también es llamado protocolo de intercambio de claves aunque realmente no se realice un intercambio de claves dado que las mismas se generan aleatoriamente.

Protocolo de Intercambio de Claves de Diffie-Hellman

Se plantea el escenario en el cual se tiene 2 entidades A y B las cuales no se han encontrado ni intercambiado claves, pero necesitan establecer una clave secreta k para intercambiar mensajes sobre un canal inseguro.

Primero las entidades concuerdan un largo número p primo, un grupo multiplicativo cíclico $G = \left(Z_p^*, \circ\right)$ con la operación binaria definida como $\forall\, a,\ b\ \in G\ \rightarrow a \circ b = a.b\ (mod\ p)$ y un generador $\alpha \in G$, talque $2 \leq \alpha \leq p - 2$ los cuales necesitan permanecer en secreto (Aumasson, 2017). Entonces:

1. A elige un $x \in N$ de forma aleatoria y computa $X \equiv \alpha^x\ (mod\ p)$, y envía X a B. Manteniendo en secreto x.

2. B elige un $y \epsilon N$ de forma aleatoria y computa $Y \equiv \alpha^y \ (mod\ p)$, y envía Y a B. Manteniendo en secreto y.
3. A computa $k \equiv Y^x \equiv (\alpha^y)^x \ (mod\ p)$.
4. B computa $k \equiv X^y \equiv (\alpha^x)^y \ (mod\ p)$.

k es la clave secreta compartida generada independientemente por A y B.

Además de su característica de *probablemente seguro* las claves generadas por Diffie-Hellman poseen las características de *secreto perfecto hacia adelante (perfect forward secrecy o PFS)* por el cual si la clave actual se ve comprometida por un tercero esto no afectaría ninguna clave anterior generada.

El problema del Logaritmo Discreto y de Diffie-Hellman

Dado un p primo, un α generador de $Z_p^* = \{a \ \epsilon N \ / \ mcd(a,p) = 1\}$ y $\beta \in G$, entonces el logaritmo discreto de β en la base α, denotado como $\log_\propto \beta$ es igual a x, siendo x un entero, $0 \leq x \leq p-2$. Encontrar x en $\alpha^x \equiv \beta \ (mod\ p)$ se conoce como *Problema del Logaritmo Discreto* (DLP).

Una generalización de este problema suele denominarse *Problema del Logaritmo Discreto Generalizado* (GDLP) y consiste en: dado un grupo cíclico G de orden n, un generador α de G, y un elemento $\beta \in G$, encontrar un entero x, $0 \leq x \leq n-1$, tal que $\alpha^x = \beta$.

Estos problemas están relacionados con el *Problema de Diffie-Hellman* (DHP), y son de gran significancia por su presunta intratabilidad, el cual se define como: dado un p primo, un α generador de Z_p^*, y elementos $\alpha^x \ mod\ p$ y $\alpha^y \ mod\ p$, encontrar $\alpha^{xy} \ mod\ p$ (Aumasson, 2017).

Ejemplo protocolo Diffie-Hellman

Las entidades A y B concuerdan $p = \ 59393$, con su correspondiente $\alpha = 5$.

También A genera aleatoriamente $x = \ 15761$ y B genera aleatoriamente $\ y = \ 20282$, entonces:

A		B
$5^{15761} \ (mod\ 59393) \equiv 46929$		
	$\rightarrow X = \ 46929 \ \rightarrow$	
		$5^{20282} \ (mod\ p) \equiv 30644$
	$\leftarrow Y = 30644 \ \leftarrow$	
$30644^{15761} \ (mod\ 59393) \equiv \ 52715$		
		$46929^{20282} \ (mod\ 59393) \equiv \ 52715$

$k = \ 52715$ es la clave secreta generada independientemente por A y B.

En el anexo dentro de la sección “Script para intercambiar claves utilizando Diffie-Hellman”, más específicamente en la página 73, se encuentra una implementación de un escenario de intercambio de claves entre dos entidades utilizando este protocolo.

RSA

El algoritmo de RSA se puede dividir en un algoritmo de generación de claves y uno de cifrado de datos (Aumasson, 2017).

En la siguiente ilustración a partir del criptosistema asimétrico genérico presentado en la ilustración 4 se agregan al modelo los detalles de generación de claves, envío de las mismas y cálculo de cifrado.

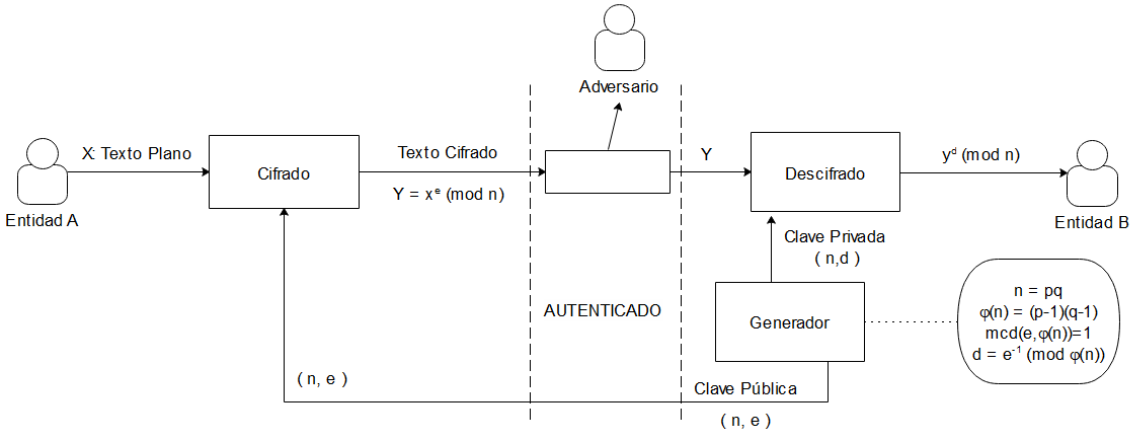

Ilustración 8 Procedimiento del Criptosistema RSA con los detalles. Adaptado de (Vaudenay, 2006)

Se asume que la entidad A desea enviar un mensaje a la entidad B.

RSA Generación de clave

1. B genera 2 números primos grandes de forma aleatoria p y q de similar tamaño, talque $p \neq q$.
2. B computa $n = pq$ y $\varphi(n) = (p-1)(q-1)$, donde a n se lo denomina el modulo.
3. B selecciona un $e \in N$, talque $1 < e < \varphi(n)$ y el $mcd(e, \varphi(n)) = 1$, e es denominado el exponente en encriptación.
4. Utilizando el algoritmo de Euclides extendido, B computa el único $d \in N$ con $1 < d < \varphi(n)$ talque $ed \equiv 1 \ (mod\ \varphi(n))$, donde d es denominado el exponente en desencriptación y es la clave privada.
5. B publica (n, e) en una base de datos pública y mantiene d, p, q y $\varphi(n)$ privados.

RSA Cifrado de Clave Pública

Escenario de cifrado: para simplificar el proceso inicialmente se asume que el mensaje en texto plano $m\ \varepsilon\ M$, convertido en un formato numérico (ya sea obteniendo el ordinal del carácter en el código ASCII[5] u otro código similar), es menor a n. También que $M = C = Z_n$ y se asume que $mcd(m, n) = 1$.

1. A obtiene la clave pública (n, e) de B de la base de datos.

[5] American Standard Code for Information Interchange.

2. A cifra m y computa $c \equiv m^e \ (mod\ n)$ utilizando la exponenciación modular.
3. A envía una conversión a texto de c a B.
   Escenario de descifrado: $B\ recibe\ c$, lo convierte en un formato numérico y utilizando d computa $m \equiv c^d \ (mod\ n)$.

En el caso de que el texto plano como valor numérico es $m \geq n$, se tiene que subdividir el texto plano en bloques numéricamente de igual tamaño en un proceso llamado agrupamiento en bloques del mensaje (message blocking).

Suponiendo que se utiliza un equivalente numérico del texto plano en base N, para algún entero $N > 1$ y se elige un entero l talque $N^l < n < N^{l+1}$ entonces se divide el mensaje en bloques de l dígitos y base N con ceros de relleno en el último bloque de ser necesario y se lo cifra por separado.

Ejemplo de Cifrado de Mensajes con RSA

La entidad A desea enviarle el mensaje "hola mundo", de forma cifrada a la entidad B que tiene por claves públicas $(n = 22961756758053506 3, e = 12971080166167661)$ y clave privada $d = 159374914931295461$.

A convierte a un formato numérico el mensaje "hola mundo" resultando $m = 2532811337764708$, utilizando un alfabeto que aceptaba letras minúsculas y el carácter de espacio.

A calcula $c \equiv 2532811337764708^{12971080166167661}\ (mod\ 22961756758053506 3) \equiv 66782217931147909$, lo convierte en la cadena de texto "hamacn hszoa" utilizando el proceso inverso al anterior paso y lo envía a B.

B recibe la cadena de texto "hamacn hszoa", la transforma en formato numérico resultando $c = 66782217931147909$ y calcula

$m \equiv 66782217931147909^{159374914931295461}\ (mod\ 22961756758053506 3) \equiv 2532811337764708$, la cual en formato numérico resulta en el mensaje "hola mundo".

En el anexo dentro de la sección "Script para cifrar mensajes con RSA", más específicamente en la página 71, se encuentra una implementación de un escenario de cifrado y descifrado de mensaje entre dos entidades utilizando este protocolo.

Prueba matemática del funcionamiento del algoritmo RSA

Dados e, d, p, q, n todos especificados en el algoritmo de generación de claves RSA y sabiendo que $n = pq$, p primo, q primo, $\varphi(n) = (p-1)(q-1)$, $ed \equiv 1\ \big(mod\ \varphi(n)\big)$ y sea $m \in Z_n$ el mensaje. Se prueba que $(m^e)^d \equiv m\ (mod\ n)$ de la siguiente manera (Liben-Nowell, 2017):

Dado que $ed \equiv 1 \ (mod\ \varphi(n))$, entonces existe un entero k tal que $ed = 1 + k\varphi(n) = 1 + k(p-1)(q-1)$ por la definición de congruencia modular.

Por el Teorema de Fermat se tiene que si $mcd(m,p) = 1$ entonces:

$$m^{p-1} \equiv 1\ (mod\ p).$$

Si se eleva cada lado de la congruencia a la potencia $k(q-1)$ se obtiene:

$$(m^{p-1})^{k(q-1)} \equiv (1)^{k(q-1)}\ (mod\ p) \rightarrow m^{k(p-1)(q-1)} \equiv 1\ (mod\ p)$$

Multiplicando cada lado de la congruencia por m se obtiene:

$$m\left(m^{k(p-1)(q-1)}\right) \equiv m(1)\ (mod\ p) \rightarrow m^{1+k(p-1)(q-1)} \equiv m\ (mod\ p) \rightarrow$$

$$m^{ed} \equiv m\ (mod\ p)$$

El otro caso que se tiene que evaluar es si $mcd(m,p) = p$ entonces existe un t tal que $m = tp$. Se observa que la validez de la última ecuación no cambia dado que

$$m^{ed} \equiv m\ (mod\ p) \rightarrow (tp)^{ed} \equiv (tp)\ (mod\ p) \rightarrow t^{ed}p^{ed} \equiv tp\ (mod\ p) \rightarrow$$

$$\rightarrow t^{ed}(0) \equiv t(0)\ (mod\ p) \rightarrow 0 \equiv 0\ (mod\ p)$$

En consecuencia $m^{ed} \equiv m\ (mod\ p)$ se cumple para todos los casos.

El mismo argumento es aplicable para $m^{ed} \equiv m\ (mod\ q)$.

Finalmente dado que $p \neq q$ por el Teorema Chino del Resto tenemos que

$$m^{ed} \equiv m\ (mod\ n)$$

Y como consecuencia $c^d \equiv (m^e)^d \equiv m\ (mod\ n)$.

Seguridad en RSA

RSA es tan seguro como difícil factorizar $n = pq$. Basa su seguridad en la conjetura que la función de cifrado es una función de un solo sentido, para encontrar d dado e el único método conocido es aplicar el algoritmo de Euclides a e y $\varphi(n)$ haciéndola computacionalmente inviable para un externo descifrar el texto (Aumasson, 2017).

La seguridad depende también del tamaño de clave, usualmente este parámetro es 1024 bits. En particular si la factorización de enteros de gran tamaño fuera fácil, entonces RSA no sería seguro porque n es público y factorizando n obtengo p y q, lo cual es suficiente para descifrar.

Esquema de firma digital RSA

La entidad A firma un mensaje $m \in M$ para que cualquier entidad B pueda verificar la firma de A y recuperar el mensaje m a partir de la firma, siendo $n = pq$ siendo p y q primos distintos, también se cuenta con un par RSA (e,d) que se obtiene vía un algoritmo de generación de claves RSA (Mollin, 2007).

Generación de la firma: Para generar la firma A debe:

1. Calcular $h = h(m)$, donde la función $h()$ aplica un hash sobre el mensaje m.
2. Computar $s \equiv h^d \ (mod\ n)$
3. El valor s obtenido es la firma de A para el mensaje m.
4. A envía el par (m, s).

Verificación: B recibe el par (m, s) y para verificar la firma s de A debe:

1. Obtener la clave pública (n, e) de A.
2. Computar $h \equiv s^e \ (mod\ n)$, con lo cual obtengo el hash enviado por A .
3. Calcular $h_2 \equiv h(m) \ \ (mod\ n)$.
4. Comparar $h = h_2$, si es verdadero se acepta la firma, si es falso se rechaza la firma.

Ejemplo de Firma Digital con RSA

La entidad B desea enviarle el mensaje "hola mundo" con su correspondiente firma digital a la entidad A, B tiene por claves públicas $(n = 229617567580535063, e = 12971080166167661)$ y clave privada $d = 159374914931295461$.

B firma el mensaje, primero calcula el hash MD5 del mensaje "hola mundo" en formato entero resultando $h = 143743596480351597544586584389714028 73$, y luego lo cifra y obtiene la firma:

$$s \equiv h^d \ (mod\ n) \equiv$$

$$\equiv 143743596480351597544586584389714028 73^{159374914931295461} \ (mod\ 229617567580535063) \equiv$$

$$s \equiv 125108287552649387$$

B envía $(\mathrm{m} = \text{"hola mundo"}, \mathrm{s} = 125108287552649387)$ a A.

A obtiene la clave pública $(n = 229617567580535063, e = 12971080166167661)$.

A descifra la firma enviada por B, y obtiene el resto entre el hash y n calculado por B del mensaje:

$$h \equiv s^e \ (mod\ n) \equiv 125108287552649387^{12971080166167661} \ (mod\ 229617567580535063)$$

$$h = 164380201910035319$$

A calcula el hash MD5 del mensaje "hola mundo" en formato entero en módulo n resultando:

$$h_2 \equiv 143743596480351597544586584389714028 73 \ (mod\ 229617567580535063)$$

$$h_2 \equiv 164380201910035319$$

A compara $h = h_2$, resultando iguales por ende la firma s del mensaje m corresponde a B.

En el anexo dentro de la sección “Script para firmar mensajes con RSA”, más específicamente en la página 72, se encuentra una implementación de un escenario de firma y verificación de mensaje entre dos entidades utilizando este protocolo.

En este capítulo se analizaron los algoritmos de Diffie-Hellman y RSA como paso previo a describir los protocolos criptográficos donde los mismos se implementan.

Capítulo 4: Protocolos Criptográficos basados en los algoritmos de DH y RSA

Un protocolo criptográfico es un algoritmo que involucra dos o más entidades, diseñado para lograr un objetivo de seguridad, como pueden ser privacidad o autenticación, a través de la utilización de la criptografía (Schoenmakers, 2019).

Día a día la utilización de comunicaciones electrónicas a través de una computadora o un dispositivo móvil forman parte de la rutina normal de una persona:

- Consultas de clima, noticias, redes sociales o homebanking.
- Gestión de correos electrónicos.
- Mensajería instantánea.

Además existen otros tipos de comunicaciones que orientan más a un ámbito reducido o pasan desapercibidas, dada su función:

- Gestión de noticias.
- Resolución de nombres de dominio.
- Virtualización de redes.
- Transferencia de archivos.
- Control de versiones.
- Conexiones remotas.
- Actualizaciones de las aplicaciones.

Todas estas acciones se realizan a través de programas informáticos. Sea un navegador web, un gestor de correos electrónicos, una aplicación móvil de mensajería instantánea u otro software. Los cuales implementan protocolos para llevar a cabo su tarea como pueden ser HTTP, SMTP, XMPP, FTP, entre otros.

Estos protocolos muchas veces requieren capacidades de seguridad que los exceden, como pueden ser acuerdos de claves, autenticación, cifrado o firma digital. Con el fin de dar soporte a estas características se utilizan protocolos criptográficos con sus correspondientes implementaciones de algoritmos criptográficos de clave pública como Diffie-Hellman y RSA.

En este capítulo se estudiarán los protocolos criptográficos seleccionados: SSL/TLS, SSH, PGP y SMIME.

SSH: Secure Shell

SSH es un protocolo para sesiones remotas seguras y otros servicios de red sobre una red insegura (Ylonen & C. Lonvick, SSH-ARCH, 2006a).

SSH fue desarrollado en 1995 por Tatu Ylonen y originalmente permite el acceso a una computadora en forma segura en sistemas operativos basados en UNIX.

El principio de SSH es implementar seguridad en el canal de comunicación de la sesión del cliente – servidor, como así también ser de fácil uso y no requerir instalación. La contrapartida es que su nivel de seguridad no es tan alto, pero aun así superior a lo que había cuando salió al mercado.

La nueva versión de SSH conocida como SSH2 salió en 2006 y utiliza una infraestructura de clave pública para autenticarse en el servidor.

Si bien no se han realizado nuevas versiones de SSH, se han generado documentos actualizados referentes a nuevas funciones o actualización de algoritmos. Entre los más importantes para este trabajo uno referente a RSA (Bider, 2018) y otro a Diffie-Hellman (Baushke, 2017).

Arquitectura del protocolo

El protocolo SSH se divide en 3 componentes: protocolo de capa de transporte, protocolo de autenticación de usuarios y protocolo de conexión.

El protocolo de capa de transporte

La capa de transporte SSH es un protocolo de transporte seguro y de bajo nivel (Ylonen & C. Lonvick, SSH-TRANS, 2006c).

Provee un canal confidencial sobre una red insegura con lo cual mejora la protección de las funcionalidades de autenticación, confidencialidad e integridad.

Los datos intercambiados se dividen en bloques de hasta aproximadamente 34 kilobytes llamados paquetes, los cuales tienen el siguiente formato:

- Tamaño del paquete en bytes.
- Tamaño del relleno del bloque en bytes.
- El contenido útil del paquete, el cual puede estar comprimido.
- Relleno del bloque de longitud aleatoria.
- Código de autenticación del mensaje (MAC), el cual asegura que la integridad de los datos esté protegida.

Durante el intercambio de claves un algoritmo de cifrado simétrico es negociado entre las partes, con esto se consigue que los campos de los paquetes viajen confidencialmente por la red.

Este protocolo requiere algoritmos de firma digital y certificados para autenticar el servidor al cliente, entre los cuales existen dos de estos algoritmos basados en RSA:

- SSH-RSA: Un algoritmo de firma digital que se utiliza para generar claves RSA.

- PGP-SIGN-RSA: Un algoritmo que genera certificados OpenPGP utilizando claves RSA.

El protocolo de autenticación de usuarios

El protocolo de autenticación SSH es un protocolo de autenticación de usuario de uso general (Ylonen & C. Lonvick, SSH-TRANS, 2006b).

Provee un conjunto de mecanismos que son utilizados para autenticar el usuario cliente en el servidor.

El único método de autenticación requerido por el protocolo y que toda implementación del mismo tiene que tener es el método por clave pública. Otros métodos como autenticación por contraseñas o basadas en el host son opcionales.

Las claves privadas se suelen almacenar en forma cifrada en la maquina cliente, y el usuario debe suministrar previamente una contraseña para que la firma sea guardada, dado que los cálculos de criptografía asimétrica son complejos esto evita que se realice procesamiento innecesario.

Cuando un cliente se desea conectar al servidor, el servidor envía su clave pública junto con un certificado. La primera conexión es crítica: el cliente puede autenticar la clave pública, por ejemplo verificando el certificado o la huella digital de la misma, o aceptar la conexión sin verificar. Luego de esto se almacena la clave pública en un archivo del cliente.

Asumiendo que la primera conexión fue establecida con éxito, todas las futuras conexiones con el mismo servidor serán seguras comparando la clave recibida con la clave pública del archivo almacenado. Si las claves no concuerdan, se emitirá una advertencia de seguridad indicando que la clave pública ha cambiado y que puede estar frente a un problema de seguridad; igualmente el usuario puede ignorar esto y aceptar la conexión. El protocolo no es restrictivo solo avisa el problema por lo cual este es uno de los principales problemas de SSH.

El cliente y el servidor ejecutan un protocolo de acuerdo de clave tal que el servidor está autenticado, y generan una clave simétrica que se utilizará para configurar un canal seguro. Luego, el cliente se autentica mediante una contraseña que se envía a través del canal seguro.

Existen muchas formas de conectarse a un equipo utilizando SSH, tal vez la forma más recurrente es utilizando el software Putty (ilustración 9).

El mismo permite conectarse a un equipo remoto a través de su dirección IP y configurar diferentes parámetros como pueden ser la prioridad de los algoritmos de intercambio de claves, los algoritmos de cifrado, entre otros.

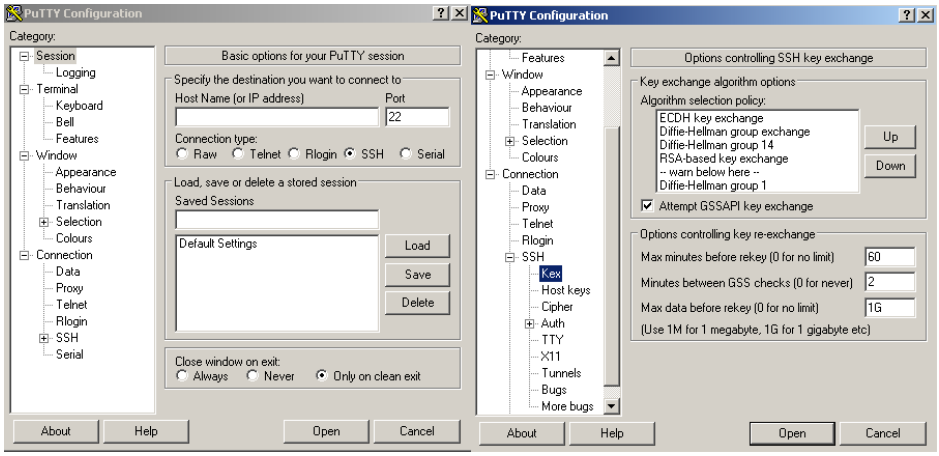

Ilustración 9 Interfaz de configuración del Software de conexión Putty

El protocolo de conexión

El protocolo de conexión SSH ha sido diseñado para ejecutarse sobre la capa de transporte SSH y los protocolos de autenticación de usuario (Ylonen & C. Lonvick, SSH-CONNECT, 2006d).

Especifica un mecanismo para combinar múltiples canales de información sobre un trasporte confidencial y autenticado, provee logins de sesión interactivos, ejecución remota de comandos y reenvío de conexiones TCP/IP[6].

Todas las sesiones terminales, conexiones reenviadas, etc., son canales. Cualquiera de los lados puede abrir un canal y varios canales se multiplexan en una sola conexión.

Los canales se identifican por números en cada extremo, los cuales pueden ser diferentes en cada lado. Las solicitudes para abrir un canal contienen el número de canal del remitente, cualquier otro mensaje relacionado con el canal contiene el número de canal del destinatario para el mismo.

Intercambio de claves en SSH2

El intercambio de claves comienza con cada lado enviando una lista de los algoritmos soportados, teniendo preferencias por defecto en cada categoría de los mismos. Puede darse el caso que no se logre un acuerdo en la elección motivo por el cual el procedimiento falla y si alguna parte envió un paquete este sea ignorado del otro lado.

El método utiliza de manera estándar autenticación explicita en el servidor si el intercambio de mensajes incluye una firma u otro método para demostrar la autenticidad. Mientras que utiliza autenticación implícita si lo que envía para autenticar es una contraseña compartida.

Se utiliza Diffie-Hellman para acordar una clave y luego obtener una clave de sesión simétrica. En el intercambio de clave se tiene que C es el cliente, S el servidor, p es un primo grande,

[6] Protocolo de control de transmisión / Protocolo de Internet, es un conjunto de protocolos que permiten la comunicación entre los ordenadores pertenecientes a una red. (Enciclopedia de CommentCaMarche.net). Recuperado de https://es.ccm.net/contents/282-tcp-ip

un generador g del subgrupo $Z_p^* = \{mcd(x,p) = 1 \wedge x < p \mid x \in N\}$ de orden $q = p - 1$, V_s es la cadena de identificación de S, V_c es la cadena de identificación de C, K_s es la clave pública de S, I_s es el mensaje inicial de S y I_c es el mensaje inicial de C, los cuales se intercambian antes del acuerdo de claves el cual se realiza de la siguiente manera (Vaudenay, 2006):

1. C genera un $x \in \{1, \dots, q-1\}$ aleatorio y calcula $e = g^x \bmod p$. C envía e a S.
2. S genera un $y \in \{1, \dots, q-1\}$ aleatorio y calcula $f = g^y \bmod p$. S recibe e. Luego computa $K = e^y \bmod p$, arma $H = hash(\ V_c \ ||\ V_s \ ||\ I_c ||\ I_s ||\ K_s ||\ e ||\ f ||\ K\ )$ el cual es firmado con la clave privada y etiquetado como s. S envía K_s, f y s a C.
3. C verifica que K_s sea la clave pública de S cotejándola con un certificado o una base de datos local. C computa $K = f^x \bmod p$, $H = hash(\ V_c \ ||\ V_s \ ||\ I_c ||\ I_s ||\ K_s ||\ e ||\ f ||\ K\ )$ y verifica la firma s para H.

Una vez finalizado el proceso el cliente y el servidor pueden usar K como clave para un cifrado simétrico.

TLS: Transport Layer Security

El objetivo principal de TLS es proporcionar un canal seguro entre dos pares que se comunican; el único requisito del transporte subyacente es un flujo de datos confiable y en orden (Rescorla, 2018).

SSL/TLS es típicamente usada por los navegadores de internet para comunicarse de una manera segura con los servidores a través de HTTP.

En la ilustración 10 se visualiza el apartado de información de página de un sitio web utilizando Mozilla Firefox, en el cual se observa la utilización de una variante del TLS para establecer la conexión entre el cliente y el sitio.

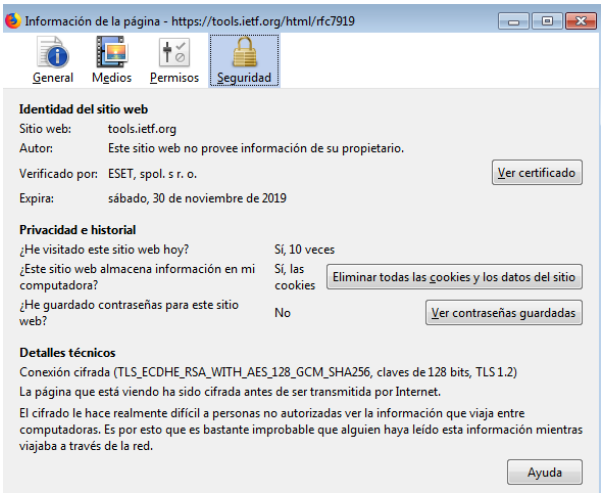

Ilustración 10 Información estándar de sitio web utilizando navegador Mozilla Firefox

También es utilizado por aplicaciones como gestores de e-mail para conectarse con sus servidores y transferencia de archivos mediante FTP. En la ilustración 11 se visualiza una conexión FTP utilizando FileZilla, en la cual se identifica claramente la utilización de TLS.

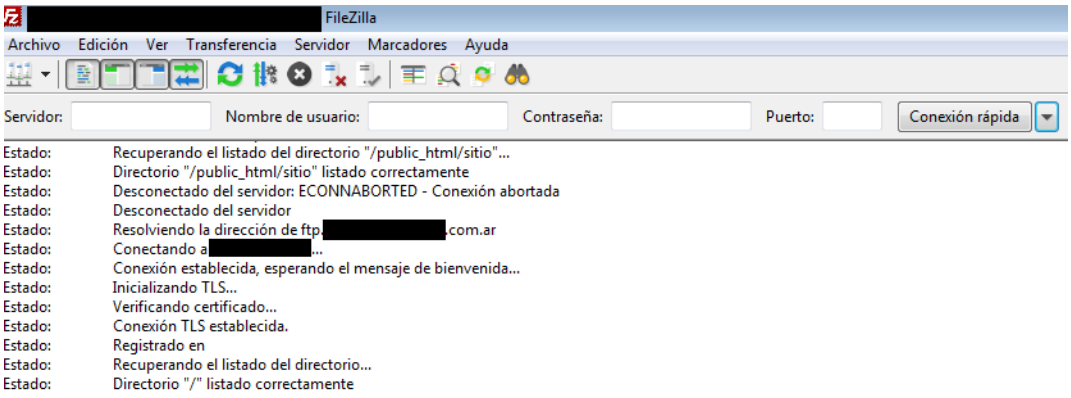

Ilustración 11 Vista de conexión de intercambio de archivos entre usuario y servidor, utilizando TLS en el software FileZilla.

SSL fue desarrollado por Netscape en 1994 con el fin de mejorar la seguridad de las comunicaciones sobre HTTP. En el año 1999 la *Internet Engineering Task Force* desarrolló Transport Layer Security, una leve mejora a SSL versión 3.0 lanzada en 1995, el cual se centró en como los números pseudo aleatorios eran generados. En el año 2018 se lanza la RFC8446 la cual actualiza TLS a la versión 1.3 con mejoras de seguridad al cifrado asimétrico, rediseño de intercambio de claves, entre otros.

TLS consiste en dos componentes principales, el protocolo de intercambio (Handshake Protocol) y el protocolo de registro (Record Protocol).

El TLS Handshake Protocol se encarga de autenticar las partes, negociar los parámetros criptográficos y establecer las claves compartidas. Está diseñado para resistir la manipulación, por lo cual un atacante no debería ser capaz de forzar el uso de un parámetro distinto al elegido sin su intervención.

El TLS Record Protocol utiliza los parámetros establecidos por el protocolo de intercambio para proteger el tráfico entre las partes, dividiéndolo en una serie de registros donde cada uno está protegido de forma independiente mediante claves de tráfico.

TLS Handshake Protocol

El protocolo puede ser dividido en 3 fases: intercambio de claves, parametrización del servidor y autenticación. La ilustración 12 describe como es la composición y secuencia de pasos de este protocolo.

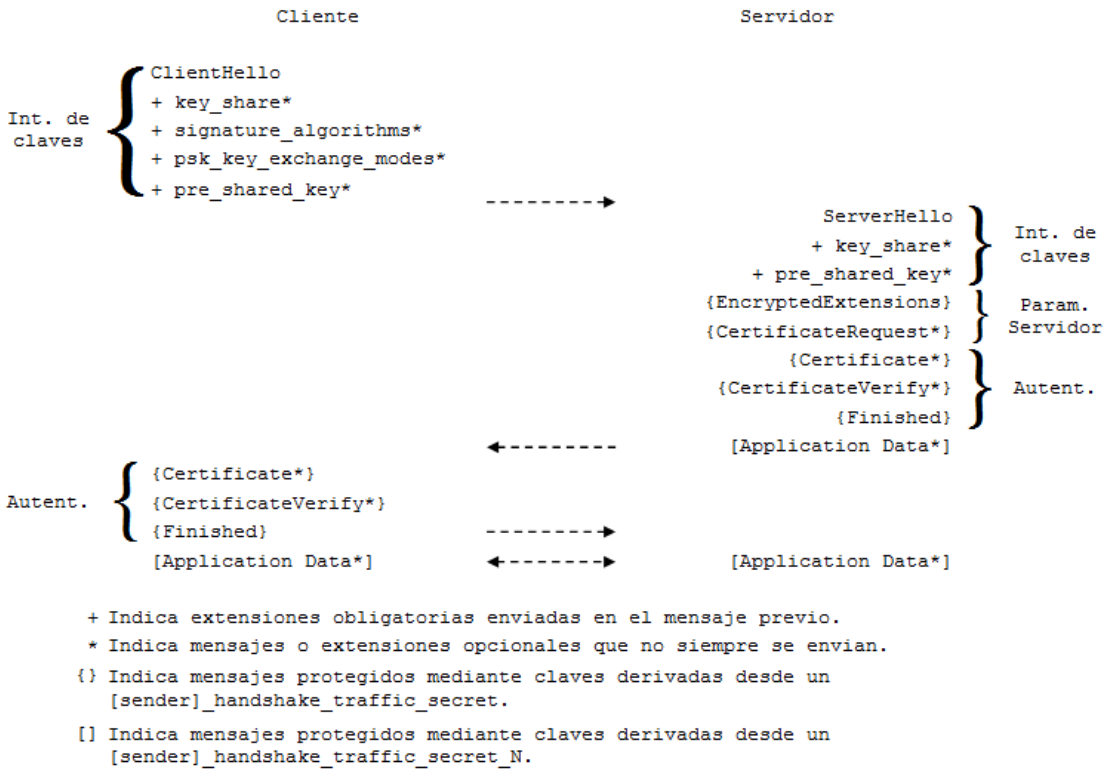

Ilustración 12 TLS handshake: Intercambio de mensajes. Adaptado de RFC8446

Intercambio de claves

Determina las capacidades de seguridad del cliente y el servidor, establece las claves compartidas y selecciona los parámetros criptográficos.

En la fase de intercambio de claves el cliente envía un mensaje denominado *ClientHello*, el cual contiene la versión del protocolo, una lista de cifrados simétricos y algoritmos de firma soportados, un conjunto de recursos compartidos de Diffie-Hellman o un conjunto de etiquetas de claves previamente negociadas y extensiones adicionales.

El servidor procesa el *ClientHello*, determina los apropiados parámetros criptográficos para la conexión y responde con un mensaje *ServerHello* el cual indica los parámetros negociados de la conexión. Esta combinación entre *ClientHello* y *ServerHello* determinan las claves compartidas.

Parametrización del servidor

En esta fase se establecen los parámetros para la autenticación a nivel cliente y el soporte del protocolo a nivel capa de aplicación. Los siguientes dos mensajes del servidor contienen información del servidor que determina el resto del protocolo de enlace:

- *EncryptedExtensions*: contiene las respuestas a las extensiones de *ClientHello* que no son necesarias para determinar los parámetros criptográficos, aparte de aquellos que son específicos para certificados individuales. Es el primer mensaje cifrado con las claves compartidas.
- *CertificateRequest*: si se desea la autenticación del cliente basada en certificados, contiene los parámetros necesarios para el certificado.

Autenticación

Esta fase se ocupa de la autenticación del servidor y provee confirmación de claves e integridad al Handshake.

Al finalizar de establecer los parámetros del servidor, el cliente y el servidor intercambian los mensajes de autenticación. TLS utiliza el mismo conjunto de mensajes cada vez que se requiere llevar a cabo la autenticación basada en certificados. Estos mensajes son:

- *Certificate*: El certificado del servidor y cualquier extensión asociada, este mensaje es omitido por el servidor si no se autentica con un certificado y por el cliente si el servidor no envió *CertificateRequest*.
  Los certificados por defecto son del tipo x509v3 y la clave pública del certificado del servidor debe ser compatible con los algoritmos de autenticación del cliente.
- *CertificateVerify*: Este mensaje se utiliza para probar explícitamente que la otra parte posee la clave privada correspondiente al certificado y brinda integridad al handshake.
  Consiste en una firma sobre todo el protocolo de handshake utilizando la clave privada correspondiente al certificado.
- *Finished*: Un código de autenticación del mensaje para todo el protocolo de handshake, este mensaje proporciona la confirmación de la clave y vincula la identidad del punto final a las claves intercambiadas.

Se cuenta con dos extensiones para indicar qué algoritmos de firma se pueden usar en firmas digitales. La extensión *signature_algorithms_cert* se aplica a las firmas en los certificados, y la extensión *signature_algorithms* se aplica a las firmas en los *CertificateVerify*. En la versión actual estas extensiones contienen varios esquemas de firma, entre los cuales se tienen algunos basados en RSA como: RSASSA-PKCS1-v1_5, RSASSA-PSS y RSA_PKCS1_SHA1.

Una vez recibidos los mensajes cifrados a través del [sender[7]]_handshake_traffic_secret desde el servidor, el cliente responde con sus mensajes de *Certificate*, *CertificateVerify* y *Finished*.

Llegado a este punto el intercambio está completo, por lo cual tanto el cliente como el servidor obtienen las claves requeridas por el TLS Record Protocol para intercambiar los datos por la capa de aplicación a través de un canal cifrado autenticado a través del server_application_traffic_ secret_N.

[7] [sender] denota el lado que envía.

El servidor puede enviar al cliente una identidad p*re_shared_key (PSK)* la cual es una clave única derivada del protocolo handshake, con la cual el cliente puede usarla para reanudar o reestablecer dicha conexión con los mismo parámetros criptográficos y claves generadas con lo cual no requiere realizar nuevamente todo el proceso.

TLS Record Protocol

El TLS Record Protocol toma los mensajes a trasmitir, los fragmenta en partes manejables denominadas bloques, protege los registros y transmite los resultados sobre una capa de registro. Los datos recibidos son verificados, descifrados, resemblados y entregados a las partes.

La capa de registro fragmenta los bloques de datos en registros del tipo *TLSPlainText* que transportan datos en segmentos de un tamaño máximo de 2^{14} bytes, estos límites se manejan dependiendo del *ContentType* del mensaje. Estos mensajes pueden fragmentarse o unirse en registros siempre y cuando no sean intercalados con registros de otro proceso.

El *TLSPlainText* contiene información del protocolo que invoca el proceso, la versión de protocolo, la longitud del siguiente segmento y la información a transmitir.

PGP: Pretty Good Privacy

PGP (Pretty Good Privacy) utiliza una combinación de cifrado de clave pública y cifrado simétrico para proporcionar servicios de seguridad para mensajes de correo electrónico y almacenamiento de datos (Callas, 2007).

En 1990 Phil Zimmermann desarrolla PGP un programa para brindar seguridad en propósitos offline como puede ser firmar y cifrar mails, archivos, etc.

Los certificados son descentralizados por ende no recaen en ninguna autoridad, no utiliza ningún parámetro público y cada usuario es libre de generar su propia clave y elegir el algoritmo de cifrado. Dentro de los cuales se suele utilizar IDEA[8] para cifrado simétrico, RSA para cifrado o firma y MD5 para hash.

Se diseñó como un programa fácil de utilizar en cualquier computadora hogareña pero brindando un nivel de cifrado similar a soluciones militares. En 1991 se libera PGP 1.0, en esa época el congreso de Estados Unidos trabajaba en una ley para regular las herramientas de cifrado (United States Senate, 1991) requiriendo que cada herramienta tenga una puerta trasera que permita al gobierno obtener el texto plano a partir de un texto cifrado y el software que se utilizó para dicha tarea. Esto generó que el programa se distribuya apresuradamente por internet de forma gratuita.

En 1992 salió PGP 2.0 a través de internet también en forma gratuita, esta versión fue desarrollada para arreglar errores de la anterior versión e implementó la utilización de certificados, aunque no utilizaba la infraestructura de clave pública sino certificados autofirmados.

[8] International Data Encryption Algorithm.

Otra barrera que frenó la distribución de PGP fue en 1993 la política de estado de considerar al software de cifrado de alto nivel como un tema de seguridad nacional prohibiendo exportarlo afuera de Estados Unidos sin una licencia por parte del departamento de estado y clasificando las herramientas de cifrado como armas de guerra. Esta limitación se aplicó al software ya desarrollado pero los libros con los códigos fuente podían circular sin inconvenientes, lo cual permitía construir versiones propias de PGP.

En el año 1998 Zimmermann para hacer frente a problemas de patentes de algunos algoritmos de cifrado realiza unos cambios en el diseño de PGP desarrollando OpenPGP. Un protocolo que combina cifrado simétrico con cifrado asimétrico para asegurar los servicios de seguridad para las comunicaciones electrónicas y el almacenamiento de datos; y lo envía al organismo responsable por estándares en internet el IETF[9].

La última versión de esta especificación es la RFC4880 (Callas, 2007) y desde el año 2016 IETF trabaja en un borrador para lograr una nueva actualización[10].

Existen varios software para poder utilizar PGP uno de los más utilizados es Kleopatra dado que funciona sobre Windows y es software libre. En la ilustración 13 se visualiza la interfaz del asistente Kleopatra y su asistente para la creación de claves que admite tanto formato X.509 como OpenPGP.

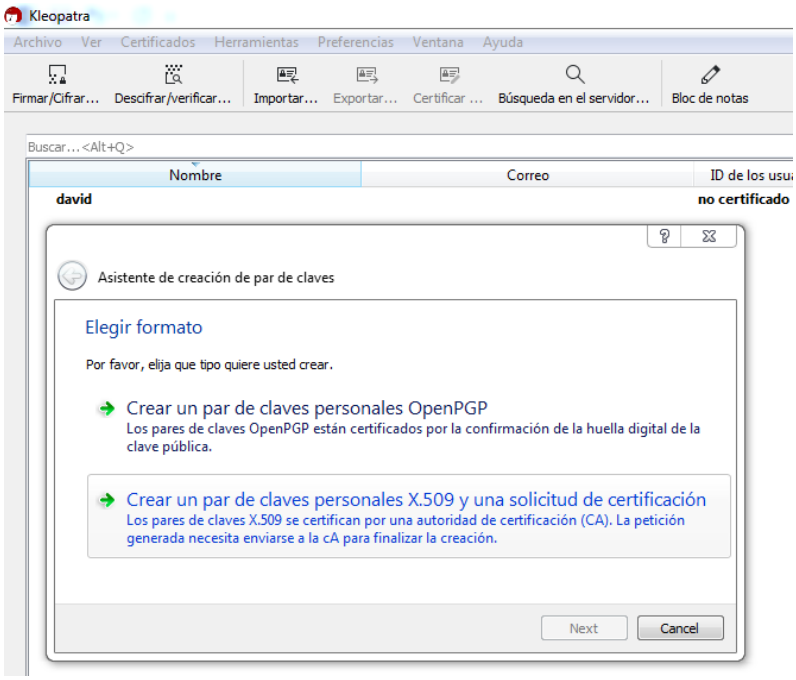

Ilustración 13 Kleopatra: Interfaz libre para administración de claves PGP

[9] Internet Engineering Task Force
[10] Disponible en https://datatracker.ietf.org/doc/draft-ietf-openpgp-rfc4880bis/

PGP define 4 protocolos para transmitir mensajes de e-mail: Autenticación, confidencialidad, compatibilidad de e-mail y segmentación; a continuación se presentan los procedimientos para llevarlos a cabo.

Autenticación

La autenticación se realiza a través de la firma digital del mensaje, este proceso en PGP también incluye la compresión de datos:

1. A crea un mensaje m, que será utilizado con el propósito de autenticarse a sí mismo con B.
2. Se crea un mensaje de resumen de 160 bits llamado $h(m)$ con el algoritmo de cifrado SHA-1.
3. A cifra $h(m)$ con su clave privada RSA k_s^a , obtiene d_a su firma digital para el mensaje m y comprime los datos en D_a, $D_a = \big(d_a\big(h(m)\big), m\big)$. A envía D_a a B.
4. B descomprime los datos recibidos y utilizando la clave publica k_p^a de A descifra y recupera $h(m)$.
5. B recrea el proceso de hash de A aplicando h a m, calculando $h(m)$, y compara este valor al $h(m)$ recibido de A.

Confidencialidad

Para asegurar confidencialidad en PGP se suele utilizar algoritmos simétricos como 3DES o IDEA para generar una clave de sesión y RSA para cifrar la clave de sesión.

1. A genera una cadena k con una longitud de 128 bits que será utilizada como una clave de sesión de único uso para el mensaje m a través de 3DES, previamente el valor fue comprimido por la función Z, así obteniendo $E_k\big(Z(m)\big)$.
2. A cifra k con la clave pública RSA de B (k_p^b) y envía $\Big(k_p^b(k), E_k\big(Z(m)\big)\Big)$ a B.
3. B descifra k con su clave privada y, luego de descomprimir, recupera m utilizando k.

Autenticación y Confidencialidad

En base a los 2 protocolos previos se desarrolla una combinación de ambos para brindar autenticación y confidencialidad.

Se desarrollan primeros los pasos 1, 2 y 3 de autenticación, siguiendo por los pasos 1 y 2 de confidencialidad. Entonces B recupera k con su clave privada RSA y usa k para recuperar D_a utilizando E_k. Por último los pasos 4 y 5 del protocolo de autenticación son ejecutados.

Compatibilidad de e-mail

PGP envía una cadena de bytes con la información sin embargo existen algunos servicios de e-mail que solo permiten transmitir información en ASCII.

La solución de PGP es trasformar la cadena de bytes en una cadena de caracteres ASCII. Esto hace que el tamaño del mensaje aumente alrededor de un tercio no obstante con las técnicas de compresión de datos esto es mitigado.

Segmentación

Los servicios de e-mail suelen tener límites de tamaño para los mensajes a enviar, esto PGP lo mitiga dividiendo el mensaje en partes más pequeñas denominadas segmentos. Los mensajes se componen básicamente de 3 partes:

- Clave de sesión: No solo cumple la función de ser la clave de sesión k, si no también que en base a la clave pública de B denominada k_p^b se calcula $I_{k_p^b} \equiv k_p^b \ (mod\ 2^{64})$, los 64 bits menos significantes de k_p^b con el propósito de identificar de una manera eficiente la clave de B.
- Firma: Puede cumplir varias funciones, entre las cuales se encuentran:
  - El timestamp t_A que corresponde al tiempo de creación de la firma de A.
  - El identificador $I_{k_p^a} \equiv k_p^a \ (mod\ 2^{64})$ similar al componente anterior.
  - Como resumen del mensaje $h(t_A, m)$.
  - Brindar 2 bytes L_1 y L_2 obtenidos de $h(t_A, m)$ los cuales permiten a B asegurarse que la clave k_p^a fue utilizada para descifrar el mensaje.
- Mensaje: La información del mensaje denominada m.

Llaveros (Key Rings)

Las claves públicas y privadas son almacenadas seguramente en las computadoras en estructuras de datos denominadas llaveros públicos y llaveros privados.

Los llaveros son archivos planos que contienen una secuencia de lista de claves. El llavero privado es almacenado solamente en la computadora de su propietario, el cual guarda el par de claves RSA propio y solo accesible para él.

En el llavero privado, la entidad A tiene los siguientes campos:

- Timestamp: el tiempo de creación t_A de $\left(k_p^a, k_s^a\right)$.
- Id de clave: $I_{k_p^a} \equiv k_p^a \ (mod\ 2^{64})$.
- Clave pública: k_p^a.

- Clave privada: k_s^a cifrada utilizando 3DES, IDEA u otro cifrado simétrico. La clave privada plana no es almacenada en la computadora por razones de seguridad, para crear la clave privada A debe:
    1. Elegir una contraseña P que utilizará para cifrar su clave privada.
    2. Cuando PGP genera una nuevo par RSA de claves $\left(k_p^a, k_s^a\right)$, pide que se le ingrese la contraseña P, y utilizando SHA-1 forma un hash $h(p)$ de 160 bits, luego PGP elimina la información de P.
    3. Cifra d_A utilizando $h(p)$ como clave y un algoritmo de cifrado simétrico E, puede ser 3DES, IDEA u otro, resultando $E_{h(p)}(d_A)$, luego elimina la información de $h(p)$. Finalmente $E_{h(p)}(d_A)$ es almacenado en el llavero privado de A.
    4. Cuando A quiere acceder a d_A, debe ingresar su contraseña. PGP realiza la operación inversa ${E^{-1}}_{h(p)}\left(E_{h(p)}(d_A)\right)$: obtiene $E_{h(p)}(d_A)$, genera $h(p)$, y descifra d_A utilizando E y $h(p)$.
- Id de usuario: ID_a definido por PGP.

En el llavero público, cada entrada individual tiene campos similares al llavero privado, con la diferencia más significativa que se agregan unos campos identificadores de confianza de los certificados. La estructura de cada entrada individual en el llavero público de A sería:

- Timestamp: el tiempo de creación de la entrada, t_B.
- Id de clave: $I_{k_p^b} \equiv k_p^b \ (mod\ 2^{64})$.
- Clave pública: k_p^b.
- Confianza propietario: un byte de bandera para la confianza en el propietario de la clave, donde la confianza es asignada por A e indica si la clave pública es confiable para firmar certificados. Cada vez que se agrega una entrada al llavero público se pregunta por el nivel de confianza que se le desea otorgar.
- Id de usuario: ID_b definido por PGP.
- Legitimidad de clave: un byte de bandera, el cual es el nivel de confianza de PGP hacia el propietario de la clave.
- Firma: Cuando se añade una entrada se pueden adjuntar una o más firmas a la misma.
- Confianza de firma: un byte de bandera, el grado de confianza de A asigna a B para certificar claves públicas.

S/MIME: Secure Multipart Internet Mail Extension

S/MIME provee un protocolo consistente para enviar y recibir la información de forma segura basándose en el estándar MIME contando con los servicios de autenticación, integridad, firma digital y confidencialidad (Schaad, 2019).

Otra especificación de e-mail que a veces se utiliza en conjunción con PGP es MIME, desarrollado por IETF para agregar información no textual como gráficos en los mensajes de e-mail.

Sin embargo MIME no tenía seguridad agregada. En 1995 la RSA Data Security Inc con apoyo de IETF mejoraron la especificación creando S/MIME.

S/MIME puede ser utilizado por los agentes de mail de usuario con propósitos de seguridad, pero no está restringido solo a este tipo de servicio sino que se utiliza en mecanismos que transfieren otros tipos de datos MIME como puede ser SIP[11] que es una arquitectura para VoIP o HTTP.

La especificación en su versión 3.0 de 1999 incluye conceptos de infraestructura de clave pública como procesamiento de certificados y listas de revocación de certificados.

El software Kleopatra permite configurar y validar los certificados S/MIME, la interfaz se muestra en la ilustración 14 e incluye diferentes parámetros de validación y verificación.

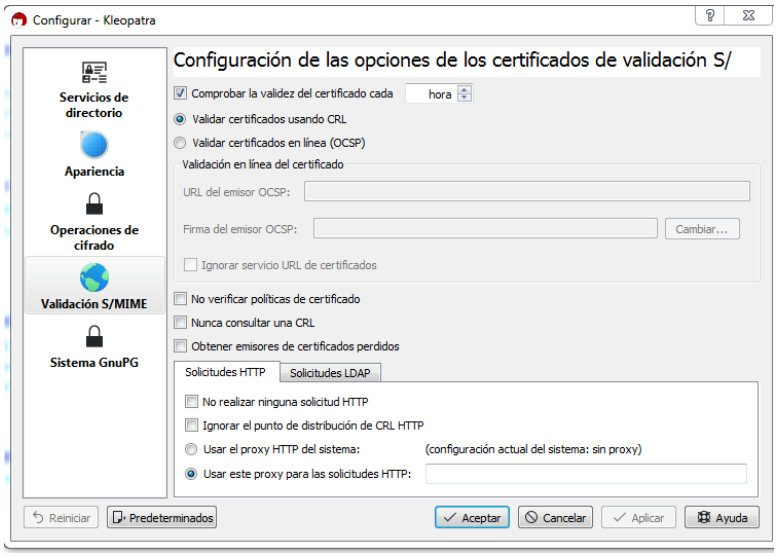

Ilustración 14 Detalles de la configuración para S/MIME en Kleopatra

Luego con la salida de su versión 3.1 en 2004 cambio algunos detalles menores como la firma digital con RSA obligatoria, codificación de archivos y compresión.

La versión 3.2 del 2010 actualiza el paquete de algoritmos de cifrado y actualiza la seguridad del protocolo en general.

La actual versión 4.0 fue especificada en 2019 la cual actualiza algoritmos de cifrado, firma digital y hash. También aumenta el tamaño de las claves utilizadas por RSA tanto para cifrar como firmar.

Dependiendo el escenario que se presente se requiere la autenticación y/o el secreto esto genera distintos protocolos a seguir:

[11] Session Initiation Protocol

S/MIME Protocolo de Autenticación, sin secreto

A desea enviar una prueba de propiedad de un mensaje m a B a través de intercambio de mensajes por e-mail. Sus programas de e-mail utilizan un algoritmo de hash h y un algoritmo de cifrado de clave pública P.

Protocolo:

1. El programa de e-mail de A crea un mensaje de resumen de m utilizando h,$h(m)$, y utiliza su clave privada d_a, obtenida de P, para cifrar y obtener $d_a(h(m))$.
2. Luego envía un mensaje e-mail S/MIME, $M = \left(m, d_A(h(m)), C(A)\right)$ a B donde $C(A)$ es el certificado X.509 de A.
3. El programa de B verifica el $C(A)$ de A y obtiene la clave pública e_A del mismo, descifra y obtiene $h(m)$.
4. El programa e-mail de B computa independientemente $h(m)$ a partir de m y la compara con la $h(m)$ recibida. Si son iguales B está convencido de la identidad de A en el mensaje dado ya que la clave privada de A se ha mantenido segura.

Este protocolo se denomina de solo firma de datos (signed-only data), puede contener cualquier número de certificados que ayuden al programa de e-mail con el trabajo de construir y verificar los certificados.

S/MIME Protocolo de secreto, sin autenticación

A desea enviar un mensaje M a B, tal que no pueda ser leído por otra entidad. Los programas de e-mail utilizado tanto por A y B comparten un común algoritmo de cifrado simétrico E y un criptosistema de clave pública del cual B tiene un par de clave pública y privada (e_B, d_B).

Protocolo:

1. El programa e-mail de A genera una clave aleatoria k, llamada clave de sesión y solamente válida para la transacción del e-mail actual.
2. Luego computa $E_k(M)$, entonces calcular $e_B(k)$, y envía el mensaje de e-mail S/MIME, $C = \left(E_k(M), e_B(k), C(A)\right)$ a B, donde $C(A)$ se genera como en el protocolo anterior.
3. El programa e-mail de B utiliza d_B para obtener k, con el cual obtiene M.

Este protocolo emplea una técnica conocida como envoltura de datos (enveloping data), la cual provee secreto pero no autenticación.

S/MIME Protocolo de autenticación y secreto

Si A y B desean enviar un mensaje el cual sea secreto y autenticado, implementaran un protocolo llamado anidación de protocolos (nesting of protocols).

De esta manera proceden a poner el mensaje m a pasar por los paso 1 y 2 en el protocolo de autenticación para obtener M. Entonces M se encripta a través de los pasos 1 y 2 del protocolo de secreto, con lo cual produce C. B recibe C, primero utiliza d_B para obtener M, luego realiza los pasos 3 y 4 del protocolo de autenticación sobre M para verificar m y la identidad de A. Este protocolo provee tanto seguridad como autenticación.

Es este capítulo se describieron los protocolos criptográficos basados en los algoritmos de Diffie-Hellman y RSA. Los usos más típicos de la criptografía asimétrica consisten en permitir que las entidades puedan intercambiar de manera segura la clave a utilizar y poder verificar a quien pertenece la información suministrada.

Capítulo 5: Conclusiones

Como resultado de este trabajo final integrador se obtuvo una descripción de los protocolos criptográficos basados en RSA y Diffie-Hellman más comúnmente empleados por las aplicaciones actuales.

Se inició con una introducción a la criptografía su historia, conceptos básicos y clasificación, marcando las diferencias que presenta la criptografía simétrica con respecto a la criptografía asimétrica haciendo énfasis en la finalidad de cada una. La criptografía simétrica como instrumento para lograr confidencialidad a través del cifrado del mensaje y la criptografía asimétrica con el fin de realizar un intercambio seguro de claves y la autenticación de las entidades por medio de la firma digital.

También en la introducción se trataron temas como las funciones hash, que sirven de soporte para la integridad y cifrado de datos, y la infraestructura de clave pública que permite, a través de la utilización de certificados, una buena solución para autenticar entidades que necesitan confiar entre sí.

Una vez establecidos los conceptos básicos de criptografía, y como paso previo a analizar los algoritmos, se presentó de forma reducida la teoría numérica que sustenta el funcionamiento de los mismos y son la base tanto de su eficacia, seguridad e eficiencia.

Se describió el protocolo de intercambio de claves Diffie-Hellman, el protocolo de cifrado RSA y el protocolo de firma digital de RSA. En los cuales se analizó el proceso detallado desde un punto vista matemático y de seguridad. Además de su importancia actual, que los hace eficientes, la efectividad del algoritmo RSA proviene del hecho de que es difícil factorizar computacionalmente enteros primos “grandes”; multiplicar dos números primos es fácil pero realizar el proceso inverso, es decir factorizar puede ser realmente difícil; el desafío de factorización RSA promulgado por los Laboratorios RSA en 1991, tiene muchos módulos aún pendientes. En diciembre de 2009, un módulo RSA de 768 bits fue factorizado por un total de 13 investigadores en un lapso de dos años usando cientos de computadoras en paralelo, una tarea equivalente a aproximadamente 2000 años de utilizar un procesador AMD de 2.2 GHz de un solo núcleo (Nisha & Farik, 2017).

La criptografía no es una ciencia nueva y tampoco es nueva su aplicación dentro de la computación, dado que la primera computadora se podría considerar realizada con el propósito de romper el cifrado de un código militar durante la segunda guerra mundial.

No obstante se estudió el impacto que tienen estos algoritmos en la tecnología actual. Siendo estos los encargados de dar soporte a los protocolos criptográficos utilizados para brindar seguridad en las comunicaciones, a través de una computadora o dispositivo móvil.

A modo resumen se realiza el siguiente cuadro, en el cual se visualizan los protocolos estudiados, su cronología, utilización y el aporte de los algoritmos Deffie-Hellman y RSA a cada uno de ellos:

	SSL/TLS	SSH	PGP	SMIME
	1994	1995	1990	1995
Ultima Especificación	RFC8446 (2018)	RFC4254 (2006). Documentos de actualización para Diffie-Hellman (2017) y RSA (2018).	RFC4880 (2007). Nueva especificación en proceso.	RFC8551 (2019).
Utilización	Soporte a: HTTPS. DNS sobre TLS. NNTPS. SMTPS. XMPP. OpenVPN.	Acceso remoto Soporte a: SFTP. Git.	Cifrar y firmar archivos y correos electrónicos. Firmar paquetes de software en Linux.	Correo electrónico. Soporte a: XMPP.
Protocolos de	Intercambio de claves. Registro.	Capa de transporte. Autenticación de usuarios. Conexión.	Autenticación. Confidencialidad. Compatibilidad de e-mail. Segmentación.	Autenticación sin secreto. Secreto sin autenticación. Autenticación y secreto
Capacidades	Intercambio de claves con Diffie-Hellman Firma Digital con RSA	Intercambio de claves con Diffie-Hellman Firma Digital con RSA	Cifrado con RSA Firma Digital con RSA	Cifrado con RSA Firma digital con RSA

Para cumplir con el último objetivo particular “Codificar los algoritmos que se crean pertinentes a fin de que el lector tenga la posibilidad de probar las técnicas y algoritmos que se desarrollan en este trabajo de investigación“ se presenta un anexo. En él se implementa en Python los algoritmos de intercambio de claves de Diffie-Hellman, cifrado y firma digital con RSA, a través de una infraestructura de clave pública en la cual las entidades a comunicarse pueden hacer uso de una autoridad de certificación para obtener los datos de la otra entidad.

Trabajos futuros

Las posibilidades brindadas por este tipo de criptografía no terminan en lo estudiado en este trabajo quedando a futuro trabajos en los cuales se analicen otras infraestructuras como Blockchain o Ethereum que permiten el funcionamiento de las criptomonedas, tema de interés tecnológico y económico actual.

Un campo a analizar es la emergente criptografía cuántica y como se podrán implementar los criptosistemas de clave pública en ella. Las computadoras cuánticas serían capaces de romper la seguridad de las implementaciones actuales basadas en la dificultad de ciertas operaciones matemáticas, por lo cual sería de interés analizar los cambios que deberían hacerse para asegurar la seguridad de la firma digital y cifrado de datos con RSA.

Otra falencia detectada en la criptografía de clave pública es su velocidad, la cual resulta ineficiente para cifrar grandes volúmenes de datos. Como trabajos futuros se podría analizar el álgebra computacional que permite a estos algoritmos funcionar y plantear algoritmos semejantes que mejoren la velocidad de generación de grandes números primos, exponenciación modular, inversos multiplicativos, búsqueda de generadores de grupo, entre otros.

Otra propuesta es desarrollar un nuevo criptosistema de clave pública basado en otro problema matemático o analizar otro enfoque. El cual sin reducir su nivel de seguridad ni tampoco las ventajas que este brinda, pueda de una forma eficiente cifrar una gran cantidad de volúmenes de datos en un tiempo aceptable.

Las redes de nueva generación (redes móviles, inalámbricas e Internet de las Cosas) imponen restricciones en cuanto al poder de procesamiento, ancho de banda y recursos de energía, lo que representa una gran limitante en la implementación de mecanismos de seguridad. En este sentido, en lo últimos años han surgido diversas propuestas para la administración de claves, procedimientos de firma digital y cifrado de datos basados en curvas elípticas y criptografía ligera, que logran niveles de seguridad equivalentes a los algoritmos convencionales basados en algoritmos Diffie-Hellman y RSA, pero que reducen la longitud de clave y, en consecuencia, el uso de los recursos computacionales.

Bibliografía

Adkins, W., & Weintraub, S. (1992). *Algebra, An Approach via Module Theory. Graduate Texts in Mathematics.* New York: Springer-Verlag.

Aumasson, J. P. (2017). *Serious Cryptography: A Practical Introduction to Modern Encryption.* No Starch Press.

Barakat, M., Eder, C., & Hanke, T. (2018). An Introduction to Cryptography. University of Kaiserslautern. Obtenido de https://www.mathematik.uni-kl.de/~ederc/download/Cryptography.pdf

Baushke, M. (2017). *RFC8268: More Modular Exponentiation (MODP) Diffie-Hellman (DH). Key Exchange (KEX) Groups for Secure Shell (SSH)*. Obtenido de https://tools.ietf.org/rfc/rfc8268.txt

Bider, D. (2018). *RFC8332: Use of RSA Keys with SHA-256 and SHA-512 in the Secure Shell (SSH) Protocol*. Obtenido de https://tools.ietf.org/rfc/rfc8332.txt

Boneh, D., & Shoup, V. (2020). *A Graduate Course in Applied Cryptography v0.5.* Stanford University.

Callas, e. a. (2007). *RFC 4880: OpenPGP Message Format*. Obtenido de https://tools.ietf.org/rfc/rfc4880.txt

Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory, 22*(6), 644-654.

Dixon, M., Kurdachenko, L., & Ya Su, I. (2014). *An Introduction to Essential Algebraic Structures.* John Wiley & Sons, Ltd.

Dorey, K., Chang-Fong, N., & Essex, A. (2017). Indiscreet Logs: Diffie-Hellman Backdoors in TLS.

Farlow, S. J. (2019). *Advanced Mathematics: A transitional Reference.* Wiley.

Galbraith, S. (2012). *Mathematics of Public Key Cryptography.* Cambridge University Press.

Harchol, Y., Abraham, I., & Pinkas, B. (2018). Distributed SSH Key Management with Proactive RSA Threshold Signatures. *Applied Cryptography and Network Security*.

Hardy, D., Richman, F., & Walker, C. (2009). *Applied Algebra: Codes, Ciphers and Discrete Algorithms, Second Edition (Discrete Mathematics and Its Applications).* Chapman and Hall/CRC, 2nd edition.

Jothy, K., Sivakumar, K., & Delsey, M. (2017). Efficient Cloud Computing with Secure Data Storage Using AES and PGP Algorithm. *International Journal of Computer Science and Information Technologies, 8*(6), 582-585.

Katz, J., & Lindell, Y. (2007). *Introduction to Modern Cryptography: Principles and Protocols.* Chapman \& Hall / CRC Press.

Kuppuswamy, P., & Al-Khalidi, S. (2014). Hybrid encryption/decryption technique using new public key and symmetric key algorithm. *International Journal of Information and Computer Security, 6*, 372.

Lenstra, A., & Verheul, E. (2001). Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 255-293.

Levin, O. (2018). *Discrete Mathematics: An Open Introduction.* Independently published.

Liben-Nowell, D. (2017). *Discrete Mathematics for Computer Science.* Wiley.

McLaren, P., Russell, G., Buchanan, W., & Tan, Z. (2019). Decrypting live SSH traffic in virtual environments. *Digital Investigation*, 109-117.

Mogollon, M. (2008). *Cryptography and Security Services: Mechanisms and Applications.* CyberTech Publishing.

Mollin, R. A. (2007). *"An Introduction to Cryptography".* Chapman \& Hall / CRC Press; 2nd edition.

Nisha, S., & Farik, M. (2017). RSA Public Key Cryptography Algorithm – A Review. *International Journal Of Scientific & Technology Research, 6*(7), 187-191.

Oppliger, R. (2016). *SSL and TLS: Theory and Practice* (2nd ed.). Artech House.

Pineda Vargas, M., Salcedo Parra, O., & Acosta Rodríguez , R. (2017). Algorithm for the optimization of RSA based on parallelization over GPU SSL/TLS Protocol. *IEEE International Conference on Smart Cloud, 1*, 294-297.

Poddebniak, D., Dresen, C., Somorovsky, J., Schwenk, J., Mülle, J., Ising, F., . . . Friedberger, S. (2018). Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels.

Raji, M., Amiri, F., & Ahmadian, M. (2016). A New secure email scheme Using Digital Signature with S/MIME. *International Journal of Computer Networks and Communications Security, 4*(3), 56-62.

Razaghpanah, A., Niaki, A., Vallina-Rodriguez, N., Sundaresan, S., Amann, J., & Gill, P. (2017). Studying TLS Usage in Android Apps. *CoNEXT '17: Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 350-362.

Rescorla, E. (2018). *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3.* Obtenido de https://tools.ietf.org/rfc/rfc8446.txt

Roh, C.-H., & Lee, I.-Y. (2018). A Study on PGP (Pretty Good Privacy) Using Blockchain. *CSA 2018: Advances in Computer Science and Ubiquitous Computing*, 316-320.

Ruohonen, K. (2014). *Mathematical Cryptology.* CreateSpace Independent Publishing Platform.

Schaad, e. a. (2019). *RFC 8551: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0.* Obtenido de https://tools.ietf.org/rfc/rfc8551.txt

Schoenmakers, B. (2019). Lecture Notes Cryptographic Protocols. *1.4*. P.O. Box 513, 5600 MB Eindhoven, The Netherlands: Department of Mathematics and Computer Science,

Technical University of Eindhoven. Obtenido de https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf

Shannon, C. (1949). Communication theory of secrecy systems. *Bell System Technical Journal, 28*, 656-715.

Shirey, R. (2007). *RFC 4949: Internet Security Glossary, Version 2*. Obtenido de https://tools.ietf.org/rfc/rfc4949.txt

Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice* (Seventh ed.). Pearson.

Stinson, D., & Paterson, M. (2018). *“Cryptography - Theory and Practice”, Textbooks in Mathematics.* Chapman \& Hall / CRC Press,4th edition.

Suprihanto, D., & Priyambodo, T. (2017). The Implementation of Pretty Good Privacy in eGovernment Applications (Case Study on the Official Scripts Electronic Applications in Bantul).

Trappe, W., & Washington, L. C. (2005). *"Introduction to Cryptography with Coding Theory".* Pearson; 2nd edition.

United States Senate. (1991). S.266 Comprehensive Counter-Terrorism Act of 1991. *Subtitle B: Electronic Communications: Section 2201, Cooperation Of Telecommunications Providers With Law Enforcement*. U.S. Library of Congress. Obtenido de https://www.congress.gov/bill/102nd-congress/senate-bill/266

Vaudenay, S. (2006). *"A Classical Introduction to Cryptography: Applications for Communications Security".* Springer US.

Weimer, F. (2015). Factoring RSA Keys With TLS Perfect Forward Secrecy.

Wu, H., & Chiu, R. (2017). Implementation a Secure Electronic Medical Records Exchange System Based on S/MIME. *Advances in Science, Technology and Engineering Systems Journal, 2*, 172-176.

Yakubov, A., Shbair, W., & State, R. (2018). BlockPGP: A Blockchain-Based Framework for PGP Key Servers. *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, 316-322.

Ylonen, T. (2019). SSH Key Management Challenges and Requirements. *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, (págs. 1-5). CANARY ISLANDS.

Ylonen, T., & C. Lonvick, E. (2006a). *RFC 4251: The Secure Shell (SSH) Protocol Architecture.* Obtenido de https://tools.ietf.org/rfc/rfc4251.txt

Ylonen, T., & C. Lonvick, E. (2006b). *RFC 4252: The Secure Shell (SSH) Authentication Protocol*. Obtenido de https://tools.ietf.org/rfc/rfc4252.txt

Ylonen, T., & C. Lonvick, E. (2006c). *RFC 4253: The Secure Shell (SSH) Transport Layer Protocol*. Obtenido de https://tools.ietf.org/rfc/rfc4253.txt

Ylonen, T., & C. Lonvick, E. (2006d). *RFC 4254: The Secure Shell (SSH) Connection Protocol*. Obtenido de https://tools.ietf.org/rfc/rfc4254.txt

Anexo

En este anexo se encuentran los códigos de los algoritmos mencionados en el trabajo, junto a funciones extras que permiten que el lector pueda simular un entorno de intercambio de claves con Diffie-Hellman, cifrado con RSA o firma digital con RSA.

Los ejemplos mostrados en el trabajo fueron obtenidos a través los algoritmos aquí desarrollados.

Los códigos son de autoría propia, los cuales se pueden utilizar para fines educativos y de investigación. Como así también cualquier error que se le puedan detectar o mejora que se le puedan realizar es siempre bienvenida.

Se utilizó Python en su versión 3.7.2. No obstante los algoritmos funcionarían con una versión mínima 3.6, la cual introdujo entre sus característica la librería *secrets* que permite la generación de números aleatorios con fines criptográficos.

Algunas acotaciones referentes al código:

- El algoritmo de Euclides Extendido: se seleccionó un enfoque iterativo de este algoritmo dada la simplicidad que presentó.
- El algoritmo de cálculo de inverso multiplicativo modular: si bien en la mayoría de la bibliografía se realiza por el Teorema de Euler motivado por su sencillez conceptual. Este Teorema solo es aplicable si el modulo es primo y es menos eficiente comparado con el algoritmo de Euclides, esto generó la selección de este algoritmo para la implementación.
- El algoritmo de MessageBlocking: se ocupa este método para dividir el mensaje en bloques para luego cifrar con RSA, en la bibliografía se da ejemplos de este proceso pero nunca brinda un algoritmo que se pueda adaptar a las necesidades del lector.
- El algoritmo generador: encargado de buscar un generador para un grupo multiplicativo, al igual que sucede con MessageBlocking, ningún libro de criptografía desarrolla métodos para encontrarlos de una forma que no sea una búsqueda lineal. Mientras que el algoritmo desarrollado surgió luego de mucha investigación en sitios dedicados a la programación.
- La clase CA: se desarrolló un modelo básico en el cuál solo almacena y brinda claves públicas, dado que no es el objetivo del trabajo modelar al detalle una infraestructura de clave pública.
- Obtención de los números primos para RSA: la generación del par de primos (p, q) se realizó utilizando el método *randbits(k)* de la librería *secrets*. Esto asegura que el par de números aleatorios tengan un tamaño fijo, pero no tiene en cuenta algunas condiciones extras que deben tener estos números para que sean seguros criptográficamente.

Funciones auxiliares a los algoritmos

Incluyen las funciones matemáticas necesarias realizar Diffie-Hellman y RSA, como así también las funciones para Message Blocking.

```
## ------------------- Librerias ---------------------

## --> math
## https://docs.python.org/3/library/math.html
## This module is always available. It provides access to the
mathematical functions defined by the C standard.

import math
## --> secrets
## https://docs.python.org/3/library/secrets.html
## The secrets module is used for generating cryptographically strong
random numbers suitable for managing data
## such as passwords, account authentication, security tokens, and
related secrets.

## secrets.randbelow(n)
##       Return a random int in the range [0, n).

## secrets.randbits(k)
##       Return an int with k random bits.

import secrets

## ------------------- Funciones ---------------------

def esPrimo(n):
    """Devuelve si el parámetro "n" es primo o no.

    Parámetros:
    n -- Número entero positivo.

    Return:
    Bool -- True si es primo, False si es compuesto.

    """
    if n<2:
        return False
    if n>2 and n%2==0:
        return False
    for i in range(3,round(math.sqrt(n))+1,2):
        if(n%i==0):
            return False
    return True



def obtenerPrimo(p,b):
    """Devuelve p si este es primo, sino genera aleatoriamente otro
entero "p"
       con una longitud de "b" bits y repite el proceso.

    Parámetros:
```

```
    p -- Número entero positivo.
    b -- Cantidad de bits.

    Return:
    Int -- Número primo.

    """
    if esPrimo(p)==True:
        return p
    else:
        return obtenerPrimo(secrets.randbits(b),b)

def mcd(a,b):
    """Devuelve el máximo comun divisor de 2 números.

    Parámetros:
    a -- Número entero positivo.
    b -- Número entero positivo.

    Return:
    Int -- Máximo comun divisor.

    """
    if b == 0:
        return a
    else:
        return mcd(b, a % b)

def obtenerCoPrimo(n,li,ls):
    """Obtiene un número "p" coprimo con "n" con la condición que li < p
< ls.

    Parámetros:
    n  -- Número entero positivo.
    li -- Número entero positivo, limite inferior.
    ls -- Número entero positivo, limite superior.

    Return:
    Int -- Número p, coprimo con n en el intervalo requerido.

    """
    e = (li + 1) + secrets.randbelow(ls - 1)
    if(mcd(e,n)==1):
        return e
    else:
        return obtenerCoPrimo(n,li,ls)




def euclidesExt(a, b):
    """
    Obtiene una tupla basada en el Algoritmo Extendido de Euclides
también denominado teorema de Bezout.
    De la forma ax +by = d = mcd(a,b) -> ( d , x , y ).
```

```
    Fuente:
http://www.wikiwand.com/es/Discusi%C3%B3n:Algoritmo_de_Euclides


    Parámetros:
    a -- Número entero positivo.
    b -- Número entero positivo.

    divmod(a, b): Take two (non complex) numbers as arguments and return
a pair of numbers consisting of their quotient
    and remainder when using integer division. For integers, the result
is the same as (a // b, a % b)

    Return:
    Tuple -- ( d , x , y ).

    """
    (s, t, s1, t1) = (1, 0, 0, 1)
    while b != 0:
        (q, r) = divmod(a, b)
        (a, s, t, b, s1, t1) = (b, s1, t1, r, s - s1*q, t - t1*q)
    return (a, s, t)

def inversoModN(a,n):
    """
    Obtiene el inverso multiplicativo de a modulo n.

    ab ≡ a(a^(-1)) ≡ 1 (mod n)

    Basado en el algoritmo de euclidesExt que genera ax +by = d =
mcd(a,b) y devuelve ( d , x , y ).
    El b en cuestión requerido es x.

    Parámetros:
    a -- Número entero positivo, que se requiere inverso.
    n -- Número entero positivo, modulo.

    Return:
    Int -- Devuelve b.

    """
    inv = euclidesExt(a,n)[1]
    while inv<0 or inv>=n:
        if inv<0:
            inv=inv+n
        if inv>=n:
            inv = inv -n
    return inv


def expModular(a,b,n):
    """
    Sea n∈Z^+,n>1 y  a,b,x ∈ Z se requiere calcular el valor de x ≡ a^b
(mod n).
```

```
    Convertir el número b en su expresión binaria b = b_k b_(k-1)….b_2
b_1 b_0.
    Primero se analiza b_0, se asigna p = a  y:
        - Si b_0 = 1 almacenar c_0 ≡ p (mod n)
        - Si b_0 = 0 almacenar c_0 = 1.
    Luego ∀ i tal que 0<i≤k, se asigna p=p^2 y:
        - Si b_i = 1 almacenar c_i ≡ p (mod n).
        - Si b_i = 0 almacenar c_i = 1 .
    Se resolverá el valor x multiplicando entre si los c_i.


    Parámetros:
    a -- Número entero positivo.
    b -- Número entero positivo.
    n -- Número entero positivo, modulo.

    Return:
    Int -- Devuelve x.

    """
    c = 1
    p = a%n
    while b>0:
        if b%2==1:
            c = (c*p)%n
        b = b//2
        p = pow(p,2)%n
    return c


def cadenaValida(cadena):
    """
    Determina si la cadena ingresada está conformada por caracteres
alfabeticos válidos o no.

    Parámetros:
    cadena -- Cadena de Texto

    Return:
    Bool -- False si detecta un caracter no valido, True caso contrario.

    """
    alfabetoPermitido =
set(["|","a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p",
"q","r","s","t","u","v","w","x","y","z"," "])
    cadena = set(cadena)
    caracteresNoValidos = cadena - alfabetoPermitido
    if len(caracteresNoValidos)==0:
        return True
    else:
        return False
  def messageBlocking(clave,msje,base,f):
    """
    Dividir el texto plano en bloques, cuando el texto plano como valor
numérico es m ≥ n.
```

```
    Se tiene que subdividir el texto plano en bloques numéricamente de
igual tamaño en un proceso llamado message blocking.
    Suponiendo que se utiliza un equivalente numérico del texto plano en
base  N, para algún entero N>1 y se elige un entero l talque N^l < n <
N^(l+1)
    entonces se divide el mensaje en bloques de l dígitos y base N con
ceros de relleno en el último bloque de ser necesario y se lo cifra por
separado.


    Parámetros:
    clave -- [array]Clave Pública.
    msje  -- Texto en claro.
    base  -- Longitud de alfabeto utilizado.
    f     -- bandera de cifrado / descifrado.

    Return:
    String -- [array] Mensaje en bloques.

    """
    t = messageBlockingTamBloques(clave,msje,base)
    b = []
    if t==0:
        b.append(msje)
        return b
    if f =='d':t = t+1
    if f =='e':
        while len(msje)%t!=0:
            msje = msje + "|"
    for x in range(0,len(msje),t):
        c = msje[x:x+t]
        if f =='d' and x == len(msje):
            while len(msje)%t!=0:
                c = "|" + c
            b.append(c)
        else: b.append(c)
    return b

def messageBlockingTamBloques(clave,msje,base):
    """
    Función auxiliar la cual determina el tamaño de los bloques.


    Parámetros:
    clave -- [array]Clave Pública.
    msje  -- Texto en claro.
    base  -- Longitud de alfabeto utilizado.

    Return:
    Int -- Tamaño de cada bloque.

    """
    # primero se considera si no es necesario el message blockin cuando n
< N
    if clave[2]<base:
        return 0
```

```
    corte = False
    i = 2
    while corte == False:
        x = pow(base, i)
        if x >= clave[2]:
            return i-1
        i = i+1


def factores(n):
      """Devuelve un array con los factores de un número entero.

      n= p_1^(e_1 ) * p_2^(e_2 ) * … *p_k^(e_k ) =>  [

            [p_1,e_1],

            [p_2,e_2],

            ....

            [p_k,e_k]

      ]

    Parámetros:
    n -- Número entero positivo.

    Return:
    Array[Int] -- Array de factores con exponentes de un número entero
positivo.

    """
    lista = []
    if esPrimo(n):
        lista.append([n,1])
        return lista
    c = 0
    if n%2==0:
        while n%2 == 0:
            n = n // 2
            c = c + 1
        lista.append([2,c])
    i = 3
    while i<=n:
        c = 0
        if n%i==0:
            while n%i == 0:
                n = n // i
                c = c + 1
            lista.append([i,c])
        i = i + 2
    return lista


def generador(p):
```

```
    """Devuelve un generador del grupo p: g, talque 2 <= g <= p-2 y
mcd(g,p) = 1

      Fuente: https://cp-algorithms.com/algebra/primitive-root.html

      First, find ϕ(n) and factorize it.
      Then iterate through all numbers g∈[1,n], and for each number, to
check if it is primitive root, we do the following:

      Calculate all g^(ϕ(n)pi)( mod n ).
      If all the calculated values are different from 1, then g is a
primitive root.

    Parámetros:
    p -- Número entero positivo primo.

    Return:
    Int -- Generador del grupo

    """
    q = [row[0] for row in factores(p-1)]
    for i in range(2,p):
        if mcd(p,i)==1:
            corte = 1
            for j in range(0,len(q)):
                aux = expModular(i,(p-1)/q[j],p)
                if aux==1:
                    corte = 0
                    break
            if corte == 1:
                return i
```

Clase Entidad

Es la clase que modela a las entidades que necesitan comunicarse, ya sea para intercambiar claves, mensajes cifrados o mensajes firmados.

```
## ------------------- Clases ---------------------
from class_RSA import *
from class_DH import *

class Entidad:
    """
    Clase que modela las entidades que desean  comunicarse.

    Attributes
    ----------
    nombre: string
        Nombre identificatorio de la entidad.
    rsa: rsa
        Objeto que maneja todos los algoritmos RSA.
    dh: dh
        Objeto que maneja todos los algoritmos DH.
    kp: dict
        Clave publica en formato {'n': ..., 'e': ...}
```

```
    bd: array[nombre,e,n]
        Base de datos propias con información de las entidades que
previamente se ha comunicado.

    Methods
    -------
    generacionClave()
        Utilizando RSA genera las claves y devuelve las claves publicas y
privadas.
    publicar()
        Publica [nombre,kp].
    añadirDatos(array[int])
        Recibe los datos de clave publica de otra entidad y los almacena
en la bd local.
        Primera valida si ya los tiene, si los tiene omite añadir
        Luego verifica si recibió datos, si no es asi omite.
    textoCifrado(entidadDestino,msje)
        Retorna el texto cifrado basandose en el msje y clave publica de
la entidad.
    descifrar(msje)
        Recibe el texto cifrado y retorna  el texto plano.

    """
    def __init__(self,nombre=''):
        self.nombre = nombre
        self.rsa = RSA()
        self.dh = DH()
        self.kp = {}
        self.bd = []
    def generacionClave(self):
        self.kp = self.rsa.generarClaves()
    def publicar(self):
        return [self.nombre,self.kp['e'],self.kp['n']]
    def buscar(self,nombre):
        for record in self.bd:
            if record[0]==nombre:
               return record
        return False;
    def añadirDatos(self,datos):
        if datos != False:
             self.bd.append(datos)
    def textoCifrado(self,entidadDestino,msje):
        texto = self.rsa.cifrarTexto(self.buscar(entidadDestino),msje)
        return texto
    def descifrar(self,msje):
        texto = self.rsa.descifrarTexto(msje)
        return texto
```

Clase Autoridad de Certificación

Es la clase que modela a la entidad de certificación, que tiene por objetivo en una infraestructura de clave pública, almacenar las claves públicas de las entidades y brindar la facilidad de obtención de las mismas para otras entidades.

```
class CA:
    """
    Clase que representa el CA


    Attributes
    ----------
    bd: [array[string]]
        Base de datos de las claves publicas.

    Methods
    -------
    añadir(array[nombre,e,n])
        Recibe desde una entidad los datos necesarios para guardar la
clave pública.
    buscar(string nombre)
        Busca una entidad por nombre y devuelve los datos de la clave
pública.

    """
    def __init__(self):
        self.bd = []
    def añadir(self,inf):
        if(self.buscar(inf[0])==None):
            self.bd.append(inf)
        else: print("registro ya cargado");
    def buscar(self,nombre):
       for record in self.bd:
        if record[0]==nombre:
           return record
        return False;
```

Clase Diffie-Hellman

Es la clase que modela al protocolo Diffie-Hellman, con el objetivo de intercambiar claves.

```
#Versión Python 3.7.2

## ------------------- Librerias ---------------------

## --> math
## https://docs.python.org/3/library/math.html
## This module is always available. It provides access to the
mathematical functions defined by the C standard.

import math

## --> secrets
## https://docs.python.org/3/library/secrets.html
## The secrets module is used for generating cryptographically strong
random numbers suitable for managing data
## such as passwords, account authentication, security tokens, and
related secrets.
```

```
## secrets.randbelow(n)
##      Return a random int in the range [0, n).

## secrets.randbits(k)
##      Return an int with k random bits.

import secrets

## --> funciones
## libreria de funciones matematicas auxiliares
## obtenerPrimo(p,b)
##      Devuelve p si este es primo, sino genera aleatoriamente otro
entero "p"
##      con una longitud de "b" bits y repite el proceso.
## expModular(a,b,n)
##      Calcula el valor de x ≡ a^b  (mod n).
## generador(p)
##      Devuelve generador del grupo p: g, talque 2 <= g <= p-2 y
mcd(g,p) = 1


from funciones import *

class DH:
    """
    Clase que representa el algoritmo Diffie-Hellman.


    Attributes
    ----------
    bits : int
        Cantidad de bits para los numeros aleatorios
    p : int
        Número primo aleatorio.
    g : int
        Generador de p.
    e : int
        Exponente.
    X : int
        Valor que se calcula como X ≡ g^e (mod p) y es utilizado para
enviar a la otra entidad.
    k : int
        Clave.
    Methods
    -------
    generarPrimo()
        Metodo que se encarga de generar el número primo p.
        Utiliza:
            - obtenerPrimo()
    generador()
        Metodo que se encarga de encontrar un generador g de p


    """
    def __init__(self,nombre=''):
```

```
        self.b = 13
        self.p = 0
        self.g = 0
        self.e = 0
        self.X = 0
        self.k = 0
    def generador(self):
        self.g = generador(self.p)
    def generarPrimo(self):
        self.p = obtenerPrimo(secrets.randbits(self.b),self.b)
    def generarE(self):
        self.e = secrets.randbits(self.b)
    def generarX(self):
        self.X = expModular(self.g,self.e,self.p)
    def generarClave(self,x):
        self.k = expModular(x,self.e,self.p)
```

Clase RSA

Es la clase que modela al protocolo RSA, con el objetivo de cifrar y/o firmar mensajes.

```
## ------------------- Librerias ---------------------

## --> math
## https://docs.python.org/3/library/math.html
## This module is always available. It provides access to the
mathematical functions defined by the C standard.

import math

## --> secrets
## https://docs.python.org/3/library/secrets.html
## The secrets module is used for generating cryptographically strong
random numbers suitable for managing data
## such as passwords, account authentication, security tokens, and
related secrets.

## secrets.randbelow(n)
##      Return a random int in the range [0, n).

## secrets.randbits(k)
##      Return an int with k random bits.

import secrets

## --> hashlib
## https://docs.python.org/3.7/library/hashlib.html
## This module implements a common interface to many different secure
hash and message digest algorithms.


import hashlib
```

```
## --> funciones
## libreria de funciones matematicas auxiliares
## esPrimo(n)
##      Devuelve si el parámetro "n" es primo o no.
## obtenerPrimo(p,b)
##      Devuelve p si este es primo, sino genera aleatoriamente otro
entero "p"
##      con una longitud de "b" bits y repite el proceso.
## mcd(a,b)
##      Devuelve el máximo comun divisor de 2 números.
## obtenerCoPrimo(n,li,ls)
##      Obtiene un número "p" coprimo con "n" con la condición que li < p
< ls.
## euclidesExt(a, b)
##      Obtiene una tupla basada en el Algoritmo Extendido de Euclides
también denominado teorema de Bezout.
##      De la forma ax +by = d = mcd(a,b) -> ( d , x , y ).
## inversoModN(a,n)
##      Obtiene el inverso multiplicativo de a modulo n.
## expModular(a,b,n)


from funciones import *

class RSA:
    """
    Clase que representa el algoritmo RSA.


    Attributes
    ----------
    bits : int
        Cantidad de bits para los numeros aleatorios
    alfabeto : int
        Caracteres que podran ser cifrados / descifrados.
        El caracter "|" fue elegijo como relleno de bloque, por ende hay
que eliminar su representacio nvisual
    p : int
        Número primo aleatorio.
    q : int
        Número primo aleatorio.
    n : int
        Modulo p*q , talque p≠q.
    phiN : int
        Phi de n = (p-1)(q-1).
    e : int
        Exponente de Cifrado.
    d : int
        Exponente de Descifrado y  Clave Privada.
    Methods
    -------
    generacionClave()
        Metodo que se encarga de generar los números necesarios para el
funcionamiento de RSA.
        Ocupa los métodos:
         - generarPrimos()
```

```
            Genera 2 números primos  p y q, talque p≠q.
        - generarN()
            Computa n = pq, donde a n se lo denomina el modulo.
        - generarPhiN()
            Computa φ(n) = (p-1)(q-1).
        - generarE()
            Se selecciona un e, talque 1<e<φ(n) y el mcd(e,φ(n) )=1.
        - generarD()
            Utilizando el algoritmo de Euclides extendido, se computa el
único d
            1<d<φ(n) talque ed≡1 (mod φ(n)).

    """
    def __init__(self,nombre=''):
        self.alfabeto =
["|","a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q",
"r","s","t","u","v","w","x","y","z"," "]
        self.b = 10
        self.p = 0
        self.q = 0
        self.n = 0
        self.phiN = 0
        self.e = 0
        self.d = 0
    def generarClaves(self):
        self.generarPrimos()
        self.generarN()
        self.generarPhiN()
        self.generarE()
        self.generarD()
        return {'n':self.n, 'e': self.e}
    def generarPrimos(self):
        self.p = obtenerPrimo(secrets.randbits(self.b),self.b)
        self.q = obtenerPrimo(secrets.randbits(self.b),self.b)
        while(self.p==self.q):
            self.q = obtenerPrimo(secrets.randbits(self.b),self.b)
    def generarN(self):
        self.n = self.p*self.q
    def generarPhiN(self):
        self.phiN = (self.p-1)*(self.q-1)
    def generarE(self):
        self.e = obtenerCoPrimo(self.phiN,1,self.phiN)
    def generarD(self):
        self.d = inversoModN(self.e,self.phiN)
    def cifrarTexto(self,clave,msje):
        b = messageBlocking(clave,msje,len(self.alfabeto),'e')
        s = []
        N = len(self.alfabeto)
        alfabeto = self.alfabeto
        for x in range(0,len(b)):
            salida =""
            c = 0
            i= 0
            for y in range(len(b[x]), 0, -1):
                c = c+(alfabeto.index(b[x][y-1]))*pow(N,i)
                i = i+1
```

```
            c = expModular(c,clave[1],clave[2])
            salida=alfabeto[c%N] +salida
            for i in range(0,len(b[x])):
                c  = c // N
                if c>=N:
                    cadena = alfabeto[c%N]
                else:
                    cadena = alfabeto[c]
                salida = cadena + salida
            s.append(salida)
        cGenerado =''.join(s)
        return cGenerado
    def descifrarTexto(self,msje):
        alfabeto = self.alfabeto
        N = len(alfabeto)
        claves = ["",self.e,self.n]
        b = messageBlocking(claves,msje,N,'d')
        s = []
        for x in range(0,len(b)):
            salida =""
            c = 0
            i= 0
            for y in range(len(b[x]), 0, -1):

                c = c+(alfabeto.index(b[x][y-1]))*pow(N, i)
                i = i+1
            c = expModular(c,self.d,self.n)
            while c > N:
                salida=alfabeto[c%N] +salida
                c  = c // N
            salida = alfabeto[c%N]+salida
            s.append(salida)
        cGenerado =''.join(s).replace("|", "")
        return cGenerado
    def firmarMensaje(self,mensaje):
        h = int(hashlib.md5(mensaje.encode('utf-8')).hexdigest(),16)
        return expModular(h,self.d,self.n)
    def verificarMensaje(self,mensaje,s,clave):
        h = expModular(s,clave[1],clave[2])
        h2 = int(hashlib.md5(mensaje.encode('utf-8')).hexdigest(),16) %
clave[2]
        if h==h2:
            return True
        else:
            return False
```

Script para cifrar mensajes con RSA

En el siguiente código se muestra como es el proceso de cifrar un mensaje utilizando RSA y una autoridad de certificación. Dejando en claro que el texto a cifrar está limitado a los caracteres alfabéticos en minúsculas y el carácter espacio, para ampliar al conjunto de entrada se deberá modificar la variable *alfabetoPermitido* dentro de la función *cadenaValida()* en el archivo de funciones.

```
##Versión Python 3.7.2

## ------------------- Clases ---------------------

from class_Entidad import *
from class_CA import *

## ------------------- Main ---------------------

##defino las dos entidades que intercambiaran mensajes, A envia mensaje a
B
A = Entidad("A")
B = Entidad("B")

## B - Generacion de clave
B.generacionClave()

## B - Almacenamiento de clave en repositorio
R = CA()
R.añadir(B.publicar())

## A obtiene datos de B si no los tiene ya
if A.buscar("B")==False:
    A.añadirDatos(R.buscar("B"))

## A cifra su mensaje
## -- solo se condierara un alfabeto formado por los caracteres a-z y
caracter espacio
cadena = input("Ingrese la cadena que desea cifrar y enviar: ")
corte = False
while corte==False:
    if cadenaValida(cadena):
        corte = True
    else:
        print("Cadena no válida")
        cadena = input("Ingrese otra cadena: ")
textoCifrado = A.textoCifrado("B",cadena)
print("Texto Cifrado: ",textoCifrado)
textoDescifrado = B.descifrar(textoCifrado)
print("Texto Descifrado: ",textoDescifrado)
```

Script para firmar mensajes con RSA

En el siguiente código se muestra como es el proceso de firmar un mensaje utilizando RSA y verificar el mismo obteniendo los datos de una autoridad de certificación. Dejando en claro que el texto a firmar está limitado a los caracteres alfabéticos en minúsculas y el carácter espacio, para ampliar al conjunto de entrada se deberá modificar la variable *alfabetoPermitido* dentro de la función *cadenaValida()* en el archivo de funciones.

```
##Versión Python 3.7.2

## ------------------- Clases ---------------------
```

```
from class_Entidad import *
from class_CA import *

## ------------------- Main ---------------------

##defino las dos entidades que intercambiaran mensajes, B envia mensaje
firmado a A
A = Entidad("A")
B = Entidad("B")

## B - Generacion de clave
B.generacionClave()


## B - Almacenamiento de clave en repositorio
R = CA()
R.añadir(B.publicar())

## B firma su mensaje
## -- solo se condierara un alfabeto formado por los caracteres a-z y
caracter espacio
cadena = input("Ingrese la cadena que desea cifrar y enviar: ")
corte = False
while corte==False:
    if cadenaValida(cadena):
        corte = True
    else:
        print("Cadena no válida")
        cadena = input("Ingrese otra cadena: ")

s = B.rsa.firmarMensaje(cadena)

## A verifica mensaje
if A.rsa.verificarMensaje(cadena,s,R.buscar("B"))==True:
    print("Mensaje y firma corresponden")
else:
    print("Mensaje y firma no corresponden")
```

Script para intercambiar claves utilizando Diffie-Hellman

En el siguiente código se muestra como es el proceso para intercambiar claves utilizando el protocolo Diffie-Hellman. Recordando que el protocolo en si no intercambia claves, sino que genera una común.

```
##Versión Python 3.7.2

## ------------------- Clases ---------------------

from class_Entidad import *


## ------------------- Main ---------------------
```

```
##defino las dos entidades que intercambiaran mensajes, A envia mensaje a
B
A = Entidad("A")
B = Entidad("B")

## genero los valores p y g, las entidades guardan estos valores
dh = DH()
dh.generarPrimo()
dh.generador()
A.dh = dh
B.dh = dh

## Generan los valores x
A.dh.generarE()
B.dh.generarE()

## Generan los valores X
A.dh.generarX()
B.dh.generarX()


## Se intercambian los valores X y generan la clave comun
A.dh.generarClave(B.dh.X)
B.dh.generarClave(A.dh.X)
```