

PROYECTO FINAL DE INGENIERÍA

ESTUDIO COMPARATIVO DE ALGORITMOS DE CRIPTOGRAFÍA LIVIANA

Barrera, Elizabeth Gabriela del Valle – LU1034506
Ingeniería Informática

Tutor:
Wehbe, Ricardo Abraham, UADE

02 de noviembre de 2020



UNIVERSIDAD ARGENTINA DE LA EMPRESA
FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

Agradecimientos

Personalmente:

Gracias a todas aquellas personas que confiaron en mí y lograron que yo también vuelva a hacerlo. Gracias por la paciencia y la persistencia en ayudarme a no bajar los brazos. Gracias a todos y a cada uno de ustedes. Principalmente:

- ✓ Gracias a mi eterna compañera mamá: soy lo que soy, y estoy, gracias a vos.
- ✓ Gracias a mamá y a papá: sin ustedes yo no estaría acá.
- ✓ Gracias a mi hermano de la vida Fyer: siempre estuviste, estás y vas a estar, en las buenas y en las malas, siempre me apoyaste, a pesar de todo.
- ✓ Gracias a mi querido Fer: por la contención y calma, y por ayudarme a volver a creer.
- ✓ Gracias a mi amiga Nati: por estar siempre, y demostrarme que la diferencia de edad no es un problema para la amistad.

Académicamente:

- ✓ Quisiera agradecer principalmente a mi tutor, que a través de los seminarios de transferencia del grupo de investigación me ayudó a definir temas, me dio las bases de la Criptografía, respondió veloz y detalladamente ante mis consultas, y estuvo siempre atento ante todas las entregas.
- ✓ Gracias a todos los docentes que en mi vida me enseñaron todo lo que sé, desde el colegio, observatorio, hasta mi queridísima UADE.
- ✓ Gracias a mi constante revisor papá, que conté con su percepción de los detalles más mínimos en cada entrega.
- ✓ Gracias a mis lectores críticos, papá, mamá, Nati, Reba, Kenn y Fyer, que generaron comentarios y recomendaciones muy enriquecedores.
- ✓ Gracias a Fer por sus conocimientos técnicos de Python, entre otros.
- ✓ Gracias eternas a UADE, porque su impecable organización y estructura me permitieron volver al aula.
- ✓ Gracias a todos mis alumnitos que me ayudaron a aprender a comunicar.

Gracias a la vida, por las segundas oportunidades...

*“Working hard is important
but there is something that matters even more:
Believing in yourself”*

HP

Resumen

La Criptografía es tan antigua como la comunicación. En todas las épocas las personas buscaron comunicarse de forma privada sin que un tercero intruso pudiera descifrar el contenido. Con el transcurso del tiempo, el canal de comunicación fue evolucionando, y con él el medio de asegurar los datos.

La actualidad no es la excepción y el carácter abierto de internet complejizó la protección de estos, generando un gran problema en la privacidad; que se vio intensificado con el boom de los dispositivos pequeños inteligentes interconectados (como *IoT* o *WSN*), que toman datos del medio y los comparten por la red, generando así cada vez más y más datos (surgimiento del *Big Data*), los cuales deben asegurarse.

El principal inconveniente de estos dispositivos es el bajo poder de cómputo que poseen, lo que requiere de adaptaciones ligeras de los métodos criptográficos, y determinarse niveles de seguridad aceptables, que engloban lo denominado *Criptografía Liviana*.

El presente trabajo, denominado “*Estudio Comparativo de Algoritmos de Criptografía Liviana*”, se realizó según un método iterativo incremental. En este se profundizó en las bases matemáticas y los diferentes algoritmos conocidos, seleccionándose a la *Criptografía Asimétrica Liviana* como el alcance a estudiar. Posteriormente se realizaron implementaciones livianas, con artilugios matemáticos que mejoran la eficiencia de cómputo, para luego llevar a cabo comparaciones entre ellas. Los algoritmos vistos son: RSA Liviano, D-Rabin (que fue descartado por no considerarse método de encriptación), Diffie-Hellman con Curvas Elípticas y ElGamal con Curvas Elípticas.

Finalmente, se concluyó que, a igual grado de seguridad requerida, las soluciones con curvas elípticas resultan ser las más eficientes en poder de cómputo. Como buena práctica se recomienda evaluar soluciones ad-hoc priorizando siempre lo particular de cada caso, y no perder la visión de estimar la seguridad requerida, seleccionando y decidiendo en función de ello. Así como también, no olvidarse de crear conciencia en la necesidad del resguardo de los datos, y promover prácticas que lo prioricen.

Palabras Clave: *Criptografía; IoT; WSN; Criptografía Liviana; Estudio Comparativo; Eficiencia; Criptografía Asimétrica Liviana; RSA; Rabin; DHKE; ElGamal; ECC; ECDH; ECEG.*

Abstract

Cryptography is as old as communication. At all times, people sought to communicate privately without a third-party intruder being able to decipher the content. Over time, the communication channel evolved, and with it the means of securing data.

Today is no exception and the open nature of the internet made their protection more complex, generating a great privacy problem; that was intensified with the boom of small interconnected intelligent devices (such as *IoT* or *WSN*), which take data from the environment and share it over the network, thus generating more and more data (arising of *Big Data*), which must be secured.

The main drawback of these devices is their low computing power, which requires light adaptations of cryptographic methods, and to determine acceptable levels of security, which cover what is called *Lightweight Cryptography*.

The present work, named "*Comparative Study of Lightweight Cryptography Algorithms*", was carried out according to an incremental iterative method. In this, the mathematical bases and the different known algorithms were deepened, selecting the *Asymmetric Lightweight Cryptography* as the scope to study. Subsequently, lightweight implementations were made, with mathematical contraptions that improve computing efficiency, and then comparisons between them were formulated. The seen algorithms are: Light RSA, D-Rabin (which was discarded because it was not considered an encryption method), Elliptic Curve Diffie-Hellman and Elliptic Curve ElGamal.

Finally, it was concluded that, with the same level of security required, the solutions with elliptic curves turn out to be the most efficient in computing power. As a good practice, it is recommended to evaluate ad-hoc solutions, always prioritizing the particularities of each case, and the estimation of the required security level should be considered, so choices and decisions should be based on them. Also, it is encouraged to raise awareness of the need to safeguard data and promote practices that prioritize that.

Keywords: *Cryptography; IoT; WSN; Lightweight Cryptography; Comparative Study; Efficiency; Asymmetric Lightweight Cryptography; RSA; Rabin; DHKE; ElGamal; ECC; ECDH; ECEG.*

Contenido

1. Introducción	7
2. Antecedentes	9
2.1. Marco Teórico.....	9
2.1.1. Introducción	9
2.1.2. La Matemática en la Criptografía	11
2.1.3. Proceso de Encriptación	12
2.1.4. Herramientas Criptográficas	14
2.1.4.1. Criptografía Simétrica	14
2.1.4.2. Criptografía Asimétrica	16
2.1.4.3. Funciones Hash	19
2.1.5. Llevando la Teoría a la Práctica.....	19
2.2. Estado del Arte.....	20
2.2.1. Artículos Relacionados	21
2.2.2. Relativo al Presente PFI	22
2.3. User Research	23
3. Hipótesis.....	24
4. Metodología	25
4.1. Criptografía Simétrica.....	25
4.1.1. DES: <i>Data Encryption Standard</i>	25
4.1.2. AES: <i>Advanced Encryption Standard</i>	26
4.2. Criptografía Asimétrica	28
4.2.1. Algunos Conceptos Matemáticos Necesarios	29
4.2.2. RSA: <i>Rivest, Shamir, Adleman</i>	33
4.2.2.1. RSA Liviano	37
4.2.3. Rabin	42
4.2.3.1. Rabin Liviano: <i>Esquema Diferencial de Rabin (D-Rabin)</i>	45
4.2.4. DHKE: <i>Diffie-Hellman Key Exchange</i>	47
4.2.5. ElGamal.....	50
4.2.6. ECC: <i>Elliptic Curve Cryptography</i>	52
4.2.6.1. ECDH: <i>Elliptic Curve Diffie-Hellman</i>	60
4.2.6.2. ECEG: <i>Elliptic Curve ElGamal</i>	61
4.2.6.3. ECLC: <i>Elliptic Curve Lightweight Cryptography</i>	63
5. Resultados.....	64
6. Conclusiones	67
7. Bibliografía	73
Anexo A: Encuesta	75
1. Introducción	75
2. Información Personal	75
3. Seguridad en Dispositivos.....	78
3.1. Eligiendo la 1 ^{ra} opción: <i>Poseo dispositivos pequeños inteligentes</i>	79
3.2. Eligiendo la 2 ^{da} opción: <i>No poseo dispositivos pequeños inteligentes</i>	79
3.3. Eligiendo la 3 ^{ra} opción: <i>No sé lo que son los dispositivos pequeños inteligentes</i>	80
4. Agradecimiento Final.....	80

Anexo B: Códigos de Implementación..... 81

- 1. Algoritmos de Apoyo..... 81
 - 1.1. Algoritmo de Exponenciación Rápida 81
 - 1.2. Algoritmo Extendido de Euclides 81
 - 1.3. Algoritmo de la Adición (*para curvas elípticas*) 82
 - 1.4. Algoritmo del Producto Punto (*para curvas elípticas*) 82
- 2. Algoritmos Livianos 83
 - 2.1. RSA Liviano..... 83
 - 2.2. D-Rabin 84
 - 2.3. Diffie-Hellman con Curvas Elípticas 84
 - 2.4. ElGamal con Curvas Elípticas 85
- 3. Medición de Tiempos 86

Anexo C: Cronograma de Actividades..... 88

- 1. Propuesto..... 88
- 2. Actualizado al 06/06 88
 - 2.1. Comentarios 88
- 3. Actualizado al 22/08 89
 - 3.1. Comentarios 89
- 4. Actualizado al 10/10 90
 - 4.1. Comentarios 90
- 5. Actualizado al 02/11 90
 - 5.1. Comentarios 90

1. Introducción

Las sociedades han tenido desde tiempos remotos el problema de permitir transmitir información privada por un canal inseguro, tal que cualquier intruso que intercepte la comunicación no entienda su significado. A veces pareciera estar asociado a las comunicaciones electrónicas modernas, pero este problema es tan antiguo como la propia escritura, aunque claramente se ha ido adaptando a los distintos canales de comunicación que fueron surgiendo. Un claro ejemplo se remonta al antiguo Egipto, donde se utilizaban jeroglíficos secretos, no estándar, para evitar que ciertas personas husmearan donde no debían. La Criptografía, cuyo objeto es lograr permitir esta privacidad, ha sido utilizada de una forma u otra, en toda cultura que desarrolló lenguaje escrito. (Ortega Triguero, *et al.*, 2006) (Paar, *et al.*, 2010)

A la Criptografía se la ha vinculado con las fuerzas militares y los servicios secretos gubernamentales, dado que ha sido utilizada como herramienta de protección de secretos y estrategias nacionales. Esto no debería sorprender considerando que la Criptografía tuvo un conocido e importante rol en colaborar con el fin de la Segunda Guerra Mundial. Pero el crecimiento de las computadoras y de los sistemas de comunicaciones trajo consigo una demanda del sector privado como modo de protección de la información en medios digitales y de proveer servicios de seguridad. Al tornarse masivas las mismas, se comenzaron a necesitar algoritmos más robustos, como consecuencia de que, con la ayuda de estas, los cifrados comunes se volvieron cada vez más simples de descifrar. Asegurar la información de las comunicaciones se volvió una necesidad y numerosas investigaciones se desarrollaron en esta área. (Menezes, *et al.*, 1996) (Kapoor, *et al.*, 2008)

En la actualidad, millones de computadoras intercambian constantemente información por Internet, una red pública y en continuo crecimiento, al alcance de todos y desde cualquier lugar del mundo. El carácter abierto de Internet supone un problema para la comunicación privada y compromete la información que almacenan las computadoras, la cual es vital para muchas organizaciones (por ejemplo, los bancos). La demanda generalizada de protección de la información ha despertado el interés por la Criptografía, tanto de empresas (por el negocio que supone la seguridad informática), como de universidades (por el conocimiento científico y técnico que requiere la Criptografía actual). (Ortega Triguero, *et al.*, 2006)

En los últimos años, la moda de fabricar dispositivos electrónicos priorizó reducir su tamaño, reducir los costos de producción e incrementar su conectividad. Estos dispositivos

inteligentes, que son capaces de resolver pequeños cálculos y recolectar datos, se vuelven más universales día a día. Pero resulta que toda la información recolectada por estos objetos puede dar una idea del comportamiento de su usuario o su entorno, con lo cual se hace necesaria su protección. (Lara-Nino, *et al.*, 2018) Este tipo de dispositivos suelen tener un bajo poder de cómputo. Es por esto por lo que su cifrado no puede llevarse a cabo con los métodos más comunes y efectivos, lo cual generó el surgimiento de métodos ligeros que permiten un nivel de seguridad aceptable para este tipo de dispositivos. (Dutta, *et al.*, 2019) (Biryukov, *et al.*, 2017)

El presente trabajo buscó comparar los diferentes mecanismos de este tipo ligero de cifrado, denominado Criptografía Liviana, en eficiencia eléctrica y poder de cómputo. En primera instancia se realizó una revisión de los principios matemáticos y herramientas de la Criptografía, que sirvieron como materia prima para explicar los diferentes algoritmos ya conocidos. Posteriormente se seleccionó una de las principales categorías, la Criptografía Asimétrica Liviana, y se realizaron implementaciones de sus algoritmos, haciendo hincapié en priorizar eficiencia y aceptar un nivel de seguridad tolerable. En última instancia, se desarrolló una evaluación comparativa cuyas conclusiones permitieron un relevamiento de los algoritmos actuales. Estas conclusiones buscan proporcionar una idea más clara sobre los posibles métodos para asegurar dispositivos interconectados, lo que permite prevenir pérdida, robo y/o corrupción de datos.

Como resultado se obtuvo un estudio ordenado que ayuda a organizar algoritmos existentes para poder establecer el estado del arte actual de la Criptografía Asimétrica Liviana (que, con el nivel de profundidad pretendido, no existía al momento) y tener una medida comparativa del nivel de eficiencia de cada implementación, y así permitir a futuro evaluar, según el nivel de seguridad requerido, el mejor algoritmo a aplicar para cada caso particular.

En tanto a la estructura del informe, luego de esta *Introducción*, se referencian los *Antecedentes* previos al actual estudio. Allí podrá encontrarse el *Marco Teórico*, el *Estado del Arte* y el *User Research*. El primero muestra los conceptos básicos necesarios para poder comprender este trabajo; el segundo hace referencia a los estudios presentes al día de la fecha, similares a este, para evidenciar el aporte del presente trabajo; y el último presenta una encuesta con conclusiones sobre el cuidado que tienen las personas con sus datos que viajan por la red. A continuación, se enuncia la *Hipótesis* que se quiere validar, para luego entrar en la *Metodología*, que muestra el desarrollo propiamente dicho de este estudio. Allí se encontrarán

los algoritmos base de las herramientas criptográficas, juntamente con su fundamentación matemática, algoritmos necesarios de apoyo y consejos de implementación. También se hallan las implementaciones de algoritmos livianos relacionados con cada una de ellas, para compararlos. Se continúa con los *Resultados* del desarrollo del trabajo, con la validación de la *Hipótesis* antes planteada, que vendrán seguidos de las *Conclusiones* finales de todo el estudio. Finalmente, la Bibliografía citada. A posteriori se pueden apreciar tres anexos: el *Anexo A* muestra los datos crudos de la encuesta realizada para el *User Research*, el *Anexo B* tiene todos los códigos de implementación efectuados en lenguaje Python, y el *Anexo C*, el cronograma de las actividades propuesto y sus respectivas actualizaciones.

2. Antecedentes

Para poder comenzar con el desarrollo del estudio realizado propiamente dicho, primero deben explicitarse ciertos Antecedentes. Se empezará con el *Marco Teórico* que explicará tópicos básicos necesarios para comprender el desarrollo del estudio. Se continuará con el *Estado del Arte* que pondrá en contexto sobre lo estudiado previamente sobre el tema. Y finalmente se mostrará una encuesta dentro del *User Research* que permitió conocer el estado de conocimiento de los individuos sobre el cuidado de sus datos virtuales.

2.1. Marco Teórico

Se partirá de la definición de ciertos conceptos conocidos para comprender el desarrollo del presente *Proyecto Final de Ingeniería* (PFI). El objetivo es esclarecer y definir de una forma más rigurosa algunos conceptos que suelen conocerse vagamente, así como también introducir conceptos nuevos que se van a utilizar posteriormente en el PFI.

2.1.1. Introducción

Se comenzará por la definición de **Criptografía**: La criptografía es la ciencia de escritura secreta con el objetivo de esconder el significado de un mensaje, (Paar, *et al.*, 2010) para poder comunicarse en presencia de adversarios (Rivest, *et al.*, 1990), basándose en herramientas matemáticas para lograr sus objetivos (Menezes, *et al.*, 1996). *El problema es proteger la información digital en un entorno distribuido, globalmente accesible y sin fronteras materiales* (Wehbe, 2020).

La criptografía busca proteger la información, a través de los siguientes **objetivos**:

- 👑 *Confidencialidad*: protección de la información contra la divulgación no autorizada.
- 👑 *Integridad*: protección de la información contra la modificación no autorizada. Se debe tener la habilidad de detectar la manipulación de datos por entidades no autorizadas. Incluye inserción, borrado y sustitución.
- 👑 *Disponibilidad*: garantía de acceso para los usuarios legítimos.
- 👑 *Autenticación de entidades*: verificación de alguna de las identidades a través de una evidencia que la corrobore.
- 👑 *Autenticación de datos de origen*: verificación de que una segunda entidad es la fuente original de un conjunto de datos. Implica integridad implícitamente (si un mensaje fue modificado, la fuente ha cambiado).
- 👑 *No repudio*: procedimiento que permite que una entidad no pueda negar haber tomado parte en una transacción. Cuando surge una disputa sobre una entidad que niega haber tomado acciones, es necesario un medio de resolución.
- 👑 *No duplicación*: protección contra copias ilícitas de la información.
- 👑 *Anonimato*: resguardo de la identidad de una entidad, de la fuente de una información o del originante de una transacción.

(Wehbe, 2020) (Menezes, *et al.*, 1996)

El marco de trabajo de la criptografía se define mediante el encauce de los objetivos fundamentales de la misma, tanto de forma teórica como de forma práctica: la confidencialidad, la integridad de los datos, la autenticación (tanto de las entidades, como de los datos) y el no repudio. El resto de los objetivos pueden derivarse de los anteriores. (Menezes, *et al.*, 1996)

La criptografía posee diferentes herramientas para alcanzar sus objetivos. Para determinar cuál sería recomendable utilizar para cada caso, se evalúan los siguientes **criterios**:

- 👑 *Nivel de seguridad*: usualmente es difícil de cuantificar, y suele darse en términos del número de operaciones requeridas, usando el mejor método actualmente conocido para lograr el objetivo (típicamente se define un límite superior). A veces se lo denomina factor de trabajo.
- 👑 *Funcionalidad*: se necesita combinar herramientas para lograr ciertos objetivos de seguridad. Cuál es la más efectiva para cierto objetivo se determina a través de las propiedades de la herramienta.

- 👑 *Método de operación*: las herramientas aplicadas a varios modos con diferentes inputs mostrarán diferentes características (la funcionalidad de la herramienta puede variar dependiendo su uso).
- 👑 *Performance*: se refiere a la eficiencia de la herramienta en cierto modo de operación.
- 👑 *Facilidad de implementar*: hace referencia a la dificultad de llevar la herramienta a la práctica (a nivel software o hardware).

(Menezes, *et al.*, 1996)

La criptografía está compuesta básicamente por dos etapas: **encriptar** y **desencriptar**. Se define encriptar como toda transformación a la que se vea sujeto un mensaje de tal manera de convertirlo en algún otro irreconocible para las partes no interesadas, denominado generalmente “*criptograma*”, texto cifrado o texto encriptado (llamamos “*texto plano*” al mensaje original). Toda encriptación se asocia a otra transformación inversa denominada desencriptar o descifrar, que permite recuperar el mensaje original a partir del criptograma. (Liberatori, 2006)

2.1.2. La Matemática en la Criptografía

Una *biyección* es una relación entre dos conjuntos X e Y en la cual a cada $x \in X$ asigna un único elemento $y = f(x)$ del conjunto Y (a lo que llamamos *función*), en la cual si existen $x_1, x_2 \in X$ con $x_1 \neq x_2$, $f(x_1) \neq f(x_2)$ (denominado *función inyectiva*) y además, para cada $y \in Y$ existe siempre un $x \in X$ tal que $f(x) = y$. Sea g una función que asigna a cada $y \in Y$ un único elemento $x = g(y)$ del conjunto X , se la denomina *función inversa* de f si satisface que $f(g(x)) = x$ (f siempre debe ser inyectiva, pero si además es una biyección, su inversa también lo será). (Aguilar Márquez, *et al.*, 2016)

En criptografía, las biyecciones se utilizan como herramienta para encriptar mensajes, y a su inversa para desencriptarlos (se debe considerar que, si la función no es una biyección, no va a ser posible siempre desencriptar a un único mensaje).

Una función f de un conjunto X a un conjunto Y se la denomina **función unidireccional** (o de un sentido, “*one-way*”) si $f(x)$ es “*simple*” de computar para todo $x \in X$, pero para todo elemento $y \in Y$ es computacionalmente “*impossible*” encontrar algún $x \in X$ tal que $f(x) = y$ (se denomina “*impossible*” a extremadamente costoso e irrealizable en un tiempo prudencial).

Es importante tener en cuenta la diferencia entre el poco trabajo de computar $f(x)$, para algún x dado, y la cantidad de trabajo en encontrar un x para un $f(x)$ dado (que se presume con costo “muy grande”). Se suponen costos “grandes” a aquellos no polinomiales, llegando a la idea de que no existe un algoritmo eficiente para calcular la función inversa (y de este modo “no se pueda” descifrar un mensaje cifrado). (Menezes, *et al.*, 1996)

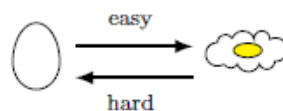


Figura 1: funciones unidireccionales (Wehbe, 2020).

Una **función unidireccional con puerta trampa** (“*trapdoor one-way*”) es una función unidireccional del conjunto X al conjunto Y con la propiedad adicional de que, dada cierta información extra, se vuelve realizable encontrar para algún $y \in Y$, un $x \in X$ tal que $f(x) = y$. (Menezes, *et al.*, 1996)

Todavía no está clara la existencia de alguna función unidireccional real, por el motivo que no hay pruebas de la existencia de tales funciones con una definición rigurosa de “simple” o “imposible” computar. Como su existencia es desconocida, la existencia de las funciones unidireccionales con puerta trampa es también desconocida, pero hay un buen número de candidatos a fines prácticos. (Menezes, *et al.*, 1996)

2.1.3. Proceso de Encriptación

Una **clave** e perteneciente al conjunto K , unívocamente determina una biyección desde el conjunto M , al que pertenece el texto plano m , al conjunto C al cual pertenece el criptograma c . Esta biyección E_e se denomina **función de encriptación** (o transformada de encriptación) y es necesario que sea una biyección, de otro modo el proceso inverso no sería unívoco. La **función de descifrado** D_d (o transformada de descifrado) es aquella biyección inversa, determinada por la clave d (también perteneciente al conjunto K). Aplicar la transformada E_e a un mensaje m del conjunto M se suele denominar **encriptar** (así mismo aplicar la transformada D_d al criptograma c , se lo denomina **descifrar**). Un **esquema de encriptación** consiste en definir una transformada de encriptación E_e (con clave e) y una transformada de descifrado D_d (con clave d), con la propiedad de que para cada $e \in K$ hay una única clave $d \in K$ tal que

$$D_d = E^{-1}_e \quad (1)$$

o sea

$$D_d(E_e(m)) = m \quad (2)$$

para todo $m \in M$ (recupero de mensaje). (Menezes, *et al.*, 1996)

Las claves son necesarias dado que es útil tener transformadas que sean muy similares pero que estén caracterizadas por una clave, de este modo si alguna transformada de encriptación o de desencriptación es revelada, no se debe rediseñar el esquema completo de encriptación, sino simplemente cambiar la clave (lo cual es un proceso simple, que se recomienda realizarlo frecuentemente). (Menezes, *et al.*, 1996)

Para poder explicar el proceso de encriptación es necesario primero definir ciertos conceptos. Un **canal inseguro** es aquel en el cual entidades diferentes de aquellas a las cuales la información está intencionada pueden reordenarla, borrarla, insertarla, o leerla (a diferencia de un **canal seguro**, en el cual no se tienen dichas habilidades). Un **adversario** (también conocido como atacante u oponente) es una entidad diferente del emisor y el receptor del mensaje, que trata de violar la seguridad provista entre dicho emisor y receptor, y usualmente trata de jugar el papel del legítimo emisor o del legítimo receptor. Los adversarios pueden ser pasivos (solo monitorea el canal de comunicación, viéndose vulnerada únicamente la confidencialidad de los datos) o activos (puede borrar, agregar, o alterar en cualquier forma la transmisión del canal, o sea se vulnera la integridad y la autenticación, además de la confidencialidad). (Menezes, *et al.*, 1996)

El siguiente gráfico muestra un esquema de encriptación entre el emisor Alice y el receptor Bob, que buscan intercambiar un mensaje por un canal inseguro (Menezes, *et al.*, 1996). Alice encripta el texto plano m que desea enviarle a Bob a través de un canal inseguro, en el cual existe la presencia de un adversario. Alice encripta el mensaje con la transformada E_e determinada por la clave e , obteniendo el criptograma c que lo emite por el canal inseguro. Bob recibe el criptograma c y lo desencripta con la transformada D_d determinada por la clave d , recuperando el mensaje de texto plano m .

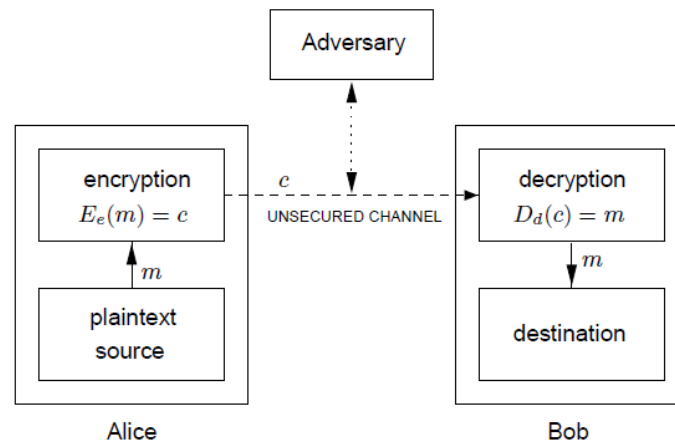


Figura 2: comunicación entre dos entidades con encriptación (Menezes, et al., 1996).

Una premisa fundamental, el *Principio de Kerckhoffs* (Kerckhoffs, 1883), dice que en la criptografía el conjunto M, C, K y las transformadas E_e, D_d son de público conocimiento. Al quererse comunicar dos entidades utilizando un esquema de encriptación, lo único que se mantiene en secreto es el par de claves (e, d) que se están utilizando, y que deben ser seleccionadas. Se puede presuponer una ganancia extra en seguridad mantener en secreto las transformadas E_e y D_d , pero no se puede basar la seguridad de todo el esquema sólo en ello, mejor conocido como *Security by Obscurity* (mantener a las transformadas en secreto es una tarea que la historia ha demostrado ser muy difícil). (Menezes, et al., 1996) (Kerckhoffs, 1883)

2.1.4. Herramientas Criptográficas

La criptografía se puede clasificar en tres grandes ramas: la *Criptografía Simétrica*, la *Criptografía Asimétrica* y las *Funciones Hash*. Cada una posee diferentes características y distintos objetivos. Es por esto por lo que, si se requiere compararlas, es necesario realizarlo independientemente una de otra. Se estudiarán a continuación cada una de ellas.

2.1.4.1. Criptografía Simétrica

Considerando un esquema de encriptación con las transformadas de encriptación y de desencriptación E_e y D_d respectivamente, se dice que el esquema es simétrico si para cada par de claves de encriptación y de desencriptación asociado (e, d) , es computacionalmente “simple” determinar la clave d conociendo sólo la clave e (y viceversa). En gran parte de los esquemas criptográficos, la clave e suele ser igual a la clave d (de allí el nombre “simétrico”).

Uno de los principales problemas que atraviesa la criptografía simétrica es acordar un método seguro de intercambio de claves. (Menezes, *et al.*, 1996)

Una simple analogía de la criptografía simétrica podría verse en una caja fuerte con una cerradura muy segura, que solo Alice y Bob tienen una copia de la llave (*claves iguales*). Alice pone el mensaje en la caja fuerte (*encriptación del texto plano*), cerrando la misma con su copia de la llave (*clave*). Para leer el mensaje, Bob utiliza su copia de la llave (*la misma clave*) y abre la caja fuerte para recuperarlo (*desencriptación del texto cifrado*). (Paar, *et al.*, 2010)



Figura 3: analogía de la encriptación simétrica, una caja fuerte con una cerradura (Paar, *et al.*, 2010).

El siguiente diagrama muestra al emisor que encripta el texto plano m con la transformada E_e generada mediante la clave e , obteniendo el criptograma c (la clave e debe ser compartida con el receptor a través de un canal seguro). El criptograma c se transmite por el canal inseguro y el receptor luego realiza la desencriptación mediante la transformada D_d generada con la clave d que se calcula eficientemente sólo conociendo la clave e compartida por el emisor (recuperando así el mensaje m).

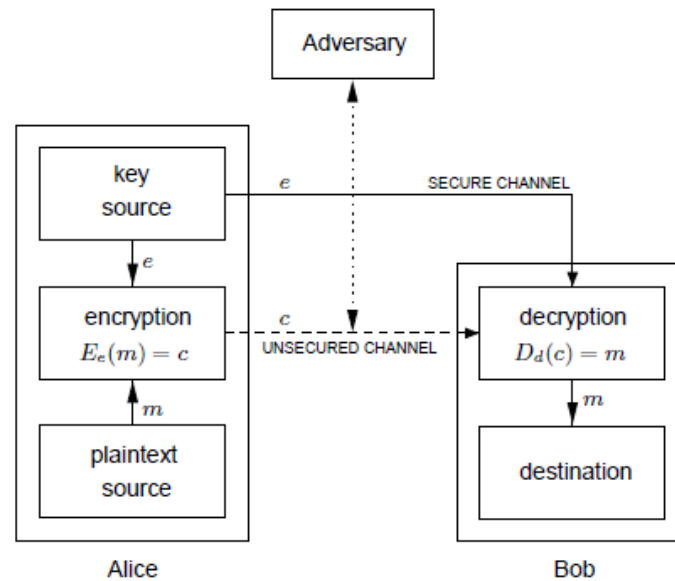


Figura 4: comunicación entre dos entidades utilizando encriptación, con un intercambio de claves por un canal seguro (Menezes, et al., 1996).

Los dos esquemas de criptografía simétrica conocidos son el cifrado por bloques (“*block cipher*”) en el cual el texto plano se divide en cadenas de una longitud prefijada denominadas bloques, y se encripta un bloque a la vez; y el cifrado de flujo (“*stream cipher*”) en el cual se encripta símbolo a símbolo. (Menezes, et al., 1996)

2.1.4.2. Criptografía Asimétrica

Sea E_e y D_d las transformadas de encriptación y de desencriptación respectivamente, en la criptografía asimétrica se supone que conociendo E_e es “*imposible*” determinar el mensaje m tal que $E_e(m) = c$, para algún criptograma c dado (lo cual implica que dada la clave de encriptación e no se puede determinar la clave de desencriptación d). En este caso E_e es vista como una función unidireccional con puerta trampa, en la cual la clave d representa la información adicional necesaria para poder computar eficientemente la función inversa, y así poder desencriptar el criptograma. (Menezes, et al., 1996)

En este esquema la clave e es pública, con lo cual cualquier emisor puede encriptar mensajes hacia el receptor, pero sólo él puede desencriptarlos (con su clave privada d). (Menezes, et al., 1996)

Una analogía interesante podría ser los antiguos buzones de correo, en los cuales cualquier persona puede poner una carta en él, pero solo el cartero, que posee la llave, puede recuperar las cartas. Yendo a una analogía similar a la planteada para criptografía simétrica,

imaginemos una caja fuerte con doble cerradura; la primera es una cerradura para depositar mensajes (*encriptar el texto plano*), la llave de esta (*clave pública*) la poseen todos aquellos que vayan a dejar mensajes allí, pero la llave de la segunda cerradura (*clave privada*) la posee solo la persona que vaya a recuperar los mensajes (*desencriptar el texto cifrado*). (Paar, et al., 2010)



Figura 5: analogía de la encriptación asimétrica, una caja fuerte con una cerradura pública para depositar mensajes y una cerradura privada para recuperarlos (Paar, et al., 2010).

En el siguiente diagrama se puede ver al emisor encriptar el texto plano m con la transformada E_e generada, utilizando la clave pública e del receptor (que se comparte por un canal inseguro). Se envía el criptograma c por un canal inseguro (que puede ser el mismo por el cual se compartió la clave pública) al receptor que lo desencripta con la transformada D_d generada con su clave privada d , recuperando así el mensaje m .

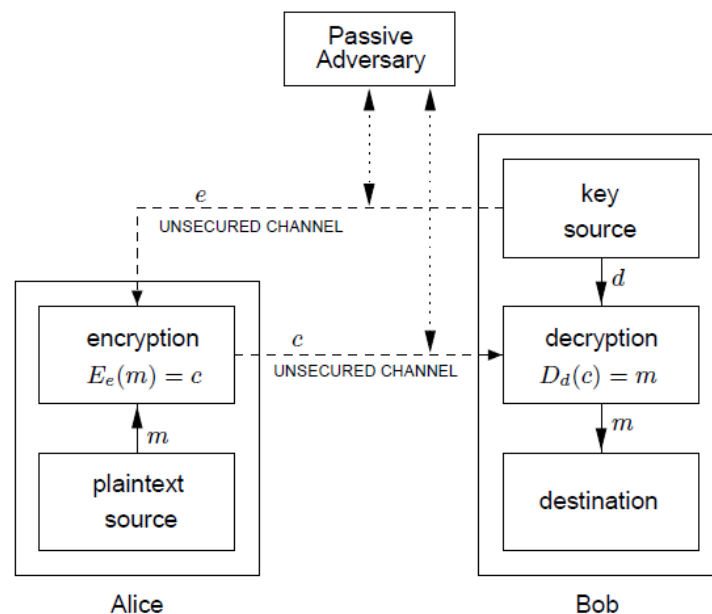


Figura 6: encriptación utilizando técnicas de criptografía asimétrica (Menezes, et al., 1996).

Pareciera que la criptografía asimétrica es un sistema ideal, dado que no requiere un canal seguro para pasar la clave de encriptación (lo cual implica que dos entidades pueden

comunicarse en forma segura, a través de un canal inseguro, sin nunca haberse encontrado para intercambiar claves). Pero no es el caso. Como se ve en la figura a continuación, un adversario activo puede frustrar el sistema (desencriptar mensajes intencionados para otra entidad) sin romper el sistema criptográfico. Este tipo de ataque es un tipo de *Impersonificación*. Aquí el adversario se hace pasar por el receptor Bob, enviando una clave pública e' que la emisora Alice asume incorrectamente que es la clave pública de Bob. El adversario luego intercepta el mensaje encriptado de Alice a Bob, lo desencripta con su propia clave privada d' , lo re-encripta con la real clave pública de Bob e , y se lo envía a Bob. Es por esto por lo que se debe hacer énfasis en la importancia de autenticar las claves públicas, para lograr que los datos de origen sean auténticos (Alice debe estar convencida de que está encriptando con la legítima clave pública de Bob). Es precisamente por esto por lo que en el diagrama anterior se suponía un adversario pasivo. (Menezes, *et al.*, 1996)

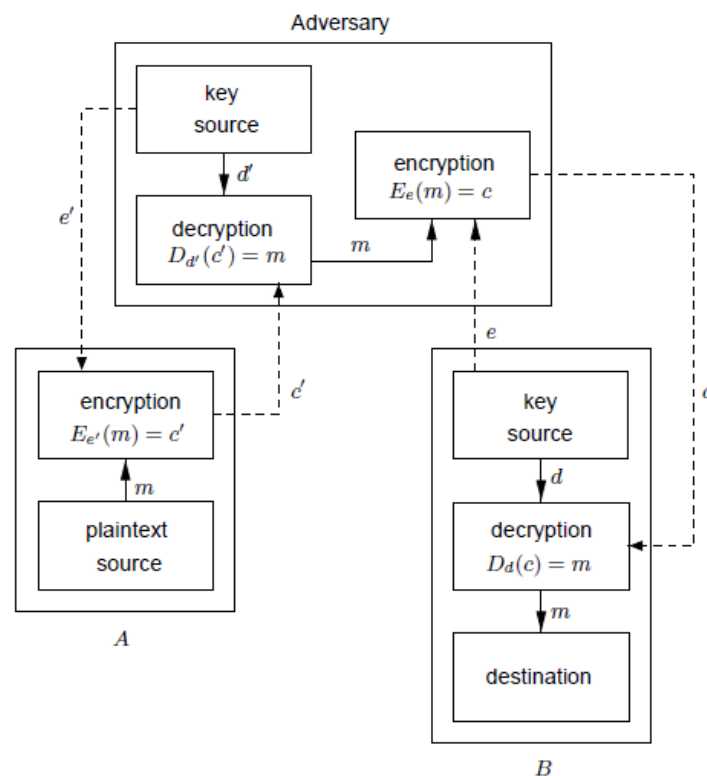


Figura 7: un ataque de Impersonificación en una comunicación entre dos interlocutores (Menezes, *et al.*, 1996).

Una sintética comparación entre la criptografía simétrica y la criptografía asimétrica se puede ver en la siguiente tabla:

Tabla I: criptografía simétrica vs criptografía asimétrica (Wehbe, 2020).

Propiedad	Cripto. simétrica	Cripto. asimétrica
Rapidez	rápido ●●	lento ●●
Implementación en hardware	fácil ●	difícil ●
Longitud de claves	128 bits ●●	1024 bits ●●
Intercambio de claves	difícil ●●	fácil ●●
Gestión de claves	difícil ●	no tan difícil ●
Fundamentos	empíricos ●	empíricos ●
Restricciones	sí ●	sí ●●

2.1.4.3. Funciones Hash

Las funciones hash son funciones computacionalmente eficientes para mapear cadenas binarias de largo arbitrario a nuevas cadenas de longitud fija (denominadas “*digesto*”), las cuales sirven de representación de las cadenas ingresadas. Estas funciones deben garantizar que no se pueden encontrar dos inputs diferentes que generan el mismo valor digesto, y que, dado un digesto específico, no se puede encontrar el input del que proviene. (Menezes, *et al.*, 1996)

Las funciones hash suelen utilizarse para garantizar integridad y autenticación, a través de la comparación de digestos históricos.

2.1.5. Llevando la Teoría a la Práctica

Un **protocolo criptográfico** es un algoritmo definido por una secuencia de pasos que especifica precisamente las acciones requeridas entre dos o más entidades para alcanzar un objetivo específico de seguridad. En los protocolos se suelen complementar herramientas criptográficas diferentes, como los algoritmos de criptografía simétrica, los de criptografía asimétrica y las funciones hash. En el protocolo de acuerdo de clave más usado, la criptografía asimétrica se utiliza como medio de intercambio de claves, para posteriormente utilizar la criptografía simétrica para encriptar los mensajes, propiamente dicho (haciendo uso de los beneficios de ambas herramientas). (Menezes, *et al.*, 1996)

Existen dispositivos inteligentes pequeños interconectados cuya desventaja, en cuanto a la seguridad, viene de la mano de su poder de cómputo. Al ser generalmente de dimensiones reducidas, su procesador suele ser de baja capacidad, lo cual genera que los métodos de cifrado más comunes y de alta efectividad no sean aplicables (por falta de espacio, o por baja velocidad

y, en definitiva, elevados tiempos). Es por esto por lo que surgen diversos métodos ligeros que engloban lo denominado **Criptografía Liviana** (“*Lightweight Cryptography*”), que permiten un nivel de seguridad aceptable para este tipo de dispositivos (Dutta, *et al.*, 2019) (Biryukov, *et al.*, 2017). Entre este tipo de dispositivos se encuentran los mayormente conocidos como IoT (*Internet of Things*, Internet de las cosas) que son dispositivos interconectados que toman datos del medio y los transmiten, para permitir su manejo remoto y para mejorar el funcionamiento de estos a través de algún análisis (Li, *et al.*, 2015). También se puede nombrar a las WSN (*Wireless Sensor Networks*, Redes de sensores inalámbricos), que son mallas de nodos que actúan de sensores y se encuentran interconectados de forma inalámbrica, y se utilizan principalmente para monitorear el ambiente, por lo que suelen estar desplegadas en zonas abiertas y, obviamente, desprotegidas (Chen, *et al.*, 2008). Todos estos dispositivos están conectados a alguna red, con lo cual la seguridad de estos es de vital importancia, ya sea por motivos de índole personal (privacidad) como también de índole legal (datos sensibles, por ejemplo). En ambos casos, la protección de los dispositivos, de sus datos y del tránsito de estos es imprescindible para permitir que continúe su crecimiento.

Mediante la Criptografía Liviana se busca evitar posibles ataques a estos dispositivos conectados. Los dispositivos pequeños suelen presentar fallas de seguridad muchas veces conocidas, poseen *passwords* por defecto que suelen mantenerse, entre otras negligencias por parte de los usuarios. La toma de control de estos dispositivos por un adversario significa la entrada a la red doméstica o corporativa, que puede implicar una eventual utilización de dichos dispositivos para actividades ilícitas. (Wehbe, 2020)

2.2. Estado del Arte

Existen limitados compendios sobre implementaciones de algoritmos de criptografía liviana, y son todavía más escasos los que poseen un amplio grado de detalle. La criptografía liviana se torna de suma necesidad con el boom masivo de los dispositivos pequeños interconectados, surgiendo incluso nuevos tipos de artefactos inteligentes. A nivel académico se pueden observar los siguientes artículos comparativos entre diferentes algoritmos. Cabe aclarar que las herramientas criptográficas se clasifican en criptografía simétrica, criptografía asimétrica y funciones hash, con los cual la criptografía liviana necesita separar las comparaciones de estas tres herramientas, de manera independiente (comparaciones entre algoritmos de diferentes clasificaciones no tendrían sentido, dado que sus objetivos son

diferentes). Es por esto por lo que se puede apreciar en los diferentes artículos que se hace principal énfasis en un tipo de herramienta o en otro.

2.2.1. Artículos Relacionados

(Bhardwaj, *et al.*, 2017) realiza un resumen muy conciso, recorriendo los principales algoritmos de criptografía simétrica liviana y de criptografía asimétrica liviana para dispositivos IoT (*Internet of Things*, Internet de las Cosas). Las herramientas criptográficas son comentadas con un mínimo detalle dado que su principal objetivo es explicar la arquitectura y vulnerabilidades de los dispositivos IoT. Se realiza una comparación entre la eficiencia de distintas implementaciones de algoritmos simétricos para encriptar, y otra entre los principales algoritmos asimétricos para autenticar.

Por su parte (Biryukov, *et al.*, 2017) muestra un exhaustivo compendio del estado del arte de la criptografía liviana simétrica. La cantidad de algoritmos evaluados es colosal, dentro de los cuales existen algoritmos propietarios, públicos (clasificados en diferentes categorías, dentro de las cuales encontramos algunas funciones hash) y propios de agencias gubernamentales. Los autores presentan estándares y algoritmos recomendados, así como también modas de diseño, y un interesante recorrido por las restricciones que debe mitigar la criptografía liviana (tanto en hardware como en software). Cabe destacar que las comparaciones realizadas entre los diferentes algoritmos (modo de operación, concesiones mutuas, entre otras) tienen un nivel de detalle impecable. Los tecnicismos de las bases matemáticas y la teoría esencial necesaria están sintetizados, dado que no es el objetivo del documento.

(Hammad, *et al.*, 2017) muestra un completo e interesante compendio de las funciones hash para criptografía liviana. La explicación inicial introduce claramente el tema de una manera simple, e incluye las construcciones para criptografía liviana de funciones hash. El presente artículo detalla claramente un gran número de funciones hash livianas, explicando con precisión cada una de ellas, y mostrando algunas aplicaciones. Concluye con un análisis comparativo entre dichas funciones de una forma muy completa.

(Eisenbarth, *et al.*, 2007) efectúa un resumen sobre implementaciones de criptografía liviana. Realiza comparaciones entre los principales algoritmos simétricos. Además, propone un diseño de procesador basado en curvas elípticas (algoritmo de criptografía asimétrica de punta, también conocido como ECC), tanto en hardware como en software, haciendo una comparación de performance contra otros diseños.

Paralelamente (Mendez, *et al.*, 2018) hace un estudio similar a (Bhardwaj, *et al.*, 2017), mostrando la arquitectura IoT y sus vulnerabilidades, labrando un amplio análisis de los principales objetivos de la criptografía a nivel aplicado. Se comentan algunos protocolos muy vagamente. Lo interesante que muestra este artículo es la comparación que realiza entre las citas sobre seguridad en dispositivos IoT en diferentes publicaciones.

(Kapoor, *et al.*, 2008) por su parte, resume los principios de la criptografía simétrica y asimétrica, mostrando aplicaciones en tarjetas inteligentes. Selecciona la criptografía asimétrica y explica claramente RSA (el algoritmo asimétrico más utilizado en la actualidad), señalando sus inconvenientes para la aplicación en criptografía liviana. Así introduce ECC y sus parámetros electivos. Haciendo énfasis en las ventajas sobre RSA, realiza un paralelismo entre los dos algoritmos, basado en seguridad, espacio requerido y eficiencia. En los anexos se amplía con más detalle sobre la matemática referida a ECC.

2.2.2. Relativo al Presente PFI

Tanto (Bhardwaj, *et al.*, 2017) como (Mendez, *et al.*, 2018) hacen un estudio general de las herramientas criptográficas livianas, tanto simétricas como asimétricas, de una forma mucho más global que la pretendida por este PFI, y la cantidad de algoritmos comparada o explicada es en general baja. En tanto a (Biryukov, *et al.*, 2017) y (Hammad, *et al.*, 2017), realizan un análisis un tanto más específico, detallando y ampliando una herramienta en particular (criptografía simétrica liviana el primero, y funciones hash livianas el segundo), como es el caso del presente PFI. Para el caso de la criptografía asimétrica, (Eisenbarth, *et al.*, 2007) hace hincapié en dicha herramienta, pero no desde un punto comparativo de algoritmos, sino proponiendo un diseño y comparándolo con otros. En tanto (Kapoor, *et al.*, 2008) se enfoca en ECC principalmente y realiza una comparación contra RSA desde un punto de vista más general. El mismo artículo hace principal hincapié en lo mucho que se ha escrito sobre ECC, pero lo poco referido a su uso en criptografía, considerándolo una desventaja sobre RSA. Cabe aclarar que estos dos últimos artículos están, además, relativamente desactualizados.

El presente PFI busca hacer un estudio comparativo de la criptografía asimétrica. Como queda de manifiesto, no existen antecedentes de un estudio riguroso entre algoritmos en esta herramienta en particular (el único que se asemeja es (Kapoor, *et al.*, 2008), pero no profundiza demasiado). Lo expuesto previamente evidencia el aporte y el valor agregado de este PFI. Se pretende además ser riguroso en los fundamentos teóricos y en los conceptos matemáticos

requeridos, ninguno expuesto en los anteriores artículos. La profundidad que exhibe (Biryukov, *et al.*, 2017) para criptografía simétrica, es la aspirada en este PFI para asimétrica.

2.3. User Research

Se realizó una encuesta entre 325 personas para determinar el grado de conocimiento y de compromiso de la gente, respecto a la seguridad de sus dispositivos. Entre los mismos se distinguieron personas de diferente sexo y edad, así como también de diversos estratos sociales y educativos. El 69% de los encuestados manifestó conocer sobre la importancia de cuidar la información de su computadora, pero el porcentaje se reduce a 54% de los que efectivamente aplican medios para realizar este cuidado. En tanto a los dispositivos pequeños inteligentes interconectados, el 70% afirmó poseer alguno, y sólo el 10% aseveró no conocer lo que eran. En el caso de las personas que no lo poseen, resultó interesante que solo el 6% contestó que no lo hacen porque son inseguros, contra la mayoría que no lo hacen porque no les interesa. Lo sorprendente fue que de las personas que afirmaron poseerlos, el 80% aseguró conocer la importancia de tomar medidas para asegurar la información que transita.

En tanto a la educación de los encuestados, el 71% de las personas con educación superior completa aseguró conocer sobre la ciberseguridad, contra un 67% de los que poseían estudios secundarios. El rango etario que manifestó más conocimiento sobre la ciberseguridad es el de entre 31 a 45 años (con un 77% asegurando conocerla), aunque el menor número lo muestran las personas de más de 60 años, sigue siendo un valor elevado (62%). Hasta los 60 años, cerca del 69% de los encuestados afirmaron poseer dispositivos inteligentes pequeños, luego el número desciende hasta 57%. Los mayores de 60 años muestran el mayor número de desconocimiento de estos dispositivos, con un 19%. El 70% de los hombres aseguró poseer este tipo de dispositivos (contra el 65% de las mujeres), mientras que de las personas que viven solas, el número asciende a 78% (contra un 66% de las personas que viven acompañadas). Resulta interesante que, de las personas que saben que deben proteger la seguridad de su computadora, solo el 73% lo hace efectivamente.

Cabe aclarar que los menores de 18 años fueron desestimados por el bajo número de respuestas de este rango etario.

Se esperaba el bajo número de personas que aseguraron no comprar los dispositivos pequeños inteligentes por ser estos inseguros, lo cual deja de manifiesto que las personas no están al tanto de lo expuestos que resultan con este tipo de dispositivos. El bajo número de

personas que desconocían lo que estos dispositivos eran, no se lo esperaba. Es claramente importante que los dispositivos pequeños inteligentes se aseguren mediante herramientas criptográficas correctas y específicas para su uso particular, dado que extrapolando la información proveniente de la ciberseguridad (que resulta más conocida masivamente), son muchas las personas que, sabiendo sobre la necesidad de proteger los datos de su computadora, no lo hacen. Por el lado de los dispositivos pequeños se espera el mismo comportamiento, o incluso peor, dado que los mismos no son tan conocidos ni masivos. Fue un poco inesperado el alto número de personas que poseían dispositivos inteligentes y estaban al tanto del cuidado que debían tener con los datos de estos (es probable que se hayan referido a cuidados más físicos que criptográficos).

Para los datos crudos, ver Anexo A.

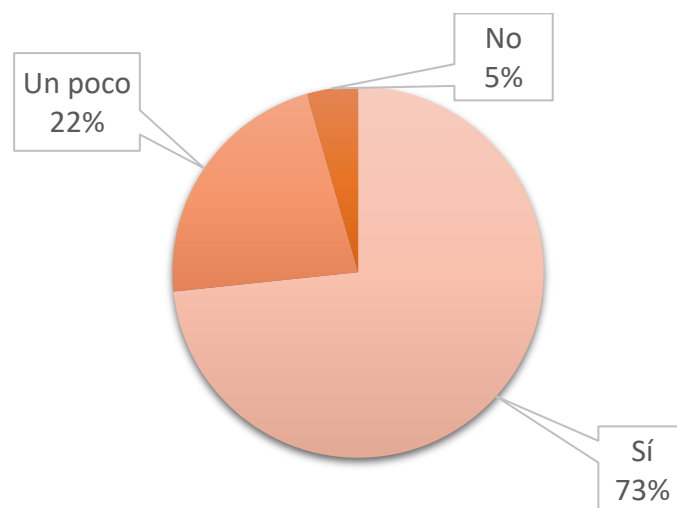


Figura 8: personas que, conociendo sobre la importancia de cuidar sus datos digitales, efectivamente lo hacen.

3. Hipótesis

En el presente documento se busca comparar diferentes algoritmos de criptografía asimétrica en implementaciones realizadas, priorizando eficiencia. Esto se debe a que se las busca utilizables para entornos restringidos. De los algoritmos a estudiar, se percibía mayor eficiencia en uno de ellos. Es por ello por lo que la hipótesis planteada es la siguiente:

“Las implementaciones de algoritmos de sistemas criptográficos con curvas elípticas son más eficientes”.

Aclaración: se verá en detenimiento a continuación cuáles son los distintos algoritmos de criptografía, y se explicará qué son las curvas elípticas.

4. Metodología

Para poder estudiar los métodos modernos de criptografía liviana, es necesario comenzar por las bases que plantean los métodos iniciales utilizados (y que se siguen utilizando) como medio de protección de datos con altos niveles de seguridad. En primer término, se empezará con los métodos de criptografía simétrica, que poseen igual clave de encriptación y de desencriptación, para luego continuar con la criptografía asimétrica, en la cual las claves resultan diferentes.

4.1. Criptografía Simétrica

En este tipo de cifrado, como se vio previamente, el emisor utiliza la misma clave para encriptar que el receptor para desencriptar (o esta última es de fácil deducción desde la primera).

4.1.1. DES: *Data Encryption Standard*

El primer algoritmo de criptografía simétrica más difundido fue el Estándar de Encriptación de Datos (DES, *Data Encryption Standard*). El mismo fue por mucho tiempo el algoritmo de cifrado por bloques más utilizado, pero a pesar de que ya no sea considerado seguro contra ciertos atacantes, por resultar ser el algoritmo simétrico más estudiado, sus principios han inspirado varios cifrados actuales, con lo cual sirve de base para entender algunos algoritmos modernos. Al volverse públicamente disponible en los años 80, resultó más simple de analizar su estructura interna de cifrado. Cabe aclarar que, en esta época, la comunidad de investigadores civiles de criptografía creció y DES sufrió un gran escrutinio. De todas formas, recién en 1990 se encontraron serias debilidades. DES fue estándar hasta 1999. (Paar, *et al.*, 2010)

El algoritmo DES se basa en la topología de *Feistel*, cuya idea es una red que utiliza un cifrado simple reiteradas veces, de una forma iterativa, para producir un sistema de cifrado más fuerte. Se realizan varias vueltas, con diferentes sub-claves que derivan de la clave principal, separando cada bloque en su parte izquierda y su parte derecha, trabajándolos independientemente, y retroalimentando el cifrado con fragmentos del bloque anterior. La idea es utilizar una combinación de cifrados simples, repetidas veces, y lograr un resultado mucho más fuerte que si se utilizara cada método por sí solo. Para el caso de DES, la red *Feistel* toma

bloques de texto plano de 64 bits como input para producir bloques de texto cifrado de 64 bits, luego de 16 vueltas, con un tamaño efectivo de clave de 56 bits. (Wehbe, 2020)

DES se basa en las siguientes operaciones básicas de cifrado:

👑 *sustitución*: reemplazo de símbolos, o grupos de símbolos, por otros símbolos, o grupos de símbolos, y

👑 *transposición*: permutación de símbolos.

Individualmente la simple sustitución o permutación no provee un alto nivel de seguridad, pero combinándolos es posible obtener un cifrado fuerte, cuya idea ya había sido propuesta por Claude Shannon, padre de la *Teoría de la Información* (Shannon, 1949). La sustitución se dice que agrega **Confusión** al proceso de encriptación, mientras que la transposición agrega **Difusión**. La primera pretende hacer una relación entre la clave y el texto cifrado lo más compleja posible, mientras que la segunda se refiere a la reorganización o extendido de bits en el mensaje para que cualquier redundancia en el texto plano se distribuya por el texto cifrado, escondiendo así propiedades estadísticas del texto plano. (Menezes, *et al.*, 1996) (Paar, *et al.*, 2010)






Cabe destacar que, como todo cifrado por bloques, se debe encriptar siempre en bloques de igual tamaño, y para evitar que el último sea de diferente tamaño, se completa el excedente con un “relleno”, denominado *Padding*. (Wehbe, 2020)

4.1.2. AES: *Advanced Encryption Standard*

Con el transcurso del tiempo se fue observando que el algoritmo DES resultaba cada vez menos seguro, con lo cual se comenzó la búsqueda de un nuevo estándar. En 2001 el algoritmo de cifrado de *Rijndael* (Daemen, *et al.*, 2003) fue seleccionado, pasando a denominarse Estándar de Encriptación Avanzado (AES, *Advanced Encryption Standard*). AES resulta ser el cifrado simétrico más ampliamente utilizado; dentro de los estándares comerciales que lo incluyen se encuentran el protocolo de seguridad de internet (IPSec, *Internet Protocol Security*), la encriptación estándar de Wi-Fi IEE 802.11i, el protocolo de seguridad de red de Shell (SSH, *Secure Shell*), el teléfono de internet Skype, entre otros. Resulta un protocolo altamente seguro dado que, a la fecha, el mejor ataque es el de *fuerza bruta*. AES no se basa en la red de *Feistel* (como su predecesor), sus operaciones básicas usan aritmética de *campos de Galois* (los cuales se verán con detenimiento en el siguiente tipo de criptografía) y proveen fuerte *Difusión* y *Confusión*. (Paar, *et al.*, 2010)

El algoritmo de AES es un cifrado en bloques de 128 bits cada uno, o 16 bytes, el cual soporta tres diferentes tamaños de clave (128, 192 y 256 bits). El mismo trabaja a partir de varias vueltas, cuyo número depende del tamaño de la clave: para una clave de 128 bits, se realizan 10 vueltas; si se selecciona en cambio una clave de 192 bits, el número de vueltas es de 12; y para el caso de una clave de 256 bits, resultan 14 vueltas. El algoritmo se basa en tres tipos de capas, que se completan en cada vuelta (excepto la vuelta cero que ejecuta una sola capa, además ocurre una pequeña variación en la última vuelta). En cada vuelta se produce una sub-clave, derivada de la clave principal, de 128 bits (sin importar el tamaño de la clave principal utilizada). Este algoritmo trabaja a nivel de byte, a diferencia de DES, es por esto por lo que se lo suele conocer como un protocolo orientado a byte (en contraste con DES, que se lo identifica como un protocolo orientado a bit). (Wehbe, 2020)

Las capas antes mencionadas son:

- 
Capa de adición de la clave (Key Addition): la sub-clave, que deriva de la clave principal, opera un XOR al estado actual (se aplica a todas las vueltas);
- 
Capa de sustitución de byte (Byte Substitution, S-Box): cada elemento del estado es transformado no linealmente a través de tablas con propiedades matemáticas especiales (logrando así que el XOR de dos textos cifrados no sea el XOR de los mensajes planos correspondientes), incorporando *Confusión* a los datos (se aplica a todas las vueltas, salvo a la cero);
- 
Capa de difusión (Diffusion): provee *Difusión* a todos los bits del estado, la misma consiste en dos subcapas que producen operaciones lineales:
 - 
Capa de intercambio de filas (ShiftRows): permuta los datos del estado a nivel de byte (se aplica a todas las vueltas, salvo a la cero),
 - 
Capa de mezcla de columnas (MixColumn): operación de matriz que combina (mezcla) las columnas de cuatro bytes en una, que resulta en una operación muy importante, dado que aquí la influencia de los bytes es difundida a través de todo el estado (se aplica a todas las vueltas, salvo a la cero y la última).

Una vuelta normal (sin considerar la vuelta cero) consiste en la aplicación de las tres capas según la siguiente secuencia: capa de sustitución de byte – capa de difusión (capa de intercambio de filas – capa de mezcla de columnas) – capa de adición de la clave. Cabe aclarar

que se agrega una operación de XOR con sub-claves al comienzo y al final del proceso, que se suele denominar blanqueamiento de clave (*Key Whitening*). Es por esto último que el número de sub-claves resulta en una unidad más que el número de vueltas. (Wehbe, 2020) (Paar, *et al.*, 2010)

Aclaración: el ataque de *fuerza bruta* es aquel que agota todas las posibilidades, haciendo una simple “*prueba y error*”. Se supone que el adversario obtiene una pequeña parte de texto plano que antes estaba cifrado y trata de descifrar el texto cifrado con todas las posibles claves. Si el texto resultante coincide con el texto plano, se está en presencia de la clave correcta, si no, pasa a la siguiente clave posible. Un ataque de fuerza bruta siempre es posible en teoría, si es viable o no depende de la cantidad de claves existentes, o sea, del tamaño de la clave. Esto ocurre dado que, si probar todas las claves posibles lleva mucho tiempo, se dice que el cifrado es computacionalmente seguro contra ataques de fuerza bruta. (Paar, *et al.*, 2010)

4.2. Criptografía Asimétrica

Este método de cifrado, como ya se comentó previamente, se apoya en el uso de funciones unidireccionales, las cuales son de fácil cómputo para encriptar, y de “*imposible*” cálculo inverso, para descifrar (volviéndose factible su resolución con algún dato extra). Por “*imposible*” se refiere a inviable computacionalmente, o sea, de una complejidad exponencial. Aquí existen dos claves diferentes, una pública y una privada: la primera para encriptar, difundida libremente, y la segunda para descifrar por el receptor, que solo la conoce él. La clave pública no es necesario mantenerla en secreto, lo cual resulta una gran ventaja respecto a los sistemas simétricos, puesto que la distribución de la clave pública es mucho más simple que para el caso de un sistema simétrico, donde el problema reside en el canal seguro necesario para transportar la clave simétrica (que no es posible enviar por el canal inseguro por el cual se transmite el mensaje, que resultaría lo más conveniente). Los esquemas asimétricos son sustancialmente más lentos que los simétricos, es por esto por lo que en la práctica se utiliza un esquema asimétrico para transportar la clave simétrica, que luego es utilizada para encriptar los datos con un cifrado simétrico. (Menezes, *et al.*, 1996) (Paar, *et al.*, 2010)

Existen tres familias de algoritmos de criptografía asimétrica que son de relevancia práctica, la clasificación se basa en el problema computacional por detrás (que resulta de la función unidireccional antes mencionada). Las mismas son:

- 👑 *Esquema de factorización de enteros*: estos esquemas asimétricos se basan en el hecho de que es difícil factorizar números enteros grandes;
- 👑 *Esquema de logaritmo discreto*: estos se basan en el problema de logaritmo discreto en campos finitos;
- 👑 *Esquemas de curvas elípticas*: una generalización del algoritmo de logaritmo discreto son los esquemas de curvas elípticas.

(Paar, *et al.*, 2010)

Las primeras dos familias fueron propuestas en los años 70 y la tercera en los 80s. No existen ataques conocidos contra ninguno de estos esquemas si los parámetros, especialmente los operandos y los tamaños de las claves, son elegidos cuidadosamente. Es de esperar que mientras más grandes sean ambos parámetros, más seguro el algoritmo se torna. Para comparar el nivel de seguridad de los algoritmos, se utiliza la denominada *seguridad de n bits*, que implica que el mejor ataque conocido requiere 2^n pasos (dicho nombre proviene de que en criptografía simétrica un nivel de seguridad de n posee una clave de n bits). En la siguiente tabla se muestra la recomendación de cuatro niveles de seguridad, en la cual algoritmos de las dos primeras familias requieren operadores y claves muy grandes, mientras que en la tercera familia son significativamente más pequeños, aunque el doble que los de cifrado simétrico con igual fuerza criptográfica. Para proveer seguridad a largo plazo (de varias décadas) se debería elegir un nivel de seguridad de 128 bits, en los cuales se puede apreciar que para las tres familias se requieren claves muy grandes. (Paar, *et al.*, 2010)

Tabla II: niveles de seguridad recomendados para algoritmos asimétricos y simétricos (Paar, et al., 2010).

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

4.2.1. Algunos Conceptos Matemáticos Necesarios

En esta sección se hará un paréntesis para comprender algunas estructuras matemáticas que nos permitirán luego construir los principales sistemas criptográficos.

Se llama al conjunto de **números enteros** \mathbb{Z} , a los números tales que

$$\mathbb{Z} = \{ \dots, -2, -1, 0, 1, 2, \dots \} \tag{3}$$

Sea $a, b \in \mathbb{Z}$, se define a $c \in \mathbb{Z}$ como el *máximo común divisor*, o $c = mcd(a, b)$, al entero más grande seleccionado de todos los divisores que son comunes a ambos números. Se define al conjunto de **números naturales** \mathbb{N} , como los enteros tales que

$$\mathbb{N} = \{1, 2, \dots\} \tag{4}$$

Un número $n \in \mathbb{N}$, con $n \geq 2$, es un *número primo* si sus únicos divisores positivos son 1 y el propio n . Dos números $a, b \in \mathbb{N}$ se los define *coprimos*, o mutuamente primos, si $mcd(a, b) = 1$. (Wehbe, 2020)

Se define a un **grupo** $(G, *)$ como un conjunto G con la operación binaria $*$ tal que cumpla con las siguientes propiedades:

- 👑 cerradura: para todo $a, b \in G$, $a * b \in G$;
- 👑 asociatividad: para todo $a, b, c \in G$, $a * (b * c) = (a * b) * c$;
- 👑 existencia de la identidad: existe un elemento $e \in G$ llamado *elemento neutro* o elemento identidad, tal que $a * e = e * a = a$, para todo $a \in G$;
- 👑 existencia del inverso: para todo $a \in G$ existe un elemento $b \in G$ llamado *elemento inverso* de a , tal que $a * b = b * a = e$.

Aclaración: la operación $*$ puede ser cualquier operación existente o no previamente, que sea definida y cumpla con dichas propiedades. Si además de lo anterior, resulta que la operación del grupo es conmutativa (o sea que para todo $a, b \in G$, $a * b = b * a$), se dice que el grupo es un **grupo abeliano** o conmutativo. (Wehbe, 2020)

Un grupo importante es el grupo multiplicativo \mathbb{Z}_n^* asociado a n . Este grupo es definido como el grupo que contiene todos los números que son *coprimos* con n en \mathbb{Z}_n (por ejemplo $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$). En el caso que n sea un número primo, $\mathbb{Z}_n^* = \{1, 2, \dots, n - 1\}$ (por ejemplo $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$). No se debe confundir \mathbb{Z}_n con \mathbb{Z}_n^* ; \mathbb{Z}_n es el conjunto finito $\mathbb{Z}_n = \{0, 1, 2, \dots, n - 1\}$. (Wehbe, 2020)

Se define el *orden de un elemento* en un grupo de la siguiente manera: sea $(G, *)$ un grupo y $a \in G$, el orden de a , denotado por $ord(a)$, es el entero positivo más pequeño tal que

$$a^k = \underbrace{a * \dots * a}_{k \text{ veces}} = 1 \tag{5}$$

donde 1 es la identidad de G sobre $*$. Si se siguen multiplicando los resultados por a (luego de alcanzar la identidad), se comienzan a repetir los resultados, denotando así un comportamiento

cíclico. Un grupo $(G,*)$ que contiene un elemento α con orden máximo $ord(\alpha) = |G|$ se llama **grupo cíclico**. Los elementos con orden máximo son llamados elementos primitivos o *generadores*. Un elemento α de un grupo G con orden máximo es llamado generador dado que todo elemento $a \in G$ puede ser escrito como una potencia $\alpha^i = a$, de este elemento por algún entero i . Por ejemplo: 2 es un elemento primitivo de \mathbb{Z}_{11}^* .

$$2^1 = 2 \tag{6}$$

$$2^2 = 4 \tag{7}$$

$$2^3 = 8 \tag{8}$$

$$2^4 = 5 \pmod{11} \tag{9}$$

$$2^5 = 10 \pmod{11} \tag{10}$$

$$2^6 = 9 \pmod{11} \tag{11}$$

$$2^7 = 7 \pmod{11} \tag{12}$$

$$2^8 = 3 \pmod{11} \tag{13}$$

$$2^9 = 6 \pmod{11} \tag{14}$$

$$2^{10} = 1 \pmod{11} \tag{15}$$

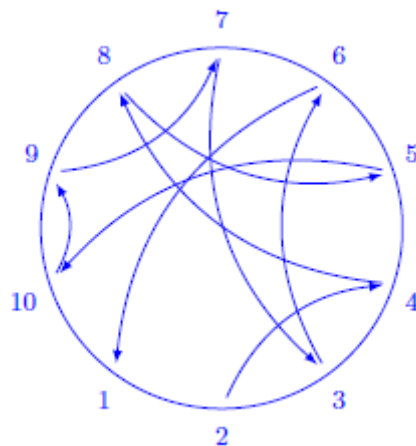


Figura 9: comportamiento cíclico de \mathbb{Z}_{11}^* (Wehbe, 2020).

Este aparente comportamiento arbitrario del exponente con respecto a los elementos en el grupo es lo que vuelve interesantes a los grupos cíclicos para sistemas criptográficos. Se demuestra

que, para todo número primo p , $(\mathbb{Z}_p^*, *)$ es un grupo cíclico; y sea $(G, *)$ un grupo cíclico, entonces para todo $a \in G$: $a^{|G|} = 1$, y $ord(a)$ divide a $|G|$. (Wehbe, 2020)

Se define a un **anillo** $A (A, +, *)$ como un conjunto con dos operaciones binarias $+$ y $*$, tales que cumplen con las siguientes propiedades:

- 👑 todos los elementos de A forman un grupo abeliano con la operación $+$ y el elemento neutro 0 ;
- 👑 la operación $*$ es asociativa, o sea para todo $a, b, c \in A$, $a * (b * c) = (a * b) * c$, y hay un elemento $1 \in A$, llamado elemento neutro (o elemento identidad), tal que $a * 1 = 1 * a = a$, para todo $a \in A$;
- 👑 la operación $*$ es distributiva respecto de $+$: para todo $a, b, c \in A$, $a * (b + c) = (a * b) + (a * c)$, y $(a + b) * c = (a * c) + (b * c)$.

Aclaración: nuevamente las operaciones $+$ y $*$ pueden resultar ser cualquier operación existente o no, que esté bien definida y cumpla con las propiedades expresadas. Por ejemplo: \mathbb{Z}_m con la adición y la multiplicación módulo m , es un anillo (denominado anillo entero). Ahora bien, se necesita una estructura con dos operaciones, denominadas adición y producto (además de implícitamente la sustracción y la división). El anillo entero es muy débil, dado que la multiplicación no es invertible). (Wehbe, 2020)

Así se define un **campo** $(F, +, *)$ como un conjunto F con dos operaciones binarias $+$ y $*$ (usualmente adición y producto), tales que cumplen las siguientes propiedades:

- 👑 todos los elementos de F forman un grupo aditivo con la operación $+$ y el elemento neutro 0 ;
- 👑 todos los elementos de F excepto el 0 forman un grupo multiplicativo con la operación $*$ y el elemento neutro 1 ;
- 👑 el producto es distributivo respecto de la adición: para todo $a, b, c \in F$, $a * (b + c) = (a * b) + (a * c)$, y $(a + b) * c = (a * c) + (b * c)$.

Por ejemplo: el conjunto \mathbb{Z} con las operaciones $+$ y $*$ forman un campo. Unos campos muy interesantes son los campos finitos, o **campos de Galois**. El número de elementos en el campo es llamado su *orden* o cardinalidad. Un campo de orden m solo existe si m es una potencia prima, o sea $m = p^n$, para algún entero positivo n y número primo p . En este caso p es llamado el *característico* del campo finito (por ejemplo, hay campos finitos con 11 elementos o con $343 = 7^3$ elementos, pero no con 12 elementos). El ejemplo más simple es el caso de los

campos primos. El anillo entero \mathbb{Z}_p es denotado como $GF(p)$ y se lo llama campo primo, o campo de Galois con un número primo de elementos. Todos los elementos no cero de $GF(p)$ tienen un inverso multiplicativo. Queda aclarar que la aritmética en $GF(p)$ se realiza módulo p . (Wehbe, 2020)

4.2.2. RSA: Rivest, Shamir, Adleman

En 1977, Ronald Rivest, Adi Shamir y Leonard Adleman propusieron un esquema que se volvió ampliamente utilizado (Rivest, *et al.*, 1978): el esquema de criptografía asimétrica RSA (nombrado tras las siglas de sus inventores). En este caso, la función unidireccional en la que se basa el algoritmo es el *problema de factorización de enteros*: multiplicar dos números primos grandes es computacionalmente (y manualmente) simple, pero factorizar el producto resultante es “*imposible*” (o sea, inviable computacionalmente). (Paar, *et al.*, 2010)

El proceso de RSA involucra cuatro pasos: generación de la clave, distribución de la clave, encriptación y desencriptación. Para explicarlos se considerará que Bob quiere enviar un mensaje cifrado a Alice, o sea Bob debe encriptar el texto plano con la clave pública de Alice y Alice desencriptar el texto cifrado con su clave privada.

1. Generación de la clave:
 - a. Alice elige dos números primos grandes diferentes, p y q , que se mantienen privados.
 - b. Alice calcula $n = p \cdot q$, denominado *módulo de encriptación*.
 - c. Alice calcula $\phi(n) = (p - 1) \cdot (q - 1)$, cuyo valor se mantiene privado.
 - d. Alice elige un número entero e tal que $1 < e < \phi(n)$ y $\text{mcd}(e, \phi(n)) = 1$, el cual se designa como *exponente de la encriptación*.
 - e. Alice calcula, mediante el *Algoritmo Extendido de Euclides*, el único entero d , con $1 < d < \phi(n)$, tal que $e \cdot d \text{ mod } \phi(n) = 1$ (o sea, el inverso multiplicativo $d = e^{-1} \text{ mod } \phi(n)$), denominado *exponente de la desencriptación*.
 - f. Se define a (n, e) como la clave pública de Alice, y a d como su clave privada.
2. Distribución de la clave:
 - a. Alice transmite mediante el canal inseguro su clave pública (n, e) a Bob, y mantiene su clave privada d , que nunca es transmitida.
3. Encriptación:

- a. Bob obtiene la clave pública de Alice (n, e) , de forma auténtica.
 - b. Bob encripta el mensaje en texto plano m que quiere enviar a Alice calculando $c = m^e \text{ mod } n$ (mediante el *Algoritmo de Exponenciación Rápida*), con $m \in \{0, \dots, n - 1\}$.
 - c. Bob transmite el texto cifrado c a Alice (por el canal inseguro).
4. Descriptación:
- a. Alice recupera el mensaje m utilizando su clave privada d , calculando $m = c^d \text{ mod } n$ (mediante el *Algoritmo de Exponenciación Rápida*).

(Wehbe, 2020) (Menezes, *et al.*, 1996)

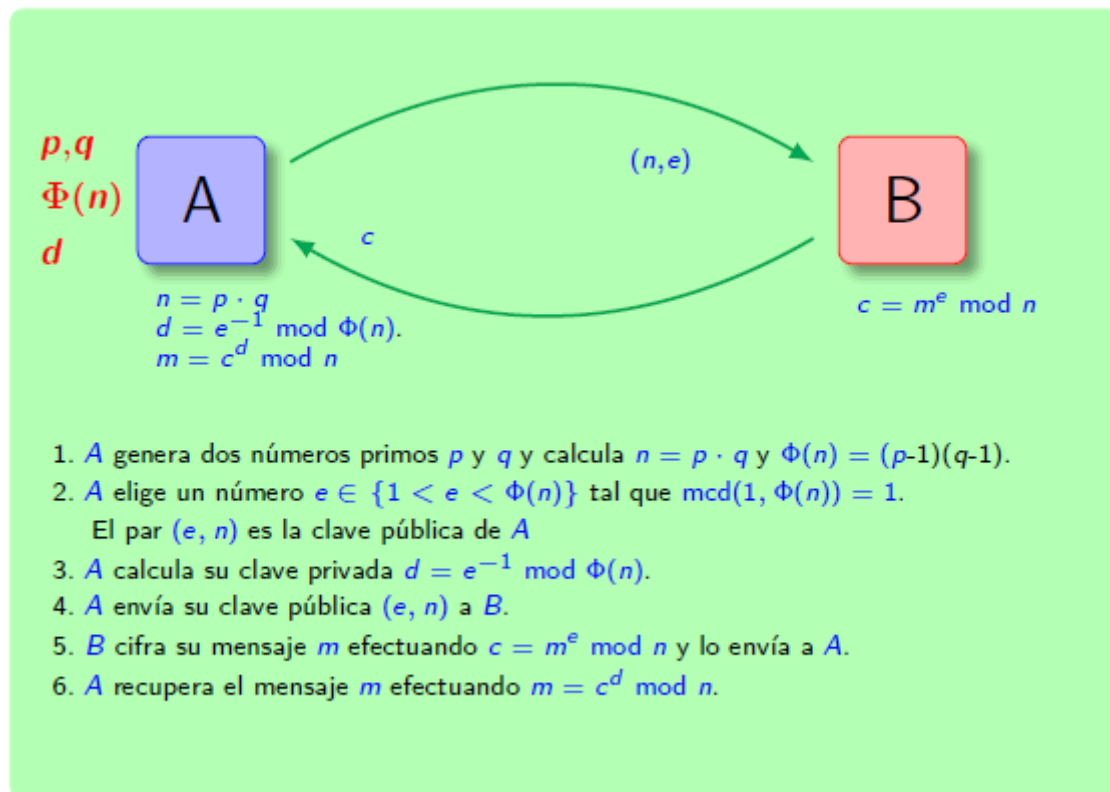


Figura 10: el sistema criptográfico RSA (Wehbe, 2020).

Nota: cuando en el paso 3a se hace referencia a la obtención de la clave pública en forma auténtica, se quiere evitar un ataque de *Impersonificación*. Se explicarán a continuación el *Algoritmo Extendido de Euclides* (nombrado en el paso 1e) y el *Algoritmo de Exponenciación Rápida* (mencionado en los pasos 3b y 4a).

Dado que los exponentes son números muy grandes, se necesita algún mecanismo para mejorar la complejidad de cómputo. Es por esto por lo que se opta por buscar algún método más eficiente para computar exponenciaciones discretas con grandes exponentes. El *Algoritmo de Exponenciación Rápida* se basa en iteraciones en las cuales primero se escanean los bits del exponente representado en binario, de izquierda a derecha (del bit más significativo al menos significativo), se eleva al cuadrado el resultado acumulado, y luego, solo si el bit escaneado es igual a 1, se multiplica el resultado acarreado, por la base. Se inicializa el resultado en el valor de la base y se obvia el primer bit, dado que se lo presupone igual a 1. (Paar, *et al.*, 2010) A continuación, puede apreciarse este algoritmo (para el código de la implementación, ver Anexo B):

```

Square-and-Multiply for Modular Exponentiation
Input:
base element  $x$ 
exponent  $H = \sum_{i=0}^t h_i 2^i$  with  $h_i \in \{0, 1\}$  and  $h_t = 1$ 
and modulus  $n$ 
Output:  $x^H \bmod n$ 
Initialization:  $r = x$ 
Algorithm:

1  FOR  $i = t - 1$  DOWNTO 0
1.1   $r = r^2 \bmod n$ 
      IF  $h_i = 1$ 
1.2   $r = r \cdot x \bmod n$ 
2  RETURN ( $r$ )
    
```

Figura 11: Algoritmo de Exponenciación Rápida (Paar, *et al.*, 2010).

El *Algoritmo Extendido de Euclides* por su parte, nos permite calcular de una forma eficiente los inversos multiplicativos y los coeficientes de Bézout. Sean $a, b \in \mathbb{Z}$ y $d = \text{mcd}(a, b)$, existen enteros x e y tales que $a \cdot x + b \cdot y = d$, la cual es denominada *Identidad de Bézout*, siendo x e y los coeficientes de Bézout. Para el caso en el cual $\text{mcd}(a, b) = d = 1$, $y \bmod a$ resulta ser el inverso multiplicativo de $b \bmod a$ (pues $(y \cdot b) \bmod a = 1$), así como $x \bmod b$, el inverso multiplicativo de $a \bmod b$ (pues $(x \cdot a) \bmod b = 1$). (Wehbe, 2020)

El presente algoritmo cuenta con tres valores por iteración que se actualizan según los últimos dos valores de cada uno (se puede pensar como un vector de tres elementos, en el cual deberíamos guardar tres valores por elemento, porque necesitamos los calculados en las últimas dos iteraciones para computar el valor actual), además existe un cuarto elemento por iteración que utiliza dos valores dentro de los anteriores mencionados. Los primeros tres valores

mencionados (r , s y t de ahora en adelante), realizan una resta entre el anteúltimo y último valor de cada uno multiplicado por el cuarto valor comentado (de ahora en adelante q), el cual resulta una división entera entre los últimos dos valores de r . El algoritmo finaliza cuando r se vuelve cero, y los valores devueltos son el último valor de s y t (que resultan ser x e y , coeficientes de Bézout, respectivamente) y el anterior valor de r (que es el $mcd(a, b)$, siendo a y b los valores que se ingresan al algoritmo). Para la implementación se guardan sólo los valores necesarios y se sobrescriben los que se dejan de necesitar.

El algoritmo se basa en el *Algoritmo de Euclides* tradicional, que sólo provee el $mcd(a, b)$, el cual utiliza como principio que el $mcd(a, b) = mcd(b, r)$, con r resto de dividir a y b (que se realiza para cada iteración), y que $mcd(a, 0) = a$ (usado para finalizar la iteración); así como también la sustitución del resto, despejado de la ecuación

$$a = b \cdot q + r \tag{16}$$

iteración a iteración (siendo q cociente y r resto de dividir a y b). (Wehbe, 2020) Se muestra a continuación el *Algoritmo Extendido de Euclides* (para el código de la implementación, ver Anexo B):

```

int[4]          Euclid (int a, int b);
sol = new int[5]; // The initialisation
r0 = a;
r1 = b;
s0 = 1;
s1 = 0;
t0 = 0;
t1 = 1;
q1 = r0 ÷ r1;
repeat until (ri+1 == 0) { // The process
    ri+1 = ri-1 - qi * ri;
    si+1 = si-1 - qi * si;
    ti+1 = ti-1 - qi * ti;
    qi+1 = ri ÷ ri+1;
}
sol[0] = ri;
sol[1] = si-1;
sol[2] = si;
sol[3] = ti-1;
sol[4] = ti;
return (sol);

```

Figura 12: Algoritmo Extendido de Euclides (Wehbe, 2020).

La prueba de funcionamiento de RSA se puede apreciar de una forma muy simple. Por definición:

$$c = m^e \bmod n \quad (17)$$

y se desea comprobar que:

$$c^d \bmod n = m \quad (18)$$

$$(m^e \bmod n)^d \bmod n = m \quad (19)$$

por potencia de potencia

$$m^{e \cdot d} \bmod n = m \quad (20)$$

Se sabe que, como d es el inverso multiplicativo de $e \pmod{\phi(n)}$:

$$e \cdot d \bmod \phi(n) = 1 \quad (21)$$

y por definición de módulo, esto último implica que:

$$e \cdot d = 1 + k \cdot \phi(n) \quad (22)$$

para algún número entero k . Entonces:

$$m^{e \cdot d} \bmod n = m^{1+k \cdot \phi(n)} \bmod n \quad (23)$$

por producto de potencias de igual base y potencia de potencia

$$m^{e \cdot d} \bmod n = m^1 \cdot m^{k \cdot \phi(n)} \bmod n = m \cdot (m^{\phi(n)})^k \bmod n \quad (24)$$

Por *Teorema de Euler*

$$m^{e \cdot d} \bmod n = m \quad (25)$$

(Wehbe, 2020) (Menezes, *et al.*, 1996)

Nota: se explicará a continuación el *Teorema de Euler* arriba mencionado. Sea $n \in \mathbb{N}$, se define a la *Función de Euler* asociada a n , usualmente denotada por $\phi(n)$, a la cantidad de números $x \in \{1, \dots, n\}$ tal que x y n son *coprimos*. Un caso especial ocurre cuando $n = p \cdot q$, con p y q *números primos*. En este caso, $\phi(n) = (p - 1) \cdot (q - 1)$. El *Teorema de Euler* dice: sea $n \geq 2$, para todo $a \in \mathbb{Z}_n^*$ (o sea a y n son coprimos), $a^{\phi(n)} \bmod n = 1$. (Wehbe, 2020) Esto último implica que, en la prueba de funcionamiento de arriba, $m^{\phi(n)} \bmod n = 1$.

4.2.2.1. RSA Liviano

Una adaptación liviana de RSA es la planteada por el producto de tres números primos (a diferencia de los dos números clásicos de RSA), redefiniéndose ambas claves. Esta solución

implica mayor seguridad, dado que, en vez de poseer dos componentes ocultos, ahora resultan ser tres, dificultando aún más romper el producto entre dichos tres números. También ocasiona menor uso de memoria para los números primos, dado que se logra igual longitud de clave que en el algoritmo RSA clásico con primos más pequeños. De este modo se aceleran las etapas de generación de clave y de descryptación. Para mejorar incluso más esta última etapa, se utiliza el *Teorema de los Restos Chinos* que facilita el cálculo. (Abd Zaid, *et al.*, 2018)

En la siguiente tabla se puede apreciar que el algoritmo de RSA modificado es más rápido que el estándar (especialmente en generación de clave y en descryptación). La velocidad de generación de clave mejora en un promedio de 80%, la de descryptación en un 96%, y solo un 4% en la encriptación, aproximadamente. (Abd Zaid, *et al.*, 2018)

Tabla III: comparación de tiempos entre los algoritmos de RSA y de RSA modificado (Abd Zaid, et al., 2018).

Key size	Message size	Standard RSA time			Modified RSA time		
		K-generation	Encryption	Decryption	K-generation	Encryption	Decryption
68 bit	3 bit	0.01111	0.00015	0.000151	0.00791	0.000137	0.00005
340 bit	20 bit	0.228	0.0157	0.0144	0.107	0.01005	0.00096
664 bit	94 bit	5.34	0.219	0.202	0.989	0.213	0.00643

El algoritmo propuesto presenta los siguientes pasos (para el código de la implementación, ver Anexo B):

1. Generación de la clave:

- a. Alice elige tres números primos grandes diferentes, p , q y s que se mantienen privados.
- b. Alice calcula $n = p \cdot q \cdot s$.
- c. Alice calcula $\phi(n) = (p - 1) \cdot (q - 1) \cdot (s - 1)$, cuyo valor se mantiene privado.
- d. Alice elige un número entero e tal que $1 < e < \phi(n)$ y $\text{mcd}(e, \phi(n)) = 1$.
- e. Alice calcula, mediante el *Algoritmo Extendido de Euclides*, el único entero d , con $1 < d < \phi(n)$, tal que $e \cdot d \text{ mod } \phi(n) = 1$ (o sea, el inverso multiplicativo $d = e^{-1} \text{ mod } \phi(n)$).
- f. Alice calcula, mediante el *Algoritmo Extendido de Euclides*, el entero d_p , tal que $e \cdot d_p \text{ mod } (p - 1) = 1$, y el entero d_q , tal que $e \cdot d_q \text{ mod } (q - 1) = 1$ (o sea, los inversos multiplicativos $d_p = e^{-1} \text{ mod } (p - 1)$, y $d_q = e^{-1} \text{ mod } (q - 1)$).

- g. Alice calcula, mediante el *Algoritmo Extendido de Euclides*, el entero Q tal que $q \cdot Q \bmod p = 1$ (o sea, el inverso multiplicativo $Q = q^{-1} \bmod p$), si $p > q$, o $p \cdot Q \bmod q = 1$ (o sea, el inverso multiplicativo $Q = p^{-1} \bmod q$), si $q > p$.
 - h. Se define a (n, e) como la clave pública de Alice, y a (Q, d_p, d_q, p, q) como su clave privada.
2. Distribución de la clave:
 - a. Alice transmite mediante el canal inseguro su clave pública (n, e) a Bob, y mantiene su clave privada (Q, d_p, d_q, p, q) , que nunca es transmitida.
 3. Encriptación:
 - a. Bob obtiene la clave pública de Alice (n, e) , de forma auténtica.
 - b. Bob encripta el mensaje en texto plano m que quiere enviar a Alice calculando, mediante el algoritmo de *Exponenciación Rápida*, $c = m^e \bmod n$.
 - c. Bob transmite el texto cifrado c a Alice (por el canal inseguro).
 4. Desencriptación, utilizando el concepto del *Teorema de los Restos Chinos*:
 - a. Alice calcula, mediante el algoritmo de *Exponenciación Rápida*, $m_a = c^{d_p} \bmod p$ y $m_b = c^{d_q} \bmod q$.
 - b. Alice calcula $h = (Q \cdot (m_a - m_b)) \bmod p$.
 - c. Alice recupera el mensaje $m = m_b + (h \cdot q)$.

(Abd Zaid, *et al.*, 2018)

Nota: a continuación, se explicará el *Teorema de los Restos Chinos*, comentado en el paso 4, que se utiliza para agilizar la desencriptación. El *Teorema de los Restos Chinos* dice: sean $p, q \in \mathbb{Z}$, tales que $p, q > 1$ y $\text{mcd}(p, q) = 1$, el siguiente sistema de ecuaciones tiene una solución y que es única módulo $p \cdot q$, y tiene una forma determinada:

$$\begin{cases} y \bmod p = a \\ y \bmod q = b \end{cases} \quad (26)$$

Como la solución es única, se puede ver este teorema como una biyección entre $\mathbb{Z}_{p \cdot q}$ y $\mathbb{Z}_p \times \mathbb{Z}_q$, con lo cual cualquier operación compleja en $\mathbb{Z}_{p \cdot q}$, puede realizarse en $\mathbb{Z}_p \times \mathbb{Z}_q$ con menos costo.

(Wehbe, 2020)

Para el caso de la descryptación de RSA, el objetivo es realizar la siguiente exponenciación eficientemente:

$$y = x^d \text{ mod } n \quad (27)$$

Cabe aclarar que como esto ocurre en el paso de la descryptación, el receptor posee la clave privada, o sea los números primos p y q , tales que $p \cdot q = n$. La idea principal del *Teorema de los Restos Chinos* es que se prefiere realizar dos exponenciaciones módulo algún número más pequeño, a una sola exponenciación de módulo grande. El primer paso es transformar los datos ingresados al nuevo dominio. Así obtenemos primero ambas bases

$$x_p = x \text{ mod } p \quad (28)$$

$$x_q = x \text{ mod } q \quad (29)$$

y ambos exponentes

$$d_p = d \text{ mod } (p - 1) \quad (30)$$

$$d_q = d \text{ mod } (q - 1) \quad (31)$$

A continuación, se realizan las dos exponenciaciones

$$y_p = x_p^{d_p} \text{ mod } p \quad (32)$$

$$y_q = x_q^{d_q} \text{ mod } q \quad (33)$$

Finalmente se requiere realizar la transformación inversa, para volver al dominio del problema. Se determinan los coeficientes

$$c_p = q^{-1} \text{ mod } p \quad (34)$$

$$c_q = p^{-1} \text{ mod } q \quad (35)$$

y se reconstruye el valor resultante:

$$y = (q \cdot c_p \cdot y_p + p \cdot c_q \cdot y_q) \text{ mod } n \quad (36)$$

Como los números primos casi nunca se modifican en una implementación de RSA, las expresiones $q \cdot c_p$ y $p \cdot c_q$ suelen precomputarse. (Paar, *et al.*, 2010) Lo realizado aquí no es más ni menos que el sistema propuesto inicialmente llevado al siguiente caso:

$$\begin{cases} y \bmod p = x_p^{d_p} \bmod p \\ y \bmod q = x_q^{d_q} \bmod q \end{cases} \quad (37)$$

La implementación de RSA liviano siguió los mismos pasos que el algoritmo antes comentado. En la etapa de generación de claves, se llama cuatro veces al *Algoritmo Extendido de Euclides*, para determinar los inversos multiplicativos d , d_p , d_q y Q (se puede agregar un chequeo extra para corroborar que el mcd entre los parámetros ingresados es igual a 1, dado que deben ser coprimos, aprovechando el hecho de que el algoritmo también devuelve este valor). Al seleccionar el valor e , debe chequearse que $mcd(e, \phi) = 1$ (puede utilizarse nuevamente el *Algoritmo Extendido de Euclides* para esto). Para la encriptación, se llama al *Algoritmo de Exponenciación Rápida* una única vez, para determinar el texto cifrado c ; así como también para la desencriptación, que se lo invoca dos veces, para determinar m_a y m_b .

Sugerencias finales: los números primos p y q deben seleccionarse de tal forma que factorizar $n = p \cdot q$ sea computacionalmente inviable, con lo cual se recomienda elegir números suficientemente grandes. Por ejemplo, si se desea utilizar 1024 bits módulo n , se requiere que tanto p como q sean de 512 bits. (Menezes, *et al.*, 1996) Para el caso de RSA liviano, al ser tres los valores, esto disminuye y se distribuye en ellos, con lo cual se permiten números más pequeños.

Otra recomendación en la selección de p y q es que la diferencia $p - q$ no debería ser muy pequeña, puesto que, sino ambos números serían muy parecidos, $p \approx q$ y entonces $p \approx \sqrt{n}$, con lo cual n puede factorizarse eficientemente simple por prueba de división de todos los enteros impares cercanos a \sqrt{n} . Es por esto por lo que se recomienda una diferencia grande. (Menezes, *et al.*, 1996)

Para el exponente de encriptación, se aconseja utilizar números no tan grandes de e , si se requiere agilizar la misma (una sugerencia puede ser utilizar un número que posea la menor cantidad de 1s en su representación binaria). Comúnmente se utiliza $e = 3$ (se requiere que ni $p - 1$, ni $q - 1$ sean divisibles por 3), o $e = 2^{16} + 1 = 65537$ (que obviamente resiste mayores ataques); este último número tiene sólo dos 1s en su representación binaria, por lo cual en el *Algoritmo de Exponenciación Rápida* se computan solamente dieciséis cuadrados y una multiplicación. (Menezes, *et al.*, 1996)

4.2.3. Rabin

Antes de continuar con la segunda familia de algoritmos, se mostrará un algoritmo no tan conocido que se basa en otro problema difícil de determinar: *el problema de la raíz cuadrada discreta*. El mismo se apoya en que la potencia cuadrada discreta es simple de calcular, pero su inversa, la raíz cuadrada discreta, se vuelve inviable. Este protocolo se denomina Rabin y fue publicado en un informe interno del *MIT Laboratory for Computer Science* en 1979 por Michael O. Rabin (Rabin, 1979). Su seguridad se basa en la dificultad de computar raíces cuadradas discretas, cuya dificultad es equivalente al de la factorización discreta. Rabin resulta ser el primer sistema criptográfico asimétrico en el cual recuperar el texto plano completo desde el texto cifrado, se puede probar ser igual de difícil que el de factorizar. (Wehbe, 2020) (Srivastava, *et al.*, 2013)

El sistema de Rabin se basa en el problema de factorizar, dado que computar raíces cuadradas discretas de un número compuesto es simple si se conoce su factorización, pero muy complejo cuando la misma es desconocida. Un inconveniente que presenta este algoritmo es que el proceso de descryptación produce cuatro candidatos, de los cuales tres son falsos (esto se debe a que el proceso de encriptación no es una función inyectiva), y se necesita cierta información extra para poder seleccionar el texto plano correcto. (Srivastava, *et al.*, 2013)

En términos de eficiencia, la encriptación de Rabin requiere computar cuadrados modulo n , lo cual resulta más eficiente que RSA, que precisa computar potencias n . En el caso de la descryptación ambos utilizan el *Teorema de los Restos Chinos*. La desventaja de la descryptación de Rabin, como ya se comentó, viene de que el proceso produce cuatro resultados posibles, mientras que RSA solo retorna el único correcto. (Srivastava, *et al.*, 2013)

Considerando que Bob quiere enviar un mensaje a Alice, el algoritmo de Rabin funciona de la siguiente manera:

1. Generación de la clave:
 - a. Alice elige dos números primos grandes diferentes, p y q , tales que $p \bmod 4 = q \bmod 4 = 3$, que se mantienen privados.
 - b. Alice calcula $n = p \cdot q$.
 - c. Se define a n como la clave pública de Alice, y a (p, q) como su clave privada.
2. Distribución de la clave:

- a. Alice transmite mediante el canal inseguro su clave pública n a Bob, y mantiene su clave privada (p, q) , que nunca es transmitida.
3. Encriptación:
 - a. Bob obtiene la clave pública de Alice n , de forma auténtica.
 - b. Bob encripta el mensaje en texto plano m que quiere enviar a Alice calculando $c = m^2 \bmod n$, con $m \in \{0, \dots, n - 1\}$.
 - c. Bob transmite el texto cifrado c a Alice (por el canal inseguro).
4. Desencriptación:
 - a. Alice calcula las cuatro raíces cuadradas m_1, m_2, m_3 y m_4 de $c \bmod n$, mediante el concepto del *Teorema de los Restos Chinos*, de la siguiente manera:
 - i. Usando el *Algoritmo Extendido de Euclides*, Alice determina los enteros a y b que satisfacen $a \cdot p + b \cdot q = 1$ (o sea, los coeficientes de Bézout), se los puede determinar una sola vez en la etapa de Generación de la clave.
 - ii. Alice calcula $r = c^{\frac{p+1}{4}} \bmod p$, y $s = c^{\frac{q+1}{4}} \bmod q$, mediante el algoritmo de *Exponenciación Rápida*.
 - iii. Alice calcula $x = (a \cdot p \cdot s + b \cdot q \cdot r) \bmod n$, e $y = (a \cdot p \cdot s - b \cdot q \cdot r) \bmod n$.
 - iv. Las cuatro raíces cuadradas de $c \bmod n$ son: $x, -x = n - x, y$, así como también $-y = n - y$.
 - b. Alice recupera el mensaje m decidiendo (con alguna información adicional) cuál de las cuatro raíces es la correspondiente.

(Menezes, *et al.*, 1996) (Srivastava, *et al.*, 2013)

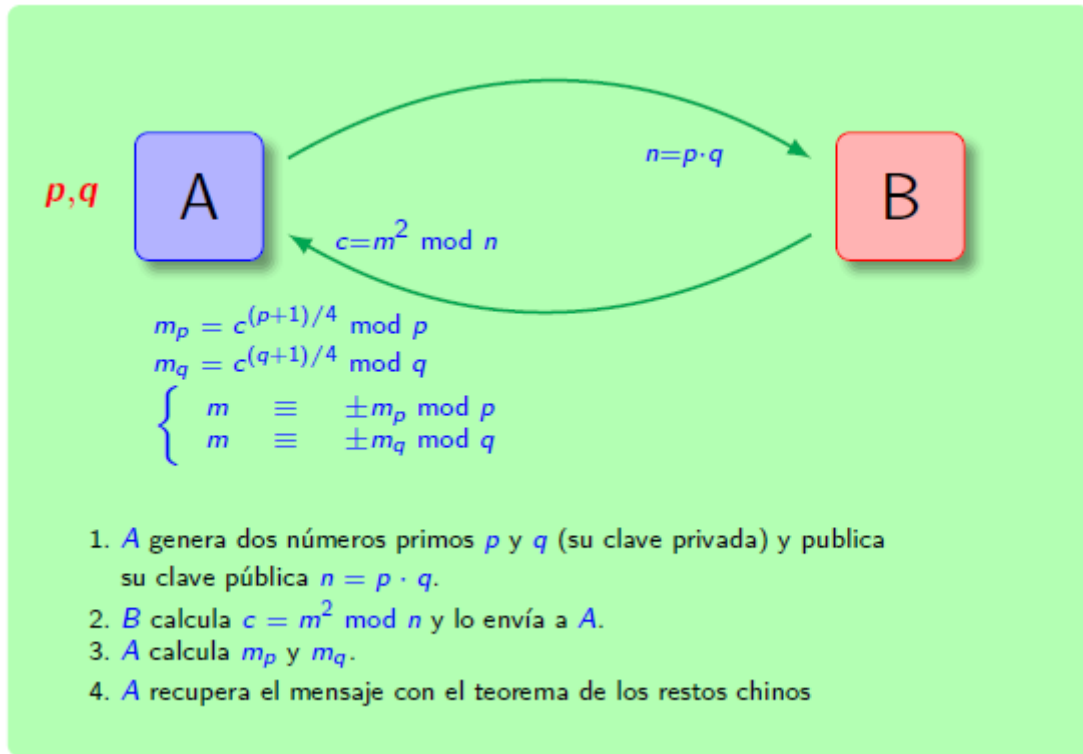


Figura 13: el sistema criptográfico de Rabin (Wehbe, 2020).

Nota: la condición $p \pmod 4 = q \pmod 4 = 3$ se solicita únicamente para simplificar el cómputo de las raíces cuadradas módulo p y q , pero se podría utilizar cualquier par de números.

En la descryptación, como $c = m^2 \pmod n$ (por encriptación), lo que se desea calcular entonces es $m = \sqrt{c} \pmod n$. El Teorema de los Restos Chinos permite llevar este cálculo de la raíz módulo n , a la raíz de los dos primos p y q (que facilitan el cómputo). De este modo se obtiene

$$m_p = r = c^{\frac{p+1}{4}} \pmod p \tag{38}$$

$$m_q = s = c^{\frac{q+1}{4}} \pmod q \tag{39}$$

que nos permiten reconstruir al texto plano m como

$$x = (a \cdot p \cdot s + b \cdot q \cdot r) \pmod n \tag{40}$$

siendo a y b los coeficientes de Bézout. Como $x^2 \pmod n = (-x)^2 \pmod n$, aparecen las otras tres raíces posibles. La condición $p \pmod 4 = q \pmod 4 = 3$ solo ayuda a operar, dado que

garantiza que el exponente sea entero. Como $p \bmod 4 = 3$, sabemos que existe un $k \in \mathbb{Z}$, tal que

$$k \cdot 4 + 3 = p \tag{41}$$

o sea

$$k = \frac{p - 3}{4} \tag{42}$$

Si sumamos 1 a ambos términos

$$k + 1 = \frac{p - 3}{4} + 1 = \frac{p + 1}{4} \tag{43}$$

que sigue siendo entero (de igual modo, para q).

Aclaración: La ambigüedad de seleccionar cuál es la raíz correcta dentro de las cuatro posibilidades, se puede solucionar agregando redundancia preespecificada al texto plano original, previo a la encriptación (por ejemplo, los últimos 64 bits del mensaje se replican). Así, solo una de las cuatro posibles raíces del texto cifrado c va a poseer redundancia, y el receptor debe seleccionar la misma como el texto plano m . Si ninguna de las cuatro raíces posee redundancia, el receptor debería rechazar c por fraudulento. (Menezes, *et al.*, 1996)

En eficiencia, la encriptación de Rabin es una operación extremadamente rápida dado que sólo involucra un único cuadrado modular, lo cual la hace más veloz que el esquema RSA (por ejemplo, con $e = 3$ dicha encriptación implica una multiplicación modular y un cuadrado modular); en tanto a la desencriptación, es más lenta, pero comparable en velocidad a la desencriptación de RSA. (Menezes, *et al.*, 1996) Es por esto último que al presente esquema se lo puede considerar interesante para entornos restringidos.

4.2.3.1. Rabin Liviano: *Esquema Diferencial de Rabin (D-Rabin)*

Una adaptación más veloz al esquema de Rabin es el planteado con un único entero, a diferencia de los dos enteros que utiliza Rabin tradicionalmente como clave privada; en este caso se plantea el uso de una única clave privada, lo cual mejora el tiempo de ejecución comparado con Rabin tradicional. El algoritmo de Rabin Diferencial puede implementarse en entornos restringidos gracias a su estructura simple, y opera más rápido en ambos flancos (servidor y cliente), haciéndolo un sistema versátil. Al generar menos primos, el sistema se

vuelve más veloz que los algoritmos tradicionales. (Kavitha, *et al.*, 2019) Funciona de la siguiente manera (para el código de la implementación, ver Anexo B):

1. Lado del transmisor (Bob):
 - a. Se selecciona el número primo p más cercano al texto plano m , tal que $p \geq m$.
 - b. Se calcula $d = p - m$.
 - c. Se calcula el texto cifrado $c = 2 \cdot (10 \cdot p + d)$.
 - d. Se envía el texto cifrado c .
2. Lado del receptor (Alice):
 - a. Se recibe el texto cifrado c .
 - b. Se calcula $c' = \frac{c}{2}$ (división entera).
 - c. Si el anteúltimo dígito de c' resulta impar se determinan: $p = \frac{c'}{10}$ (división entera) y $d = c' \% 10$. Sino: $d = (c' \% 10) + 10$ y $p = \frac{c'}{10} - 1$ (división entera).
 - d. Se recupera el texto plano $m = p - d$.

(Kavitha, *et al.*, 2019)

Se puede apreciar en los siguientes gráficos cómo el algoritmo D-Rabin disminuye los tiempos de ejecución tanto para el emisor como para el receptor, en comparación con el algoritmo Rabin tradicional. (Kavitha, *et al.*, 2019)

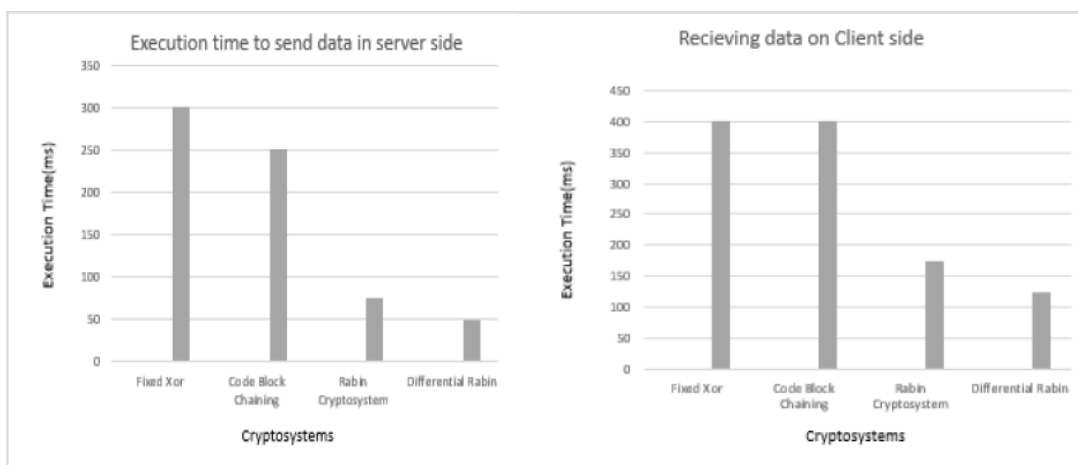


Figura 14: tiempos de ejecución en enviar y recibir datos para diferentes sistemas criptográficos, desde el lado del emisor (I) y del receptor (II) (Kavitha, *et al.*, 2019).

Se realizó la implementación siguiendo el algoritmo propuesto, que a simple vista es muy sencillo, con lo cual era de esperar tiempos mucho más cortos que el propuesto por RSA liviano. D-Rabin resulta ser altamente inseguro, dado que se aplica una función para encriptar, con operaciones muy simples, y su inversa para desencriptar. Por otro lado, no se utiliza clave (se elige un número primo para operar, pero no se envía al emisor, es el mismo emisor quien lo determina, y encima tiene que relacionarse con el texto plano que se desea encriptar); lo cual implica que cualquier receptor que conozca el esquema de encriptación (sea el intencionado o un intruso), puede desencriptar el mensaje plano. La idea de utilizar un solo número primo puede parecer interesante dado que ocasiona ganancia computacional, pero eso genera que se pierda la posibilidad de poseer una clave privada (ambos primos) y una pública (su producto).

Es principalmente por esto (por la falta de seguridad) que se desestimó la implementación D-Rabin propuesta para la comparación de esquemas livianos asimétricos. Se lo muestra (juntamente con su implementación) como un algoritmo no seguro e impráctico, y se explica sus motivos. Como última aclaración, no se encuentra la relación de este con el algoritmo de Rabin tradicional (que se basa en el problema de calcular una raíz cuadrada discreta).

4.2.4. DHKE: *Diffie-Hellman Key Exchange*

Continuando con la segunda familia de algoritmos de criptografía asimétrica, se encuentran los esquemas basados en la intratabilidad de encontrar la solución al *problema del logaritmo discreto* (DLP, *Discrete Logarithm Problem*). El algoritmo más conocido de esta familia es el de intercambio de claves de Diffie-Hellman (DHKE, *Diffie-Hellman Key Exchange*), propuesto por Whitfield Diffie y Martin Hellman en 1976 (Diffie, *et al.*, 1976), con una fuerte influencia del trabajo de Ralph Merkle. DHKE resulta ser el primer esquema de cifrado asimétrico publicado en la literatura abierta. El mismo provee una solución práctica al *problema de distribución de claves*, cuya técnica está implementada en varios protocolos criptográficos comerciales como el de seguridad de red de Shell (SSH, *Secure Shell*), seguridad de transporte en capas (TLS, *Transport Layer Security*), y el protocolo de seguridad de internet (IPSec, *Internet Protocol Security*). (Paar, *et al.*, 2010)

DHKE se basa en que la exponenciación discreta es de simple cálculo, pero su inversa (el logaritmo), es “*imposible*” (o sea, inviable computacionalmente), en grupos particularmente seleccionados, siendo esta la función unidireccional en la que se basa este algoritmo. Teniendo

en cuenta la función $\alpha^k \bmod p = \beta$ (dentro de un grupo cíclico \mathbb{Z}_p^* , con el generador α y siendo p un número primo), conociendo el grupo y el generador α , β es de simple cálculo, pero dado un valor de β determinado, calcular k es inviable. (Wehbe, 2020)

El protocolo funciona de la siguiente forma:

1. Alice y Bob se ponen de acuerdo en un número primo grande p .
2. Alice y Bob se ponen de acuerdo en un número entero $\alpha \in \{2, 3, \dots, p - 2\}$, estos valores suelen ser denominados *parámetros de dominio* y son públicos.
3. Alice elige un valor $k_{priv_A} = a \in \{2, 3, \dots, p - 2\}$, que será la clave privada de Alice.
4. Alice calcula, mediante el *Algoritmo de Exponenciación Rápida*, $k_{pub_A} = \alpha^a \bmod p$, que será la clave pública de Alice, y se la envía a Bob.
5. Bob elige un valor $k_{priv_B} = b \in \{2, 3, \dots, p - 2\}$, que será la clave privada de Bob.
6. Bob calcula, mediante el *Algoritmo de Exponenciación Rápida*, $k_{pub_B} = \alpha^b \bmod p$, que será la clave pública de Bob, y se la envía a Alice.
7. Alice recibe k_{pub_B} y calcula, mediante el *Algoritmo de Exponenciación Rápida*, $k = k_{pub_B}^{k_{priv_A}} = (\alpha^b)^a \bmod p$.
8. Bob recibe k_{pub_A} y calcula, mediante el *Algoritmo de Exponenciación Rápida*, $k = k_{pub_A}^{k_{priv_B}} = (\alpha^a)^b \bmod p$.

(Wehbe, 2020)

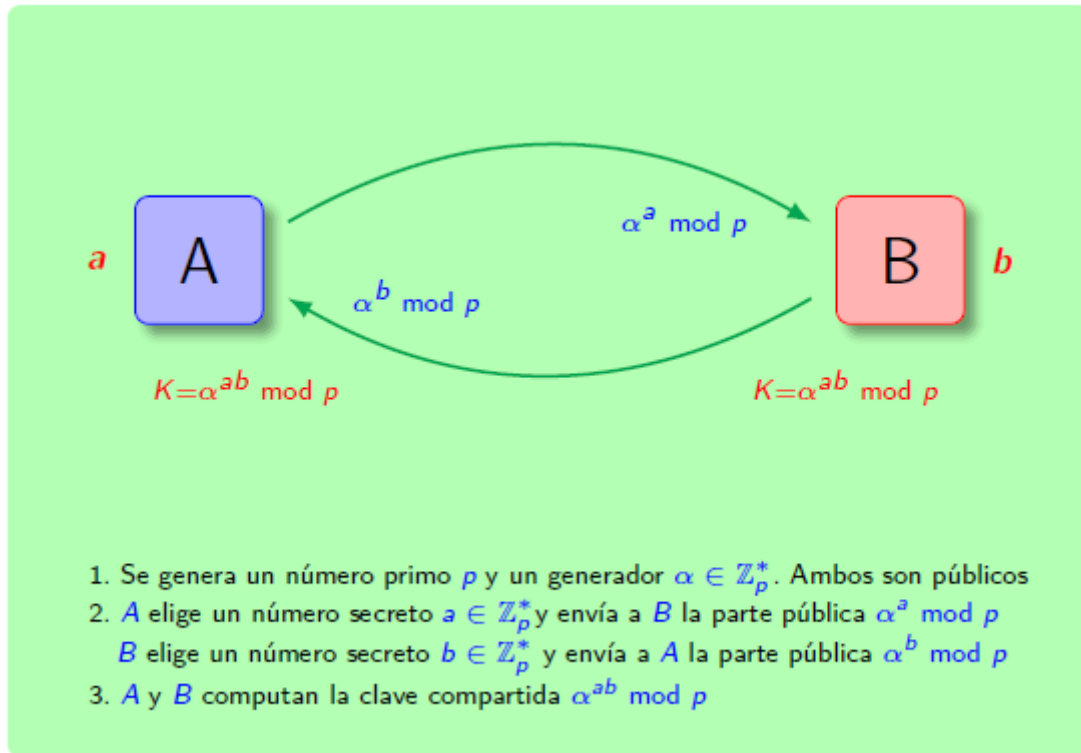


Figura 15: el protocolo de Diffie-Hellman (Wehbe, 2020).

La prueba de funcionamiento resulta de que Alice computa:

$$k_1 = k_{pub_B}^{k_{priv_A}} = (\alpha^b)^a \text{ mod } p \tag{44}$$

por potencia de potencia

$$(\alpha^b)^a \text{ mod } p = \alpha^{b \cdot a} \text{ mod } p \tag{45}$$

mientras que Bob computa:

$$k_2 = k_{pub_A}^{k_{priv_B}} = (\alpha^a)^b \text{ mod } p = \alpha^{a \cdot b} \text{ mod } p \tag{46}$$

Pero resulta que, por conmutatividad del producto:

$$k_1 = k_2 = k \tag{47}$$

k resulta ser el secreto compartido que puede ser utilizado como clave simétrica entre las dos partes (a veces la clave simétrica utilizada resulta de aplicar una *función hash* a dicho k). (Paar, et al., 2010)

Para el protocolo de intercambio de claves de Diffie-Hellman se realizó una implementación liviana, que se comentará más adelante.

4.2.5. ElGamal

Otro esquema parte de la misma familia, es el esquema de encriptación de ElGamal, propuesto por Taher ElGamal en 1985 (ElGamal, 1985). El mismo puede verse como una extensión del protocolo de DHKE, así es que su seguridad se basa en la intratabilidad del problema del logaritmo discreto. La relación entre ambos protocolos puede verse de la siguiente forma: considerando los interlocutores, ambos primero realizan un intercambio de claves Diffie-Hellman consiguiendo una clave compartida (denominada en este caso como clave de enmascarar o de sesión). La diferencia con el protocolo de Diffie-Hellman resulta en que, posteriormente, el emisor utiliza esta clave como una máscara multiplicativa para encriptar el texto plano. (Paar, *et al.*, 2010)

Suponiendo que Bob quiere enviar un mensaje a Alice, el esquema de encriptación de ElGamal funciona de la siguiente forma:

1. Generación de la clave:
 - a. Alice elige aleatoriamente un número primo grande p , y un generador α del grupo \mathbb{Z}_p^* .
 - b. Alice elige aleatoriamente un entero a tal que $1 \leq a \leq p - 2$.
 - c. Alice calcula $\beta = \alpha^a \bmod p$, mediante el *Algoritmo de Exponenciación Rápida*.
 - d. Se define a (p, α, β) como la clave pública de Alice, y a a como su clave privada.
2. Distribución de la clave:
 - a. Alice transmite mediante el canal inseguro su clave pública (p, α, β) a Bob, y mantiene su clave privada a , que nunca es transmitida.
3. Encriptación:
 - a. Bob obtiene la clave pública de Alice (p, α, β) .
 - b. Bob elige aleatoriamente un número entero k tal que $1 \leq k \leq p - 2$.
 - c. Bob calcula, mediante el *Algoritmo de Exponenciación Rápida*, $\gamma = \alpha^k \bmod p$, como clave efímera.
 - d. Bob calcula, mediante el *Algoritmo de Exponenciación Rápida*, $\delta = m \cdot \beta^k \bmod p$, siendo m el mensaje en texto plano que quiere enviar a Alice, y denominándose a $\beta^k \bmod p$ como la clave de enmascarar.

- e. Bob transmite el texto cifrado $c = (\gamma, \delta)$ a Alice (por el canal inseguro).
- 4. Descriptación:
 - a. Alice recupera la clave de enmascarar calculando, mediante el *Algoritmo de Exponenciación Rápida*, $b = \gamma^a \text{ mod } p$, utilizando su clave privada a .
 - b. Alice recupera el mensaje m calculando $m = b^{-1} \cdot \delta \text{ mod } p$, mediante el *Algoritmo Extendido de Euclides*.

(Menezes, et al., 1996) (Paar, et al., 2010)

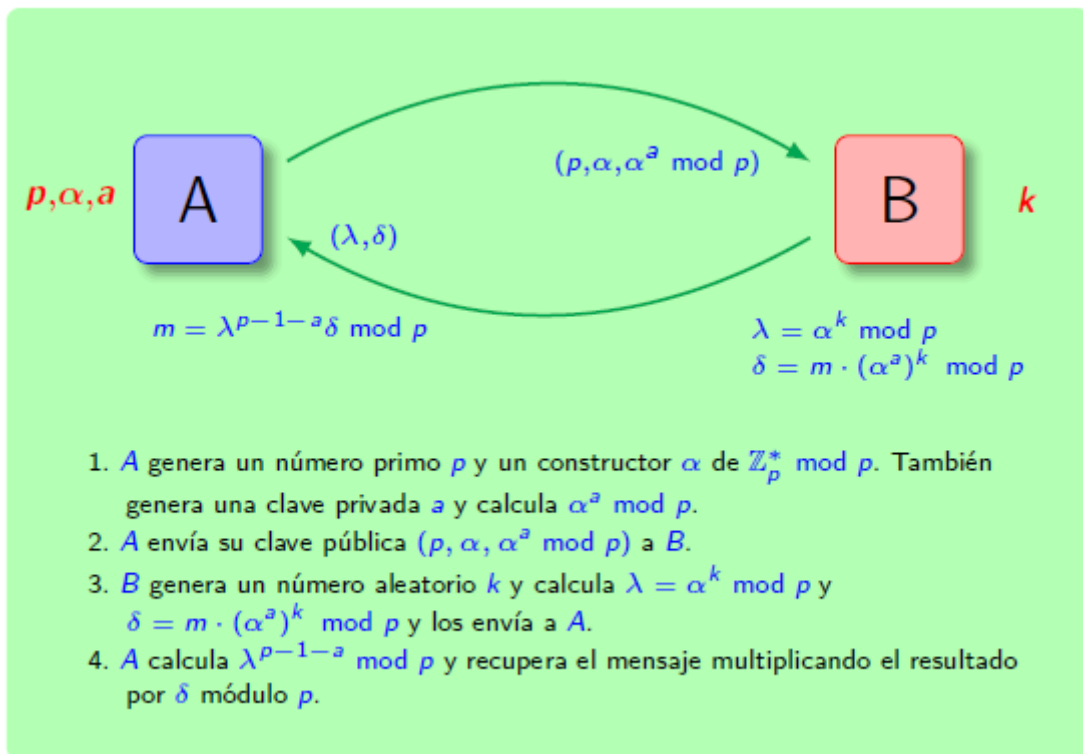


Figura 16: el protocolo de ElGamal (Wehbe, 2020).

La prueba de funcionamiento se puede apreciar claramente de la siguiente forma. Lo que se quiere demostrar es que:

$$b^{-1} \cdot \delta = m \text{ (mod } p) \tag{48}$$

Reemplazando b y δ por definición:

$$b^{-1} \cdot \delta = (\gamma^a)^{-1} \cdot \delta = (\gamma^a)^{-1} \cdot (m \cdot \beta^k) \text{ (mod } p) \tag{49}$$

asimismo, a γ y β , también por definición:

$$(\gamma^a)^{-1} \cdot (m \cdot \beta^k) = ((\alpha^k)^a)^{-1} \cdot (m \cdot \beta^k) = ((\alpha^k)^a)^{-1} \cdot (m \cdot (\alpha^a)^k) \text{ (mod } p) \tag{50}$$

Y finalmente, por potencia de potencia, producto de potencias de igual base, asociatividad y conmutatividad del producto, se obtiene:

$$((\alpha^k)^a)^{-1} \cdot (m \cdot (\alpha^a)^k) = \alpha^{-a \cdot k} \cdot m \cdot \alpha^{a \cdot k} = \alpha^0 \cdot m = m \pmod{p} \quad (51)$$

ElGamal es un esquema de encriptación probabilístico, con lo cual encriptar dos mensajes idénticos no implican dos textos cifrados iguales. (Paar, *et al.*, 2010) La desventaja de ElGamal reside en la expansión del mensaje por un factor de 2, el texto cifrado es dos veces mas largo que el texto plano correspondiente. (Menezes, *et al.*, 1996)

Para el esquema de encriptación de ElGamal se realizó una implementación liviana, que será explicada más adelante.

4.2.6. ECC: *Elliptic Curve Cryptography*

El protocolo de Diffie-Hellman, y aquellos basados en él, pueden desarrollarse en cualquier grupo en el cual el problema del logaritmo discreto resulta “*imposible*” de resolver (o sea, inviable computacionalmente), pero la exponenciación es eficiente. Otro grupo, además del visto anteriormente, que cumple dichas características, es el grupo de puntos definido por una curva elíptica en un campo finito. (Menezes, *et al.*, 1996) Y esto nos lleva a la tercera familia de algoritmos asimétricos: la Criptografía de Curvas Elípticas (ECC, *Elliptic Curve Cryptography*).

Desde su descubrimiento en 1985 por Neal Koblitz (Koblitz, 1987) y Victor Miller (Miller, 1986), la criptografía de las curvas elípticas ha sido estudiada ampliamente en la industria y la academia desde diferentes perspectivas. La mayor ventaja de ECC es el tamaño de sus claves, que es mucho más pequeño, lo cual requiere menos memoria y menor costo computacional de operaciones aritméticas, en comparación con otros sistemas criptográficos. Esto ha hecho que sean la elección ideal de implementación de criptografía asimétrica en dispositivos de recursos limitados, como dispositivos IoT, o sensores WSN, entre otros. Cabe aclarar que las operaciones de RSA que involucran claves públicas pequeñas, son más veloces que las operaciones con ECC. (Lara-Nino, *et al.*, 2018) (Paar, *et al.*, 2010)

ECC provee el mismo nivel de seguridad que las otras dos familias de algoritmos, con operadores menores (160-256 bits, contra 1024-2072 bits), con los cual se evidencian ventajas en performance (menor cómputo y ancho de banda). ECC se basa en la generalización del problema del logaritmo discreto, con lo cual protocolos basados en él, como el de intercambio

de claves de Diffie-Hellman, se pueden adaptar utilizando curvas elípticas. En ECC se utiliza otro grupo cíclico diferente al usado en el protocolo Diffie-Hellman convencional (recordando que, para su aplicación a la criptografía, debe existir allí una operación unidireccional). (Paar, *et al.*, 2010)

Una curva elíptica es un tipo especial de ecuación polinomial (una *polinomial* es una ecuación con potencias de x y potencias de y , y sus respectivos coeficientes), que se utiliza para diversas aplicaciones matemáticas. Se toma la acepción de *curva* como todo punto tal que responde a la solución de la ecuación que la representa. ECC utiliza curvas definidas en un campo finito, cuyos puntos responden a la ecuación de Weierstrass

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{52}$$

Se define a la curva elíptica sobre el campo \mathbb{Z}_p (con $q = p^m$ y $m = 1$), más conocida como *curva prima*, a aquella que responde a la ecuación de Weierstrass simplificada

$$E_p: y^2 = x^3 + ax + b \text{ mod } p \tag{53}$$

tal que p es un número primo con $p > 3$, $a, b \in \mathbb{Z}_p$, $4a^3 + 27b^2 \neq 0$. (Lara-Nino, *et al.*, 2018) (Paar, *et al.*, 2010)

Se puede ver a continuación una curva elíptica sobre los números reales, $y^2 = x^3 - 3x + 3$, que sirve de ejemplo, para dilucidar la forma de la misma. Como se puede apreciar claramente, las curvas elípticas no son elipses (el nombre viene de que juegan un papel importante al determinar la circunferencia de elipses). (Paar, *et al.*, 2010)

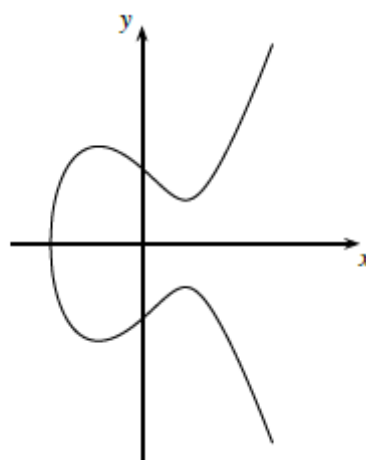


Figura 17: curva elíptica sobre los reales $y^2 = x^3 - 3x + 3$ (Paar, *et al.*, 2010).

Para ECC, como se utilizan curvas finitas, los siguientes ejemplos son más esclarecedores, cuyos gráficos, como se puede apreciar, son puntos en el plano (y no un continuo). En ambos se ve representada la curva $y^2 = x^3 - 7x + 10 \pmod p$, con $p = 19$ a la izquierda y $p = 97$ a la derecha. (Lara-Nino, *et al.*, 2018)

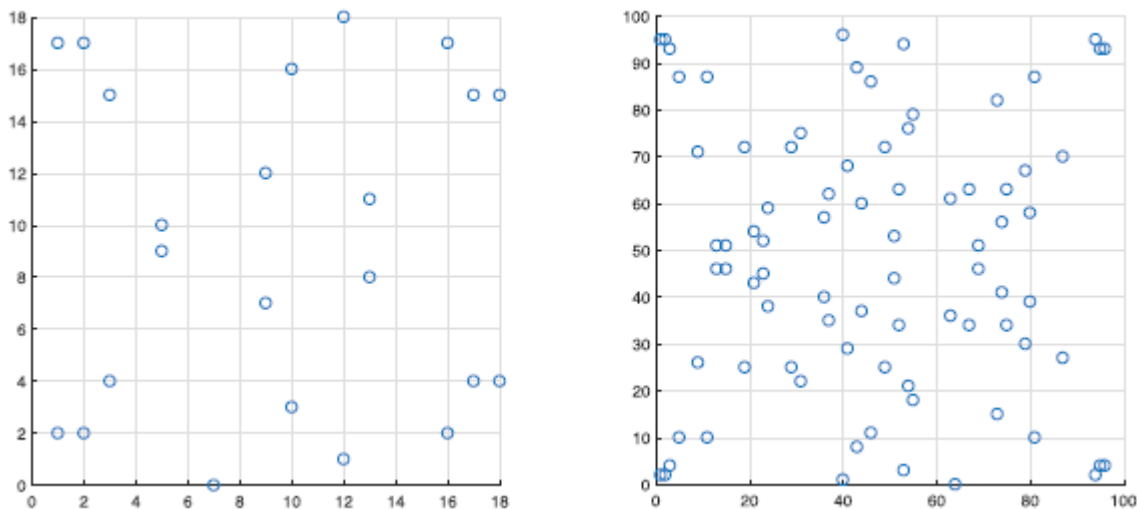


Figura 18: curva prima $y^2 = x^3 - 7x + 10 \pmod p$, con $p = 19$ (izquierda) y $p = 97$ (derecha) (Lara-Nino, *et al.*, 2018).

Llevando la curva elíptica al plano criptográfico, se opera con aritmética modular en un campo finito (se utilizan los *campos de Galois* $GF(p)$, con p número primo, como en DHKE). Lo que permite la aplicación del problema de logaritmo discreto, es que mediante ciertas restricciones y con sus operaciones, se logra formar un grupo cíclico (con generador o primitivo P). Esto último es de suma importancia dado que se conoce cómo construir un sistema criptográfico desde los grupos cíclicos (por DHKE). Y así a continuación se presenta el Problema del Logaritmo Discreto para Curvas Elípticas (ECDLP, *Elliptic Curved Discrete Logarithm Problem*). Sea la siguiente operación definida dentro las curvas elípticas:

$$\underbrace{P + P + \dots + P}_{d \text{ veces}} = d \cdot P = T \tag{54}$$

(con P primitivo y T cualquier elemento perteneciente a la curva), resolver $d \cdot P$, obteniendo T , es de simple cálculo, pero de forma inversa, o sea conociendo P y T obtener d , resulta inviable (se define así la función unidireccional). Aquí d resulta ser el secreto, o sea la clave privada, P un parámetro público, y $T = (x_T, y_T)$, la clave pública. (Wehbe, 2020) (Paar, *et al.*, 2010)

La clave privada d es un número entero, con el que se genera la clave pública $T = (x_T, y_T)$ que es un punto de la curva elíptica. Notar que estos dos valores de la clave pública no son independientes entre sí (con uno se puede determinar el otro a través de la ecuación de la curva elíptica). Es por esto por lo que se utiliza uno de ellos como clave simétrica. En la práctica a la coordenada x se le suele aplicar una *función hash* y luego así se la utiliza como clave simétrica. (Wehbe, 2020) (Paar, *et al.*, 2010)

Nota: se utiliza el símbolo “+”, que se elige arbitrariamente como denotación de la operación del grupo. Pero si se hubiera seleccionado una notación de multiplicación como “ \cdot ”, el ECDLP tendría la forma $P^d = T$, y sería más consistente con el problema del logaritmo discreto convencional en \mathbb{Z}_p^* . (Paar, *et al.*, 2010)

Yendo a los números reales, hay una interesante interpretación geométrica del ECDLP. Dado un punto inicial P , se computa $2 \cdot P, 3 \cdot P, \dots, d \cdot P = T$, efectivamente saltando atrás y adelante en la curva elíptica. Se publica el punto P (que es un parámetro público) y el punto final T (que es la clave pública). Para lograr romper el sistema criptográfico, un atacante debe descifrar cuán a menudo se *salta* por la curva elíptica, puesto que este número de saltos es el secreto d , o sea, la clave privada. (Paar, *et al.*, 2010)

Si las curvas elípticas son elegidas cuidadosamente, los mejores ataques contra el problema del logaritmo discreto para curvas elípticas son considerados más débiles que los provocados a los del problema del logaritmo discreto convencional (de DHKE) y del problema de factorización (de RSA). (Paar, *et al.*, 2010)

No es simple encontrar un grupo con las propiedades requeridas para que el problema del logaritmo discreto para curvas elípticas resulte difícil de resolver (y, por ende, tenga una aplicación a la criptografía); es por esto por lo que las curvas elípticas que se conoce que son seguras, se las reporta en la literatura y se las incluye en estándares. La criptografía asimétrica basada en curvas elípticas generalmente usa estas estructuras estandarizadas, que son seguras, pero que no se las pensó para utilizarlas en ambientes restringidos. En los últimos años, las definiciones de nuevas curvas elípticas buscan no solo alcanzar altos niveles de seguridad, sino también reducir costos operacionales y disminuir los recursos de hardware requeridos, para lograr eficiencia computacional. Estas nuevas curvas elípticas, que quedan fuera del alcance de los estándares, son atractivas para los dispositivos pequeños inteligentes. (Lara-Nino, *et al.*, 2018)

Ahora bien, para especificar el sistema criptográfico, se necesita definir primero el grupo con el cual se va a operar. El conjunto de puntos de la curva, con el punto en el infinito σ forman un grupo, cuya operación es la ley de adición. Juntamente con la regla de adición, se forma un grupo abeliano con el punto σ que sirve de elemento de identidad. Para implementar sistemas criptográficos se utilizan subgrupos cíclicos de aquellos grupos. (Lara-Nino, *et al.*, 2018)

Como se comentó, se busca un grupo cíclico para operar en el cual el problema del logaritmo discreto sea complejo. Para esto, un grupo candidato es el conjunto de puntos definidos por la curva prima, pero para esto se debe definir la operación del grupo (que denominamos adición y se realiza a continuación), la cual debe cumplir con las propiedades de grupo. Para esto se agrega el punto σ como elemento de identidad, y se define así el opuesto de un punto (que se verá más adelante). Se demuestra que los puntos en una curva elíptica juntamente con el punto σ poseen subgrupos cíclicos, bajo ciertas condiciones todos los puntos de una curva elíptica forman un grupo cíclico (Wehbe, 2020). Finalmente se debe determinar el orden del grupo para construir el sistema criptográfico, y para esto se provee un mecanismo para estimarlo (que se verá luego). Con la segunda operación (denominada producto punto, que se define luego), y cumple con las propiedades solicitadas, se obtiene el campo con el que se va a operar (que son los campos primos $GF(p)$). Y así sí se define el ECDLP propiamente dicho.

Se define la regla arbitraria de **adición** (arbitraria porque podría haberse denominado de cualquier otra forma) aplicada a dos puntos de la curva, que devuelve un tercer punto, tal que responde a la operación que se define a continuación. Sea $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, $P + Q = R$, con $R = (x_3, y_3)$, tal que

$$x_3 = s^2 - x_1 - x_2 \text{ mod } p \quad (55)$$

$$y_3 = s \cdot (x_1 - x_3) - y_1 \text{ mod } p \quad (56)$$

donde

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \text{ mod } p; & \text{si } P \neq Q \\ \frac{3 \cdot x_1^2 + a}{2 \cdot y_1} \text{ mod } p; & \text{si } P = Q \end{cases} \quad (57)$$

Se puede apreciar que existen dos casos, el primero resulta si $P \neq Q$, que se lo denomina *adición punto*; y el segundo si $P = Q$, denominado *doble punto* (puesto que en este caso $P + Q = P + P = 2 \cdot P$). (Paar, et al., 2010)

Existen interpretaciones geométricas de estas operaciones para el caso de estar en el conjunto de los números reales, que se pueden ver a continuación.

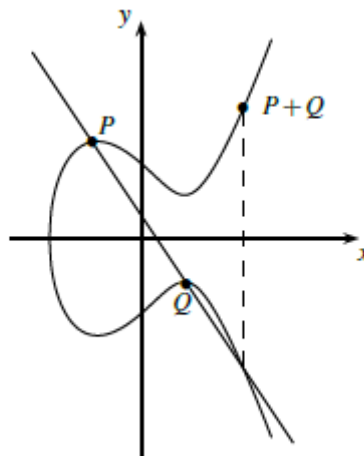


Figura 19: adición punto sobre una curva elíptica en los reales (Paar, et al., 2010).

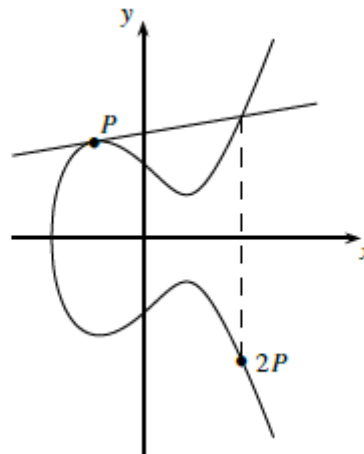


Figura 20: doble punto sobre una curva elíptica en los reales (Paar, et al., 2010).

Se debe definir la identidad (o **elemento neutro**) σ tal que

$$P + \sigma = \sigma + P = P \tag{58}$$

para todo punto P de la curva. Resulta que no existe ningún punto (x, y) que cumple con dicha condición, es por esto por lo que se define un punto abstracto en el infinito como el elemento neutro. Se define así el **inverso** $-P$ de todo elemento del grupo como

$$P + (-P) = \sigma \tag{59}$$

donde

$$-P = (x_P, -y_P) \tag{60}$$

siendo $P = (x_P, y_P)$. Para el caso de *curvas primas*, como $-y_P \bmod p = p - y_P$, se obtiene el punto

$$-P = (x_P, p - y_P) \tag{61}$$

Su interpretación geométrica para el caso de los reales se muestra a continuación. (Paar, *et al.*, 2010)

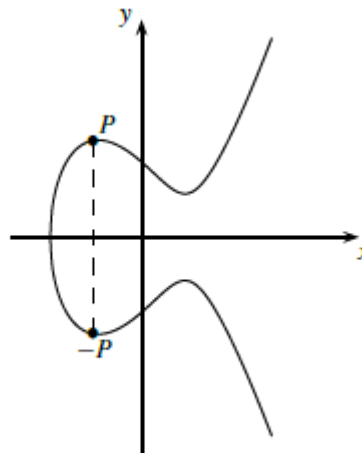


Figura 21: inverso de un punto sobre una curva elíptica en los reales (Paar, *et al.*, 2010).

Como se vio previamente, el **producto punto** se define como

$$\underbrace{P + P + \dots + P}_{d \text{ veces}} = d \cdot P = T \tag{62}$$

con P y T puntos de la curva. Se plantea una interesante implementación del producto punto para mejorar la eficiencia, que resulta análoga al *Algoritmo de Exponenciación Rápida* que utiliza cuadrados y multiplicaciones, pero a diferencia aquí se realizan dobles y adiciones. Como en el *Algoritmo de Exponenciación Rápida*, se utiliza la expresión binaria del coeficiente aquí, la cual es leída de izquierda a derecha, y realiza un doble del punto acumulado en cada

iteración y solo si el bit actual es igual a 1, realiza una adición punto de P al mismo (nuevamente se obvia el primer bit, dado que se presupone igual a 1). Para el cómputo de la adición punto y el doble punto se utiliza el *Algoritmo de la Adición* (que se verá más adelante). A continuación, se puede ver el *Algoritmo de Producto Punto* de una curva elíptica (para el código de la implementación, ver Anexo B). (Paar, *et al.*, 2010)

```

Double-and-Add Algorithm for Point Multiplication
Input: elliptic curve  $E$  together with an elliptic curve point  $P$ 
a scalar  $d = \sum_{i=0}^t d_i 2^i$  with  $d_i \in \{0, 1\}$  and  $d_t = 1$ 
Output:  $T = dP$ 
Initialization:
 $T = P$ 
Algorithm:
1  FOR  $i = t - 1$  DOWNTO 0
1.1   $T = T + T \bmod n$ 
      IF  $d_i = 1$ 
1.2   $T = T + P \bmod n$ 
2  RETURN ( $T$ )

```

Figura 22: Algoritmo de Producto Punto de una curva elíptica (Paar, *et al.*, 2010).

Para el caso de la adición o doble, se realiza la implementación de la especificación antes vista, del *Algoritmo de la Adición*. Como puede apreciarse en la definición de s , existe un numerador y un denominador. La idea es multiplicar el numerador por el inverso multiplicativo del denominador (que se calcula con el *Algoritmo Extendido de Euclides*), siendo cuidadoso de que previamente la fracción sea simplificada. Para esto último se utiliza nuevamente el *Algoritmo Extendido de Euclides* para determinar el *mcd* entre el numerador y el denominador, y en el caso de que no sea igual a 1, se los divide a ambos por él (sino no es necesario realizarlo porque la fracción ya está simplificada). Para el código de la implementación, ver Anexo B.

Los campos más comunes a utilizar para ECC son el campo primo (ya comentado, definido por un número primo p) y el campo binario (definido por una polinomial irreducible $F(x)$). Los estándares corrientes recomiendan campos específicos para diferentes razones de seguridad, la longitud en bits de p o el grado de $F(x)$ debe cumplir con los estándares para alcanzar un nivel de seguridad específico. Para los campos primos hay diferentes familias de curvas elípticas, como random, Koblitz, Montgomery, Edwards, Twisted-Edwards y las más recientes MoTE. Para campos binarios las familias conocidas son random binario, Koblitz

binario, Edwards binario y Hessian. Cada una de estas es un conjunto de construcciones denominadas *familias*, que contienen curvas para campos de diferentes tamaños. Los coeficientes para cada modelo están también definidos para cada caso, así como también los generadores, y son provistos con cada especificación (algunas poseen varios generadores). Estos valores deben ser seleccionados de un modo tal que permitan alcanzar ventajas de implementación del sistema criptográfico según objetivos específicos. (Lara-Nino, *et al.*, 2018)

4.2.6.1. ECDH: *Elliptic Curve Diffie-Hellman*

Como se comentó anteriormente, el protocolo de Diffie-Hellman, puede aplicarse a las curvas elípticas, basado en la seguridad de la intratabilidad del ECDLP. Ahora bien, dicho protocolo de intercambio de claves procede de la siguiente forma:

1. Alice y Bob se ponen de acuerdo en un número primo p .
2. Alice y Bob se ponen de acuerdo en la curva elíptica $E: y^2 \equiv x^3 + a \cdot x + b \pmod{p}$, y en un elemento primitivo P ; a estos valores se los denomina *parámetros de dominio* y se hacen públicos.
3. Alice elige un valor $k_{priv_A} = a \in \{2, 3, \dots, \#E - 1\}$, que será la clave privada de Alice.
4. Alice calcula, mediante el *Algoritmo de Producto Punto*, $k_{pub_A} = a \cdot P = A$, que será la clave pública de Alice, y se la envía a Bob.
5. Bob elige un valor $k_{priv_B} = b \in \{2, 3, \dots, \#E - 1\}$, que será la clave privada de Bob.
6. Bob calcula, mediante el *Algoritmo de Producto Punto*, $k_{pub_B} = b \cdot P = B$, que será la clave pública de Bob, y se la envía a Alice.
7. Alice recibe B y calcula, mediante el *Algoritmo de Producto Punto*, $a \cdot B = T_{AB}$.
8. Bob recibe A y calcula, mediante el *Algoritmo de Producto Punto*, $b \cdot A = T_{AB}$.

(Wehbe, 2020)

La corrección del protocolo es fácil de probar, dado que Alice calcula:

$$T_A = a \cdot B = a \cdot (b \cdot P) \tag{63}$$

y, por su parte, Bob calcula:

$$T_B = b \cdot A = b \cdot (a \cdot P) \tag{64}$$

La operación resulta asociativa (por propiedad de los grupos), con lo cual:

$$T_A = a \cdot (b \cdot P) = a \cdot b \cdot P = b \cdot (A \cdot P) = T_B = T_{AB} \quad (65)$$

(Paar, *et al.*, 2010)

Aclaración: el valor $\#E$, que se aprecia en los pasos 3 y 5 del algoritmo, es el número de puntos de la curva E . Computar todos los puntos de una curva elíptica no es nada sencillo, pero se puede aproximar el número de puntos de la misma gracias al *Teorema de Hasse*. Este teorema dice que dada una curva elíptica E módulo p , el número de puntos de la curva, denotado por $\#E$, está acotado por

$$p + 1 - 2 \cdot \sqrt{p} \leq \#E \leq p + 1 + 2 \cdot \sqrt{p} \quad (66)$$

de lo cual se desprende que el número de puntos de la curva es aproximadamente del orden del número primo p . Con lo cual, si se necesita por ejemplo una curva elíptica con 2^{160} elementos, se debe utilizar un primo de aproximadamente 160 bits. (Paar, *et al.*, 2010) (Wehbe, 2020)

Se realizó una implementación del protocolo de ECDH con cuatro partes. Las dos primeras, de generación de claves, una para el emisor y otra para el receptor, en las cuales se calcula mediante el *Algoritmo de Producto Punto*, un producto punto en cada parte. Luego del intercambio de claves, cada interlocutor realiza su propio cómputo del secreto compartido (en las otras dos partes de la implementación, una para cada uno), con un único cálculo de un producto punto, también mediante el *Algoritmo de Producto Punto*. Se presupone previo acuerdo por parte de ambos interlocutores, de la curva elíptica a utilizar, el elemento primitivo y el número primo. Para el código de la implementación, ver Anexo B.

4.2.6.2. ECEG: *Elliptic Curve ElGamal*

Otro protocolo que puede adaptarse a las curvas elípticas es el esquema de encriptación de ElGamal. El mismo se describe típicamente en el ambiente del grupo multiplicativo \mathbb{Z}_p^* , pero puede generalizarse simplemente para que funcione en cualquier grupo cíclico finito G . A este protocolo se lo denomina *Encriptación ElGamal Generalizada*. La seguridad de este esquema de encriptación se basa en la intratabilidad del problema del logaritmo discreto en el grupo G , el cual debe ser cuidadosamente elegido en eficiencia y seguridad. Uno de los grupos que cumplen con esto, que recibieron más atención, es el grupo de puntos de una curva elíptica en un campo finito. (Menezes, *et al.*, 1996)

En este caso, el protocolo procede de la siguiente forma:

1. Generación de la clave:
 - a. Alice y Bob se ponen de acuerdo en un número primo p .
 - b. Alice y Bob se ponen de acuerdo en la curva elíptica $E: y^2 \equiv x^3 + a \cdot x + b \pmod{p}$, y en un elemento primitivo P ; estos valores se hacen públicos.
 - c. Alice elige un entero a tal que $1 \leq a \leq p - 1$.
 - d. Alice calcula, mediante el *Algoritmo de Producto Punto*, $P_A = a \cdot P$.
 - e. Se define a P_A como la clave pública de Alice, y a a como su clave privada.
2. Distribución de la clave:
 - a. Alice transmite mediante el canal inseguro su clave pública P_A a Bob, y mantiene su clave privada a , que nunca es transmitida.
3. Encriptación:
 - a. Bob obtiene la clave pública de Alice P_A .
 - b. Bob debe primero mapear m , el mensaje en texto plano que quiere enviar a Alice, a un punto en la curva P_m .
 - c. Bob elige un número entero b tal que $1 \leq b \leq p - 1$.
 - d. Bob calcula, mediante el *Algoritmo de Producto Punto*, $P_B = b \cdot P$.
 - e. Bob calcula, mediante el *Algoritmo de Producto Punto* y el *Algoritmo de la Adición*, $P_E = P_m + b \cdot P_A$.
 - f. Bob transmite el texto cifrado $c = (P_B, P_E)$ a Alice (por el canal inseguro).
4. Desencriptación:
 - a. Alice calcula, mediante el *Algoritmo de Producto Punto*, $a \cdot P_B$, utilizando su clave privada a .
 - b. Alice recupera el mensaje mapeado P_m calculando, mediante el *Algoritmo de Producto Punto* y el *Algoritmo de la Adición*, $P_E - a \cdot P_B$ (teniendo en cuenta la definición de elemento inverso explicada anteriormente).

(Lara-Nino, *et al.*, 2018)

La prueba de funcionamiento es muy simple. Lo que se busca demostrar es que:

$$P_E - a \cdot P_B = P_m \tag{67}$$

Reemplazando P_E por definición:

$$P_E - a \cdot P_B = (P_m + b \cdot P_A) - a \cdot P_B = P_m + b \cdot P_A - a \cdot P_B \quad (68)$$

asimismo, P_A y P_B , también por definición:

$$P_m + b \cdot P_A - a \cdot P_B = P_m + b \cdot (a \cdot P) - a \cdot P_B = P_m + b \cdot (a \cdot P) - a \cdot (b \cdot P) \quad (69)$$

Finalmente, por asociatividad y conmutatividad del producto, se obtiene:

$$P_m + b \cdot (a \cdot P) - a \cdot (b \cdot P) = P_m + a \cdot b \cdot P - a \cdot b \cdot P = P_m \quad (70)$$

Se realizó la implementación del protocolo ECEG, siguiendo los siguientes tres pasos. En la generación de claves, por parte del receptor, se realiza un solo producto punto mediante el *Algoritmo de Producto Punto*. El emisor calcula dos productos puntos con el *Algoritmo de Producto Punto*, y una adición punto mediante el *Algoritmo de la Adición*, para encriptar el mensaje en texto plano. El receptor desencripta el texto cifrado a través de un solo producto punto con el *Algoritmo de Producto Punto*, y una adición punto mediante el *Algoritmo de la Adición*; esta última operación como técnicamente es una sustracción, antes de la adición, se determina primero el elemento inverso ($-P$) del punto situado a la derecha del signo “-”. Se presupone previo acuerdo por parte de ambos interlocutores, de la curva elíptica a utilizar, el elemento primitivo y el número primo. Para el código de la implementación, ver Anexo B.

4.2.6.3. ECLC: *Elliptic Curve Lightweight Cryptography*

Se conoce como Criptografía Liviana de Curvas Elípticas (ECLC, *Elliptic Curve Lightweight Cryptography*) a aquellas aplicaciones de la criptografía de curvas elípticas adecuadas para ambientes restringidos. Este tipo de criptografía liviana engloba la definición de los protocolos de las curvas elípticas, los parámetros de dominio, los algoritmos, y las técnicas de implementación, diseñados para proveer seguridad en este tipo de entorno. En la bibliografía no se ha demostrado si el hecho de tornar livianos los algoritmos de criptografía de curvas elípticas se debe a los modelos matemáticos subyacentes, o a decisiones de diseño e implementación. Recordando que la ECC recae en la dificultad de encontrar el logaritmo discreto de curvas elípticas definidas en algún grupo, el orden de dicho grupo está directamente relacionado con la seguridad de la solución, así como también con la complejidad de evaluar las operaciones allí. Mejorando la performance, el tamaño, o la energía consumida por una realización de ECLC, no se debería comprometer la seguridad del sistema. Es por esto por lo que el tamaño del campo subyacente debe definirse de acuerdo con el nivel de seguridad

recomendado del ECC. En algunos escenarios, se puede decidir que la información que está siendo protegida no requiere seguridad a largo plazo. Por ejemplo, existen sensores WSN que por definición están destinados a durar solo algunas semanas o meses. En estos casos, usar curvas elípticas en un grupo que garantizan seguridad por cientos de años no sería requerido. (Lara-Nino, *et al.*, 2018)

5. Resultados

Para medir la complejidad temporal de las implementaciones antes explicadas, se realizó un programa muy simple en Python que llama a cada algoritmo e informa su tiempo de ejecución (para el código de la implementación, ver Anexo B). Para determinar estos tiempos se compara la hora inicial del procesador (un momento antes de empezar el algoritmo), y la hora final (al terminar el mismo), haciendo una simple resta. Como los tiempos son de ordenes muy pequeños (y la respuesta resultaba cero, por redondeo), se realizaron reiteradas corridas de cada algoritmo, en iguales condiciones (podría luego dividirse el número obtenido por la cantidad de veces que se ejecutó el algoritmo para determinar el tiempo medio de corrida, pero no se lo realizó dado que, solo se pretende una comparación de tiempos entre los diferentes algoritmos y, como las condiciones entre ellos son las mismas, el número obtenido resulta igual de útil, y así además se evitan números muy pequeños, generándose una especie de normalización). Se consideran iguales condiciones, y por lo tanto tiempos comparables, dado que se utilizan coeficientes, parámetros y variables de igual orden (tamaño), igual estado del procesador (los diferentes algoritmos se corren uno a continuación del otro, en el mismo programa, o sea, en el mismo momento), e igual cantidad de repeticiones para cada implementación. Los tiempos obtenidos fueron los siguientes:


```

Cantidad de ejecuciones: 13500
-----
Tiempos RSA Liviano
Generación de clave: 0.11269903182983398
Encriptación: 0.10571742057800293
Desencriptación: 0.12366771697998047
-----
Tiempos D-Rabin
Emisor: 0.004986763000488281
Receptor: 0.008976221084594727
-----
Tiempos ECDH
Generación de clave Alice: 0.2593057155609131
Generación de clave Bob: 0.24733805656433105
Secreto compartido Alice: 0.28523683547973633
Secreto compartido Bob: 0.32512974739074707
-----
Tiempos ECEG
Generación de clave: 0.254319429397583
Encriptación: 0.5684788227081299
Desencriptación: 0.32218074798583984

```

Figura 23: tiempos de ejecución de las implementaciones.

Antes de realizar ninguna discusión de los resultados se quiere aclarar que, como ya se comentó previamente, el algoritmo de Rabin liviano propuesto (D-Rabin) debe desestimarse y no tenerse en cuenta, por problemas de seguridad y por no contar con una clave existente. Era de esperar que los tiempos resultaran menores.

Como último comentario, para las implementaciones con curvas elípticas (ECDH y ECEG) se utilizó la curva elíptica $y^2 = x^3 + 2x + 2 \pmod{17}$, con primitivo $P = (5,1)$. Todos los parámetros ingresados fueron del mismo orden de magnitud (10^1).

Como se puede apreciar en los tiempos obtenidos, todos resultaron del mismo orden de magnitud (10^{-1} segundos, las 13.500 ejecuciones), lo cual tiene concordancia con la hipótesis planteada, y se explicará a continuación. La hipótesis decía “*Las implementaciones de algoritmos de sistemas criptográficos con curvas elípticas son más eficientes*”, y la comparación realizada fue en eficiencia temporal con igual orden de parámetros, obteniéndose igual eficiencia en la ejecución. Recordando que, para igual grado de seguridad, los parámetros necesarios son mucho menores en los sistemas criptográficos con curvas elípticas, el hecho de obtener igual complejidad temporal con iguales parámetros implica directamente que, para igual tipo de seguridad, los sistemas criptográficos con curvas elípticas van a precisar menores parámetros que los demás sistemas (en este caso, RSA Liviano). Concluyendo así que, de

realizar una comparación para igual grado de seguridad, los parámetros de los sistemas con curvas elípticas precisarán menor tamaño, lo cual generará menor tiempo de ejecución, demostrándose que son más eficientes. Es por esto por lo que la hipótesis se la considera validada.

Aclaración: como se ve a simple vista, aunque los tiempos resultaron de igual orden, los de RSA Liviano fueron menores que los de ECC, esto ocurre debido a que las claves usadas son muy pequeñas. Resulta que, a claves pequeñas, RSA es más eficiente computacionalmente que ECC (debido a las respectivas operaciones, como ya se había comentado). Esto no cuestiona la validación de la hipótesis puesto que la eficiencia a igual grado de seguridad sigue siendo mucho mejor en ECC (las claves serían más grandes, y las de RSA mucho mayores).

6. Conclusiones

La criptografía data de los inicios de la comunicación. Desde épocas remotas se ha buscado proteger mensajes de terceros intrusos. Con el tiempo lo que se ha ido modificando fue el canal; y con el canal, el medio de asegurarlo. Paredes en las pirámides, papel, telégrafo, radio, televisión, internet. El modo de asegurar el acceso de los intrusos dependió de ellos.

La criptografía busca esconder un mensaje, con la posibilidad de que luego se lo pueda reconstruir. Para esto utiliza funciones matemáticas y datos complementarios que permiten la encriptación y la desencriptación. Estos datos extras son las claves, que se guardan de forma privada (para desencriptar) o que son públicas (para encriptar). Dentro de la criptografía se pueden distinguir dos grandes ramas en función del comportamiento de estas claves. La Criptografía Simétrica, que posee igual clave de encriptación y de desencriptación; y la Asimétrica, que posee claves diferentes. El presente trabajo se basó principalmente en profundizar en este último tipo de sistema de cifrado.

La criptografía ha jugado un papel importante para proteger datos de seguridad nacional, académicos, bancarios, financieros, empresariales, etc.; pero con la difusión de internet, la cantidad de estos datos a resguardar aumentó radicalmente. Internet provocó que el volumen a preservar aumentara en órdenes de magnitud.

Las computadoras conectadas a la red cada vez han sido más, y la llegada de los teléfonos inteligentes generó más y más transporte de datos. En la actualidad no solo los teléfonos inteligentes están conectados a la red, cada vez hay más dispositivos que lo hacen (heladera, lavarropas, aires acondicionados, casas inteligentes, etc.); los cuales toman datos del ambiente y los transmiten. Esta transferencia ocurre para guardar datos estadísticos que luego permiten mejoras, o para realizar acciones (como encender, cambiar parámetros, programar sucesos, interactuar entre ellos, etc.). Esto genera una transmisión de datos permanente, datos que parecen inofensivos, y otros que no. Este tipo de dispositivos están en los hogares, comercios, empresas, bancos, organismos públicos, instituciones gubernamentales, hospitales, etc.; y los datos recolectados y transmitidos resultan de diferente tipo (por ejemplo, los datos clínicos de un hospital, que deben protegerse por ley). También existen otro tipo de datos, los que creemos más inofensivos como la hora en la que se enciende la luz, la hora de ingreso y egreso de un domicilio, la hora de apagar las luces, etc. Este tipo de datos no deben tomarse a la ligera, dado que una simple ingeniería social puede permitir recrear un patrón de comportamiento, y así, por ejemplo, ingresar a un domicilio en el momento en el cual no están

sus habitantes. Y no solo los datos en sí deben protegerse, sino también los propios dispositivos. Tomar acceso de una red privada (ya sea de un domicilio, o de un comercio), por ejemplo, puede ocasionar que se cometan actos ilícitos en nombre del dueño de la red. Es por esto por lo que asegurar los datos y dispositivos, es primordial. Y esto se logra con la Criptografía. Los ataques a los dispositivos en general explotan alguna de sus vulnerabilidades o de la red, con lo cual en base a esto se pueden clasificar para permitir enfocarse en el método de aseguramiento a la vulnerabilidad más explotada, y así lidiar con ella de la mejor forma.

La criptografía tradicional a veces no puede aplicarse, dado que el poder de cómputo de los dispositivos más pequeños es bajo; entonces se deben modificar los algoritmos conocidos, para adaptarse a estos entornos más restringidos. Además, uno de los principales problemas que enfrenta la seguridad de estos dispositivos es la falta de estándares, que facilitarían el modo de protección de estos.

El presente estudio buscaba realizar un compendio de los algoritmos de criptografía liviana, para esto se debían revisar los algoritmos base conocidos y sus fundamentos matemáticos. Se pretendía realizar un camino de aprendizaje de las bases de la criptografía, que resultó enriquecedor. Permitted comprender y mostrar la simpleza y elegancia detrás de todas las operaciones, que pueden resultar a veces abrumadoras (porque *“siempre la forma más simple, es la mejor”*). Se recordó el poder de las matemáticas y se logró reivindicar su uso, mostrando un campo de aplicación que muchos no conocen (en general no se espera que un matemático esté protegiendo datos de un banco y, al fin y al cabo, es lo que ocurre).

La criptografía ha utilizado áreas especiales de la matemática como la teoría de números para generar mecanismos muy prácticos como la encriptación de clave pública y firma digital. Tales usos no fueron siquiera imaginados tiempo atrás. Incluso el famoso matemático Hardy, fue tan lejos como para presumir sobre su falta de utilidad: “tanto Gauss como otros matemáticos no tan importantes pueden ser justificados en pavonearse de la existencia de una sola ciencia, ante todo, la cual debe permanecer muy remota y lejos de las actividades humanas ordinarias; debe mantenerse ligera y limpia”. (Menezes, *et al.*, 1996)

Si se proyecta la criptografía a lo largo de su *“vida”*, se puede entender a la criptografía simétrica como los inicios (ya está muy estudiada, lo que implica así un nivel de tranquilidad alto en la seguridad, dado que ya pasó por muchos intentos de ser quebrantada). Posteriormente se puede ver a la criptografía asimétrica, principalmente RSA, como lo vigente ya conocido (también muy examinada y conocida en profundidad, y con un alto nivel de serenidad sobre su

seguridad). Finalmente, dentro de los sistemas asimétricos, se encuentra la criptografía de curvas elípticas como lo nuevo conocido, por profundizar (esto implica que no posee esa tranquilidad antes mencionada, por el poco tiempo que se ha tenido para ser estudiada, lo cual genera cierta desconfianza). Este tipo de criptografía sigue en crecimiento, por ejemplo, ya existen ciertos estándares, pero no precisamente livianos.

El desarrollo más revolucionario de la historia de la criptografía aparece cuando en 1976 Diffie y Hellman publicaron *Nuevas Direcciones en Criptografía*. Dicho artículo introdujo un concepto revolucionario de la criptografía de clave pública (o criptografía asimétrica) y también proveyó de un nuevo e ingenioso método de intercambio de claves, la seguridad del cual está basada en la intratabilidad del problema del logaritmo discreto. Mas allá de que los autores no tuvieron para ese tiempo una realización práctica del esquema de encriptación de clave pública, la idea fue clara y generó amplio interés y actividad en la comunidad criptográfica. En 1978 Rivest, Shamir y Adleman descubrieron el primer esquema práctico de clave pública, de encriptación y de firma, ahora referido como RSA. El esquema de RSA se basa en otro problema matemático difícil, la intratabilidad de la factorización de enteros grandes. Una de las contribuciones más significativas proveídas por la criptografía asimétrica es la firma digital. En 1991 el primer estándar internacional de firma digital (ISO/IEC 9796) fue adoptado. Se basa en el esquema asimétrico de RSA. La búsqueda de nuevos esquemas asimétricos, mejoras de mecanismos criptográficos existentes, y pruebas de seguridad continúan en crecimiento veloz. Varios estándares e infraestructuras que involucran la criptografía están siendo puestos en acción. Productos seguros están siendo producidos buscando cumplir con las necesidades de seguridad de una sociedad de información intensa. (Menezes, *et al.*, 1996) La firma digital fue uno de los tópicos que hubiera sido interesante indagar en el presente trabajo.

El objetivo amplio de este documento era proveer datos comparativos de los algoritmos, analizarlos, y así ofrecer información que permita tomar decisiones, eligiendo la mejor opción para cada caso particular.

Se concluyó que la criptografía simétrica es siempre mucho más eficiente que la asimétrica, pero siempre está el problema del intercambio de la clave (dado que es la misma). Es por esto por lo que suele recomendarse un enfoque intermedio: la utilización de la criptografía asimétrica para el intercambio de claves (con una periodicidad estipulada), y luego realizar el proceso de cifrado con un sistema simétrico.

En tanto a la eficiencia, se recomienda primero evaluar el nivel de seguridad requerido, dado que a veces se pretende un nivel no tan necesario (no siempre una seguridad máxima es mejor, porque, si el dispositivo tarda un tiempo muy alto en realizar el cifrado, deja de ser funcional). Luego se debería buscar una solución ad-hoc, una adaptación de los métodos existentes, específica para la solución propuesta (se debe siempre priorizar lo particular de cada caso), y buscar artilugios matemáticos que simplifiquen los cálculos y los vuelvan más eficientes. Es imprescindible siempre primero estudiar el caso a fondo (evaluar, analizar y cuantificar), en todo momento con datos reales, y luego tomar la decisión del nivel de seguridad o de la implementación a utilizar.

Para el caso de las implementaciones, se trata siempre de buscar estrategias de agilización. Los algoritmos revisados fueron RSA, Rabin, Diffie-Hellman, ElGamal y Curvas Elípticas (para Diffie-Hellman y ElGamal). Se realizó una implementación liviana de RSA (que necesitó de la implementación del *Algoritmo de Exponenciación Rápida* y del *Algoritmo Extendido de Euclides*), una implementación liviana de Rabin, una de Diffie-Hellman con curvas elípticas y una de ElGamal con curvas elípticas (que requirieron del *Algoritmo de Adición* y el *Algoritmo de Producto Punto* para operar con curvas elípticas). De ello se concluye

👑 en general:

- ✓ elegir parámetros acordes a la seguridad requerida;

👑 de RSA Liviana:

- ✓ elegir el exponente de encriptación con la menor cantidad de 1s (para agilizar la exponenciación rápida),
- ✓ toda exponenciación (en la encriptación y en la desencriptación) se agiliza muchísimo con la exponenciación rápida, y el *Algoritmo Extendido de Euclides* resulta una simple herramienta para calcular inversos multiplicativos y *mcd*,
- ✓ la desencriptación se simplifica con el uso del *Teorema de los Restos Chinos*;

👑 de D-Rabin:

- ✓ no se considera un algoritmo de encriptación, puesto que las operaciones que realiza el receptor pueden ser realizadas por cualquier intruso, con lo cual no existe seguridad alguna. Es por ello por lo que directamente fue desestimado y no se lo tomó como dato para realizar conclusiones;

- 👑 para curvas elípticas en general:
 - ✓ la curva elíptica y el tamaño del campo deben seleccionarse adecuadamente, según la seguridad requerida (existen curvas tabuladas con su/s generador/es),
 - ✓ el *Algoritmo de la Adición* utiliza el *Algoritmo Extendido de Euclides* para determinar el inverso multiplicativo del denominador en la definición del parámetro auxiliar, y también para simplificar la fracción,
 - ✓ el *Algoritmo del Producto Punto*, que se utiliza para agilizar el producto punto, resulta muy similar al de exponenciación rápida,
 - ✓ se debe tener cuidado de no operar con el producto punto con enteros que generen la identidad, o sea el punto en el infinito;

- 👑 de ECDH:
 - ✓ resulta sumamente simple y transparente cada paso de la generación del secreto compartido (es increíble la simpleza y el poder del propio algoritmo de Diffie-Hellman);

- 👑 de ECEG:
 - ✓ en la encriptación y desencriptación se utilizan el *Algoritmo del Producto Punto* y el *Algoritmo de la Adición* (debe tenerse precaución en el último paso de la desencriptación sobre el modo de determinarse el opuesto del punto a la derecha).

Los resultados obtenidos fueron los esperados: a igual tamaño de parámetros, igual orden de complejidad temporal. Conociendo que, para igual grado de seguridad se requieren parámetros mucho más pequeños para las curvas elípticas, se concluye que las curvas elípticas resultan más eficientes que los demás algoritmos livianos asimétricos, resultando así las mismas más eficientes para aplicaciones livianas.

Para el desarrollo del documento se utilizó un proceso que se asemeja al de la *Metodología Agile*. Resultó iterativo incremental, con etapas similares a los *sprint*, cuyos *entregables* resultaron ser las presentaciones preliminares, en las cuales cada una era un producto terminado en sí, que iba agregando valor al entregable anterior. Así se alcanzó el *MVP* (Producto Mínimo Viable), que es el presente documento, que además posee futuros *releases*. En el futuro se pretende continuar ampliando los algoritmos a comparar y realizar el mismo

estudio para las otras herramientas criptográficas (criptografía simétrica liviana y funciones hash livianas).

El resguardo de los datos es de vital importancia, con lo cual protegerlos, debería ser una prioridad. Esto tiene que ser acompañado por un cambio cultural; las personas deben comprender lo necesario de este cuidado y lo peligroso de no hacerlo. Se debe empujar esta revolución de la información para evitar que los datos no sean confiables (por el lado obvio, y por el lado de que son ellos los que pueden perjudicar la seguridad de las personas al quedar expuestos o modificados). Es responsabilidad de los individuos que sí entienden lo primordial de esta protección, educar y explicar al resto sobre herramientas básicas y cuidados mínimos que se pueden aplicar (porque muchas veces el problema es sólo negligencia del usuario por desconocimiento). Porque al fin y al cabo la responsabilidad es del que conoce, es por esto por lo que se necesita asegurar los dispositivos de fábrica, programar de forma segura y sortear vulnerabilidades que se pueden evitar. Y principalmente, capacitar.

Es primordial conocer la seguridad que provee y la que no provee cada método, porque así se pueden tomar decisiones correctas. Sin los datos certeros, eso resulta imposible y genera incertidumbre, que puede evolucionar en una vulnerabilidad (es preferible un nivel aceptable conocido, que nos permite elegir hasta dónde se puede asegurar y hasta dónde el dispositivo queda vulnerable, así como también conocer dónde están dichas vulnerabilidades).

No debe olvidarse que los datos son el futuro, es la moneda de cambio y lo que permite generar ganancias. Si estos datos no son protegidos, todo se vuelve efímero y caótico. El futuro es de los datos y de los que los pueden entender; la decisión que se debe tomar no es si se deben proteger los dispositivos, es qué futuro se busca: uno caótico o uno productivo. Solo hay que elegir, y elegir bien implica tener todos los datos que nos permitan tomar la decisión correcta (datos confidenciales, datos íntegros, datos disponibles y datos auténticos).

7. Bibliografía

Abd Zaid Mustafa and Hassan Soukaena Lightweight Rsa Algorithm Using Three Prime Numbers, *International Journal of Engineering & Technology*, 7 (4.36) pp. 293-295. - 2018.

Aguilar Márquez A., Bravo Vázquez F., Gallego Ruiz H., Cerón Villegas M. and Reyes Figueroa R. Cálculo diferencial, Pearson Educación. - 2016.

Bhardwaj Isha, Kumar Ajay and Bansal Manu A Review on Lightweight Cryptography Algorithms for Data Security and Authentication in IoTs, 4th International Conference on "Signal Processing, Computing and Control", ISPCC, IEEE. - 2017.

Biryukov Alex and Perrin Léo-Paul State of the Art in Lightweight Symmetric Cryptography, *IACR Cryptology ePrint Archive 2017:511*. - 2017.

Chen Haiguang, Wu Huafeng, Zhou Xi and Gao Chuanshan Agent-based Trust Model in Wireless Sensor Networks, *Proc. Int. Conf. Netw. Archit. Storage (NAS)*, pp. 359-364. - 2008.

Daemen Joan and Rijmen Vincent AES Proposal: Rijndael, National Institute of Standards and Technology. - 2003.

Diffie and Hellman New Directions in Cryptography, *IEEE Tans. on Information Theory* 22 (6), pp. 644-654. - 1976.

Dutta Indira, Bayoumi Magdy and Ghosh Baskar Lightweight Cryptography for Internet of Insecure Things: a Survey, *Poc. of the 9th Annual IEEE Computing and Communication Workshop and Conf. (CCWC)*. - 2019.

Eisenbarth Thomas, Paar Christof, Poschmann Axel, Kumar Sandeep and Uhsadel Leif A Survey of Lightweight-Cryptography Implementations, *IEEE Design and Test of Computers*. - 2007.

ElGamal Taher A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469-472. - 1985.

Hammad Baraa Tareq, Jamil Norziana, Rusli Mohd Ezanee and Z'aba Muhammad Reza A survey of Lightweight Cryptographic Hash Function, *International Journal of Scientific & Engineering Research Volume 8, Issue 7*. - 2017.

Kapoor Vivek, Abraham Vivek Sonny and Singh Ramesh Elliptic Curve Cryptography, *ACM Ubiquity*, Volume 9, Issue 20. - 2008.

Kavitha V. P., Rajkumar A. R. F., Logeshwaran S. R. Dhanasekar V. and Ezhil R. Lightweight Secured D-Rabin Cryptosystem for IoT, *International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 6 (Special Issue)*. - 2019.

Kerckhoffs Auguste La cryptographie militaire, *Journal de sciences militaires IX*, pp. 5-38 y pp. 161-191. - 1883.

Koblitz N. Elliptic curve cryptosystems, *Mathematics of Computation*, 48, pp. 203-209. - 1987.

Lara-Nino Carlos Andrés, Diaz-Perez Arturo and Morales-Sandoval Miguel Elliptic curve lightweight cryptography: A survey, *IEEE Access*, vol. 6, pp. 72514-72550. - 2018.

Li Shancang, Xu Li Da and Zhao Shanshan The Internet of Things: a Survey, *Inf Syst. Front.* 17, pp. 243-259. - 2015.

Liberatori Mónica C. Desarrollo de Encriptado AES en FPGA, Tesis presentada para obtener el grado de Magíster en Redes de Datos. - 2006.

Mendez Diego, Papapanagiotou Ioannis and Yang Baijian Internet of Things: Survey on Security and Privacy, Information Security Journal: A Global Perspective, vol. 27, no. 3, pp. 162-182. - 2018.

Menezes Alfred J., van Oorschot Paul C. and Vanstone Scott A. Handbook of Applied Cryptography, CRC Press. - 1996.

Miller V. Uses of elliptic curves in cryptography, Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science, 218, pp. 417-426. - 1986.

Ortega Triguero Jesús J., López Guerrero Miguel Ángel and García del Castillo Crespo Eugenio C. Introducción a la criptografía: historia y actualidad, Ediciones de la Universidad de Castilla. - 2006.

Paar Christof and Pelzl Jan Understanding Cryptography, A Textbook for Students and Practitioners, Springer. - 2010.

Rabin Michael O. Digitalized signature and public key cryptosystem as intractable as factorization, MIT Laboratory for Computer Science Technical Report, MIT/LCS/TR-212. - 1979.

Rivest Ronald L. and van Leeuwen J. (editor) Handbook of Theoretical Computer Science, pp. 719–755, Elsevier Science Publishers. - 1990.

Rivest, Shamir and Adleman A Method for Obtaining Digital Signature and Public-Key Cryptosystems, Comm. of the ACM 21 (2), pp. 120-126. - 1978.

Shannon Claude Communication Theory of Secrecy Systems, Bell System Technical Journal 28, pp. 656-715. - 1949.

Srivastava Arpit Kumar and Mathur Abhinav The Rabin Cryptosystem & analysis in measure of Chinese Remainder Theorem, International Journal of Scientific and Research Publications, Volume 3, Issue 6, ISSN 2250-3153, pp. 493-496. - 2013.

Wehbe Ricardo A. Conceptos fundamentales de seguridad y criptografía, Seminario de Transferencia. - 2020.

Anexo A: Encuesta

La encuesta realizada para el User Research se desarrolló de forma online, y estuvo disponible desde la fecha 24/05/2020 hasta el 03/06/2020, obteniendo 325 respuestas. A continuación, se adjuntan las preguntas y resultados previos a su análisis y limpieza realizados para sacar las conclusiones del desarrollo del presente PFI.

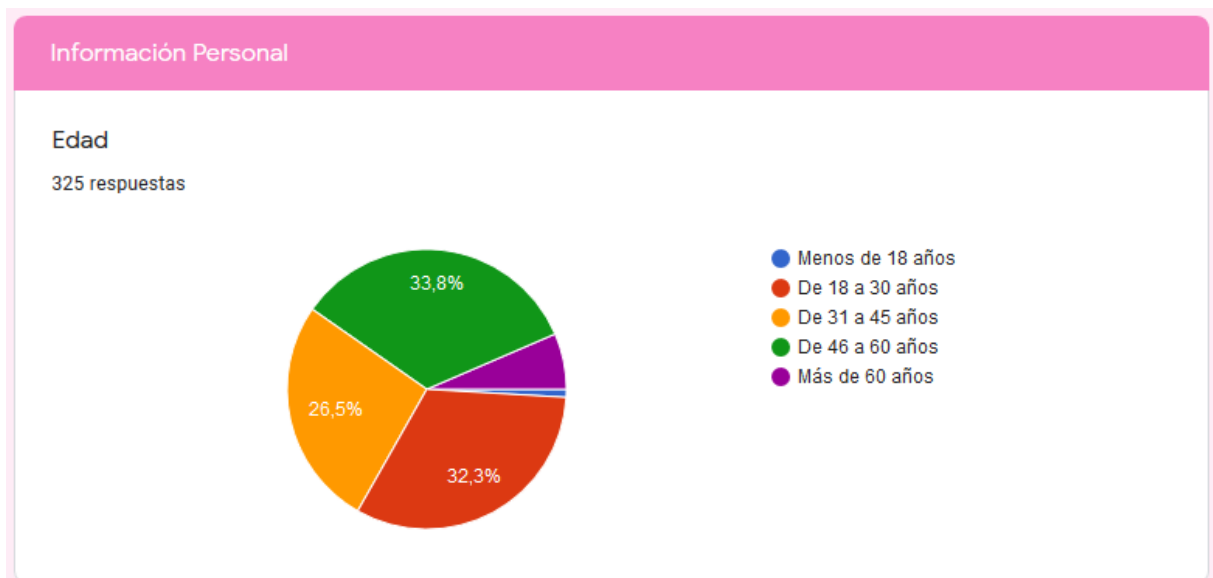
1. Introducción

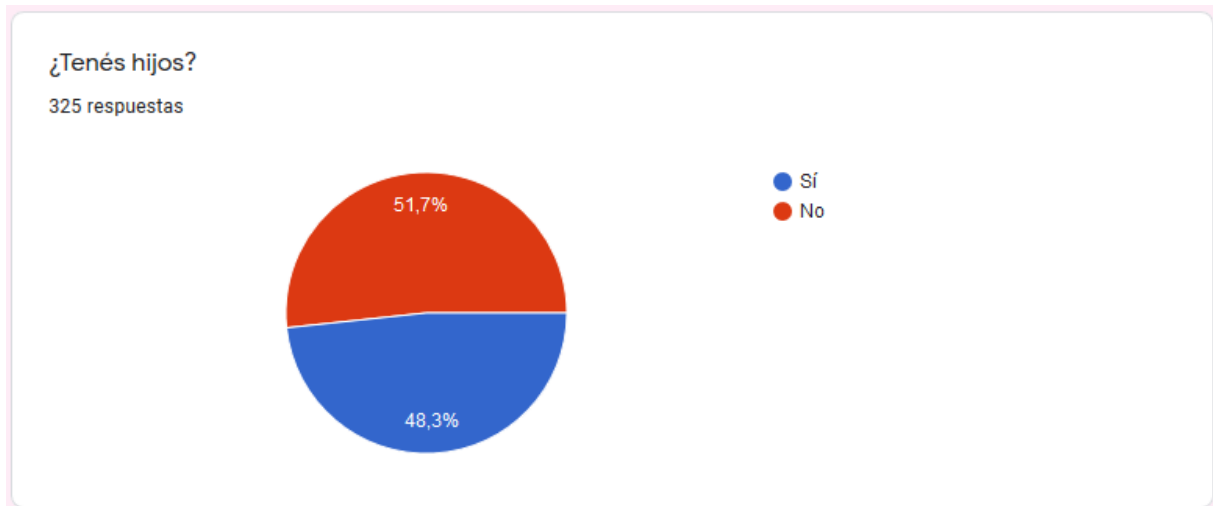
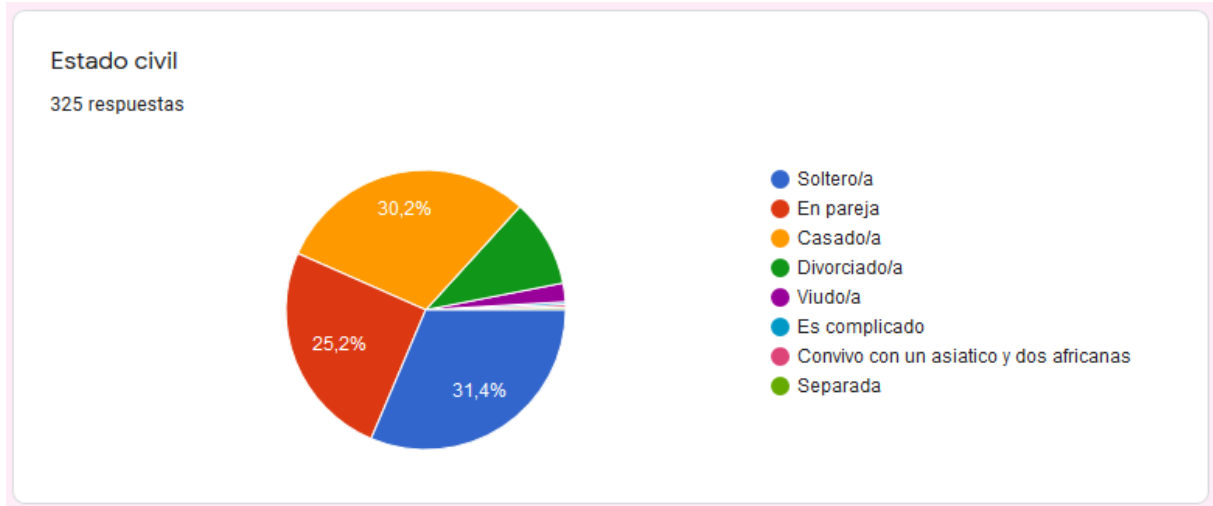
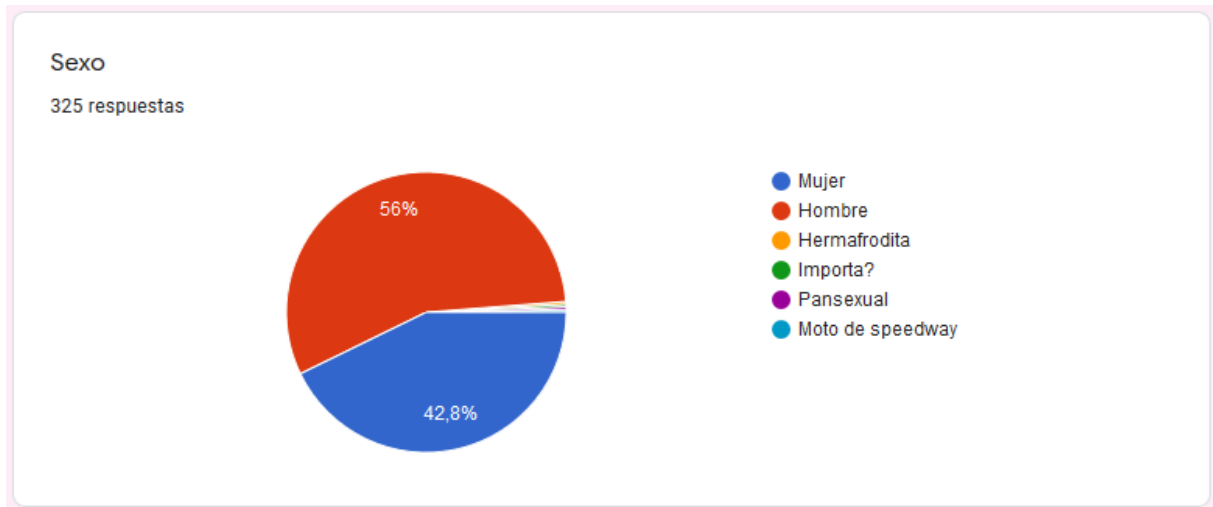
Seguridad en Dispositivos Inteligentes Pequeños

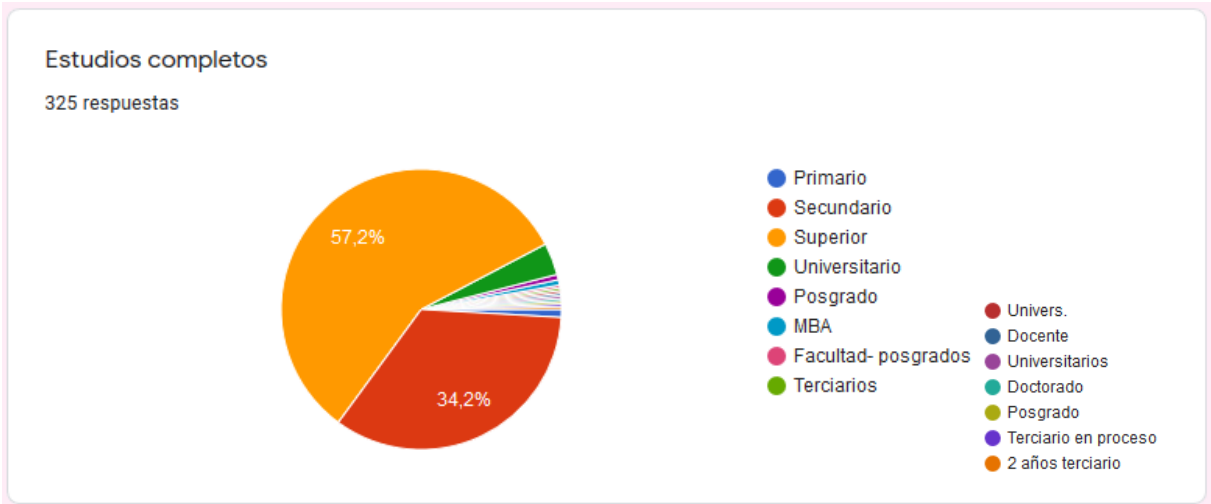
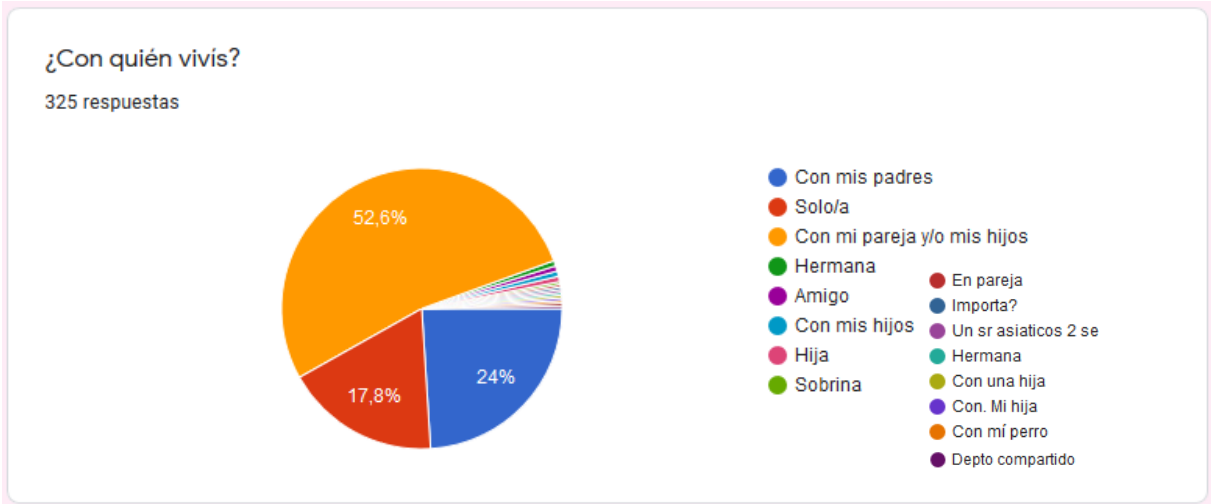
En la actualidad existen dispositivos pequeños de uso cotidiano, a los que se los denomina "inteligentes". Los mismos se conectan a alguna red y proveen información sobre su uso para mejorar el funcionamiento y también para realizar actividades. Un claro ejemplo son los lavarropas con wifi, o las luces automáticas del hogar; componentes de lo que también se conoce como "casas inteligentes".

Un detalle a tener en cuenta es la seguridad de estos dispositivos. La presente encuesta pretende determinar cuánto se conoce de la seguridad de los mismos, en una población masiva.

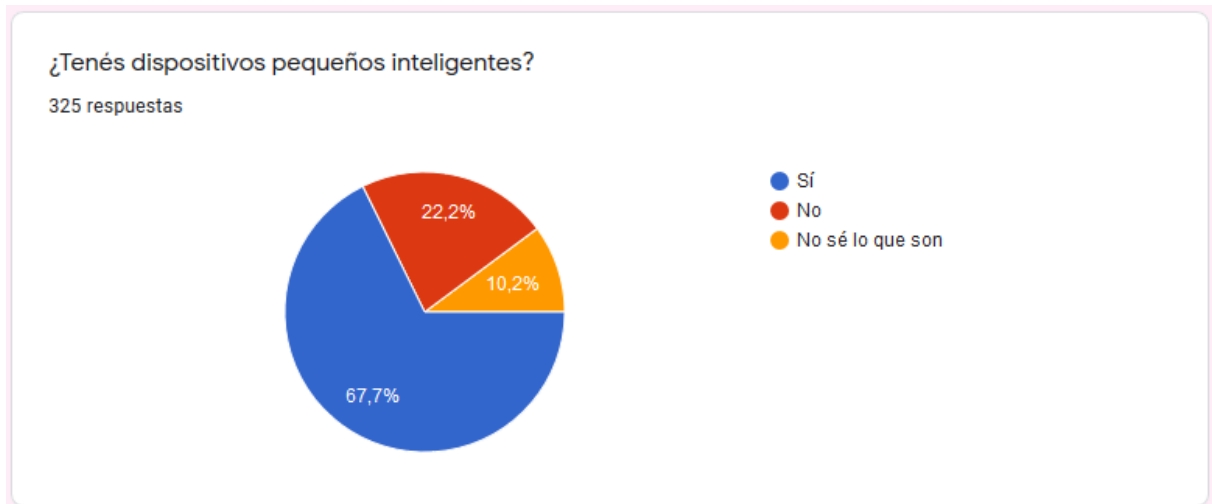
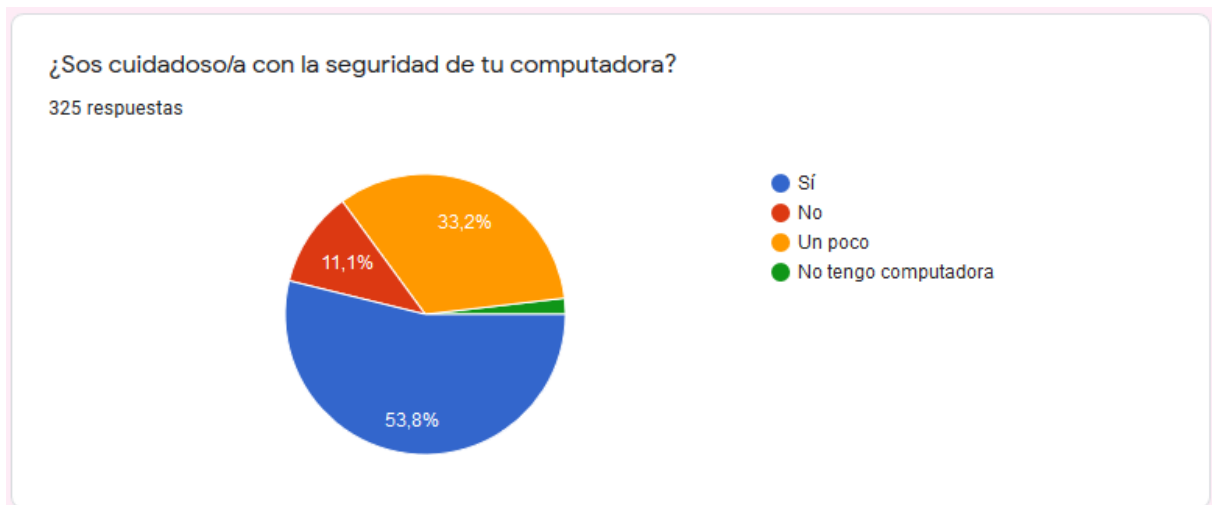
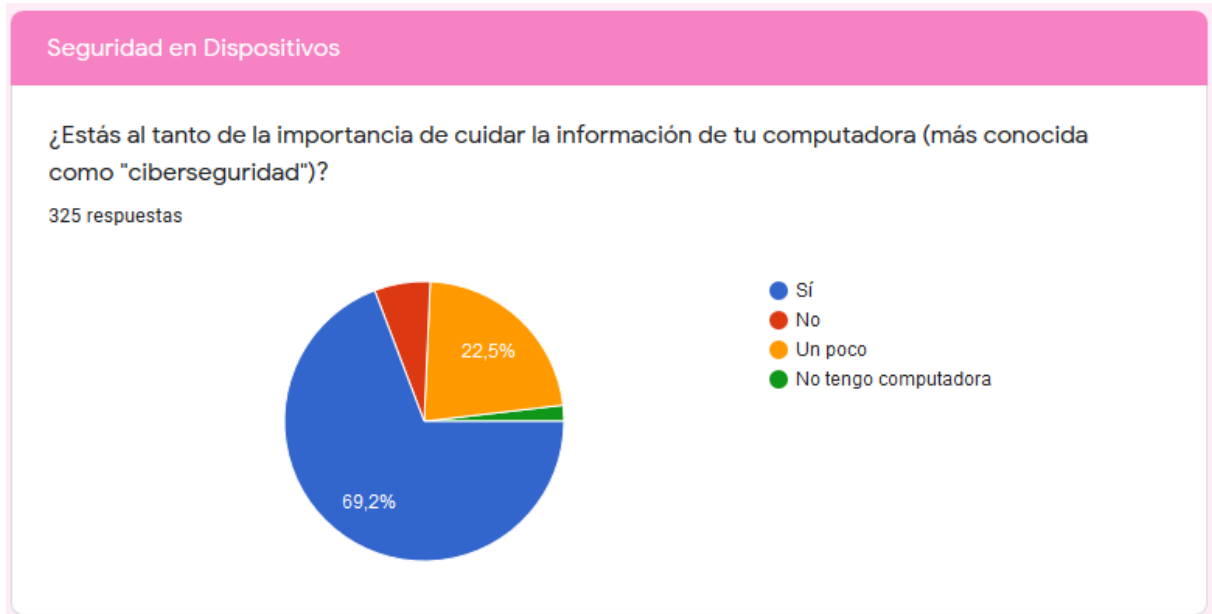
2. Información Personal







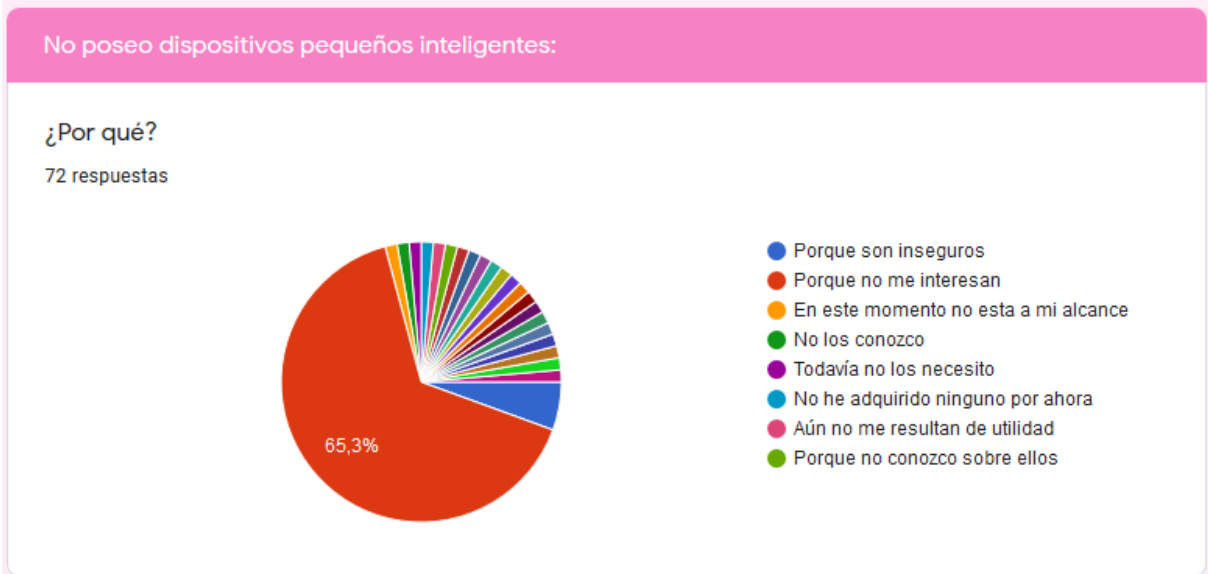
3. Seguridad en Dispositivos



3.1. Eligiendo la 1^{ra} opción: *Poseo dispositivos pequeños inteligentes*



3.2. Eligiendo la 2^{da} opción: *No poseo dispositivos pequeños inteligentes*



- No he adquirido ninguno por el mom...
- PRESUPUESTO
- Porque no sé usarlo
- Me resultan caros para el beneficio que aportan
- Costo
- No tengo por ahora
- No tengo buen manejo de la nueva t...
- No me acostumbro a usarlos
- No he tenido oportunidad de comprar
- No me hace falta
- Porq supongo q son caros y no podrí...
- al momento no tuve necesidad de co...
- No conocer su utilidad
- Aún no compre, pero quisiera.
- Bi me son útiles por el momento.

Aclaración:

Estos dispositivos transfieren información a la red en la que se encuentran conectados. Es importante estar al tanto de que los mismos, al igual que las computadoras, son vulnerables ante ataques externos. Con lo cual es necesario que sean asegurados mediante algún mecanismo que proteja los datos transferidos.

3.3. Eligiendo la 3^{ra} opción: *No sé lo que son los dispositivos pequeños inteligentes*

En la actualidad existen dispositivos pequeños de uso cotidiano, a los que se los denomina "inteligentes". Los mismos se conectan a alguna red y proveen información sobre su uso para mejorar el funcionamiento y también para realizar actividades. Un claro ejemplo son los lavarropas con wifi, o las luces automáticas del hogar; componentes de lo que también se conocen como "casas inteligentes".

Estos dispositivos transfieren información a la red en la que se encuentran conectados. Es importante estar al tanto de que los mismos, al igual que las computadoras, son vulnerables ante ataques externos. Con lo cual es necesario que sean asegurados mediante algún mecanismo que proteja los datos transferidos.

4. Agradecimiento Final

¡Muchas Gracias!

Tu información ha sido de mucha ayuda. Te agradezco por tu tiempo.

La presente encuesta se ha desarrollado para el Proyecto Final de Ingeniería de Elizabeth Barrera, actual estudiante de UADE.

Anexo B: Códigos de Implementación

1. Algoritmos de Apoyo

1.1. Algoritmo de Exponenciación Rápida

```

1. def expRapida(base, exp, n):
2.     ''' Ingresa un entero base, un entero exponente y un entero
3.     n (módulo de congruencia)
4.     Devuelve un entero
5.     '''
6.     # Se expresa el exponente en binario, guardándolo en la lista
7.     e
8.     e = [int(i) for i in bin(exp)[2:]]
9.     # Exponenciación Rápida:
10.    r = base
11.    k = len(e)
12.    for i in range(1,k): # se deja de lado el primer elemento
13.        que se supone 1
14.        r = (r * r) % n
15.        if (e[i] == 1):
16.            r = (r * base) % n
17.    return r
    
```

1.2. Algoritmo Extendido de Euclides

```

1. def euclides(a, b):
2.     ''' Ingresan 2 enteros a y b
3.     Devuelve 3 enteros
4.     '''
5.     s0 = 1
6.     s1 = 0
7.     t0 = 0
8.     t1 = 1
9.     q = a // b
10.    r1 = a - q * b
11.    r0 = b
12.    while (r1 != 0):
13.        aux = s1
14.        s1 = s0 - q * s1
15.        s0 = aux
16.        aux = t1
17.        t1 = t0 - q * t1
18.        t0 = aux
19.        q = r0 // r1
20.        aux = r1
21.        r1 = r0 - q * r1
22.        r0 = aux
23.    'r0 es el mcd(a,b), s1 y t1 son los coeficientes de Bézout,
24.    y permiten calcular inversos multiplicativos (b^-1 mod a = t1
25.    mod a)'
26.    return (r0, s1, t1)
    
```

1.3. Algoritmo de la Adición (*para curvas elípticas*)

```

1. from AlgExtEucl import euclides
2.
3. def sumaCE(a, p, P, Q):
4.     ''' Ingresan 2 puntos de la CE: P = (x1, y1), Q = (x2, y2),
5.     y el coeficiente a y el módulo p de la CE
6.     Devuelve el punto R = (x3, y3) (la suma)
7.     '''
8.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ 
9.     if (P!=Q): # suma
10.         s = Q[1] - P[1] # numerador, s auxiliar
11.         aux = Q[0] - P[0] # denominador
12.         r0, s1, t1 = euclides(s, aux) # se busca mcd(num, den)
13.         if (r0 != 1): # si mcd(num, den) != 1 -> simplifica
14.             s = s // r0
15.             aux = aux // r0
16.             r0, s1, t1 = euclides(p, aux) # se busca el inverso
17.             # multiplicativo del denominador
18.             aux = t1 % p
19.             s = (s * aux) % p
20.         else: # doble
21.             s = 3 * P[0]**2 + a # numerador, s auxiliar
22.             aux = 2 * P[1] # denominador
23.             r0, s1, t1 = euclides(s, aux) # se busca mcd(num, den)
24.             if (r0 != 1): # si mcd(num, den) != 1 -> simplifica
25.                 s = s // r0
26.                 aux = aux // r0
27.             r0, s1, t1 = euclides(p, aux) # se busca el inverso
28.             # multiplicativo del denominador
29.             aux = t1 % p
30.             s = (s * aux) % p
31.             x3 = (s**2 - P[0] - Q[0]) % p
32.             y3 = (s * (P[0] - x3) - P[1]) % p
33.             R = (x3, y3)
34.         return R
    
```

1.4. Algoritmo del Producto Punto (*para curvas elípticas*)

```

1. from AlgExtEucl import euclides
2.
3. def prodPuntoCE(a, p, k, P):
4.     ''' Ingresan un escalar k, un punto P de la CE, y el módulo p
5.     de la CE
6.     Devuelve el punto T (T = k.P)
7.     '''
8.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ 
9.     # Se expresa el coeficiente en binario, guardándolo en la
10.    lista d (sin el primer elemento)
11.    d = [int(i) for i in bin(k)[3:]]
12.    T = P
13.    for i in d:
14.        T = sumaCE(a, p, T, T)
15.        if (i == 1):
16.            T = sumaCE(a, p, T, P)
17.    return T
    
```

2. Algoritmos Livianos

2.1. RSA Liviano

```

1. from AlgExtEucl import euclides
2. from ExpRap import expRapida
3.
4. def rsaLivGenClave():
5.     ''' Alice genera su clave pública y privada
6.     '''
7.     p = 97 # elige Alice
8.     q = 83 # elige Alice
9.     s = 89 # elige Alice
10.    n = p * q * s
11.    fi = (p-1) * (q-1) * (s-1)
12.    e = 95 # elige Alice
13.    r0, s1, t1 = euclides(fi, e) # controlar que mcd(fi, e) = 1
    (con r0)
14.    d = t1 % fi # el inverso multiplicativo de e (mod fi)
15.    r0, s1, t1 = euclides(p-1, e)
16.    dp = t1 % (p-1) # el inverso multiplicativo de e (mod p-1)
17.    r0, s1, t1 = euclides(q-1, e)
18.    dq = t1 % (q-1) # el inverso multiplicativo de e (mod q-1)
19.    r0, s1, t1 = euclides(p, q)
20.    if (p > q):
21.        Q = t1 % p # el inverso multiplicativo de q (mod p)
22.    else: # considero p!=q
23.        Q = s1 % q # el inverso multiplicativo de p (mod q)
24.    cPub = (n, e)
25.    cPriv = (Q, dp, dq, p, q)
26.    return (cPub, cPriv)
27.
28. def rsaLivEncriptacion(cPub):
29.     ''' Bob encripta el mensaje con la clave pública de Alice
30.     '''
31.     # expRapida(base, exp, módulo), cPub = (n, e)
32.     m = 4228 # texto plano que Bob quiere enviar encriptado
33.     return expRapida(m, cPub[1], cPub[0])
34.
35. def rsaLivDesencriptacion(c, cPriv):
36.     ''' Alice desencripta el mensaje con su clave privada
37.     '''
38.     # expRapida(base, exp, módulo), cPriv = (Q, dp, dq, p, q)
39.     ma = expRapida(c, cPriv[1], cPriv[3])
40.     mb = expRapida(c, cPriv[2], cPriv[4])
41.     h = (cPriv[0] * (ma-mb)) % cPriv[3]
42.     return (mb + (h * cPriv[4]))
    
```

2.2. D-Rabin

```

1. def rabinLivEmisor():
2.     ''' Bob encripta el mensaje para Alice
3.     ...
4.     p = 4229 # elige el primo más cercano a m (p >= m)
5.     m = 4228 # texto plano
6.     d = p - m
7.     return (2 * (10 * p + d))
8.
9. def rabinLivReceptor(c):
10.    ''' Alice desencripta el mensaje de Bob
11.    ...
12.    cp = c // 2
13.    a = (cp // 10) % 10 # anteuúltimo dígito
14.    if (a % 2):
15.        p = cp // 10
16.        d = cp % 10
17.    else:
18.        d = (cp % 10) + 10
19.        p = cp // 10 - 1
20.    return (p - d)
    
```

2.3. Diffie-Hellman con Curvas Elípticas

```

1. from ProdPuntoCE import prodPuntoCE
2. p = 17
3. a = 2
4. P = (5, 1)
5.
6. def DHKEgenClaveA(p, a, P):
7.     ''' Alice genera su clave pública y privada
8.     ...
9.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
10.    ka = 10 # clave privada de Alice
11.    kA = prodPuntoCE(a, p, ka, P) # clave pública de Alice
12.    return (ka, kA)
13.
14. def DHKEgenClaveB(p, a, P):
15.     ''' Bob genera su clave pública y privada
16.     ...
17.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
18.    kb = 11 # clave privada de Bob
19.    kB = prodPuntoCE(a, p, kb, P) # clave pública de Bob
20.    return (kb, kB)
21.
22. def DHKEsecCompA(p, a, ka, kB):
23.     ''' Alice calcula el secreto compartido
24.     ...
25.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
26.    T = prodPuntoCE(a, p, ka, kB) # secreto compartido
27.    return T
28.
29. def DHKEsecCompB(p, a, kb, kA):
30.     ''' Bob calcula el secreto compartido
31.     ...
32.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
33.    T = prodPuntoCE(a, p, kb, kA) # secreto compartido
34.    return T
    
```

2.4. ElGamal con Curvas Elípticas

```

1. import ProdPuntoCE
2. p = 17
3. a = 2
4. P = (5, 1)
5.
6. def ECElGGenClave(p, a, P):
7.     ''' Alice genera su clave pública y privada
8.     '''
9.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
10.    ka = 10 # clave privada de Alice
11.    Pa = ProdPuntoCE.prodPuntoCE(a, p, ka, P) # clave pública de
    Alice
12.    return (ka, Pa)
13.
14. def ECElGEncriptacion(p, a, P, Pa):
15.     ''' Bob encripta el mensaje con la clave pública de Alice
16.     '''
17.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
18.     Pm = (0, 6) # texto plano que Bob quiere enviar encriptado
    (mapeado a CE)
19.     kb = 11
20.     Pb = ProdPuntoCE.prodPuntoCE(a, p, kb, P)
21.     Paux = ProdPuntoCE.prodPuntoCE(a, p, kb, Pa)
22.     Pe = ProdPuntoCE.sumaCE(a, p, Pm, Paux)
23.     return (Pb, Pe)
24.
25. def ECElGDesencriptacion(p, a, Pb, Pe, ka):
26.     ''' Alice desencripta el mensaje con su clave privada
27.     '''
28.     # E:  $y^2 = (x^3 + a*x + b) \pmod p$ , P primitivo
29.     Paux = ProdPuntoCE.prodPuntoCE(a, p, ka, Pb)
30.     return ProdPuntoCE.sumaCE(a, p, Pe, (Paux[0], p-Paux[1])) #
    -P = (x, -y) = (x, p-y) (mod p)

```

3. Medición de Tiempos

```

1. import RSALiv
2. import RabinLiv
3. import DHKEecc
4. import ECElGamal
5. from time import time
6.
7. def medirTiemposRSALiv(cant):
8.     print("Tiempos RSA Liviano")
9.
10.    inicio = time()
11.    for i in range(cant):
12.        cPub, cPriv = RSALiv.rsaLivGenClave()
13.    fin = time()
14.    print("Generación de clave:", fin-inicio)
15.
16.    inicio = time()
17.    for i in range(cant):
18.        c = RSALiv.rsaLivEncryptacion(cPub)
19.    fin = time()
20.    print("Encriptación:", fin-inicio)
21.
22.    inicio = time()
23.    for i in range(cant):
24.        m = RSALiv.rsaLivDesencriptacion(c, cPriv)
25.    fin = time()
26.    print("Desencriptación:", fin-inicio)
27.
28. def medirTiemposRabinLiv(cant):
29.     print("Tiempos D-Rabin")
30.
31.     inicio = time()
32.     for i in range(cant):
33.         c = RabinLiv.rabinLivEmisor()
34.     fin = time()
35.     print("Emisor:", fin-inicio)
36.
37.     inicio = time()
38.     for i in range(cant):
39.         m = RabinLiv.rabinLivReceptor(c)
40.     fin = time()
41.     print("Receptor:", fin-inicio)
42.
43. def medirTiemposDHKEecc(cant):
44.     print("Tiempos ECDH")
45.     p = 17
46.     a = 2
47.     P = (5, 1)
48.
49.     inicio = time()
50.     for i in range(cant):
51.         ka, kA = DHKEecc.DHKEgenClaveA(p, a, P)
52.     fin = time()
53.     print("Generación de clave Alice:", fin-inicio)
54.
55.     inicio = time()
56.     for i in range(cant):
57.         kb, kB = DHKEecc.DHKEgenClaveB(p, a, P)
58.     fin = time()
59.     print("Generación de clave Bob:", fin-inicio)
60.
61.     inicio = time()
    
```

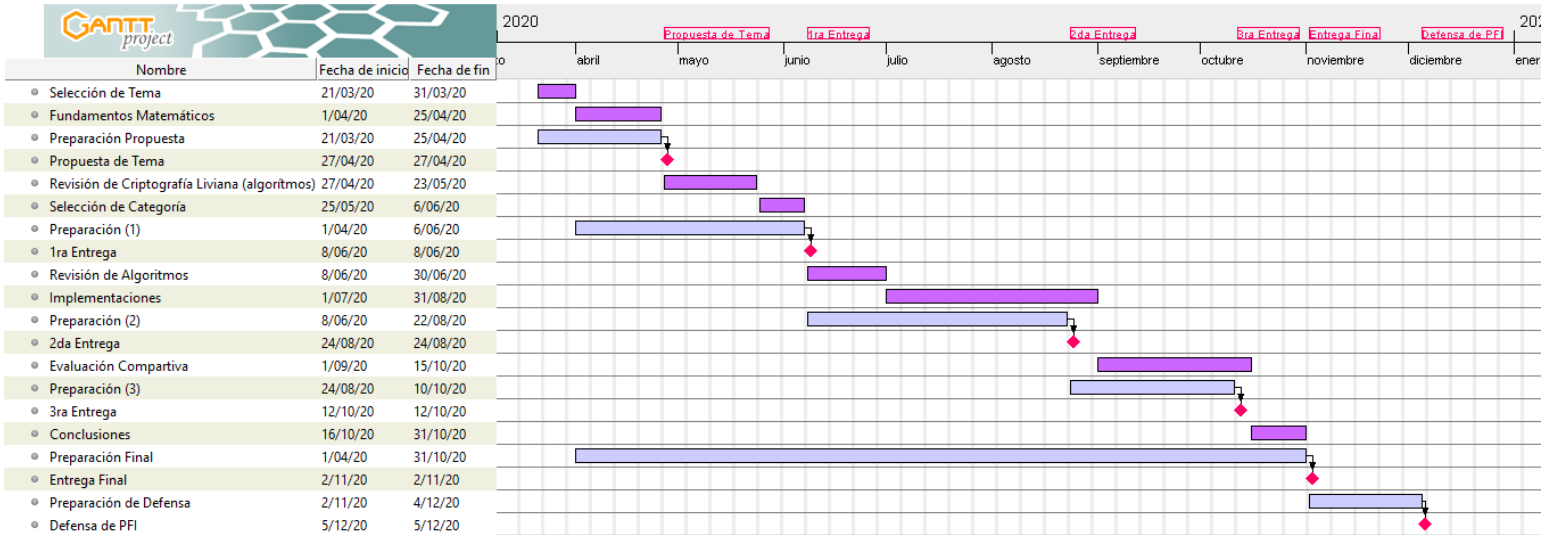
```

62.     for i in range(cant):
63.         Ta = DHKEecc.DHKESecCompA(p, a, ka, kB)
64.         fin = time()
65.         print("Secreto compartido Alice:", fin-inicio)
66.
67.         inicio = time()
68.         for i in range(cant):
69.             Tb = DHKEecc.DHKESecCompB(p, a, kb, kA)
70.             fin = time()
71.             print("Secreto compartido Bob:", fin-inicio)
72.
73. def medirTiemposECElGamal(cant):
74.     print("Tiempos ECEG")
75.     p = 17
76.     a = 2
77.     P = (5, 1)
78.
79.     inicio = time()
80.     for i in range(cant):
81.         ka, Pa = ECElGamal.ECElGGenClave(p, a, P)
82.         fin = time()
83.         print("Generación de clave:", fin-inicio)
84.
85.         inicio = time()
86.         for i in range(cant):
87.             Pb, Pe = ECElGamal.ECElGEncriptacion(p, a, P, Pa)
88.             fin = time()
89.             print("Encriptación:", fin-inicio)
90.
91.         inicio = time()
92.         for i in range(cant):
93.             Pm = ECElGamal.ECElGDesencriptacion(p, a, Pb, Pe, ka)
94.             fin = time()
95.             print("Desencriptación", fin-inicio)

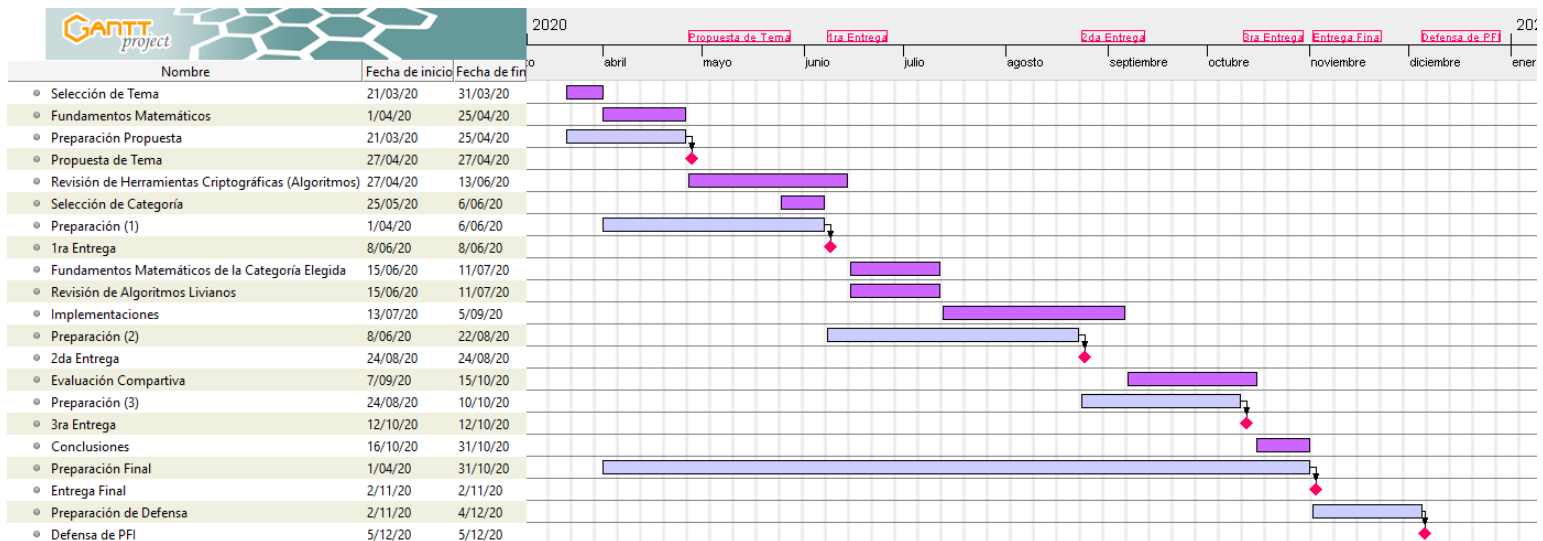
```

Anexo C: Cronograma de Actividades

1. Propuesto



2. Actualizado al 06/06

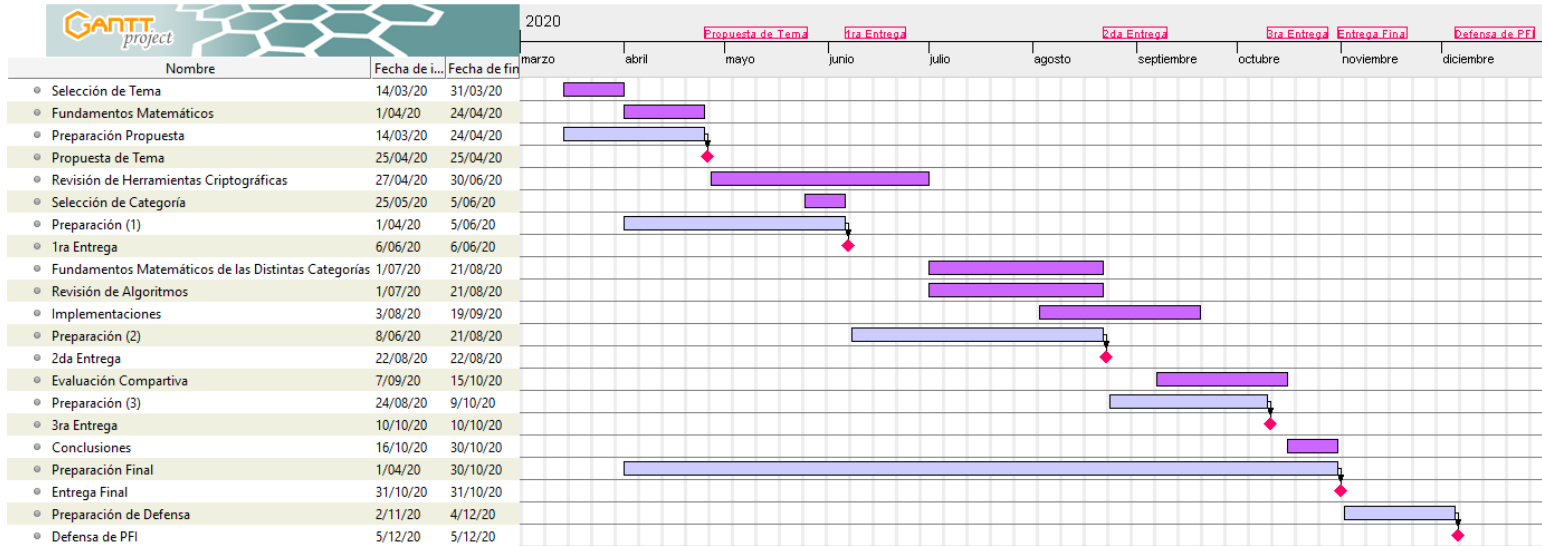


2.1. Comentarios

La tarea “Revisión de Criptografía Liviana (algoritmos)”, que fue renombrada como “Revisión de Herramientas Criptográficas (Algoritmos)” dado que presentaba más semejanza con la misma, se retrasó y produjo la modificación en la fecha de finalización. Esto se ocasionó dado que se revisó funciones hash y criptografía asimétrica, faltando ver los principales algoritmos de criptografía simétrica. Esto implicó un atraso en la tarea “Revisión de Algoritmos” (que se la renombró como “Revisión de Algoritmos Livianos”), tanto en fecha de inicio como de fin. Se creó una nueva tarea paralela con la anterior denominada “Fundamentos

Matemáticos de la Categoría Elegida”. Se retrasó así el inicio y finalización de la tarea “Implementaciones”, pero se previó una reducción de esta. Por último, en la tarea “Evaluación Comparativa” se vio modificada su fecha de inicio, pero se mantuvo la de fin.

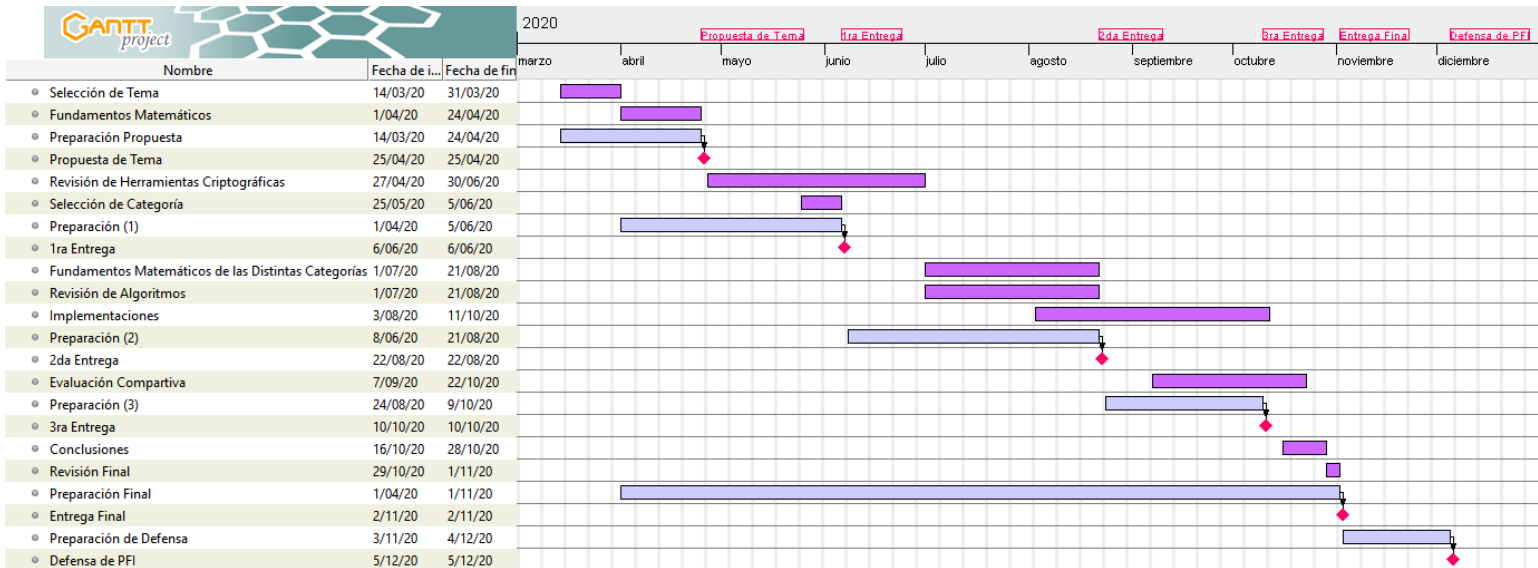
3. Actualizado al 22/08



3.1. Comentarios

La tarea “Revisión de Herramientas Criptográficas (Algoritmos)”, que fue renombrada como “Revisión de Herramientas Criptográficas”, se prolongó y produjo la modificación en la fecha de finalización. Esto se ocasionó dado que faltaba rever parte de las herramientas de criptografía simétrica (que se prolongó más de lo esperado), y se completó lo ya examinado de criptografía asimétrica. Esto implicó un atraso en las tareas “Fundamentos Matemáticos de la Categoría Elegida” (que se la renombró como “Fundamentos Matemáticos de las Distintas Categorías”, puesto que se requirió revisión de la matemática de la criptografía simétrica, además de la asimétrica) y “Revisión de Algoritmos Livianos” (que se la renombró como “Revisión de Algoritmos”, puesto que se requirió rever los algoritmos no livianos, además de los livianos, para comprender su funcionamiento), tanto en fecha de inicio como de fin. Se retrasó así el inicio y finalización de la tarea “Implementaciones”, que no produjo otras modificaciones, dado que se espera comenzar la tarea “Evaluación Comparativa” paralelamente.

4. Actualizado al 10/10



4.1. Comentarios

Se prolongó la tarea “Implementaciones”, con lo cual se retrasó su fecha de finalización, posponiendo así la fecha final de la tarea “Evaluación Comparativa” (aunque su inicio no se vio modificado). Se adelantó la fecha de finalización de la tarea “Conclusiones”, para poder agregar una nueva: “Revisión Final”. Se modificó el fin de la tarea “Preparación Final”, dado que se fijó la fecha del hito “Entrega Final”, cambiando así también la fecha de inicio de la tarea “Preparación de Defensa”.

5. Actualizado al 02/11

5.1. Comentarios

Se mantuvo el último cronograma, cumpliéndose con los tiempos esperados.