ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA EN SISTEMAS E INFORMÁTICA

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS E INFORMÁTICA

TEMA: IMPLEMENTACIÓN DE UN ALGORITMO BASADO EN CRIPTOGRAFÍA NEURONAL Y CRIPTOGRAFÍA BASADA EN ADN PARA GENERACIÓN DE UNA CLAVE PÚBLICA Y ENVÍO DE MENSAJES SEGUROS

AUTOR: SALGUERO DOROKHIN, ÉDGAR LUIS

DIRECTOR: ING. FUERTES DÍAZ, WALTER MARCELO PhD.

SANGOLQUÍ

2019

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERIA EN SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, *"**IMPLEMENTACION DE UN ALGORITMO BASADO EN CRIPTOGRAFIA NEURONAL Y CRIPTOGRAFIA BASADA EN ADN PARA GENERACION DE UNA CLAVE PUBLICA Y ENVIO DE MENSAJES SEGUROS**"* fue realizado por el señor ***Salguero Dorokhin, Édgar Luis*** el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto, cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de las Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 16 de enero del 2019

Ing. Fuertes Díaz, Walter Marcelo PhD.

C.C.: 1707017701

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERIA EN SISTEMAS E INFORMÁTICA

AUTORÍA DE RESPONSABILIDAD

Yo, ***Salguero Dorokhin, Édgar Luis***, declaro que el contenido, ideas y criterios del trabajo de titulación: ***Implementación de un algoritmo basado en criptografía neuronal y criptografía basada en ADN para generación de una clave pública y envío de mensajes seguros*** es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de las Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 16 de enero del 2019

Édgar Luis Salguero Dorokhin

C.C.: 1717262628

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERIA EN SISTEMAS E INFORMÁTICA

AUTORIZACIÓN

Yo, ***Salguero Dorokhin, Édgar Luis*** *autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación:* ***Implementación de un algoritmo basado en criptografía neuronal y criptografía basada en ADN para generación de una clave pública y envío de mensajes seguros*** *en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.*

Sangolquí, 16 de enero del 2019

Édgar Luis Salguero Dorokhin

C.C.: 1717262628

DEDICATORIA

A mi familia, que me ha apoyado durante la realización de este proyecto de investigación. Su amor y paciencia fue el combustible que necesitaba para poder terminarlo, que sin ellos no me hubiese sido posible.

A mi madre Galina, que me supiste entender y que siempre te has preocupado por mí. Tu esfuerzo y perseverancia a pesar de los problemas ha sido lo que me ha impulsado a trabajar duro en este proyecto. A mi hermano Andrés, que siempre estuviste ahí para mí, siempre me has enseñado cosas y seguiré aprendiendo de ti. A mi abuela Sara, que nunca faltó los momentos en que me demostrabas tu cariño. A mi padre Luis, que, a pesar de que sólo estuviste durante el inicio de esta etapa en mi vida, fue tiempo suficiente para aprender de ti lo más importante y cualidades que me guiarían durante todo este trayecto hasta terminarlo.

Gracias a todos ustedes soy la persona que soy ahora. Han sido el complemento perfecto a mi formación académica, y supieron formar mi carácter y cultivar en mí cualidades necesarias para desenvolverme en el día a día.

AGRADECIMIENTO

Agradezco a todos los que estuvieron involucrados directa o indirectamente en la elaboración de este proyecto de investigación. Por sobre todo agradezco a Jehová Dios, que fue la mente maestra detrás de todo este trabajo. Gracias a él tuve la capacidad física y mental para realizarlo. Su diseño sin igual en las neuronas del cerebro y en el ADN de las células fue la inspiración necesaria para tratar de imitar una pequeña parte de su funcionamiento.

Agradezco de todo corazón a mi familia, que siempre estuvo ahí para mí, apoyándome en todo momento. Su amor y paciencia me dio fuerzas para seguir adelante y no rendirme.

Agradezco a mis amigos y compañeros que estuvieron siempre a mi lado. Muchas cosas que conozco y domino ahora fueron gracias a su paciencia y amistad. Su apoyo fue fundamental y me permitió formarme como persona. El superar retos y disfrutar los buenos ratos hizo de esta etapa una que nunca olvidaré.

Agradezco al Ing. Walter Fuertes por su paciencia y dirección durante la elaboración y ejecución de este proyecto. Con amabilidad y firmeza, dirigió este proyecto para que resulte ser un trabajo del que puedo estar orgulloso.

Finalmente, agradezco a la Universidad de las Fuerzas Armadas ESPE, por brindarme la oportunidad de prepararme académicamente, al proporcionarme personal docente que no sólo se enfocó en mi formación académica y profesional, sino también en lo personal.

Gracias de todo corazón.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En la actualidad, las tecnologías que almacenan y procesan la información crecen cada día, aumentando a la par las amenazas que ponen en peligro dicha información. Por ello, la seguridad informática se vuelve cada vez más importante para proteger la información de personas mal intencionadas que buscan tener un acceso no autorizado. Puesto que muchos algoritmos criptográficos tradicionales ya han sido vulnerados, ha surgido la necesidad de técnicas criptográficas más seguras. En el presente trabajo se propone la implementación de dos nuevos algoritmos criptográficos en una aplicación móvil de chat. Por un lado, la criptografía neuronal, que emplea una estructura modificada de una red neuronal artificial llamada Tree Parity Machine, permite la generación de una clave criptográfica de 512 bits de longitud, sin la necesidad de enviarla por un medio inseguro. Y, por otro lado, la criptografía basada en ADN, que emplea dicha clave para el cifrado y descifrado de los mensajes dividiéndolos en bloques de longitud fija. Para cada uno de los algoritmos se realizan modelos de ataques y mediciones de su nivel de seguridad ante cada uno de ellos mediante análisis estadístico. Para ello, se ha realizado una investigación del estado actual de los dos algoritmos criptográficos, además de simulaciones y ataques en un ambiente controlado. Posteriormente, para el desarrollo de la aplicación móvil de chat se siguen los estándares y metodologías ágiles que permitan un desarrollo sostenible y resistente a cambios.

PALABRAS CLAVE:

- **CRIPTOGRAFÍA**
- **CRIPTOGRAFÍA NEURONAL**
- **TREE PARITY MACHINE**
- **CRIPTOGRAFÍA BASADA EN ADN**
- **EVALUACIÓN DE LA SEGURIDAD**

ABSTRACT

Nowadays, technologies that store and process information grow every day, unluckily the threats that endanger this information have also increased. Therefore, computer security becomes extremely important to protect the information from malicious people who seek to reach unauthorized access. Due to the fact that many traditional cryptographic algorithms have already been broken, the necessity for safer cryptographic techniques has emerged. Consequently, the present work proposes the implementation of two new cryptographic algorithms in a mobile chat application. On one hand, neural cryptography, which uses a modified structure of an artificial neural network called Tree Parity Machine, permits the generation of a cryptographic key of 512 bits in length, which does not need to be sent through an insecure medium. On the other hand, DNA-based cryptography uses this key to encrypt and decrypt messages by dividing them into blocks of fixed length. Models of attacks and measurements of their level of security have been developed for each one of the algorithms, by means of statistical analysis. For instance, an investigation of the current state of the two cryptographic algorithms has been done, as well as simulations and attacks in a controlled environment. Later, standards and agile methodologies that permit not only sustainable development, but also which have been proven to be invulnerable to changes are followed in order to design the mobile chat application.

KEYWORDS:

- **CRYPTOGRAPHY**
- **NEURAL CRYPTOGRAPHY**
- **TREE PARITY MACHINE**
- **DNA-BASED CRYPTOGRAPHY**
- **SECURITY EVALUATION**

CAPÍTULO I

INTRODUCCIÓN

1.1 Antecedentes

En la actualidad, las tecnologías que almacenan y procesan la información crece cada día, aumentando a la par las amenazas que ponen en peligro dicha información (Jain & Bhatnagar, 2014). Por ello, la seguridad informática se vuelve cada vez más importante para proteger la información de personas mal intencionadas que buscan tener un acceso no autorizado.

Las redes neuronales han atraído mucha atención en los últimos años como un modelo computacional de cómo funciona el cerebro humano (Klimov, Mityagin, & Shamir, 2002). Hoy en día el área de la inteligencia artificial sigue siendo muy activa, y atrae a investigadores interdisciplinarios de una amplia variedad de antecedentes.

No es sorprendente que los investigadores también hayan intentado utilizar redes neuronales en Criptografía. En enero de 2002 (Kanter, Kinzel, & Kanter, 2002) propuso un nuevo protocolo de intercambio de claves entre dos partes A y B. Este utiliza la nueva noción de desincronización caótica, que hace posible que dos sistemas débilmente interactivos converjan, aunque cada uno de ellos (visto individualmente) continúa moviéndose de manera caótica. La criptografía neuronal es un enfoque reciente que tiene como objetivo resolver el problema de intercambio de claves con la computación no clásica a través del entrenamiento de redes neuronales con los mismos patrones de entrada. Recientemente, existe un gran interés en la comunidad criptográfica para estudiar la seguridad de las implementaciones de protocolos criptográficos. El análisis de tiempo y análisis de potencia son los mecanismos más conocidos y exitosos para obtener información sobre los parámetros secretos del protocolo sin necesidad de resolver un problema difícil (Allam, Abbas, & El-Kharashi, 2013).

Por otro lado, debido a la gran capacidad de almacenamiento que posee el ADN (Ácido Desoxirribonucleico), en el que un gramo tiene una capacidad de 108 Terabytes, es altamente susceptible a almacenar información por encima de los medios convencionales de almacenamiento (Niazi & Brown, 2015). El primero en emplear ADN para realizar cálculos fue (Adleman, 1994) en el que resolvió el problema del camino más corto demostrando la alta capacidad de procesamiento en paralelo que tiene el ADN. A partir de ese estudio se han derivado varios estudios donde se emplea ADN para resolver problemas de cálculo. Pero el primero en dar un enfoque a la seguridad informática fue (Gehani, LaBean, & Reif, 2003), en el cual emplea una técnica de sustitución utilizando librerías de un solo uso.

1.2 Planteamiento del problema

En el mundo globalizado de hoy día, existen peligros de enviar información por medios digitales, tales como: transacciones bancarias, credenciales, entre otros. Según (Singh, Maakar, & Kumar, 2013) los algoritmos de generación de claves, como los empleados por RSA (Rivest, Shamir y Adleman) emplean propiedades de los números primos. Existen ciertas exclusiones en los valores semilla que hacen que una clave sea segura o no. Además, la generación periódica de esa clave para asegurar la seguridad puede resultar en consumo de recursos innecesarios de cómputo ya que cada cierto tiempo es necesario extender el tamaño de la clave (Meneses, y otros, 2016). Como muchos algoritmos criptográficos tradicionales (como DES, RSA, entre otros) ya han sido vulnerados por muchos atacantes, ha surgido la necesidad de técnicas criptográficas más seguras, como lo muestra (Upadhyaya, 2015) y (Ayala, Fuertes, Galárraga, Aules, & Toulkeridis, 2017).

1.3 Justificación

Actualmente, con el avance de la tecnología, las capacidades de cómputo se incrementan lo que resulta en que los actuales sistemas criptográficos que basan su

seguridad en la complejidad matemática sean vulnerados. Para mitigar esto, se incrementa la longitud de la llave para dificultar su descubrimiento.

Este problema se incrementa aún más debido a investigaciones en materia de computación cuántica. Es por esta razón que es primordial buscar nuevos sistemas criptográficos que no requieran grandes capacidades de cálculo y sean inmunes a personas no autorizadas que tratan de descifrarlos.

1.4 Objetivos

1.4.1 Objetivo general

Implementar un algoritmo de criptografía basado en redes neuronales y en ADN en una aplicación móvil de chat que permita una comunicación segura entre dos terminales mediante la generación de una clave y cifrado del canal.

1.4.2 Objetivos específicos

- Describir los actuales sistemas de cifrado, su clasificación y su seguridad, y analizar los nuevos sistemas de cifrado.
- Definir los sistemas de criptografía (neuronal y ADN) así como sus algoritmos y su complejidad, y las seguridades que implementan las aplicaciones móviles.
- Diseñar e implementar un algoritmo criptográfico que combine la criptografía neuronal y criptografía basada en ADN en una aplicación móvil de chat, y realizar pruebas de criptoanálisis.
- Realizar las pruebas de concepto, evaluar los resultados, validarlos y difundirlos ante la comunidad científica.

1.5 Alcance

En este proyecto se implementará un algoritmo de cifrado y descifrado que combine los dos nuevos métodos actuales de criptografía neuronal y criptografía basada en ADN. Este nuevo algoritmo será probado en una aplicación móvil de chat que se desarrollará a lo largo del presente proyecto, con la finalidad de corroborar el envío de mensajes seguros. Por ello, los mensajes no contarán con un formato definido, es decir, que los datos enviados están estructurados únicamente de texto.

La aplicación móvil no contará con todas las funcionalidades que tienen las actuales aplicaciones de chat que existen en el mercado. Más bien, será una aplicación destinada a enviar y recibir mensajes de texto. Además, la aplicación será únicamente compatible con dispositivos Android debido a las facilidades de desarrollo que ofrece y su acceso al medio.

1.6 Definición de la investigación

Para la realización del presente proyecto se utilizará como metodología de investigación la investigación aplicada. Como lo menciona (Bickman & Rog, 2009), dicha metodología se compone de dos fases principales: planificación y ejecución. Durante la primera fase se define el alcance de la investigación y se desarrolla un plan integral. Durante la segunda fase se implementa y se supervisa el plan propuesto, seguido de actividades de informe y seguimiento.

1.7 Metodología de desarrollo

Para el desarrollo del algoritmo y la aplicación se empleará la metodología Extreme Programming, que es una metodología ágil de desarrollo (Canós, Letelier, & Penadés, 2003). Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

CAPÍTULO II

ESTADO DE LA CUESTIÓN Y MARCO TEÓRICO

En este capítulo se especifica los fundamentos teóricos/técnicos que se aplicaron en el desarrollo del algoritmo criptográfico y de la aplicación móvil de chat, en el cual se describe las técnicas, herramientas y métodos, que apalancan el presente estudio.

2.1 Seguridad de la información

El Comité en Sistemas de Seguridad Nacional de los Estados Unidos define la seguridad de la información como “la protección de la información y de los sistemas de información contra el acceso no autorizado, uso, divulgación, interrupción, modificación o destrucción para proporcionar confidencialidad, integridad y disponibilidad” (Committee on National Security Systems, 2015).

De acuerdo con (Barrett, 2018), las amenazas a la seguridad informática “ponen en riesgo la seguridad, la economía, la seguridad pública y la salud de una nación”. En una organización, dichas amenazas “perjudican su balance final, aumentando los costos y disminuyendo sus ingresos, y afectan su capacidad innovar y ganar y mantener clientes”. Por ello, las organizaciones “deben tener un enfoque constante e iterativo para identificar, evaluar y administrar el riesgo en la ciberseguridad”, para proteger “sus activos de información, así como su reputación, posición legal, personal y otros activos tangibles o intangibles” (Nieles, Dempsey, & Pillitteri, 2017).

Ante este escenario, el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (National Institute of Standards and Technology, NIST) lanzó un proyecto, convocando a organizaciones de los sectores público y privado, con el objetivo de realizar un marco de trabajo para mejorar la ciberseguridad de las infraestructuras críticas (Barrett, 2018). Este marco se detalla a continuación.

2.2 Marco de trabajo para mejorar la ciberseguridad

Este marco proporciona y establece un lenguaje común para comprender, gestionar y expresar el riesgo de seguridad cibernética para las partes interesadas. Se puede usar para ayudar a identificar y priorizar acciones para reducir el riesgo de seguridad cibernética. Además, es una herramienta para alinear los enfoques de políticas, negocios y tecnología para manejar ese riesgo. Se puede usar para administrar el riesgo de seguridad cibernética en todas las organizaciones o se puede enfocar en la entrega de servicios críticos dentro de una organización (Barrett, 2018).

Este marco está compuesto por tres partes: el núcleo, los niveles de implementación y los perfiles. Estos componentes se explican a continuación.

2.2.1 Núcleo del marco

El núcleo del marco proporciona un conjunto de actividades para lograr resultados específicos y referencias de ejemplos de orientación. También presenta los resultados clave identificados por los interesados como útiles para gestionar el riesgo de ciberseguridad. Se compone de 4 elementos: funciones, categorías, subcategorías y referencias informativas. La **Figura 1** muestra la estructura del núcleo del marco.

Las funciones organizan actividades básicas en su más alto nivel y ayudan a una organización a expresar su gestión del riesgo de ciberseguridad mediante la organización de la información. Esto permite tomar decisiones de gestión de riesgos, abordar las amenazas y disminuir las vulnerabilidades. También se alinean con las metodologías existentes para la gestión de incidentes y ayudan a mostrar el impacto de las inversiones en ciberseguridad. De acuerdo a (Barrett, 2018), las funciones son:

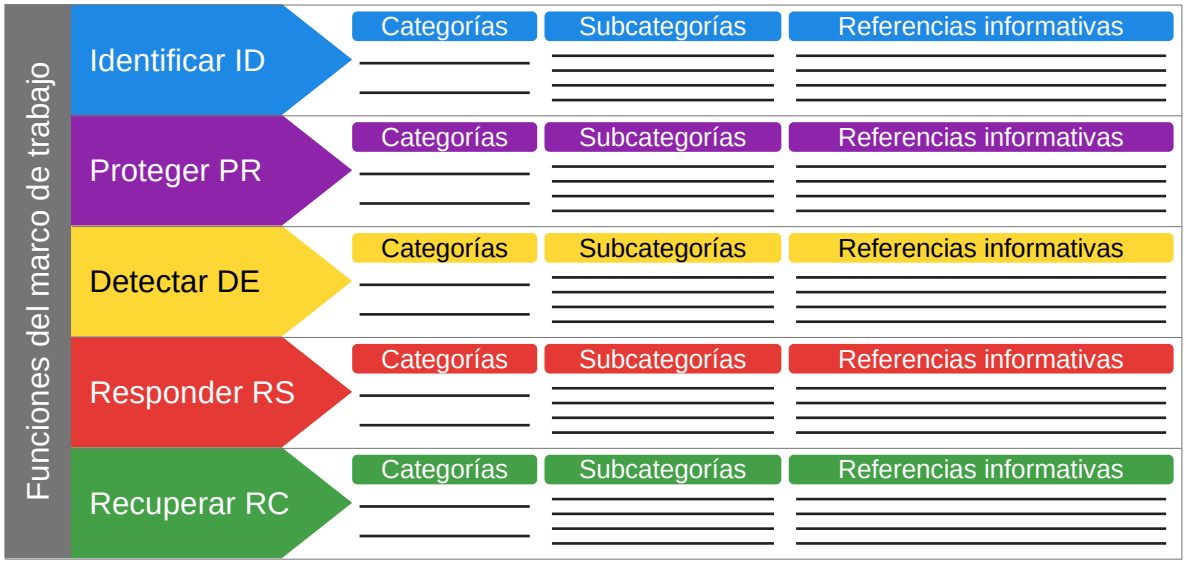


Figura 1. Estructura del núcleo del marco de trabajo
Fuente: (Barrett, 2018)

1. Identificar: Desarrolla un entendimiento organizacional para administrar el riesgo de ciberseguridad para sistemas, personas, activos, datos y capacidades.

2. Proteger: Desarrolla e implementa las salvaguardas apropiadas para proteger la entrega de servicios críticos.

3. Detectar: Desarrolla e implementa actividades apropiadas para detectar la ocurrencia de un evento de ciberseguridad.

4. Responder: Desarrolla e implementa actividades apropiadas para tomar medidas con respecto a un incidente de ciberseguridad detectado.

5. Recuperar: Desarrolla e implementa actividades apropiadas para mantener los planes de resiliencia y restaurar cualquier capacidad o servicio que se haya deteriorado debido a un incidente de ciberseguridad.

Las categorías son las subdivisiones de una función en grupos de resultados de seguridad cibernética estrechamente vinculados a las necesidades y actividades particulares. Las subcategorías dividen aún más una categoría en resultados específicos de actividades técnicas y/o de gestión proporcionando un conjunto de resultados que,

aunque no son exhaustivos, ayudan a respaldar el logro de los resultados en cada categoría. Finalmente, las referencias informativas son secciones específicas de estándares, pautas y prácticas comunes entre los sectores de infraestructura crítica que ilustran un método para lograr los resultados asociados con cada subcategoría.

2.2.2 Niveles de implementación del marco

Los niveles de implementación proporcionan un contexto sobre cómo una organización ve el riesgo de ciberseguridad y los procesos establecidos para gestionar ese riesgo. Describen un grado creciente de rigor y sofisticación en las prácticas de gestión de riesgos de ciberseguridad. Ayudan a determinar hasta qué punto la gestión de riesgos de ciberseguridad se basa en las necesidades empresariales y se integra en las prácticas generales de gestión de riesgos de una organización.

Cada organización debe determinar el nivel deseado, asegurándose de que el nivel seleccionado cumpla con los objetivos organizacionales, sea factible de implementar y reduzca el riesgo de ciberseguridad a activos y recursos críticos a niveles aceptables para la organización.

2.2.3 Perfiles del marco

Los perfiles son la alineación de las funciones, las categorías y las subcategorías con los requisitos comerciales, la tolerancia al riesgo y los recursos de la organización. Un perfil permite a las organizaciones establecer una hoja de ruta, describiendo el estado actual o el estado objetivo deseado, para reducir el riesgo de ciberseguridad que está alineada con los objetivos organizacionales y sectoriales, considera los requisitos legales y las mejores prácticas de la industria, y refleja las prioridades de gestión de riesgos. La comparación de perfiles (por ejemplo, el perfil actual y el perfil objetivo) puede revelar las brechas que deben abordarse para cumplir con los objetivos de la gestión de riesgos de ciberseguridad.

2.3 Servicios de seguridad

De acuerdo con (Barker, 2016), la criptografía se puede emplear para cubrir varios servicios de seguridad básicos como: confidencialidad, integridad, autenticación, autorización, y no repudio, además de trazabilidad, los mismos que se detallan a continuación.

2.3.1 Confidencialidad

La confidencialidad es la propiedad por la cual la información no se divulga a terceros no autorizados. La confidencialidad se logra mediante el cifrado para hacer que la información sea ininteligible excepto por entidades autorizadas, y puede volverse inteligible usando el descifrado. Para que el cifrado sea confidencial, el algoritmo criptográfico y el modo de operación deben diseñarse e implementarse de modo que una parte no autorizada no pueda determinar las claves secretas o privadas asociadas con el cifrado o pueda derivar directamente el texto sin usar las claves correctas.

2.3.2 Integridad

La integridad de los datos es una propiedad por la cual los datos no se han modificado de forma no autorizada desde que se crean, transmiten o almacenan. La modificación incluye la inserción, eliminación y sustitución de datos. Los mecanismos criptográficos, tales como códigos de autenticación de mensajes o firmas digitales, pueden usarse para detectar (con una alta probabilidad) modificaciones accidentales y deliberadas de un adversario. Los mecanismos no criptográficos también se utilizan a menudo para detectar modificaciones accidentales, pero no son muy confiables al detectar modificaciones deliberadas.

2.3.3 Autenticación

Existen dos tipos de servicios de autenticación mediante criptografía: autenticación de integridad y autenticación de origen.

- Se utiliza el servicio de autenticación de integridad para verificar que los datos no se hayan modificado, es decir, este servicio proporciona protección de integridad.
- Se utiliza el servicio de autenticación de origen para verificar la identidad del usuario o sistema que creó la información.

Se pueden usar varios mecanismos criptográficos para proporcionar servicios de autenticación. Los mecanismos que se utilizan comúnmente para proporcionar autenticación son las firmas digitales o códigos de autenticación de mensajes, además de algunas técnicas de acuerdo clave.

2.3.4 Autorización

La autorización se refiere a proporcionar una sanción o permiso para realizar una función o actividad de seguridad. La autorización se considera como un servicio de seguridad que a menudo es compatible con un servicio criptográfico. Normalmente, la autorización se otorga solo después de la ejecución de un servicio de autenticación de origen exitoso.

2.3.5 No repudio

En la administración de claves, el no repudio es un término asociado con las claves de firma digital y los certificados digitales que vinculan el nombre del sujeto del certificado a una clave pública. Cuando se indica no repudio para una clave de firma digital, significa que las firmas creadas por esa clave admiten no solo la integridad habitual y los servicios de autenticación de origen de las firmas digitales, sino que también pueden (dependiendo del contexto de la firma) indicar compromiso por el sujeto del certificado.

2.3.6 Trazabilidad

La trazabilidad proporciona la capacidad de rastrear algo a través de un proceso a un punto a lo largo de su curso. Puede ser hacia adelante o hacia atrás a través del proceso

y determina, según sea necesario, su origen, su historia y las condiciones a las que fue sometido. (Hoyle, 2017)

2.4 Criptología

La criptología es la disciplina de la criptografía y el criptoanálisis y de su interacción (Bauer, Cryptology, 2011). Su objetivo es la confidencialidad de la información. La criptología se divide en Criptografía y Criptoanálisis, como lo muestra la **Figura 2**.

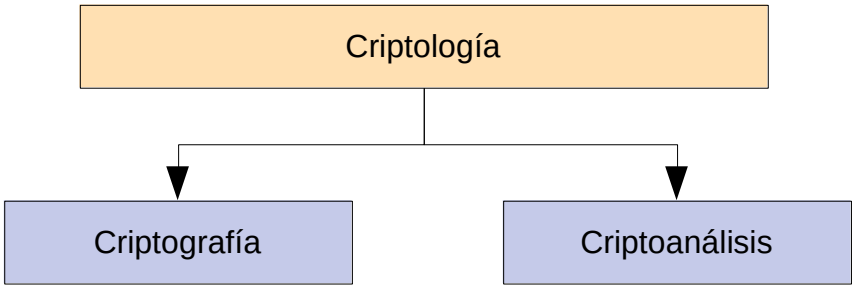


Figura 2. Criptología

2.4.1 Criptografía

La criptografía es la ciencia matemática y la disciplina de escribir un mensaje en texto cifrado, usualmente mediante la traducción del mensaje de acuerdo con una o algunas llaves criptográficas (frecuentemente cambiantes), con el objetivo de proteger un secreto de adversarios, interceptores, intrusos, opositores y/o atacantes (Bauer, Cryptology, 2011). Trata de mecanismos para garantizar la integridad, técnicas para intercambiar claves secretas y protocolos para autenticar usuarios. La criptografía implica el estudio de técnicas matemáticas para asegurar la información digital, los sistemas y los cálculos distribuidos contra los ataques adversos (Katz & Lindell, 2014).

2.4.1.1 Criptosistema

Un sistema criptográfico o criptosistema consiste de un algoritmo de cifrado, un algoritmo de descifrado, y de textos sin formato (mensajes), textos cifrados y de llaves

criptográficas (Bauer, Cryptosystem, 2011). En términos generales, un criptosistema puede ser dividido en dos esquemas (Pelosi, 2011):

- Esquema de cifrado simétrico
- Esquema de cifrado asimétrico

En un criptosistema simétrico (también llamado criptosistema de llave privada), se emplea la misma clave para las operaciones de cifrado y descifrado, y, por lo tanto, la clave debe ser compartida y mantenida en secreto por las partes que realizan dichas operaciones (Kaliski, Symmetric Cryptosystem, 2011). En 1949, Claude Shannon fue el primero en definir formalmente (con definición matemática) la criptografía simétrica (van Tilborg, 2011). La **Figura 3** muestra el modelo propuesto por Shannon de la criptografía simétrica.

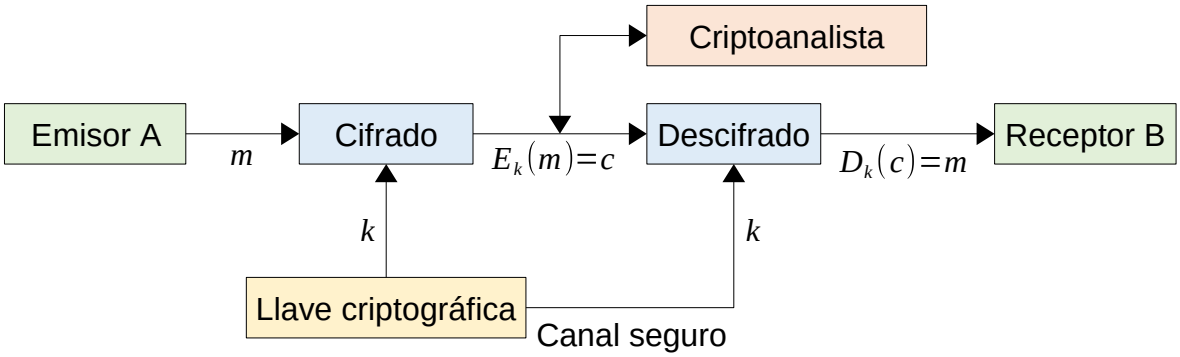


Figura 3. Modelo de criptografía simétrica
Fuente: (Shannon, 1949)

En el modelo de Shannon, un emisor A desea mandar un mensaje m a un receptor B de tal manera que el receptor sea el único que pueda leerlo. Mientras tanto, un criptoanalista tiene la posibilidad de leer los mensajes que se envían a través del medio inseguro. Para que solo el receptor sea capaz de leerlo, tanto el emisor como el receptor comparten una misma llave criptográfica k con la cual el emisor cifra el mensaje obteniendo un texto cifrado c mediante un algoritmo criptográfico E. El receptor recibe el texto cifrado, y, mediante un algoritmo de descifrado D obtiene el mensaje original del

emisor. El texto cifrado se envía por un medio inseguro, mientras que la llave criptográfica se la comparte mediante un canal seguro.

Existen dos formas para cifrar los datos: cifrado por bloques y cifrado de flujo. El cifrado por bloques especifica un algoritmo criptográfico para calcular el texto cifrado de n-bits para un mensaje de n-bits dado, junto con un algoritmo de descifrado para calcular el mensaje de n-bits correspondiente a un texto cifrado de n-bits dado (Knudsen, 2011). Una llave criptográfica de k-bits tiene 2^k permutaciones de n-bits. A menudo, al cifrado por bloque se lo llama cifrado iterativo, por lo que el texto cifrado se lo calcula a través de operaciones iterativas sobre el mensaje junto con la llave criptográfica. Otros algoritmos calculan un conjunto subclaves a partir de la llave criptográfica para realizar el cifrado. A estos algoritmos se los llama algoritmos de generación de llaves.

El cifrado por bloques tiene 5 modos de operación (Preneel, 2011):

- Modo ECB (Electronic Code Book)
- Modo CBC (Cipher Block Chaining)
- Modo OFB (Output FeedBack)
- Modo CTR (Counter)
- Modo CFB (Cipher FeedBack)

En el modo ECB, después del relleno, el texto plano se divide en bloques de n-bits y el cifrado y descifrado se aplica a cada bloque. El modo CBC es el más popular en el cifrado por bloques. En este modo, el mensaje se divide en t bloques de n-bits y le añade un valor inicial antes del cifrado a cada bloque, y, luego, el bloque resultante es el valor inicial para el siguiente bloque a cifrar. El modo OBF transforma el cifrado por bloques en un cifrado de flujo síncrono. Este modo emplea solo la operación de cifrado y consiste en una máquina de estados finitos inicializada con un valor inicial de n-bits. El estado está cifrado y el resultado del cifrado se utiliza como flujo de claves y como retroalimentación

al estado inicial. El modo CTR es otra forma de transformar el cifrado por bloques en un cifrado de flujo síncrono. Al igual que el modo OBF, este modo emplea solo la operación de cifrado y consiste de una máquina de estados finitos inicializada con un valor entero de n-bits. El estado es cifrado para obtener el flujo de clave, y el estado es actualizado como un contador mediante la operación $\mathrm{mod}\, 2^n$. Finalmente, el modo CFB transforma el cifrado por bloques en un cifrado de flujo auto-sincronizado. Al igual que el modo OBF y CTR, este modo emplea solo la operación de cifrado y consiste en una máquina de estados finitos con un valor inicial de n-bits. El estado se cifra y los m-bits más significativos del resultado se agregan al bloque del mensaje de m-bits y el texto cifrado resultante se retroalimenta al estado.

El cifrado de flujo opera con una transformación variable en el tiempo en dígitos individuales del mensaje a diferencia del cifrado por bloques que opera con una transformación fija en bloques grandes del mensaje (Canteaut, 2011).

Dentro de los algoritmos de cifrado simétrico por bloques más populares y con mayor seguridad se encuentra el algoritmo Rijndael (Advanced Encryption Standard AES) que proporciona un mapeo de bloques del mensaje a bloques de texto cifrado y viceversa bajo una clave de cifrado (Daemen & Rijmen, 2011). El algoritmo AES admite todas las combinaciones de longitudes de bloque y longitudes de clave que son un múltiplo de 32 bits con un mínimo de 128 bits y un máximo de 256 bits.

Por otro lado, un criptosistema asimétrico (también llamado criptosistema de llave pública) es aquel en el que se emplean claves diferentes para las operaciones de cifrado y descifrado, y donde una de las claves puede hacerse pública sin comprometer el secreto de la otra clave (Kaliski, Asymmetric Cryptosystem, 2011). En el modelo clásico para el cifrado asimétrico, una operación de cifrado E que implica una clave pública $PubKey$ se puede revertir mediante una operación D de descifrado que implica una clave privada coincidente $PrivKey$. Es decir, dado un mensaje M, un texto cifrado C se calcula como lo muestra la **Ecuación 1**:

$$C = E(PubKey, \quad M) \qquad \textbf{Ecuación 1}$$

y el mensaje se recupera como lo muestra la **Ecuación 2**:

$$M = D(PrivKey, \quad C) \qquad \textbf{Ecuación 2}$$

El uso actual es más general y cubre otros aspectos (como los esquemas de firma digital y los protocolos de establecimiento de claves) donde algunas operaciones involucran que una clave puede hacerse pública y otras involucran que una clave coincidente deba mantenerse en secreto. Se implementan ampliamente, especialmente en entornos donde las partes inicialmente no se encuentran en una posición conveniente para compartir secretos entre sí. Algunos ejemplos son el cifrado RSA (Rivest, Shamir y Adleman), el protocolo de establecimiento de claves Diffie-Hellman, el estándar de firma digital y la criptografía con curva elíptica.

Las implementaciones a menudo emplean una combinación de criptosistemas asimétricos y simétricos: el primero para establecer una clave secreta compartida y el segundo para proteger los mensajes con la clave. Dichos sistemas se llaman criptosistemas híbridos.

En noviembre de 2008, (Al Hasib & Haque, 2008) realizaron una comparación entre los algoritmos de cifrado más usados de los criptosistemas simétricos y asimétricos: AES y RSA respectivamente. Se evaluó su rendimiento y seguridad contra ataques como: ataque por fuerza bruta, ataque matemático, ataque de tiempo y ataque de texto cifrado escogido (Chosen Ciphertext Attack CCA). Se obtuvo los siguientes resultados:

- El algoritmo AES es rápido y seguro comparado con otros algoritmos de cifrado simétrico. El algoritmo AES también es rápido al cifrar bloques de datos de gran tamaño.
- El algoritmo RSA también es considerado seguro con una clave grande (por ejemplo, de 2048 bits).

- El mayor inconveniente de AES es el intercambio de la clave.
- El mayor inconveniente de RSA es la sobrecarga de cómputo debido a la clave.
- Ya que el algoritmo RSA tiene una funcionalidad superior y el algoritmo AES es rápido, se los combina en criptosistemas híbridos, el primero para establecer la llave criptográfica y el segundo para cifrar los datos.

2.4.1.2 Administración de la llave criptográfica

La administración de la llave criptográfica involucra las operaciones de generación, distribución, almacenamiento, actualización y cancelación de claves empleadas en el cifrado y descifrado. Uno de los mayores problemas asociados con la criptografía es la distribución segura de estas claves a las partes comunicantes correspondientes. Esto se conoce como distribución de claves o establecimiento de claves (Lloyd & Adams, 2011).

2.4.2 Criptoanálisis

El criptoanálisis es la disciplina de descifrar un texto cifrado sin tener acceso a la llave criptográfica, normalmente recuperando, de forma parcial o total, el mensaje original, e incluso, la llave criptográfica empleada, en casos favorables para el atacante al reconstruir todo el sistema criptográfico utilizado. Siendo este el peor caso posible para el lado atacado, un nivel aceptable de seguridad debe descansar completamente en la clave (Bauer, Cryptanalysis, 2011).

Existen varios modelos de ataques para el cifrado por bloques. En el presente estudio se emplearon dos modelos de ataques, que son ataque de fuerza bruta y ataque de texto plano escogido. A continuación de detalla cada uno de ellos.

2.4.2.1 Ataque de fuerza bruta

En el ataque de fuerza bruta, o búsqueda exhaustiva de la clave, el criptoanalista intenta descifrar el texto cifrado con cada posible clave hasta que encuentra la correcta

(Wiener, 2011). Este tipo de ataques requieren un elevado costo computacional, pero su implementación no requiere un esfuerzo superior por parte del criptoanalista. El algoritmo que ejecute este ataque debe tener la capacidad de generar una clave y emplearla para descifrar un texto cifrado. Si el texto encontrado es igual al texto original, la clave probada será la correcta.

2.4.2.2 Ataque de texto plano escogido

El ataque de texto plano escogido (Chosen Plaintext Attack CPA) es un escenario en el que el atacante tiene la capacidad de elegir el mensaje y ver su texto cifrado correspondiente (Biryukov, 2011). Uno de estos escenarios emplea el Criterio Avalancha (AVAL). El criterio AVAL es una propiedad criptográfica importante de los cifrados de bloque que dice que un pequeño número de diferencias de bits en el mensaje de entrada conduce a una “avalancha” de cambios, es decir, da como resultado un gran número de diferencias de bits de texto cifrado (Vergili & Yucel, 2001).

2.4.3 Nuevos criptosistemas emergentes

Como lo menciona (Zhang, Liu, Ma, & Cheng, 2017), “hay tres ramas prominentes de la criptografía: la criptografía moderna, la criptografía basada en ADN, y la criptografía cuántica, cada una con sus problemas y desafíos para los cuales no existen una solución conocida hasta ahora”. Y la llegada del criptoanálisis cuántico “ha suscitado una creciente preocupación por la seguridad de los protocolos criptográficos modernos”, como lo menciona (Padilla, Meyer-Baese, & Foo, 2018), y por tal motivo, la criptografía neuronal pretende ser una solución al problema de la distribución de la clave. A continuación, se detallan cada uno de estos nuevos criptosistemas: la criptografía neuronal y la criptografía basada en ADN.

2.4.3.1 Criptografía neuronal

El algoritmo, propuesto inicialmente por (Kanter, Kinzel, & Kanter, 2002), emplea una estructura modificada de una red neuronal artificial llamada Tree Parity Machine o TPM.

Este tipo de red neuronal, en forma de árbol como lo muestra la **Figura 4**, se compone de una sola capa oculta de K neuronas, y cada neurona de esta capa tiene n nodos de entrada con N pesos sinápticos w. En la capa de salida existe una sola neurona que produce una salida binaria de valores $\{-1,\ \ 1\}$. El objetivo de la red neuronal TPM es que dos redes neuronales, con la misma estructura, inicialicen sus pesos sinápticos de forma completamente aleatoria, y, luego de proveer unos valores a la entrada y tomar en cuenta sus salidas, lleguen a tener los mismos pesos.

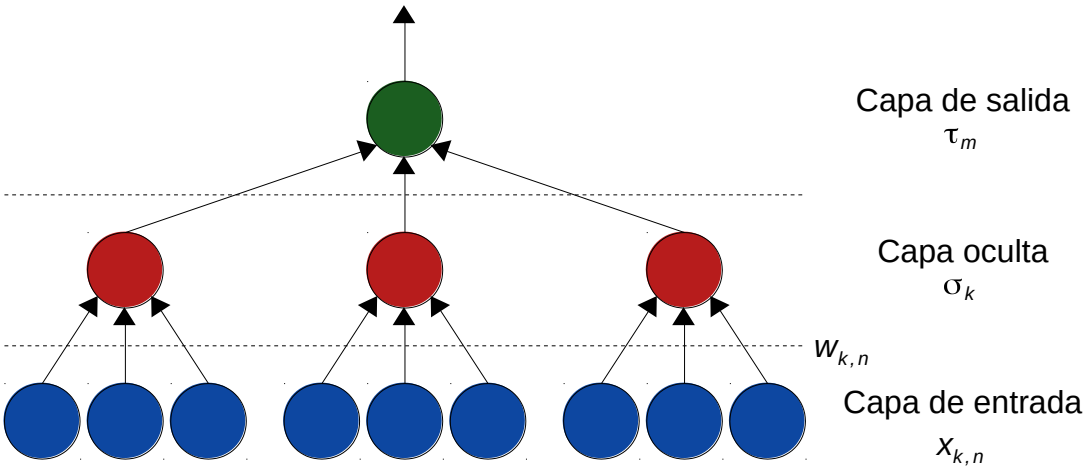


Figura 4. Tree Parity Machine de K=3 y N=3
Fuente: (Padilla, Meyer-Baese, & Foo, 2018)

Como lo menciona (Gupta, Nanda, Chhikara, Gupta, & Jain, 2018), para lograr la sincronización de los pesos sinápticos de ambas redes neuronales, se pueden emplear una de tres reglas de aprendizaje. Estas son: regla hebbiana, regla anti-hebbiana y camino aleatorio. Cada una de estas reglas toman en cuenta los valores de entrada generados, así como las salidas de ambas redes neuronales, para actualizar sus pesos sinápticos únicamente si las salidas de ambas redes son iguales. Las ecuaciones de cada uno se detallan a continuación.

1. Regla hebbiana:

$$w_{i,j} = \gamma\left(w_{i,j} + x_{i,j} * \tau * \theta(\sigma_i,\ \ \tau) * \theta(\tau^A,\ \ \tau^B)\right)$$

2. Regla anti-hebbiana

$$w_{i,j} = \gamma\left(w_{i,j} - x_{i,j} * \tau * \theta(\sigma_i, \quad \tau) * \theta(\tau^A, \quad \tau^B)\right)$$

3. Camino aleatorio

$$w_{i,j} = \gamma\left(w_{i,j} + x_{i,j} * \theta(\sigma_i, \quad \tau) * \theta(\tau^A, \quad \tau^B)\right)$$

El autor mencionado anteriormente, analiza y concluye que la regla hebbiana tiene un mejor rendimiento que las demás en cuanto a porcentaje de error en la sincronización y menor cantidad de iteraciones necesarias para alcanzarla. Por ello, en el presente trabajo se empleará esta regla para la sincronización.

Varios investigadores, como (Padilla, Meyer-Baese, & Foo, 2018), (Anikin, Makhmutova, & Gadelshin, 2016), (Gupta, Nanda, Chhikara, Gupta, & Jain, 2018) y (Pattanayak & Ludwig, 2017), concluyen que la red neuronal TPM es una estructura que permite el intercambio de una llave criptográfica mediante un canal inseguro, que, con determinados parámetros de configuración inicial, hacen muy difícil la tarea a un criptoanalista establecer la llave criptográfica que se genera. Actualmente, se investiga las configuraciones óptimas que permiten una sincronización más rápida para que la llave resultante sea criptográficamente segura.

2.4.3.2 Criptografía basada en ADN

Para hablar de criptografía basada en ADN es necesario hablar primero de los procesos que tiene el ADN en la biología celular y molecular. En (Lodish, Berk, Matsudaira, Kaiser, & Scott, 2005) se describe los procesos que presenta el ADN, dos de los más importantes son la transcripción y la traducción, los cuales ocurren cuando se van a producir moléculas proteicas dentro de la célula, como lo muestra la **Figura 5**. Durante la transcripción se copia una parte del ADN de un gen para sintetizar finalmente en una hebra de Ácido Ribonucleico o ARN. Tanto el ADN como el ARN son escritos con las cuatro bases nitrogenadas de Adenina (A), Timina (T), Guanina (G) y Citosina (C), siendo el caso especial para la hebra de ARN en la cual se reemplaza la Timina por Uracilo (U). Durante la traducción, la hebra de ARN formada anteriormente se emplea

para la síntesis de proteínas la cual es traducida al lenguaje de 20 aminoácidos de las proteínas.

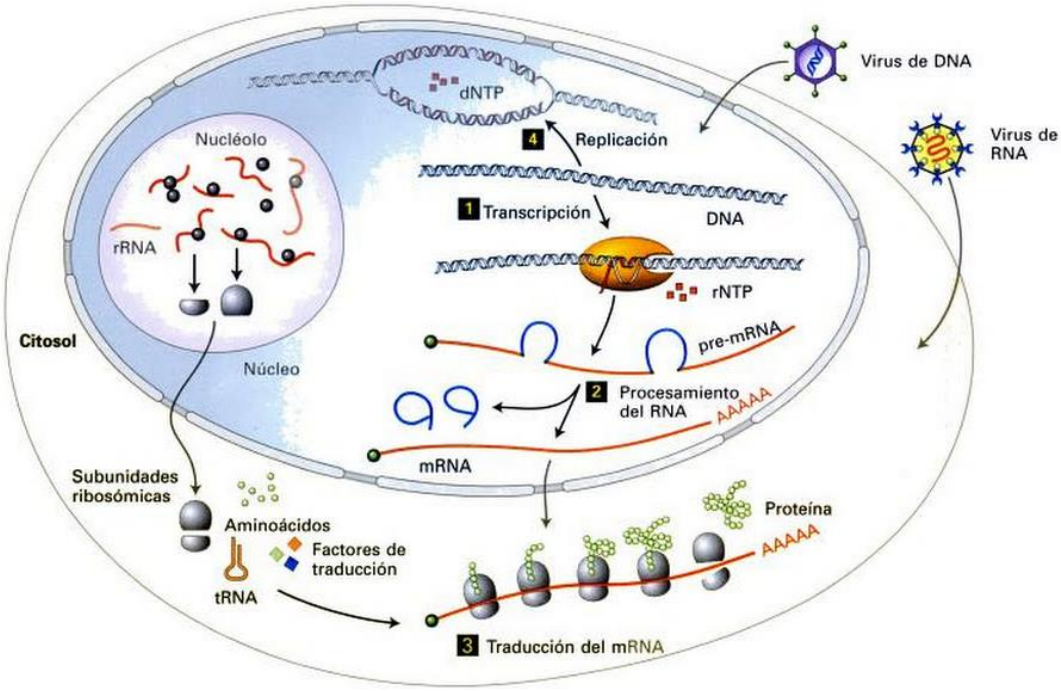


Figura 5. Vista general de cuatros procesos genéticos básicos
Fuente: (Lodish, Berk, Matsudaira, Kaiser, & Scott, 2005)

Actualmente, los investigadores plantean varias soluciones para asemejar el comportamiento del ADN biológico con sus diferentes procesos y aplicarlo a la criptografía basada en ADN. En (Kalsi, Kaur, & Chang, 2018), propone que para cifrar la información primero se debe realizar la generación de la clave criptográfica con las cuatro bases nitrogenadas mediante un algoritmo genético garantizado su aleatoriedad. Luego, esa clave y el texto a cifrar se las pasan a información binaria de acuerdo a una equivalencia propuesta de cada base a un par en base 2. A cada una de las cadenas resultantes se aplica el proceso de transcripción calculando su cadena complementaria, y finalmente, mediante el uso de una tabla diccionario, se aplica el proceso de traducción calculando la cadena cifrada al combinar ambas cadenas de texto y clave convirtiéndolas en base hexadecimal. Para el proceso de descifrado se aplica las operaciones contrarias y en secuencia inversa.

En (Al-Mahdi, Shahin, Fouad, & Alkhaldi, 2018), propone el proceso de cifrado de una forma diferente. Con la clave criptográfica, obtenida mediante un proceso de generación resultando en una cadena de ADN, se generan subclaves las cuales serán usadas para los procesos de cifrado y descifrado. Luego, tanto el texto a cifrar como las subclaves, son transformados a base binaria y, mediante un cifrado por bloques de longitud fija, se aplica el proceso de traducción calculando la cadena cifrada al combinar ambos elementos. El autor concluye que este algoritmo “muestra eficiencia términos de tiempo de cifrado y descifrado, de efecto de avalancha y de resistencia de la clave secreta en contra de ataques”.

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN

En este capítulo se detalla la metodología empleada para el desarrollo, así como el diseño e implementación de la aplicación de chat que combina los dos criptosistemas descritos en el capítulo anterior.

3.1 Metodología de desarrollo

Para el desarrollo de la aplicación se utilizó la metodología ágil Extreme Programming (XP), ya que, como lo menciona (Pressman & Maxim, 2015), es la metodología más ampliamente usada pues emplea un enfoque orientado a objetos y engloba un conjunto de reglas y prácticas que ocurren dentro del contexto de cuatro actividades iterativas, como lo muestra la **Figura 6**, que son: planificación, diseño, desarrollo y pruebas.

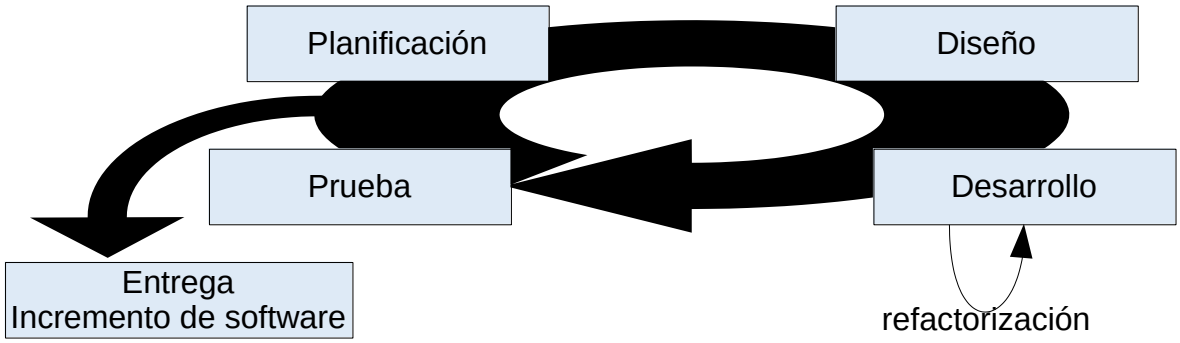


Figura 6. Proceso de Extreme Programming
Fuente: (Pressman & Maxim, 2015)

Cada iteración produce un software entregable con funcionalidad especificada por el usuario, y, con cada iteración, se incrementan estas funcionalidades. Debido a que los requisitos para el presente desarrollo pueden ser cambiantes y, en ocasiones imprecisos, la metodología XP es ideal (Canós, Letelier, & Penadés, 2003).

En comparación con la metodología ágil SCRUM, XP satisface los postulados del Manifiesto Ágil de mejor manera (Mendes Calo, Estevez, & Fillottrani, 2010). Además, XP

es útil para proyectos con grupos de trabajo pequeños. Para el caso de grupos de trabajo con una sola persona, (Ravikant & Umphress, 2008) menciona que XP puede ser aplicado manteniendo los valores de simplicidad, comunicación, retroalimentación y coraje o valentía. Por estos motivos, se ha seleccionado XP como metodología ágil de desarrollo de la aplicación móvil de chat. A continuación, se detallan las actividades realizadas dentro del proceso de la metodología XP.

3.1.1 Planificación

Para la actividad de planificación dentro de la metodología XP, se crean y redactan las historias de usuario que describen la o las salidas esperadas, las características y las funcionalidades de la aplicación y lo que el usuario espera obtener con cada acción. Posteriormente, se ordena de acuerdo a su prioridad y se agrupan aquellas que puedan generar un producto entregable al usuario organizándolas en mapas de historias. En la **Tabla *1*** hasta la **Tabla *6*** se describen las historias de usuario identificadas durante la elicitación de requerimientos.

Tabla 1
Historia de usuario HU01

ID:	HU01
Nombre:	Registrar usuario
Descripción:	El usuario podrá ingresar su información personal para poder registrarse en la aplicación
Entrada:	Nombre de usuario, correo electrónico y contraseña
Salida:	Mensaje de confirmación de cuenta creada
Proceso:	1. El usuario ingresa sus datos 2. La aplicación valida los datos ingresados

CONTINÚA

	3. Se despliega un mensaje de confirmación de éxito o error al crear la cuenta
Precondiciones:	Ninguna
Postcondiciones:	La cuenta del usuario se habrá creado
Prioridad:	Alta

Tabla 2
Historia de usuario HU02

ID:	HU02
Nombre:	Ingreso de usuario
Descripción:	El usuario podrá acceder a toda su información
Entrada:	Correo electrónico y contraseña
Salida:	Pantalla principal de la aplicación
Proceso:	1. El usuario ingresa sus credenciales 2. La aplicación valida los datos ingresados 3. Se muestra la pantalla principal de la aplicación o un mensaje de error
Precondiciones:	El usuario debe tener una cuenta creada
Postcondiciones:	La aplicación abre la pantalla principal con los datos del usuario
Prioridad:	Alta

Tabla 3
Historia de usuario HU03

ID:	HU03
Nombre:	Ver y modificar perfil
Descripción:	El usuario podrá ver y/o modificar su perfil en la aplicación
Entrada:	Imagen y/o estado
Salida:	Mensaje de confirmación de perfil actualizado
Proceso:	1. El usuario selecciona una imagen de su galería 2. La aplicación actualiza la foto de perfil del usuario con la imagen seleccionada 3. El usuario ingresa su estado en forma de texto 4. La aplicación actualiza el estado del usuario con el estado ingresado
Precondiciones:	El usuario debe tener una cuenta y haber ingresado exitosamente
Postcondiciones:	La cuenta del usuario se habrá actualizado
Prioridad:	Baja

Tabla 4
Historia de usuario HU04

ID:	HU04
Nombre:	Enviar solicitud de amistad

CONTINÚA

Descripción:	El usuario podrá enviar solicitudes de amistad a otros usuarios
Entrada:	El usuario al que se le envía la solicitud
Salida:	Mensaje de confirmación envío exitoso
Proceso:	1. El usuario selecciona la cuenta de otro usuario 2. El usuario selecciona Enviar Solicitud 3. La aplicación envía una solicitud al otro usuario
Precondiciones:	El usuario debe tener una cuenta y haber ingresado exitosamente
Postcondiciones:	La solicitud de amistad se habrá enviado exitosamente
Prioridad:	Media

Tabla 5
Historia de usuario HU05

ID:	HU05
Nombre:	Aceptar y/o eliminar solicitud de amistad
Descripción:	El usuario podrá aceptar y/o eliminar solicitudes de amistad de otros usuarios
Entrada:	El usuario que envía la solicitud
Salida:	Mensaje de confirmación de amistad establecida
Proceso:	1. El usuario selecciona la cuenta de otro usuario

CONTINÚA

	2. El usuario selecciona Aceptar/Eliminar Solicitud 3. La aplicación agrega a la lista de amigos el usuario aceptado
Precondiciones:	El usuario debe haber recibido una solicitud de amistad de otro usuario
Postcondiciones:	El otro usuario es agregado a la lista de amigos exitosamente
Prioridad:	Media

Tabla 6
Historia de usuario HU06

ID:	HU06
Nombre:	Enviar y/o recibir mensajes
Descripción:	El usuario podrá enviar y/o recibir mensajes de otros usuarios
Entrada:	El usuario al que se le envía o del que se recibe un mensaje
Salida:	Mensaje enviado con éxito
Proceso:	1. El usuario selecciona el perfil de otro usuario 2. El usuario escribe el mensaje y selecciona Enviar 3. El otro usuario recibe el mensaje
Precondiciones:	El usuario debe tener una amistad con el usuario al que desea mandar el mensaje

CONTINÚA

Postcondiciones:	El mensaje habrá sido enviado y recibido de forma exitosa
Prioridad:	Alta

Dada la prioridad que tiene cada historia de usuario, se procede a ordenarlas y agruparlas de tal manera que puedan generar un producto entregable al final de cada iteración de la siguiente manera:

1. Iteración 1: HU01 y HU02
2. Iteración 2: HU04 y HU05
3. Iteración 3: HU06
4. Iteración 4: HU03

Además de los requisitos funcionales descritos en las historias de usuario, la aplicación tiene requisitos no funcionales que mantienen su calidad, como los requerimientos de seguridad. Esta aplicación requiere que todos los mensajes sean cifrados para su envío y que, únicamente los usuarios autorizados puedan ser capaces de descifrarlos.

3.1.2 Diseño

3.1.2.1 Diagrama de arquitectura

Según (Pressman & Maxim, 2015), un diagrama de arquitectura se enfoca en la estructura del sistema de software y es útil para mostrar la distribución física de los componentes entre las plataformas de hardware y los ambientes de ejecución. En tal sentido, la **Figura 7** muestra la arquitectura del sistema.

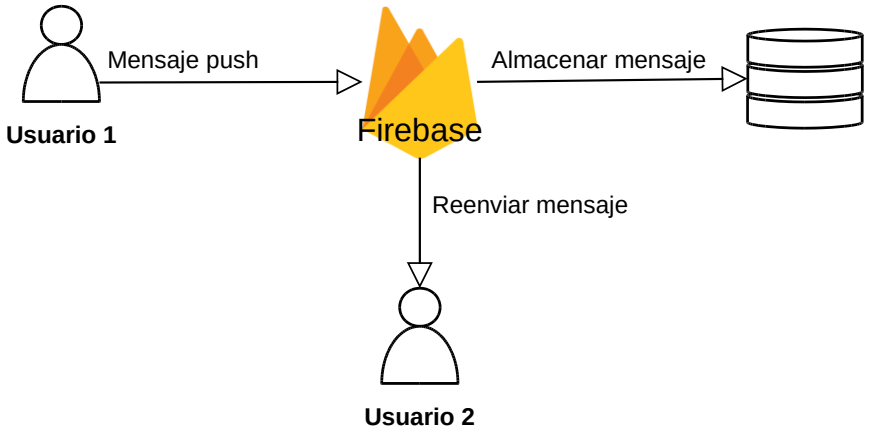


Figura 7. Diagrama de arquitectura de Firebase
Fuente: (Google, 2018), (Microsoft, 2018)

Como se aprecia en la **Figura 7**, se muestra la arquitectura de la aplicación para el envío y recepción de mensajes. El Usuario 1 envía un mensaje push a la base de datos de Firebase. Cuando Firebase lo recibe, lo guarda en su almacenamiento e inmediatamente lo reenvía al Usuario 2.

Posteriormente, se creó un servicio REST que permita generar un valor aleatorio con probabilidades determinadas. Dicho valor es requerido por las dos redes TPM determinar la cantidad de pasos que requieren durante el proceso de sincronización. La **Figura 8** muestra el diagrama de arquitectura empleado para esta operación.

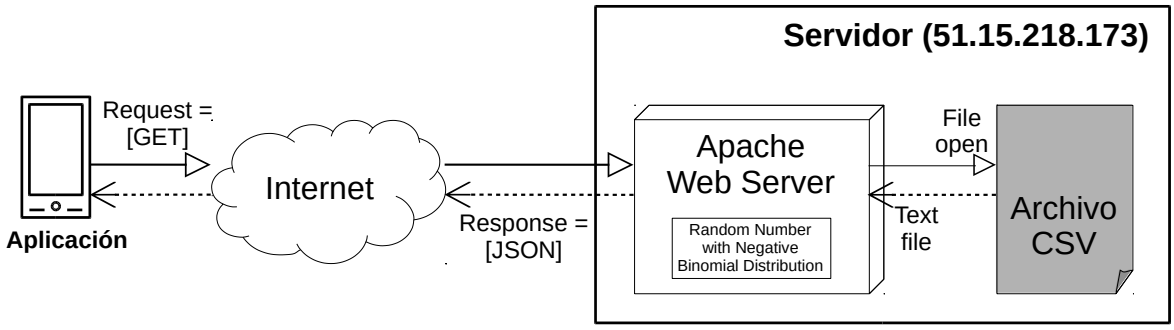


Figura 8. Diagrama de arquitectura del servicio web

La aplicación enviará una solicitud GET al servidor web (51.15.218.173) que ejecuta Apache, específicamente al método RandomNumberWithNegativeBinomialDistribution. Dicho método consulta el archivo CSV con las probabilidades y genera un número

aleatorio. Finalmente, envía una respuesta en formato JSON con el número generado. Por ejemplo, una respuesta esperada podría ser la siguiente.

```
{
    "number" : 211
}
```

El valor de `"number"` tendrá el número aleatorio generado, y será empleado por la aplicación para establecer la cantidad de pasos totales que usarán las redes TPM para sincronizar sus pesos sinápticos.

3.1.2.2 Diagrama de secuencia

Según (Sommerville, 2016), los diagramas de secuencia son utilizados para mostrar las interacciones entre los actores y los objetos en un sistema. Este tipo de diagramas muestran la secuencia de interacciones que tienen lugar durante una determinada tarea. De acuerdo con esta definición, la **Figura 9** muestra el diagrama de secuencia de la aplicación de chat.

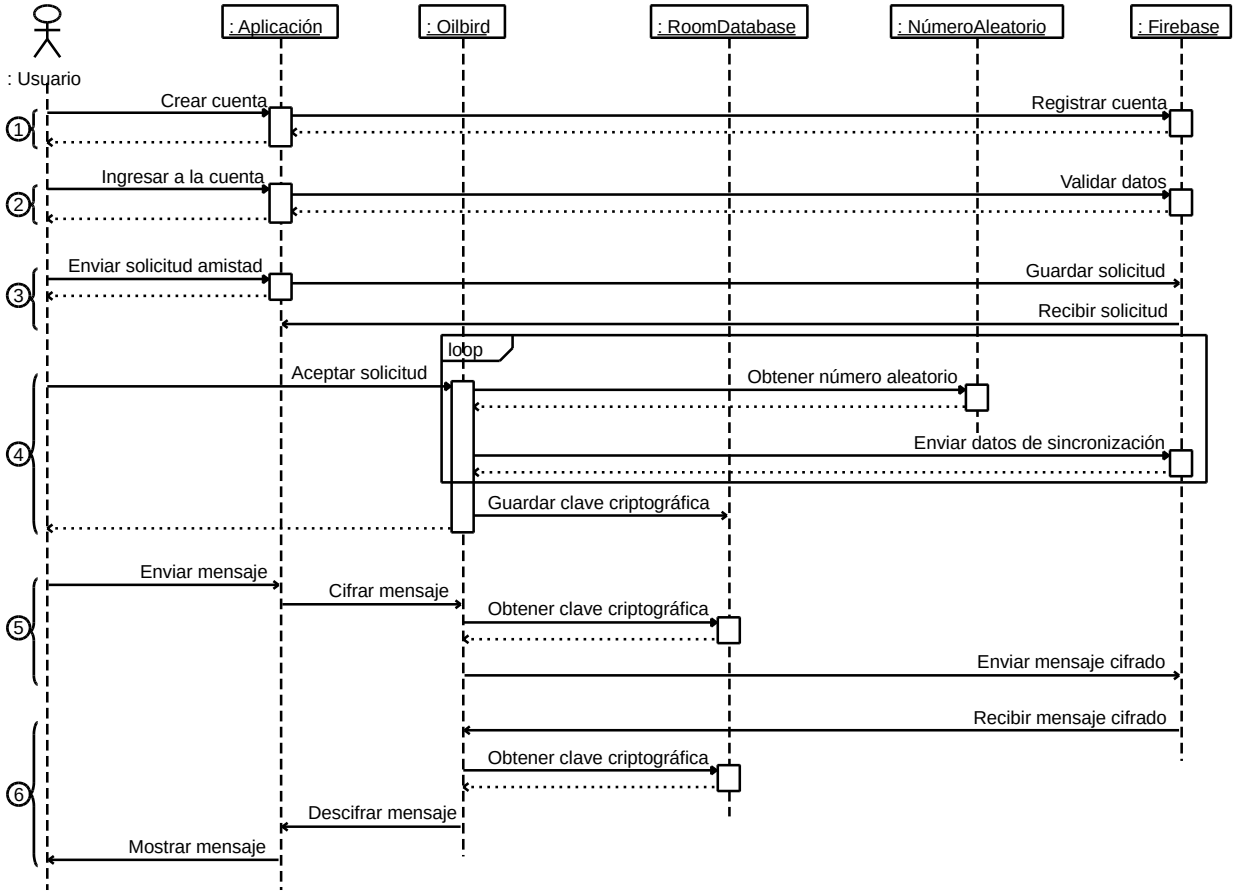


Figura 9. Diagrama de secuencia de la aplicación

En el diagrama de secuencia de la **Figura 9**, se muestran las principales acciones del usuario en la aplicación. En el punto 1, está la creación de una cuenta. Aquí, el usuario ingresa su información en la aplicación, la cual la envía a Firebase para su registro. Cuando la cuenta ha sido registrada, Firebase envía un mensaje de confirmación y la aplicación muestra un mensaje de cuenta creada. El punto 2 es cuando un usuario ingresa a la aplicación. Para ello, ingresa su información en la aplicación, la cual la envía a Firebase para ser validada. Si Firebase encuentra el registro en su base de datos, envía un mensaje de confirmación, caso contrario, envía un mensaje de error. Cuando un usuario ingresa exitosamente a la aplicación, podrá mandar solicitudes de amistad a otros usuarios, como lo muestra el punto 3. Al enviar la solicitud, la aplicación enviará un mensaje asíncrono a Firebase, el cual se encargará de almacenar dicha solicitud. En este caso la aplicación solo mostrará un mensaje de confirmación de envío de solicitud.

Firebase enviará la solicitud al usuario deseado, el cual podrá aceptarlo o rechazarlo. Cuando un usuario acepta una solicitud de amistad, entrará en un bucle que terminará cuando las redes TPM de los dos usuarios han terminado de sincronizarse, como lo muestra el punto 4.

Para llevar a cabo esto, la aplicación del primer usuario solicita un número generado aleatoriamente por el servidor Web que tiene el servicio REST. Luego, envía al otro usuario los datos de sincronización, el cual retorna una respuesta. Cuando las redes han cumplido la cantidad de pasos, se ejecuta un paso adicional de verificación, el cual, en caso de ser exitoso, termina el proceso de sincronización y almacena la clave criptográfica generada, caso contrario, inicia un nuevo proceso.

Para el proceso de envío de un mensaje entre dos usuarios, como lo muestran los puntos 5 y 6, el usuario ingresa el texto que desea enviar en la aplicación, la cual cifra la información con la clave generada durante el proceso de sincronización. Posteriormente, la aplicación envía el mensaje con el texto cifrado a Firebase, el cual lo almacena y lo reenvía al usuario destinatario. Cuando la aplicación recibe un mensaje, descifra el texto con la clave criptográfica generada y lo muestra al usuario.

3.1.2.3 Diagrama de clases

Según (Pressman & Maxim, 2015) y (Sommerville, 2016), un diagrama de clases provee una vista estática o estructural de una sistema cuando es desarrollado bajo un paradigma orientado a objetos. La **Figura 10** muestra el diagrama de clases de la aplicación. Las clases `FirebaseAuth` y `FirebaseUser` sirven para identificar y autorizar a un usuario para realizar una determinada tarea, y las clases `DatabaseReference` y `StorageReference` son para tener una referencia de la base de datos y el almacenamiento de archivos en Firebase.

Figura 10. Diagrama de clases de la aplicación

Todas las demás clases están organizadas en dos paquetes: `ec.edgarsalguero.smartsecuritychat` y `ec.edgarsalguero.oilbird`. Las clases del paquete `smartsecuritychat` tienen toda la lógica de la aplicación y la parte gráfica de la aplicación. Las clases de paquete `oilbird` tienen toda la lógica del proceso de sincronización (clase `TreeParityMachine`) y de los procesos de cifrado y descifrado de la información (clase `DNACrypt`). En este paquete también se encuentran las clases para el proceso de consulta del servicio web y la generación de un texto aleatorio (clases `NegativeBinomialDistribution` y `Sonar` respectivamente). Adicionalmente, en este paquete se encuentran las clases que manejan la persistencia de la información. La **Figura 11** muestra las clases que se encargan del almacenamiento local de la información.

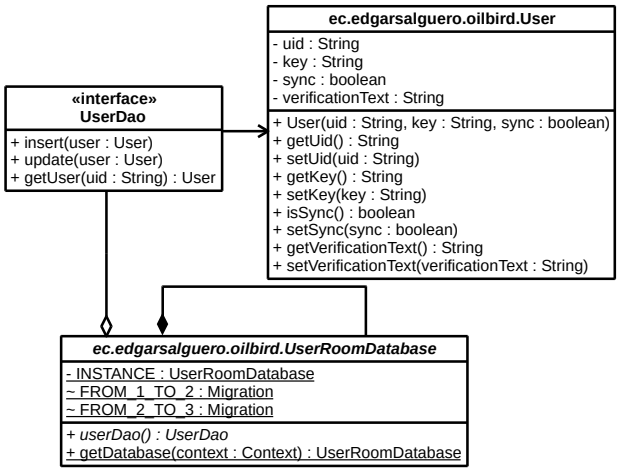


Figura 11. Diagrama de clases para la persistencia de la información

La clase `User` tiene la información del usuario que se almacena en la base de datos (SQLite). La interface `UserDao` es el objeto de acceso a datos con las operaciones de inserción, actualización y lectura de los datos almacenados. La clase `UserRoomDatabase` se encarga del almacenamiento en la base de datos local, la cual maneja migraciones en caso de cambios a la estructura original de los datos.

3.1.3 Estructura de la base de datos en Firebase

Como Firebase espera un mensaje en formato JSON, se estructura los datos de manera que pueda ser leído. Para este punto, los datos que son necesarios estructurar son principalmente dos: 1) cuando dos usuarios empiezan la sincronización de sus redes, y 2) cuando un usuario envía o recibe un mensaje. Para ello, se estructuró los mensajes de la siguiente manera. Para los mensajes de sincronización se estructuraron los siguientes datos:

```
{
    "sync": {
        "uid2": {
            "uid1": {
                "step": …,
                "total": …,
                "type": …,
                "output": …,
                "msg": …
            }
        }
    }
}
```

El nodo `"sync"` es la raíz de los datos que los usuarios emplearán al momento de enviar mensajes de sincronización. Cuando un usuario con identificador `"uid1"` desea enviar a otro usuario con identificador `"uid2"`, le enviará al nodo del segundo usuario su propio identificador. De esta manera, el usuario `"uid2"` podrá reconocer cambios en su nodo y tendrá la posibilidad de enviar un mensaje en respuesta al usuario `"uid1"`. Finalmente, la estructura del mensaje se compone de cinco datos. `"step"`, almacena el paso actual en el que se encuentran dentro de la sincronización las dos redes de los dos usuarios. `"total"` indica la cantidad total de pasos que realizarán las dos redes durante el proceso de sincronización. `"output"` indica la salida de la red calculada con el valor de la entrada que está en `"msg"` el cual puede tomar los valores de 1 y -1. `"msg"` indica

el mensaje que se envía al otro usuario. `"type"` indica en tipo de paso que está llevando a cabo la red, el cual puede tomar cuatro valores.

- Cuando el valor es `"update"`, la red empleará el valor de los datos `"msg"` y `"output"` como entrada y salida respectivamente para, de acuerdo a su propia salida, sincronizar o no los pesos sinápticos de su red. Luego, la aplicación enviará al otro usuario un mensaje del tipo `"sync"` con el dato `"output"` el valor de la salida de su red.

- Cuando el valor es `"sync"`, la red empleará el valor del dato `"output"` para sincronizar sus pesos sinápticos. Luego, la aplicación enviará al otro usuario un mensaje del tipo `"update"` con los datos `"msg"` un valor de entrada aleatorio, `"output"` el valor de la salida de su red con la nueva entrada, y `"step"` incrementado en 1.

- Cuando el valor de `"step"` sea igual al valor de `"total"`, la aplicación enviará un mensaje de tipo `"check"` al otro usuario. Primero, se genera un texto aleatorio y se encripta empleando los pesos sinápticos de la red como clave criptográfica, y `"msg"` toma el valor del texto cifrado. El segundo usuario recibe el mensaje e intenta descifrar el texto con sus pesos sinápticos como clave criptográfica. Finalmente lo vuelve a cifrar con su clave, y le envía al primer usuario un mensaje con el texto cifrado con el tipo `"verify"`.

- Cuando el valor del tipo es `"verify"`, la aplicación descifrará el texto con la clave criptográfica generada y comparará el texto original creado con el recibido. Si son iguales, termina el proceso de sincronización. Caso contrario, la aplicación generará una nueva red y empezará un nuevo proceso de sincronización.

Finalmente, para los mensajes de envío y recepción se estructuraron los siguientes datos.

```
{
    "messages" : {
        "uid2" : {
            "uid1" : {
                "msgid" : {
                    "from" : …,
                    "message" : …,
                    "seen" : …,
                    "time" : …,
                    "type" : …
                }
            }
        }
    }
}
```

El nodo "messages" es la raíz de los datos que los usuarios emplearán al momento de enviar mensajes de sincronización. De igual manera que con la sincronización, cuando un usuario con identificador "uid1" desea enviar a otro usuario con identificador "uid2", le enviará al nodo del segundo usuario su propio identificador. "msgid" será un identificador único para cada mensaje, ya que se contará con una lista de mensajes. Cada mensaje tendrá la siguiente estructura. "from" tendrá el identificador del usuario que envía el mensaje. "message" tendrá el texto cifrado del mensaje que se envía. "seen" tendrá el valor del tiempo en milisegundos cuando el mensaje ha sido visto por el otro usuario, y, en caso de no ser visto, tendrá un valor booleano de falso. "time" tendrá el tiempo en milisegundos cuando se envió el mensaje. Finalmente, "type" tendrá el formato que tiene el mensaje, que, para este proyecto, tendrá el valor de "text".

Para el registro de la información adicional que será útil para la aplicación, se crearon tres nodos raíz adicionales. El primero es "users", el cual tiene todos los usuarios registrados en la aplicación. El segundo es "friends" el cual se registra los amigos de cada usuario. Y el tercero es "chats" en el cual se registra la información del último mensaje enviado de cada usuario.

3.2 Algoritmo criptográfico neuronal

Para el diseño de la estructura que será usado por el algoritmo criptográfico neuronal, el cual será empleado para el establecimiento de la clave criptográfica entre dos usuarios de la aplicación móvil, se tomó en cuenta la longitud de la clave que es requerido por el algoritmo criptográfico basado en ADN. Dicha estructura debe ser la misma para los dos usuarios para poder sincronizar sus pesos sinápticos y establecer la clave criptográfica.

3.2.1 Estructura de la red neuronal

La estructura básica de una red neuronal TPM se basa en tres parámetros: el número de neuronas en la capa oculta (K), el número de entradas por neurona (N), y el rango de valores permitidos para los pesos sinápticos de cada una de las neuronas y de las entradas (L), siendo el rango $[-L,\ \ L]$, pero se ha modificado a $[-L,\ \ L-1]$ para asegurar un valor par en la cantidad de números posibles. Dependiendo de esos valores, la red neuronal generará una clave criptográfica de longitud única.

Para determinar todas las combinaciones posibles de los valores K, N y L se toma como base que la longitud de la clave debe ser de 512 bits, por lo que el valor de L en notación decimal puede ser en el rango de $[1,\ \ 2^{511}]$. Dado que $512 = 2^9$, existen 10 posibles valores de L cuya longitud en notación binaria hace que sean factores de 512 y que permitan calcular los valores de K y N con el factor restante. Por ejemplo, para el caso de $L = 2^3 = 8$, su longitud en notación binaria es 4 por lo que el factor restante es 128, ya que $4 \times 128 = 512$, por lo que $K \in \{1, 2, 4, 8\}$ y $N \in \{128, 64, 32, 16\}$ respectivamente, siendo las combinaciones totales $(K,\ \ N,\ \ L)$:

- $(1,\ \ 128,\ \ 8)$
- $(2,\ \ 64,\ \ 8)$
- $(4,\ \ 32,\ \ 8)$
- $(8,\ \ 16,\ \ 8)$

La combinación $(16,\ \ 8,\ \ 8)$ no es posible ya que no cumple la condición $K \leq N$. Para cada combinación, la longitud en notación binaria de la clave criptográfica se calcula con la fórmula de la **Ecuación 3**, resultando en 512 de todas las combinaciones.

$$\text{len}(key) = K * N * \text{len}(L_{BIN}) \qquad \textbf{Ecuación 3}$$

De las posibles combinaciones, se determinó que la combinación óptima del trío de valores $(K,\ \ N,\ \ L)$ es $(8,\ \ 16,\ \ 8)$, cuyo análisis estadístico se considera en el siguiente capítulo. De esta manera, la red neuronal tiene la siguiente estructura:

- Neuronas en la capa oculta: 8
- Número de entradas por neurona: 16
- Rango de valores permitidos para pesos y entradas: $[-8,\ \ 7]$

Para empezar la sincronización, las dos partes deben crear la misma estructura para la red neuronal, generando valores aleatorios criptográficamente seguros para cada uno de los pesos sinápticos que la componen. Luego, una de las partes genera un vector aleatorio de entrada de longitud N y calcula su salida σ. Finalmente, se le envía a la otra parte el vector y la salida para que, de acuerdo a su salida, sean actualizados a no los pesos de ambas redes de acuerdo a la regla de aprendizaje hebbiana. Algo interesante de destacar es que únicamente se actualizan los pesos sinápticos de las neuronas cuya salida parcial sea igual a la salida global de la red. Esto hace muy difícil la tarea de un criptoanalista en tratar de adivinar qué pesos debe actualizar para imitar el comportamiento de las dos redes neuronales.

Este proceso, se repite varias veces hasta que las dos redes neuronales tengan los mismos pesos sinápticos, lo cual se convierte en la clave criptográfica que emplearán ambas partes para cifrar su información. Para ello, a cada uno de los pesos se le adiciona el valor de L asegurando valores positivos para cada uno, y se los transforma a su equivalente binario de 4 bits longitud resultando en una clave de 512 bits de longitud.

Para que pueda ser usado por el algoritmo criptográfico basado en ADN, cada par de bits de la clave se transforma en su nucleótido equivalente de acuerdo a la **Tabla *7***.

Tabla 7
Equivalencia binaria para cada nucleótido

Par binario	Nucleótido
00	C
01	A
10	T
11	G

3.3 Algoritmo criptográfico basado en ADN

Una vez establecida la clave criptográfica entre las dos partes, el algoritmo criptográfico basado en ADN lo usará para cifrar y descifrar los mensajes que se envíen y reciban. Tomando en cuenta dos de los cuatro procesos genéticos que ocurren dentro de la célula (transcripción y traducción), el algoritmo trata de imitar ese mismo comportamiento al cifrar y descifrar la información con el mensaje y la clave. El algoritmo se describe a continuación.

3.3.1 Preparación

En primer lugar, a partir de la clave criptográfica se convierte a su equivalente binario tomando en cuenta la **Tabla *7***. Luego, la clave criptográfica en notación binaria es el valor semilla para generar un diccionario aleatorio con todos los valores ASCII (0 - 255) y sus equivalentes nucleótidos, por ejemplo, como lo muestra la **Tabla *8***.

Tabla 8
Ejemplo de diccionario

Representación	Decimal	Binario	ADN
⋮			

CONTINÚA

A	65	0100 0001	AACT
B	66	0100 0010	GTAC
C	67	0100 0011	TAGG
D	68	0100 0100	GGGT
⋮			

Con estos valores, el algoritmo criptográfico puede realizar las tareas de cifrado y descifrado de mensajes en formato texto.

3.3.2 Transcripción

En esta fase, el texto a cifrar es transformado, carácter a carácter, a su equivalente en notación ADN empleando el diccionario generado anteriormente. Luego, la cadena de ADN es empleada para calcular su cadena de ADN complementaria, como lo describe la **Tabla 9**.

Tabla 9
Complemento a cada nucleótido

Nucleótido	Complemento
C	G
A	T
T	A
G	C

Finalmente, se calcula su equivalente binario de la cadena de ADN complementaria dividiéndolo en bloques de 128 bits. Si la longitud de la cadena de ADN complementaria

en notación binaria no es múltiplo de 128, el último bloque tendrá una longitud menor. Y a cada bloque se lo vuelve a dividir en bloques de 8 bits.

3.3.3 Traducción

Primero, se procede a calcular el resumen del texto. Por cada bloque de 128 bits, se extrae el valor decimal de cada bloque de 8 bits y se lo multiplica por su posición dentro del bloque de 128 bits. Luego se suman todos los valores calculados y se transforma a su equivalente binario de 16 bits de longitud, obteniéndose n números, donde n es la cantidad de bloques de 128 bits formados a partir de la cadena de ADN complementaria en notación binaria. A todos los números binarios se les aplica la operación de OR exclusivo para obtener finalmente un solo número binario de 16 bits de longitud.

A continuación, se procede a calcular subclaves a partir de la clave criptográfica. Para ello, se genera una cadena que almacena en notación binaria de 8 bits de longitud las posiciones, en sentido inverso, de cada uno de los nucleótidos de la clave criptográfica en el orden: A, C, T, G. Luego, a esta cadena se la divide en bloques de 128 bits obteniéndose siempre 16 bloques.

Para el proceso de cifrado, se recorre el resumen del texto bit a bit por cada bloque del texto, y, en caso de que el bit sea 1, se aplica la operación OR exclusivo al bloque del texto con la subclave correspondiente a la misma posición del bit. Finalmente, se recorre nuevamente el resumen del texto bit a bit para rotar los bits del texto (0 – rotación hacia la derecha y 1 – rotación hacia la izquierda). Luego de haberlo aplicado a todos los bloques, se adjunta el resumen al final de cada bloque del texto, para luego aplicar la misma operación nuevamente con los primeros 16 bits de cada bloque como nuevo resumen del texto.

Para terminar, a cada bloque se lo divide en bloques de 4 bits para calcular su equivalente hexadecimal y formar el texto cifrado que será enviado. Para el proceso de descifrado, se aplica las operaciones inversas.

3.4 Combinación de los algoritmos criptográficos

Los dos algoritmos mencionados anteriormente cumplen funciones específicas dentro del criptosistema: el algoritmo criptográfico neuronal permite establecer la clave criptográfica entre dos usuarios, y el algoritmo criptográfico basado en ADN permite cifrar y descifrar la información. Aunque tienen funcionalidades diferentes, son complementarios.

En primer lugar, cuando dos usuarios quieren establecer comunicación, el algoritmo criptográfico neuronal crea y sincroniza las redes para que los pesos sinápticos sean usados como una clave criptográfica. Y, en segundo lugar, el algoritmo criptográfico basado en ADN emplea la clave criptográfica para cifrar y descifrar la información.

3.5 Implementación de la aplicación

Para la implementación de la aplicación de chat se empleó Android Studio, en su versión 3.0. Todas las interfaces de usuario siguen la secuencia lógica definida en los requisitos funcionales de la aplicación, descritas en las historias de usuario. A continuación, se muestran las interfaces de acuerdo a cada una de las historias de usuario.

3.5.1 Registro e ingreso de usuario (HU01 y HU02)

Cuando un usuario entra por primera vez a la aplicación, se le desplegará una interfaz como lo muestra la **Figura 12**.

Figura 12. Interfaz gráfica inicial de la aplicación

Si el usuario ya tiene una cuenta creada, podrá pulsar el botón "YA TENGO UNA CUENTA", después de lo cual se le desplegará un formulario, como lo muestra la **Figura 13**, en el que podrá llenar sus datos y acceder a la aplicación.

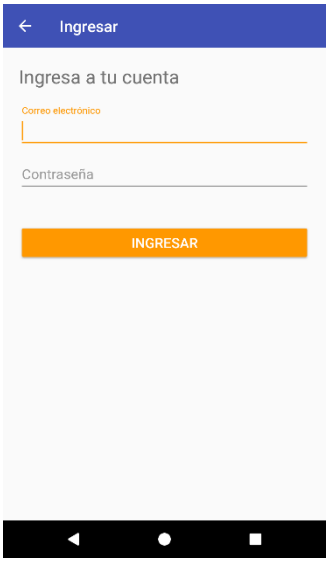


Figura 13. Formulario de ingreso a la aplicación

Si el usuario no tiene una cuenta creada, podrá pulsar el botón “CREAR UNA NUEVA CUENTA” después de lo cual se le desplegará un formulario, como lo muestra la **Figura 14**, en el que podrá ingresar sus datos de registro y acceder con ellos a la aplicación.

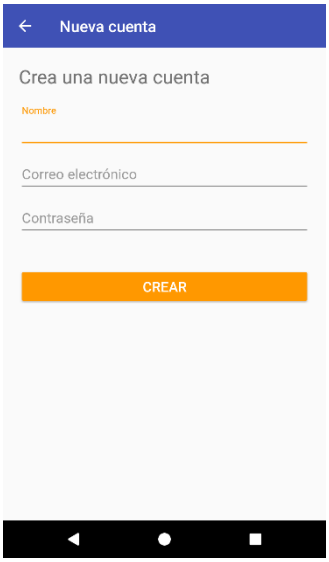


Figura 14. Formulario de registro de la aplicación

Una vez dentro de la aplicación, se mostrará la pantalla principal donde el usuario podrá acceder a las distintas opciones, como lo muestra la **Figura 15**.

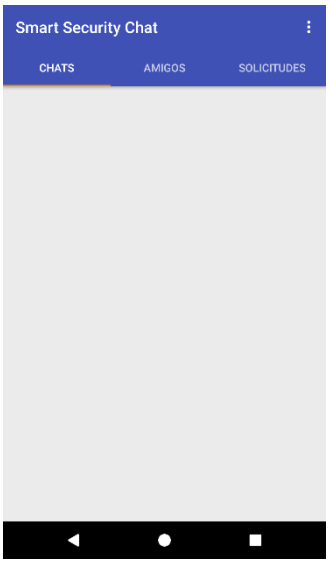


Figura 15. Pantalla principal de la aplicación

3.5.2 Enviar y aceptar solicitudes de amistad (HU04 y HU05)

Para que dos usuarios puedan enviar y/o recibir mensajes uno del otro, primero deben ser amigos dentro de la aplicación. Para ello, cuando un usuario ingresa a su cuenta dentro de la aplicación, contará con un menú en la parte superior derecha de la pantalla en el cual podrá realizar dichas acciones, como lo muestra la **Figura 16**.

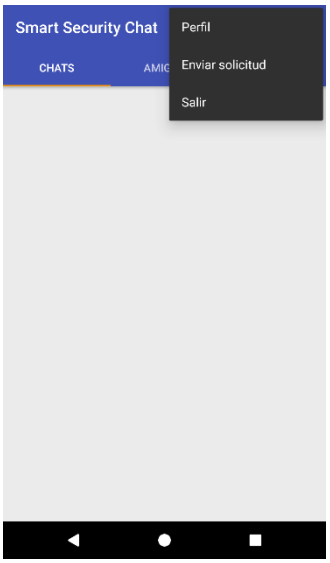


Figura 16. Menú de opciones

Cuando un usuario pulsa la opción "`Enviar solicitud`" se desplegará una lista con los usuarios registrados y a los cuales puede enviar una solicitud de amistad. La **Figura 17** muestra la pantalla que se despliega para que el usuario pueda realizar esta tarea.

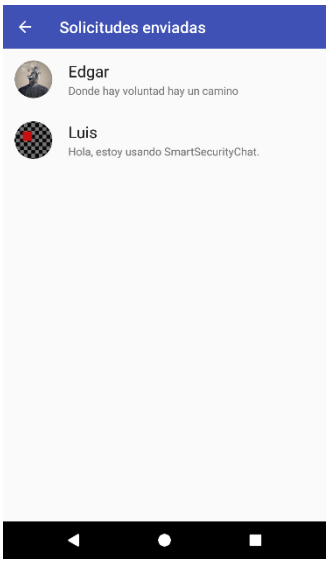


Figura 17. Pantalla de solicitudes

Cuando el usuario pulsa sobre otro usuario, se despliega la pantalla del perfil de dicho usuario, como lo muestra la **Figura 18**, junto con la opción de enviar una solicitud de amistad.

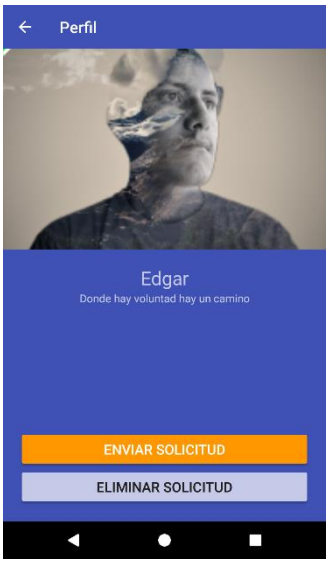


Figura 18. Pantalla de perfil de un usuario

Cuando el usuario pulsa en el botón de "`ENVIAR SOLICITUD`", la aplicación envía la solicitud al otro usuario, y muestra el perfil de dicho usuario, como lo muestra la **Figura 19**, junto con la opción de cancelar solicitud.

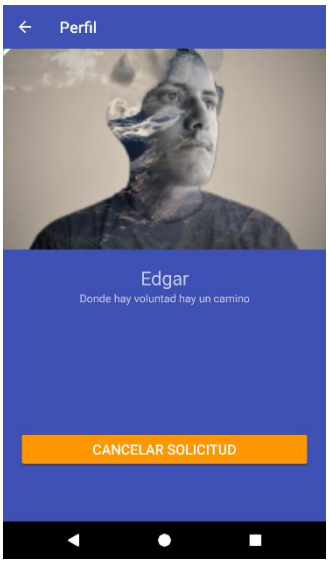


Figura 19. Pantalla de perfil de usuario con solicitud enviada

El usuario que recibe la solicitud podrá acceder a la misma interfaz, pero cuando accede al perfil del usuario del que recibe una solicitud amistad, se le mostrará el perfil de usuario con la opción de aceptar la solicitud enviada, como lo muestra la **Figura 20**.

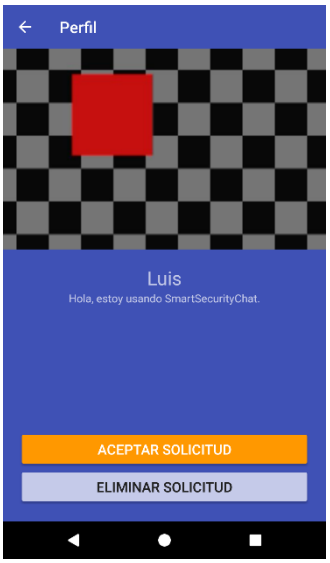


Figura 20. Pantalla de perfil de usuario cuando una solicitud ha sido enviada

En esta pantalla, el usuario tiene las opciones de "`ACEPTAR SOLICITUD`" y "`ELIMINAR SOLICITUD`". Cuando el usuario acepta la solicitud de amistad, la aplicación empieza con el proceso de creación y sincronización de las redes TPM para finalmente generar la clave criptográfica entre los dos usuarios. Durante el proceso de sincronización, se mostrará a los dos usuarios una pantalla como lo muestra la **Figura 21**.

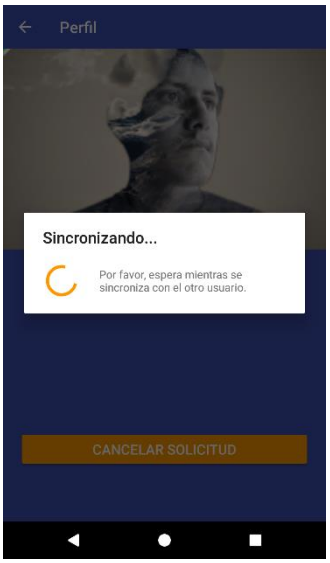


Figura 21. Mensaje de sincronización en proceso

Una vez terminado el proceso de sincronización, se agregará a cada uno el perfil del otro usuario en la pantalla de amistades, como lo muestra la **Figura 22**

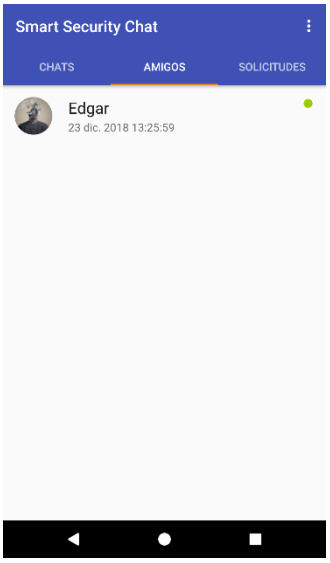


Figura 22. Pantalla de amistades del usuario

3.5.3 Enviar y recibir mensajes (HU06)

Si un usuario quiere enviar un mensaje a uno de sus amigos dentro de la aplicación, podrá ingresar a la pantalla de amistades y seleccionar el usuario a quien desea enviar un mensaje. Al seleccionarlo, se desplegará un menú de opciones como lo muestra la **Figura 23**.

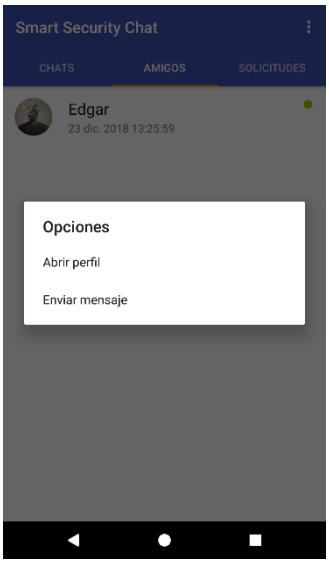


Figura 23. Menú de acciones

Al pulsar sobre la opción "`Enviar mensaje`", se mostrará la pantalla de conversaciones entre los dos usuarios, así como también la opción de enviar nuevos mensajes, como lo muestra la **Figura 24**.

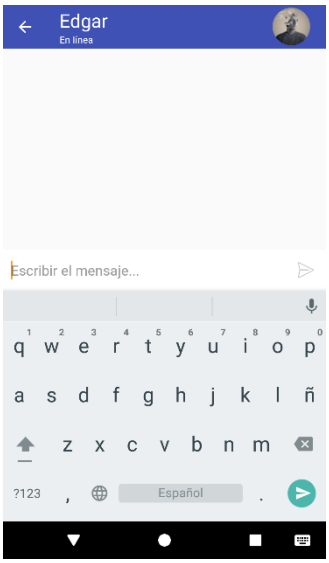


Figura 24. Pantalla de chat

Cuando un usuario ingresa y envía un mensaje de texto, el proceso de cifrado es totalmente transparente para los dos usuarios, como lo muestra la **Figura 25**.

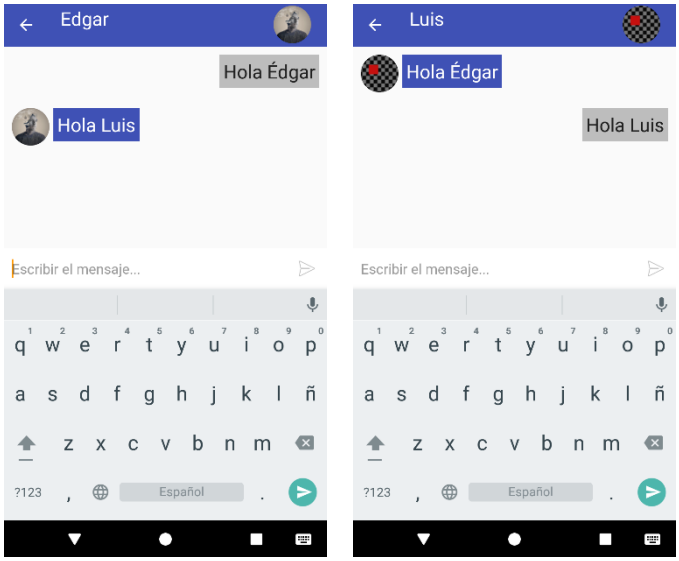


Figura 25. Pantalla de chat con mensajes enviados

3.5.4 Ver y modificar el perfil de usuario (HU03)

Si un usuario quiere modificar su perfil (foto y estado), podrá ingresar al menú mostrado en la **Figura 16** y seleccionar la opción "`Perfil`". Se le desplegará una pantalla donde

podrá ver su información actual de su perfil, y opciones para modificarlo, como lo muestra la **Figura 26**.

Figura 26. Perfil de usuario

3.6 Ejemplo de cifrado

A continuación, se muestra un ejemplo de cifrado empleando el criptosistema basado en ADN del mensaje "`mi gato está loco`" con una clave previamente establecida entre dos usuarios. Un ejemplo de clave podría ser la siguiente:

```
GTATTTTCTCAGCGGGGTCAAGCGGCATTAACCTTACAACGCTCTTCAGGCGCTTGCCAGAACAACTAGAG
AGTCTCAAACTCATCATAATGAGTCAAATCATTCTTTCATATAGCCGGAAACATCCAAGTGCGAGCTATCT
TTATACGGACGGACCACGTGTCAGTTGCCGCGGGTTCGGCCTAAGAGGCGGAGTCCTCCTCCATTGGAGGC
TGACTGTTCACATTACAAAGTATTACACTAGGTTGTAACATAG
```

Para empezar el proceso de cifrado, se crea el diccionario de todos los caracteres del código ASCII dispuestos de forma aleatoria empleando la clave como valor semilla. Entonces, el mensaje se lo transforma a secuencias de ADN, y posteriormente se calcula su secuencia complementaria. A este valor se lo transforma a su equivalente binario resultando en la siguiente cadena:

```
10110011000110101010111001000000111100110010001111011100101011101100001
00000100000100011111100111010111011011011110111001001010111011100
```

A la secuencia complementaria binaria del texto se la divide en bloques de 128 bits. Luego, a cada bloque de 128 bits se lo divide nuevamente en bloques de 8 bits, que da como resultado la siguiente matriz:

```
[[10110011, 00011010, 10101110, 01000000, 11110011, 00100011, 11011100,
10101110, 11000010, 00001000, 00100011, 11110011, 10101110, 11011011,
11011100, 10010101], [11011100]]
```

Ahora, por cada bloque de 128 bits, se calcula el producto de cada bloque de 8 bits en su notación decimal por su posición dentro del bloque de 128 bits incrementado en una unidad, para finalmente sumar todos los valores. Por ejemplo, de la siguiente manera.

```
[[179, 26, 174, 64, 243, 35, 220, 174, 194, 8, 35, 243, 174, 219, 220,
149], [220]]


[179*1+26*2+174*3+64*4+243*5+35*6+220*7+174*8+194*9+8*10+35*11+243*12+1
74*13+219*14+220*15+149*16, 220*1] = [21505, 220]
```

A cada valor obtenido se lo transforma a su equivalente binario de 16 bits de longitud y se realiza la operación de OR exclusivo entre todos ellos. Por ejemplo, como se muestra a continuación:

```
[21505, 220] = [0101010000000001, 0000000011011100]
0101010000000001 ⊕ 0000000011011100 = 0101010011011101
```

Ahora, se calculan las subclaves a partir de la clave principal. Para ello, se almacena cada posición por cada nucleótido en la clave de forma inversa en su equivalente binario de 8 bits de longitud. Luego, se crean bloques de 128 bits de longitud concatenando todos los valores calculados, lo que resulta en 16 subclaves. Por ejemplo, de la siguiente manera:

A = [00000010, 00001010, 00010011, 00010100, 00011010, 00011101, 00011110, 00100011, 00100101, 00100110, 00101111, 00111010, 00111100, 00111101, 00111111, 01000000, 01000011, 01000101, 01000111, 01001101, 01001110, 01001111, 01010011, 01010110, 01011000, 01011001, 01011100, 01100000, 01100001, 01100010, 01100101, 01101101, 01101111, 01110001, 01110111, 01111000, 01111001, 01111011, 01111111, 10000000, 10000110, 10001010, 10010000, 10010010, 10010110, 10011010, 10011101, 10100100, 10111000, 10111001, 10111011, 11000001, 11001100, 11010001, 11010111, 11011110, 11100000, 11100011, 11100101, 11100110, 11100111, 11101010, 11101101, 11101111, 11110010, 11111001, 11111010, 11111100, 11111110]

C = [00000111, 00001001, 00001100, 00010010, 00010110, 00011001, 00011111, 00100000, 00100100, 00100111, 00101001, 00101011, 00101110, 00110010, 00110100, 00111000, 00111001, 00111110, 01000001, 01001010, 01001100, 01010000, 01010010, 01010101, 01011111, 01100100, 01101000, 01101100, 01110011, 01110100, 01111010, 01111101, 01111110, 10000100, 10001000, 10001100, 10010011, 10010111, 10011011, 10011100, 10011110, 10100011, 10101001, 10101010, 10101100, 10110010, 10110101, 10110110, 10111110, 11000100, 11000101, 11000111, 11001000, 11001010, 11001011, 11010100, 11011000, 11011101, 11011111, 11100100, 11101110, 11110000, 11111011]

G = [00000000, 00001011, 00001101, 00001110, 00001111, 00010000, 00010101, 00010111, 00011000, 00101000, 00110000, 00110001, 00110011, 00110111, 00111011, 01000100, 01000110, 01001000, 01011011, 01011101, 01110010, 01110101, 01110110, 10000001, 10000011, 10000101, 10000111, 10010100, 10010101, 10011000, 10011001, 10011111, 10100001, 10100101, 10101000, 10101011, 10101101, 10101110, 10101111, 10110011, 10110100, 10111010, 10111100, 10111101, 10111111, 11000000, 11000010, 11001111,

11010000, 11010010, 11010011, 11010110, 11011010, 11101000, 11110011, 11110100, 11110111, 11111111]

T = [00000001, 00000011, 00000100, 00000101, 00000110, 00001000, 00010001, 00011011, 00011100, 00100001, 00100010, 00101010, 00101100, 00101101, 00110101, 00110110, 01000010, 01001001, 01001011, 01010001, 01010100, 01010111, 01011010, 01011110, 01100011, 01100110, 01100111, 01101001, 01101010, 01101011, 01101110, 01110000, 01111100, 10000010, 10001001, 10001011, 10001101, 10001110, 10001111, 10010001, 10100000, 10100010, 10100110, 10100111, 10110000, 10110001, 10110111, 11000011, 11000110, 11001001, 11001101, 11001110, 11010101, 11011001, 11011011, 11011100, 11100001, 11100010, 11101001, 11101011, 11101100, 11110001, 11110101, 11110110, 11111000, 11111101]

Subkeys =
[111111101111110011111010111110011111001011101111111011011110101011001
111110011011100101111000111111000001101111011010111110100001,
110011001100000110111011101110011011100010100100100111011001101010010111
010010010100100001000101010000110100000000111111101111011,
011110010111100001110111011100010110111101101101011001010110001001100000
101100000010111000101100101011000010101100101001101001111,
010011100100110101000111010001010100001101000000001111110011110100111110
000111010001011110010011000100101001000110001111000011101,
000110100001010000010011000010100000001011111011111100001110111011100101
011011111110111011101100011010100110010111100101011001000,
110001111100010111000100101111101011011010110101101100101010110010101010
010101001101000111001111010011100100110111001011110010011,
100011001000100010000100011111100111110101111010011101000111001101101100
001101000011001000101111101010101010010010100000100110011,

01001010010000010011111000111001001110000011010000110010001011100010101
10010100100100111001001000010000000011111000110010001011O,

00010010000001100000010010000011111111111111011111110100111100111110100
01101101011010110110100111101001011010000110011111110000010,

11000000101111111011110110111100101110101011010010110011101011111010111
0101011011010101110101000101001011010000110011111110011001,

10011000100101011001010010000111100001011000001110000000101110110011101O
10111001001011101010110110100100001000110010001000011101l,

00110111001100110011000100110000000101000000110000000101110001010100010OO
00000111100001110000011010000101100000000111111011111100O,

11110110111101011111000111101100111010111110100111100010111000011101110
01101101111011001110101011100111011001101110010011100011O,

11000011101101111011000110110000101001111010011010100010101000001001000
11000111110001110100011011000101110001001100000100111110O,

01110000011011100110101101101010011010010110011101100110011000110101111
00101101001010111010101000101000101001011010010010100001O,

00110110001101010010110100101100001010100010001000100001000111000001101
10001000100001000000000110000000101000000100000000011000000O1]

A continuación, se recorre el resumen del texto 0101010011011101 bit a bit, y, si el bit es igual a uno se aplica la subclave correspondiente a la posición del bit mediante la operación OR exclusivo con el texto binario complementario, lo que resulta en lo siguiente:

[000100001010010001111110111110011011010101000100111111011011000111000O
1000110100000100111110100110101101110100101100111101111000, 01111111]

Luego, se recorre una vez más el resumen del texto bit a bit, y, si el bit es igual a 0, se realiza una rotación de los bits hacia la derecha. Caso contrario, se realiza una rotación hacia la izquierda. Finalmente, se adjunta el resumen al final de cada bloque del texto. Por ejemplo, el texto quedaría de la siguiente forma:

[01000010100100011111101111100110110101010001001111110110110001110000100110100000100111110100110101101110100101100111101111000000101010011011101, 11111101010100110111011

Se realiza los mismos pasos anteriores, pero tomando como resumen del texto los primeros 16 bits de cada bloque del texto (`0100001010010001` y `1111110101010100`). El texto quedaría de la siguiente forma.

[01000010100100010100011010100010100101111111111001111110011101101010000100011110000110101011010011000100100110100011000110010111001100100101011, 11111101010100100110111

Finalmente, por cada bloque de 4 bits de longitud del texto resultante se lo transforma a su equivalente hexadecimal, lo que resulta en el texto cifrado final. Para el caso anterior, el texto cifrado sería: `429146a297ff3f3b508f0d5a624d18cb992bfd54b7`. Para realizar el proceso de descifrado, se realiza las mismas operaciones en forma inversa.

3.7 Pruebas de rendimiento

A continuación, se muestra el rendimiento del algoritmo al cifrar textos de diferentes longitudes. La **Tabla *10*** muestra el tiempo que le toma al algoritmo cifrar un mensaje.

Tabla 10
Tiempo de cifrado

Cantidad de caracteres	Tiempo de cifrado [s]
1	0.002
10	0.002
50	0.003
100	0.004

CONTINÚA

1,000	0.024
10,000	0.591
50,000	14.080

Como se puede observar en la **Tabla *10***, conforme se incrementa la cantidad de caracteres del mensaje, se incrementa el tiempo necesario para calcular el texto cifrado. Es interesante notar cómo, a partir de los 10,000 caracteres, el tiempo de cifrado se incrementa en un 2,382.4%, mientras la cantidad de caracteres se incrementa en un 500%. Con pocos caracteres, el tiempo de cifrado es similar. La **Figura 27** muestra de forma gráfica lo explicado.

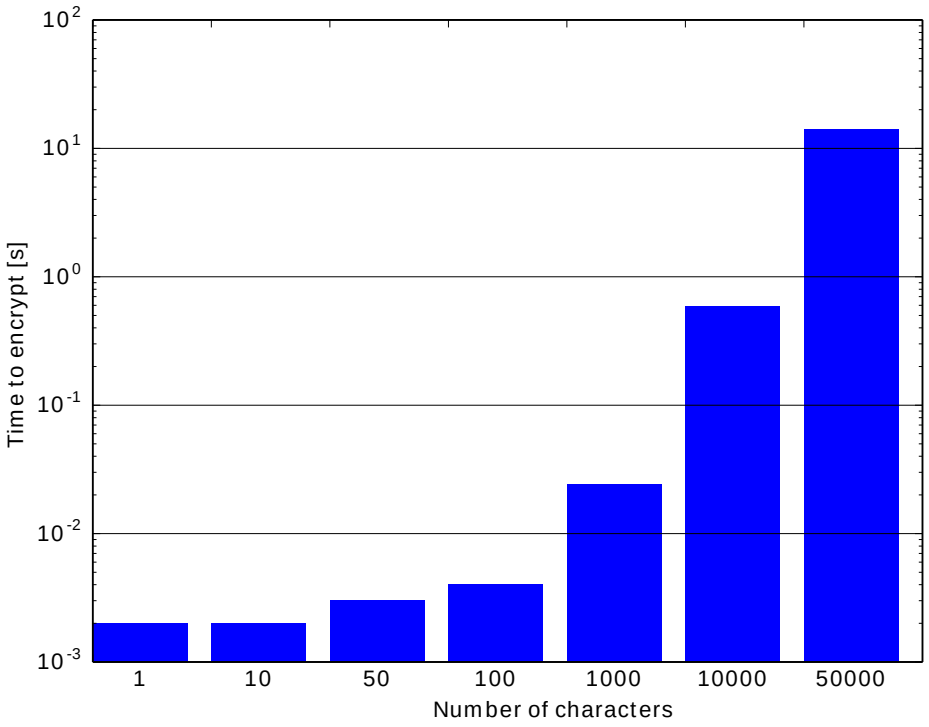


Figura 27. Evolución del tiempo de sincronización con respecto a la cantidad de caracteres

3.8 Implementación del algoritmo del modelo de ataque de fuerza bruta

El modelo de ataque de fuerza bruta (o una búsqueda exhaustiva de la clave) tiene como objetivo probar todas las claves posibles, dado un texto cifrado, para encontrar el mensaje original. Para la implementación del algoritmo que realizará este ataque, se empleó Java, ya que la librería de cifrado se encuentra en dicho lenguaje. La **Figura 28** muestra el diagrama de clases del algoritmo.

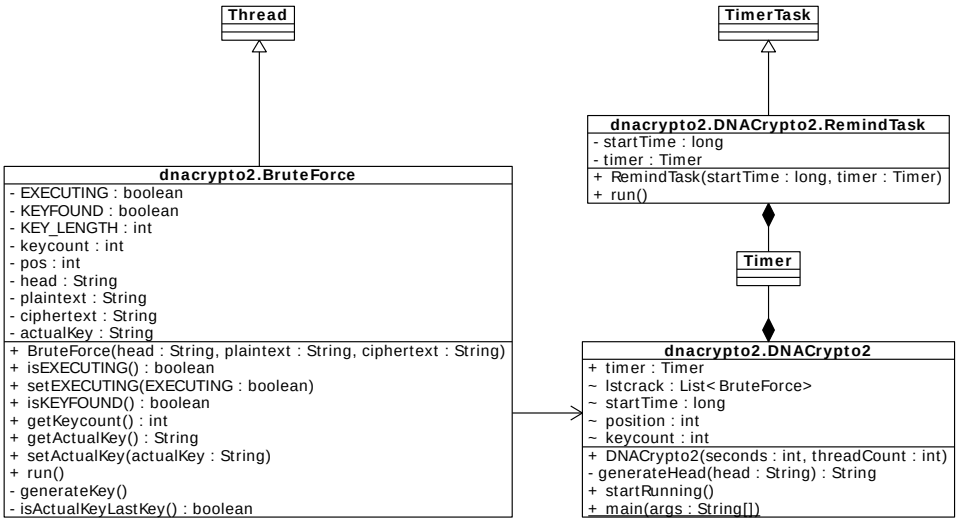


Figura 28. Diagrama de clases del algoritmo para el ataque de fuerza bruta

En la **Figura 28** se aprecia principalmente tres clases empleadas para realizar el ataque. La primera, `RemindTask`, que se encarga de controlar el tiempo del ataque. Ya que el este modelo de ataque requiere gran cantidad de tiempo para su ejecución, se realizaron pruebas con tiempos cortos para proyectarlos a la totalidad de la cantidad de claves. En total se emplearon 4 espacios de tiempos (5, 10, 20 y 3600 segundos). A continuación, se muestra el código empleado.

```
class RemindTask extends TimerTask {

        private long startTime;
        private Timer timer;

        public RemindTask(long startTime, Timer timer) {
```

```
            this.startTime = startTime;
            this.timer = timer;
        }

        @Override
        public void run() {
            int keysTested = 0;
            for (BruteForce bruteForce : lstcrack) {
                bruteForce.setEXECUTING(false);
                keysTested += bruteForce.getKeycount();
                bruteForce.stop();
            }
            System.out.println("END TIME: " +
System.currentTimeMillis() + " (" + ((System.currentTimeMillis() -
startTime) / 1000.0) + " sec.)");
            System.out.println("KEYS TESTED: " + keysTested + "\n");
            this.cancel();
            this.timer.cancel();
            this.timer.purge();

        }
    }
```

La función `run()` de esta clase se ejecutará una vez concluido el tiempo especificado al crear el objeto. De esta manera, se ejecutará por un tiempo determinado y mostrará los resultados obtenidos durante la realización del ataque. Principalmente, se mostrará el tiempo de la prueba y la cantidad de claves que se pudo probar. En segundo lugar, está la clase `BruteForce`, que se encarga de ejecutar el ataque con una clave generada. A continuación, se muestra el código empleado.

```
public class BruteForce extends Thread {

    private boolean EXECUTING = true;
    private boolean KEYFOUND = false;
    private final int KEY_LENGTH = 256;
    private int keycount = 0;
    private int pos = 0;
    private final String head;
    private final String plaintext;
    private final String ciphertext;
    private String actualKey;
```

```
    public BruteForce(String head, String plaintext, String ciphertext)
{
        this.head = head;
        this.plaintext = plaintext;
        this.ciphertext = ciphertext;
        this.actualKey = this.head;
        while (this.actualKey.length() < KEY_LENGTH) {
            this.actualKey += "A";
        }
    }

    public boolean isEXECUTING() { return EXECUTING; }
    public void setEXECUTING(boolean EXECUTING) { this.EXECUTING =
EXECUTING;
    public boolean isKEYFOUND() { return KEYFOUND; }
    public int getKeycount() { return keycount; }
    public String getActualKey() { return actualKey; }
    public void setActualKey(String actualKey) { this.actualKey =
actualKey; }

    @Override
    public void run() {
        while (EXECUTING) {
            keycount++;
            String plaintext2 = DNA2.decrypt(ciphertext, actualKey);
            if (plaintext.equals(plaintext2)) {
                EXECUTING = false;
                KEYFOUND = true;
                System.out.println("KEY FOUND: " + actualKey);
                System.out.println("TEXT: " + plaintext2);
            } else if (isActualKeyLastKey()) {
                EXECUTING = false;
            }
            generateKey();
        }
    }

    private void generateKey() {
        char[] nucleotides = {'A', 'C', 'T', 'G'};
        char[] key = actualKey.toCharArray();
        key[key.length - 1] = nucleotides[pos++];
        if (keycount == 1) {
```

```
            key[key.length - 1] = nucleotides[1];
        }
        if (pos > 3) {
            pos = 0;
        }
        if (key[key.length - 1] == 'A') {
            for (int i = key.length - 2; i >= head.length() - 1; i--) {
                if (key[i] == 'A') {
                    key[i] = 'C';
                    break;
                } else if (key[i] == 'C') {
                    key[i] = 'T';
                    break;
                } else if (key[i] == 'T') {
                    key[i] = 'G';
                    break;
                } else if (key[i] == 'G') {
                    key[i] = 'A';
                }
            }
        }
        actualKey = String.valueOf(key);
    }

    private boolean isActualKeyLastKey() {
        String lastKey = "";
        while (lastKey.length() < KEY_LENGTH) {
            lastKey += "G";
        }
        return actualKey.equals(lastKey);
    }

}
```

La clase BruteForce hereda de la clase Thread de Java, por lo que, al ejecutarla, se creará como un proceso hijo del proceso principal, ejecutándose de forma paralela con los demás procesos. En el constructor de la clase creará la cabecera de la clave que será probada por el objeto, para que a partir de dicha clave se generen el resto de claves con la misma cabecera y no existan colisiones de claves con los demás procesos. De esta manera, cada proceso hijo probará un conjunto único de claves. En la función `run()` se realiza el proceso de descifrado del texto cifrado con la clave generada. En caso de que

el texto descifrado corresponda al mensaje original, la clave generada es la clave empleada al cifrar la información, y, por lo tanto, el ataque ha sido exitoso. Caso contrario, se generará una nueva clave para repetir el proceso con la función `generateKey()`. Si la clave generada, luego de ser probada y rechazada, corresponde a la última clave posible que puede probar, se detiene el proceso actual.

Finalmente, la clase `DNACrypto2` empleará las dos clases mencionadas para crear todos los procesos hijos y realizar el ataque. A continuación, se muestra el código empleado.

```
public class DNACrypto2 {

    Timer timer;
    List<BruteForce> lstcrack;
    long startTime;
    int position = 0;
    int keycount = 0;

    public DNACrypto2(int seconds, int threadCount) {
        lstcrack = new ArrayList<>();
        String head = "";
        for (int i = 0; i < threadCount; i++) {
            head += "A";
        }
        for (int i = 0; i < (int) Math.pow(4, threadCount); i++) {
            lstcrack.add(new BruteForce(head, "mi gato esta loco",
"429146a297ff3f3b508f0d5a624d18cb992bfd54b7"));
            head = generateHead(head);
        }
        timer = new Timer();
        startTime = System.currentTimeMillis();
        timer.schedule(new RemindTask(startTime, timer), seconds *
1000);
    }

    private String generateHead(String head) {
        char[] nucleotides = {'A', 'C', 'T', 'G'};
        char[] key = head.toCharArray();
        key[key.length - 1] = nucleotides[position++];
        if (keycount == 1) {
```

```
            key[key.length - 1] = nucleotides[1];
        }
        if (position > 3) {
            position = 0;
        }
        if (key[key.length - 1] == 'A') {
            for (int i = key.length - 2; i >= 0; i--) {
                if (key[i] == 'A') {
                    key[i] = 'C';
                    break;
                } else if (key[i] == 'C') {
                    key[i] = 'T';
                    break;
                } else if (key[i] == 'T') {
                    key[i] = 'G';
                    break;
                } else if (key[i] == 'G') {
                    key[i] = 'A';
                }
            }
        }
        return String.valueOf(key);
    }

    public void startRunning() {
        System.out.println("START TIME: " + startTime);
        for (BruteForce bruteForce : lstcrack) {
            bruteForce.start();
        }
    }

    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        System.out.println("START TIME: " + start);
        DNACrypto2 bf1 = new DNACrypto2(Integer.parseInt(args[0]),
Integer.parseInt(args[1]));
        System.out.println("END TIME: " + System.currentTimeMillis() +
" (" + ((System.currentTimeMillis() - start) / 1000.0) + " sec.)");
        bf1.startRunning();
    }

}
```

En esta clase, se toma como parámetros iniciales los segundos que durará el ataque y la cantidad de threads que será creados. Adicionalmente, se emplea como mensaje el texto “`mi gato esta loco`” cuyo texto cifrado se considera como ejemplo en la subsección **3.6**. La función `generateHead()` generará una cabecera única y evitará redundancias de la clave entre dos o más procesos hijos. Finalmente, la función `startRunning()` comenzará la ejecución de cada uno de los procesos hijos que se encargarán de las tareas mencionadas anteriormente. Para cada una de las pruebas, se toma en consideración el tiempo real empleado durante el ataque y la cantidad de claves que fue posible probar. El análisis estadístico de dichos resultados se muestra en el siguiente capítulo.

CAPÍTULO 4

EVALUACIÓN DE RESULTADOS, VALIDACIÓN Y DISCUSIÓN

En este capítulo se detalla el proceso estadístico empleado para determinar la estructura óptima de las redes TPM. Además, se detalla los modelos de ataque que se emplearon para medir el nivel de seguridad del criptosistema propuesto.

4.1 Procesamiento estadístico

Para encontrar la estructura óptima de las redes TPM, se realizaron simulaciones con todas las posibles combinaciones de los valores de K, N y L que permitan la generación de una clave criptográfica de 512 bits de longitud. Debido a esto, existen varias combinaciones de los valores de K, N y L que permiten generar una clave de esa longitud. Por ello, se analizaron varias combinaciones, probando su rendimiento y seguridad, para determinar el trío de valores óptimos, como lo muestra la **Tabla *11***. Adicionalmente, se tomó en cuenta la restricción de $K \leq N$, ya que en la simulaciones realizadas se determinó que si $K > N$, la red puede tomar valores que hacen imposible alcanzar la sincronización.

Tabla 11

Combinaciones de los valores de K, N y L para generar claves criptográficas de 512 bits de longitud

L	$\text{len}(L_{BIN})$	K	N
$\mathbf{2^0 = 1}$	1	1	512
		2	256
		4	128
		8	64

CONTINÚA

		16	32
$\mathbf{2^1 = 2}$	2	1	256
		2	128
		4	64
		8	32
		16	16
$\mathbf{2^3 = 8}$	4	1	128
		2	64
		4	32
		8	16
$\mathbf{2^7 = 128}$	8	1	64
		2	32
		4	16
		8	8
$\mathbf{2^{15} = 32768}$	16	1	32
		2	16
		4	8
$\mathbf{2^{31}}$ $\mathbf{= 2147483648}$	32	1	16
		2	8

CONTINÚA

		4	4

Las combinaciones donde $L > 2^{31}$ (2^{63}, 2^{127}, 2^{255} y 2^{511}) no fueron considerados para las simulaciones por su longitud y complejidad en el cálculo. Por lo tanto, existen 24 combinaciones de los valores de K, N y L que pueden estructurar las redes TPM para su sincronización.

Para realizar las simulaciones y determinar la combinación óptima, se desarrolló un algoritmo empleando Python que permita contabilizar los resultados y generar gráficos con dichos resultados. De forma paralela, se realizó un modelo de ataque en el cual, de forma pasiva, una red atacante tratará de imitar el comportamiento de las otras dos redes, para determinar la clave criptográfica que se está estableciendo. La **Figura 29** muestra el diagrama de clases empleado para el algoritmo.

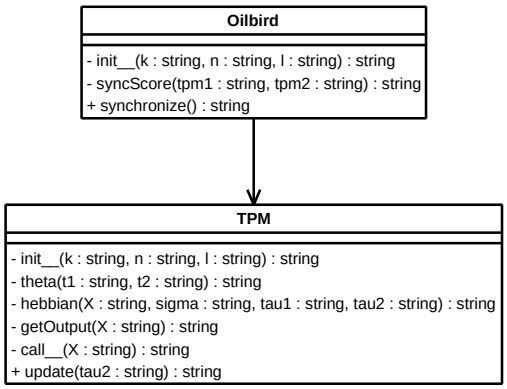


Figura 29. Diagrama de clases para las simulaciones

El diagrama de clases de la **Figura 29** tiene la clase `TPM`, que maneja todo lo relacionado con las redes Tree Parity Machine. La clase `Oilbird` se encarga de crear las dos redes que sincronizarán sus pesos y la red adicional que será la atacante. Entonces, cada una de las simulaciones consta de los siguientes pasos:

1. Crear tres redes TPM, dos autorizadas y una atacante.

2. Inicializar los contadores de la cantidad de pasos y sincronizaciones exitosas de la red atacante a cero.

3. Sincronizar los pesos de las redes incrementando los contadores según sea necesario.

A continuación, se muestra el código de la clase `Oilbird` que sigue dichos pasos en cada una de sus funciones.

```
import numpy as np
from tpm import TPM

class Oilbird:
    def __init__(self, k, n, l):
        self.k = k
        self.n = n
        self.l = l
        self.alice = TPM(k, n, l)
        self.bob = TPM(k, n, l)
        self.eve = TPM(k, n, l)

    def __syncScore(self, tpm1, tpm2):
        return 1.0 - np.average(1.0 * np.abs(tpm1.W - tpm2.W)/(2 *
self.l))

    def synchronize(self):
        sync = False
        updates = 0
        eveUpdates = 0
        while(not sync):
            x = np.random.randint(-self.l, self.l, [self.k, self.n])
            tauA = self.alice(x)
            tauB = self.bob(x)
            tauE = self.eve(x)
            self.alice.update(tauB)
            self.bob.update(tauA)
            if tauA == tauB == tauE:
                self.eve.update(tauA)
                eveUpdates += 1
            updates += 1
            score = 100 * self.__syncScore(self.alice, self.bob)
```

```
            if score >= 100:
                sync = True
        return {'updates': updates, 'eve_score':
self.__syncScore(self.alice, self.eve), 'eve_updates': eveUpdates}
```

Como primer punto se importan las librerías necesarias. En el constructor de la clase se crean las tres redes TPM con los valores de K, N y L proporcionados. La función `syncScore` calculará el porcentaje de sincronización que hayan alcanzado las redes después de una actualización. Si dicho porcentaje es 100%, las redes tienen los mismos pesos sinápticos. La función `synchronize` ejecutará la sincronización entre las redes inicializando los contadores y ejecutando un bucle que termina si las redes tienen los mismos pesos sinápticos. Esta función retorna un diccionario con los siguientes valores: la cantidad de pasos que fueron necesarios para sincronizar a las dos redes autorizadas (`updates`), el porcentaje de sincronización alcanzado por la red atacante (`eve_score`), y la cantidad de pasos realizados por la red atacante (`eve_updates`). Al tratarse de un modelo de ataque pasivo, la red atacante sólo puede actualizar sus pesos si su salida es igual a las salidas de las otras dos redes.

Para determinar la combinación óptima de los valores de K, N y L, se realizaron 500,000 simulaciones por cada una. En cada una se midió el tiempo que les tomó a las redes sincronizarse. La **Tabla *12*** muestra los resultados obtenidos de las simulaciones.

Tabla 12

Resultados obtenidos de las 500,000 simulaciones por cada combinación

Ítem	Combinación $(K,\ N,\ L)$	Pasos mín.	Pasos máx.	Pasos media	Tiempo mín. [seg]	Tiempo máx. [seg]	Tiempo media [seg]	Eve sinc. exitosas	% Eve sinc. exitosas
1	$(1,\ 512,\ 2^0)$	5	28	9.3367	0.0657	2.2718	0.4259	368680	73.736
2	$(2,\ 256,\ 2^0)$	5	25	9.3312	0.0596	1.5588	0.4253	368229	73.6458
3	$(4,\ 128,\ 2^0)$	5	32	9.3374	0.0680	2.1275	0.4263	368471	73.6942
4	$(8,\ 64,\ 2^0)$	4	27	9.3370	0.0656	1.9089	0.4233	368652	73.7304

CONTINÚA

5	$(16,\ 32,\ 2^0)$	5	32	9.3356	0.0661	1.4625	0.4249	368389	73.6778
6	$(1,\ 256,\ 2^1)$	8	45	15.5450	0.0573	1.4665	0.3824	355397	71.0794
7	$(2,\ 128,\ 2^1)$	8	25371	15.7774	0.0523	537.5299	0.3857	349495	69.899
8	$(4,\ 64,\ 2^1)$	8	4104	18.0226	0.0491	95.8512	0.3965	308691	61.7382
9	$(8,\ 32,\ 2^1)$	9	814	31.6046	0.0403	19.6915	0.4597	161148	32.2296
10	$(16,\ 16,\ 2^1)$	15	587	73.4867	0.0289	7.3628	0.5381	8102	1.6204
11	$(1,\ 128,\ 2^3)$	11	78	25.3284	0.0472	1.1114	0.2983	132715	26.543
12	$(2,\ 64,\ 2^3)$	15	289	65.2706	0.0411	2.7924	0.4699	41475	8.295
13	$(4,\ 32,\ 2^3)$	26	616	128.1582	0.0309	4.5353	0.5325	487	0.0974
14	$(\mathbf{8},\ \mathbf{16},\ \mathbf{2^3})$	**38**	**769**	**211.6590**	**0.0262**	**6.0476**	**0.5741**	**0**	**0.0**
15	$(1,\ 64,\ 2^7)$	10	71	25.0318	0.0180	0.6389	0.1679	132666	26.5332
16	$(2,\ 32,\ 2^7)$	15	372	74.9943	0.0362	1.9875	0.3599	39111	7.8222
17	$(4,\ 16,\ 2^7)$	23	712	137.9569	0.0193	2.7179	0.4572	759	0.1518
18	$(\mathbf{8},\ \mathbf{8},\ \mathbf{2^7})$	**30**	**955**	**218.2211**	**0.0342**	**3.4490**	**0.5109**	**0**	**0.0**
19	$(1,\ 32,\ 2^{15})$	7	68	21.3921	0.0105	0.2913	0.0847	144461	28.8922
20	$(2,\ 16,\ 2^{15})$	10	451	69.8188	0.0196	1.2301	0.2161	61628	12.3256
21	$(4,\ 8,\ 2^{15})$	14	777	132.0805	0.0181	2.7993	0.3200	4080	0.816
22	$(1,\ 16,\ 2^{31})$	6	84	24.9428	0.0020	0.2366	0.0549	46244	9.2488
23	$(2,\ 8,\ 2^{31})$	10	804	100.5999	0.0121	1.4193	0.1842	25006	5.0012
24	$(4,\ 4,\ 2^{31})$	9	837	134.2786	0.0032	2.0644	0.2281	9168	1.8336

En la **Tabla *12***, las columnas Pasos mín. y Pasos máx., muestran la cantidad mínima y la cantidad máxima de pasos que les tomó a las dos redes para sincronizar sus pesos. La columna Pasos media muestra el promedio de pasos de la totalidad de las simulaciones realizadas. Las columnas Tiempo mín. y Tiempo máx., muestran el tiempo

mínimo y tempo máximo (en segundos) que les tomó a las dos redes para sincronizar sus pesos. La columna Tiempo Media muestra el promedio de tiempo (en segundos) de la totalidad de las simulaciones realizadas. La columna Eve sinc. exitosas, muestra la cantidad de ocasiones en las que la red atacante logró imitar el comportamiento de las otras dos redes del total de simulaciones realizadas. Finalmente, la columna % Eve sinc. exitosas, muestra el porcentaje que representa la columna anterior con respecto al total de las simulaciones.

Como se puede observar en la **Tabla *12***, los ítems 14 y 18 que corresponden a las combinaciones $(8,\ 16,\ 2^3)$ y $(8,\ 8,\ 2^7)$ respectivamente obtuvieron los mejores resultados desde el punto de vista de seguridad frente a la red atacante (Eve). En ninguna de las 500,000 simulaciones, Eve logró imitar el comportamiento de las otras dos redes con la misma cantidad de pasos que las redes de Alice y Bob. Sin embargo, las dos combinaciones están entre las que les tomó un mayor número de pasos para lograr la sincronización (211.659 y 218.2211 respectivamente) que el resto de combinaciones, lo cual involucra un mayor tiempo promedio.

Desde el punto de vista del tiempo de sincronización, el ítem 22 que corresponde a la combinación $(1\ \ 16,\ 2^{31})$ obtuvo el mejor resultado, con un tiempo promedio de sincronización de 0.0549 segundos. Sin embargo, tiene un porcentaje de sincronización de la red atacante de 9.2488%. Cabe resaltar también que las combinaciones donde $L = 2^0 = 1$, a pesar de tener los mejores tiempos en sincronización, obtuvieron los peores resultados desde el punto de vista de seguridad, ya que la red atacante Eve logró imitar el comportamiento de las otras dos redes en aproximadamente 73.7% del total de las simulaciones.

Como se puede observar, la probabilidad de éxito de una red atacante tiene una dependencia muy notoria con respecto a los valores de K, N y L. Por ello, se procedió a realizar un análisis de los valores de K, N y L con respecto a su probabilidad. Se graficó todas las combinaciones posibles con sus respectivas probabilidades. Para ello, se

empleó la función `scatter3` GNU Octave para dibujar el diagrama de dispersión. El siguiente algoritmo muestra el código ejecutado para obtener la gráfica.

```
scatter3(K(:), N (:), L(:), [16], probabilities(:), ’.’);
set(gca, ’zscale’, ’log’);
xlabel ”K”;
ylabel ”N”;
zlabel ”L”;
colorbar;
```

Con la función `scatter3`, se cargar los parámetros de K, N y L que corresponden a los valores de los ejes x, y y z respectivamente. El valor de 16 indica el tamaño del punto a graficar. El parámetro `probabilities` indica el color que tomará el punto en la gráfica. La función `set` define la escala del eje z (valor de L) a una escala logarítmica para mejorar la visualización. A continuación, se cambian las etiquetas de los ejes a su valor respectivo. Finalmente, se mostrará una barra de color. La Figura 2 muestra el resultado del algoritmo.

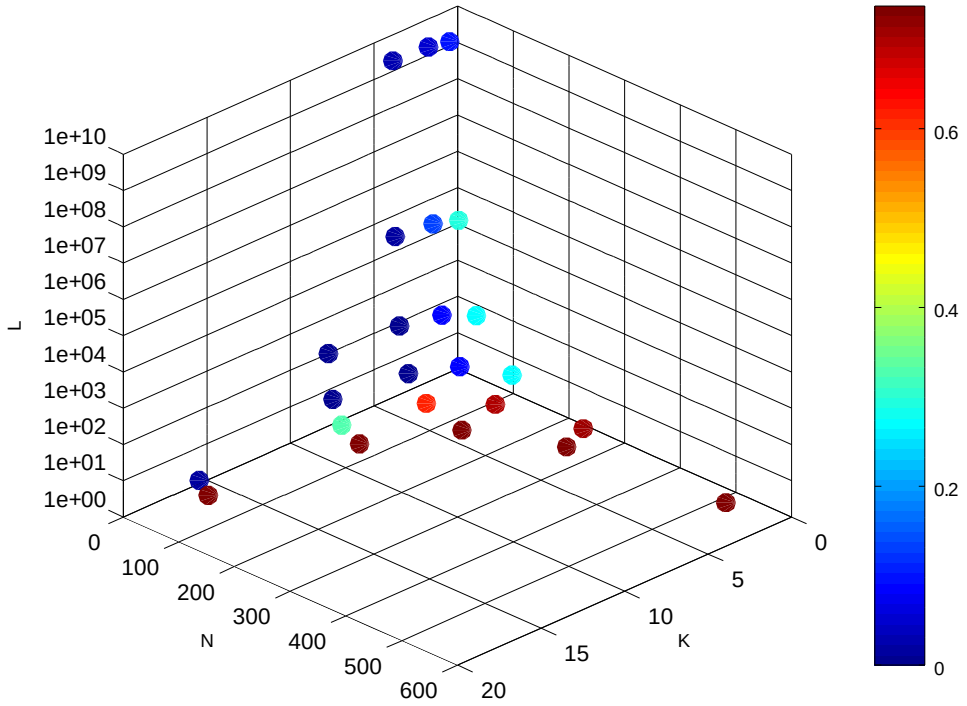


Figura 30. Diagrama de dispersión de los valores de K, N y L

La **Figura 30** muestra cómo cambian las probabilidades de acuerdo a los valores de K, N y L. En el eje x se encuentran los valores de K, en el eje y los valores de N, y en el eje z los valores de L. El color de cada punto muestra la probabilidad de éxito de la red atacante, una probabilidad baja es mejor (azul) y una alta es peor (roja).

Como se puede observar en la **Figura 30**, el valor de K no tiene mucha influencia en la probabilidad. Por otro lado, un valor alto de N tiene un impacto negativo en el resultado. Finalmente, un valor de L muy bajo, tiene un impacto negativo. Por ello, para plantear valores de K, N y L se recomienda un valor relativamente bajo de N, pero superior a K. El valor de L no debe ser muy bajo. Estas condiciones aseguran una probabilidad muy baja de un ataque pasivo exitoso.

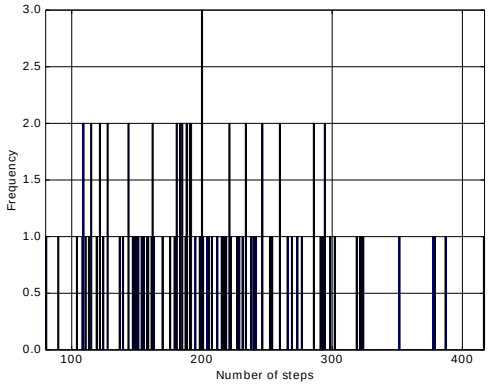
Por lo tanto, se consideran las combinaciones $(8,\ 16,\ 2^3)$ y $(8,\ 8,\ 2^7)$, ya que tienen los mejores resultados desde el punto de vista de seguridad. Para determinar la mejor combinación de las dos se realizaron un millón de simulaciones por cada una, y se obtuvo los resultados como lo muestra la **Tabla *13***.

Tabla 13

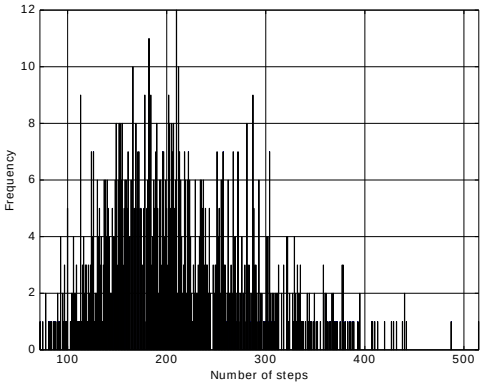
Resultados obtenidos de 1'000,000 simulaciones para las dos combinaciones

Ítem	Combinación $(K,\ N,\ L)$	Pasos mín.	Pasos máx.	Pasos media	Tiempo mín. [seg]	Tiempo máx. [seg]	Tiempo media [seg]	Eve sinc. exitosas	% Eve sinc. exitosas
1	$(8,\ 16,\ 2^3)$	40	785	211.5888	0.0780	1.5155	0.4058	0	0
2	$(8,\ 8,\ 2^7)$	27	861	218.3696	0.0312	0.9229	0.2385	2	0.0002

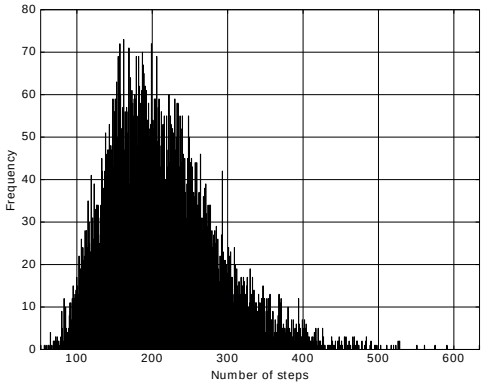
De la **Tabla *13*** se determinó que la mejor combinación para el trio de valores $(K,\ N,\ L)$ es $(8,\ 16,\ 2^3)$. Se realizaron más simulaciones para determinar la evolución de la distribución en la cantidad de pasos. Del total de las simulaciones se contó la cantidad de pasos que fueron necesarios para que las dos redes TPM se sincronicen. Se obtuvieron los histogramas como lo muestran las figuras de la **Figura 31**.



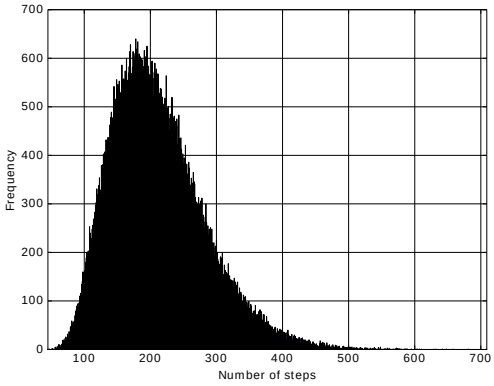
a) Histograma de 100 simulaciones



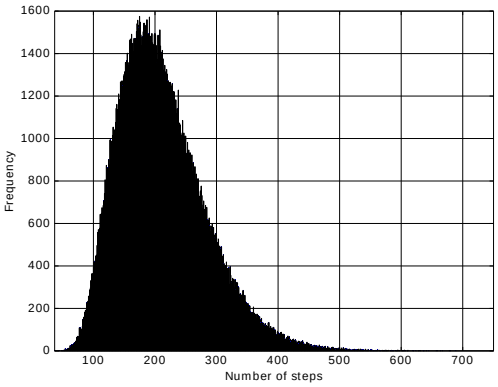
b) Histograma de 1,000 simulaciones



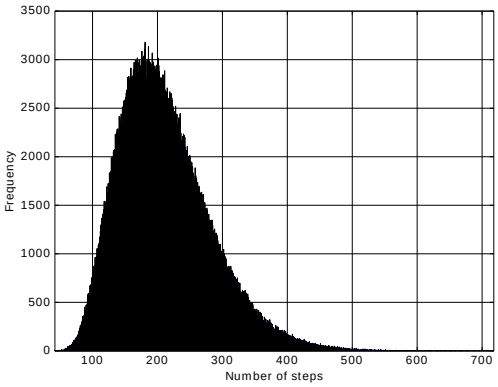
c) Histograma de 10,000 simulaciones



d) Histograma de 100,000 simulaciones

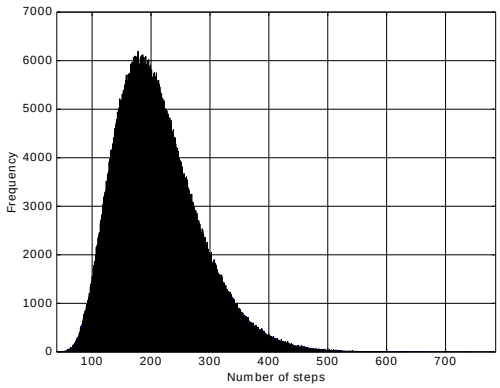


e) Histograma de 250,000 simulaciones



f) Histograma de 500,000 simulaciones

CONTINÚA



g) Histograma de 1’000,000 simulaciones

Figura 31. Evolución de la distribución en la cantidad de pasos

Durante las simulaciones adicionales, ocurrió que, en una única ocasión, la red atacante de Eve logró sincronizar sus pesos con las otras redes. La **Tabla *14*** muestra las probabilidades de un ataque exitoso de ambas combinaciones frente al total de las simulaciones realizadas.

Tabla 14

Probabilidades de éxito de una red TPM atacante de las dos combinaciones

Combinación $(K,\ N,\ L)$	Número de simulaciones	Eve sinc. exitosas	% Eve sinc. exitosas
$(\mathbf{8},\ \mathbf{16},\ \mathbf{2^3})$	2’361,100	1	0.00004
$(\mathbf{8},\ \mathbf{8},\ \mathbf{2^7})$	1’500,000	2	0.0001

Por lo tanto, la estructura óptima de la red TPM para los valores de K, N y L son 8, 16 y 8 respectivamente. Para determinar las probabilidades de cada valor discreto en el rango por el que se mueve la cantidad de pasos necesarios, se realizó el siguiente análisis estadístico.

4.1.1 Ajuste de distribución y cálculo de las probabilidades

Como se puede observar en las figuras de la **Figura 31**, la cantidad de pasos empleados en todas las simulaciones siguen una distribución discreta asimétrica positiva

o hacia la derecha. Por tanto, se realizó una descripción de la distribución discreta que siguen los datos empleando la función `descdist` de R, obteniéndose el resultado mostrado en la **Figura 32**.

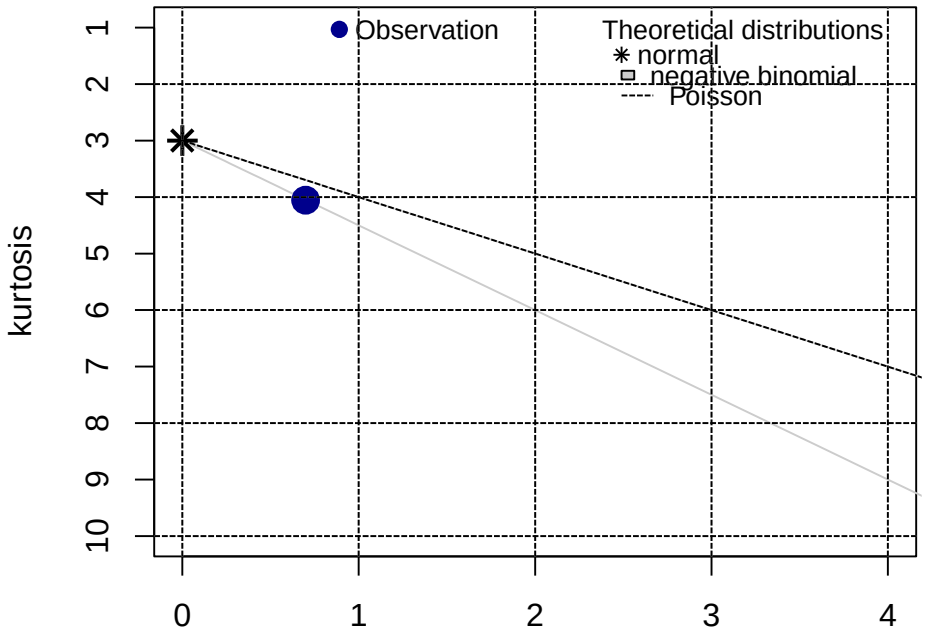


Figura 32. Gráfica de Cullen y Frey de la descripción de la distribución en la cantidad de pasos

La **Figura 32** ilustra la localización aproximada de los datos dentro de las distribuciones de probabilidad discretas. Su valor mínimo y máximo son 37 y 785 respectivamente. Su mediana es de 202 y su promedio es de 211.6149. Su desviación estándar estimada es de 71.6628, su asimetría estadística estimada es de 0.8360543 y su curtosis estimada es de 4.054793. Con estos valores, los datos se aproximan a la distribución de probabilidad Binomial Negativa. Para asegurarlo, las rectas que encierran el rango de dicha distribución se muestran en las ecuaciones **Ecuación 4** y **Ecuación 5**.

$$y = \frac{3}{2}x + 3 \qquad \textbf{Ecuación 4}$$

$$y = x + 3 \qquad \textbf{Ecuación 5}$$

Además, el punto de los datos es $(0.8360543,\ \ 4.054793)$. Al reemplazar la abscisa en la **Ecuación 4** se obtiene $y = 4.25408145$, y, al reemplazar la abscisa en la **Ecuación 5** se obtiene $y = 3.8360543$. Como $3.8360543 < 4.054793 < 4.25408145$, el punto se encuentra entre las rectas, y, por tanto, dentro del rango de la distribución Binomial Negativa. Por ello, se realizó un ajuste de la distribución, empleando la función `fitdistr` de R, obteniéndose los valores mostrados en la **Tabla *15***.

Tabla 15
Resultados del ajuste de la distribución

Parámetro	Valor	Error
Tamaño	9.4747	0.0089
Media	211.6081	0.0457

Con esos parámetros se calculó las probabilidades de cada valor discreto en el rango $[0,\ \ 1023]$ (dado que todas las observaciones están dentro de ese rango), que permita generar un número aleatorio con el cual las dos redes sincronicen sus pesos. Esto es debido a que, si se escoge un número pequeño, las redes tienen una probabilidad muy baja de éxito en la sincronización. Por otro lado, si se escoge un número grande, la red atacante tendrá más información sobre la sincronización y, debido a ello, una mayor probabilidad, de éxito en imitar el comportamiento de las dos redes. Por lo tanto, si se escoge un número aleatorio con la distribución de probabilidad calculada, las dos redes tendrán una mayor probabilidad de éxito frente a una menor probabilidad de éxito de la red atacante. Para ello, empleando R, se calculó las probabilidades con la función `dnbinom`.

```
x = c(0:1023)
y = dnbinom(x, size = 9.4747, mu = 211.6081)
write.table(y, file = ”probability.csv”, sep = ”,”, col.names = NA)
```

En primer lugar, se crea un arreglo de valores en el rango $[0,\ \ 1023]$. Luego, se calcula las probabilidades de cada valor discreto en el rango [0, 1023] siguiendo una distribución

negativa binomial de acuerdo a los valores calculados anteriormente. Finalmente, se almacenan los valores con su respectiva probabilidad en un archivo. Ejecutando el algoritmo anterior, se obtuvo las probabilidades como lo muestra la **Tabla *16***.

Tabla 16
Probabilidades para cada uno en el número de pasos

Número	Probabilidad	Probabilidad acumulada
0	1.39450272063906E-13	1.39450272063906E-13
1	1.02140347471555E-12	1.16085374677946E-12
2	4.99925585785512E-12	6.16010960463458E-12
3	1.82220717206152E-11	2.43821813252498E-11
4	5.43337331061097E-11	7.87159144313595E-11
…	…	…
187	0.00594225213845013	0.402406516717900
188	0.00594397823578930	0.408350494953689
189	0.00594434788781840	0.414294842841507
190	0.00594337481628165	0.420238217657789
191	0.00594107343515470	0.426179291092944
…	…	…
1019	1.32583323883776E-12	0.999999999995134
1020	1.28060895797495E-12	0.999999999996414
1021	1.23695343095161E-12	0.999999999997651

CONTINÚA

1022	1.19481260094881E-12	0.999999999998846
1023	1.15413426055656E-12	1

La **Tabla *16*** muestra la probabilidad y la probabilidad acumulada de cada valor discreto en el rango $[0, \quad 1023]$. Con el uso de estas probabilidades, se puede generar un número aleatorio antes de la sincronización de las dos redes para determinar la cantidad de pasos que efectuarán las dos redes para establecer sus pesos con un grado de certeza. Por ejemplo, con 189 pasos, se tiene una probabilidad del 41.4% de que las redes hayan sincronizado sus pesos.

4.2 Ataque de fuerza bruta

En el ataque de fuerza bruta, o búsqueda exhaustiva de la clave, el criptoanalista intenta descifrar el texto cifrado con cada posible clave hasta que encuentra la correcta (Wiener, 2011). Este tipo de ataques requieren un elevado rendimiento computacional, pero su implementación no requiere un esfuerzo superior por parte del criptoanalista. El algoritmo que ejecute este ataque debe tener la capacidad de generar una clave y emplearla para descifrar un texto cifrado. Si el texto encontrado es igual al texto original, la clave probada será la correcta.

En primer lugar, para realizar este ataque, se parte del hecho que el atacante conoce el mensaje sin cifrar y su texto cifrado con una clave de 512 bits. El atacante tratará de encontrar la clave probando todas las combinaciones posibles. Para agilizar las pruebas por segundo que se pueden realizar, se emplearon Hilos (Threads) que se ejecutarán de forma paralela en el procesador. Cada thread ejecutará un conjunto de claves con una cabecera igual y de la misma longitud con respecto a los demás. Por ello, como la clave se compone con los cuatro nucleótidos (A, C, T, G), la cantidad de threads que se ejecutarán serán 4^x, donde x es la longitud de la cabecera que tendrá cada thread. Por ejemplo, si $x = 2$ entonces la cantidad de threads que se ejecutarán es $4^x = 4^2 = 16$.

Cada uno de los 16 threads tendrá una cabecera de 2 nucleótidos y será diferente entre los demás. De esta manera, cada uno de los threads tendrán las siguientes claves:

- Thread 1: “AAXXXXXXXXXX…”
- Thread 2: “ACXXXXXXXXXX…”
- Thread 3: “ATXXXXXXXXXX…”
- Thread 4: “AGXXXXXXXXXX…”
- Thread 5: “CAXXXXXXXXXX…”
- Thread 6: “CCXXXXXXXXXX…”
- Thread 7: “CTXXXXXXXXX…”
- Thread 8: “CGXXXXXXXXXX…”
- Thread 9: “TAXXXXXXXXXX…”
- Thread 10: “TCXXXXXXXXXX…”
- Thread 11: “TTXXXXXXXXXX…”
- Thread 12: “TGXXXXXXXXXX…”
- Thread 13: “GAXXXXXXXXXX…”
- Thread 14: “GCXXXXXXXXXX…”
- Thread 15: “GTXXXXXXXXXX…”
- Thread 16: “GGXXXXXXXXXX…”

Por lo tanto, cada thread del algoritmo generará una clave, descifrará el texto proporcionado, y, si el texto obtenido es igual al texto original del mensaje, habrá encontrado la clave. Caso contrario, se probará con la siguiente clave. Para realizar este

ataque se midió el rendimiento que tiene dos tipos de microprocesador con diferentes características. Los microprocesadores empleados, así como características adicionales que son relevantes, se muestran en la **Tabla *17***.

Tabla 17
Recursos en hardware empleados en el ataque

Ítem	Microprocesador	Frecuencia	vCPU	Memoria
1	Intel(R) Core(TM) i7-6500U	2.50GHz	4	16GB
2	Intel(R) Xeon(R) Platinum 8124M	3.00GHz	72	144GB

Con los microprocesadores mostrados en la **Tabla *17*** se realizaron diversas pruebas de ataque con tiempos variables. Debido un ataque de fuerza bruta exitoso puede requerir grandes cantidades de tiempo, se realizaron proyecciones tomando como base tiempos cortos. Los resultados de los ataques se muestran en las tablas **Tabla *18*** y **Tabla *19***.

Tabla 18
Cantidad de claves probadas durante un tiempo determinado empleando el microprocesador Intel(R) Core(TM) i7-6500U

	5 seg.			10 seg.			20 seg.			3600 seg.		
Threads	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.
4	14277	5.12	2788	32464	10.10	3214	68911	20.12	3425	13023301	3600	3617
16	11604	5.16	2248	28695	10.15	2827	65211	20.13	3239	12942960	3600	3595
64	10337	5.21	1984	24708	10.17	2429	60322	20.15	2993	12858613	3600	3571

Tabla 19

Cantidad de claves probadas durante un tiempo determinado empleando el microprocesador Intel(R) Xeon(R) Platinum 8124M

	5 seg.			10 seg.			20 seg.			3600 seg.		
Threads	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.	Prom. claves	Tiempo [seg]	Prom. 1 seg.
4	12559	5.30	2369	48168	10.35	4653	119345	20.35	5864	28651521	3600	7958
16	32266	5.39	5986	110736	10.40	10647	206769	20.40	10135	53956184	3600	14987
64	45020	5.44	8275	114547	10.45	10961	259319	20.45	12680	48941080	3600	13593

Como se muestran en las tablas **Tabla *18*** y **Tabla *19***, entre mayor sea la cantidad de threads que se ejecutan simultáneamente, el algoritmo puede probar un mayor número de claves durante el mismo tiempo. Se puede notar también que, a menor cantidad de tiempo para realizar una prueba, el promedio de claves que se prueban por segundo también es bajo. Pero, a medida que en tiempo sube, se incrementa la cantidad promedio de claves que se prueban. Por lo tanto, pruebas con tiempos cortos arrojan como resultado datos sesgados al proyectarlos con la totalidad de las claves a probar. De esta manera, se escoge el valor mayor que corresponde a las pruebas realizadas durante 3600 segundos con 16 threads.

Por lo tanto, para calcular el tiempo total para que el algoritmo pruebe todas las combinaciones de la clave, se realiza una proporción. La **Ecuación 6** muestra el resultado.

$$\frac{3600}{53956184} * 4^{256} \approx 8.94 \times 10^{149} \qquad \textbf{Ecuación 6}$$

La **Ecuación 6** muestra que a un atacante le tomaría aproximadamente 8.94×10^{149} segundos ($\approx 2.83 \times 10^{142}$ años) probar todas las combinaciones de la clave. Por tanto, debido a que, con la computación actual, un ataque de fuerza bruta requiere una gran cantidad de tiempo, se puede asegurar que el algoritmo no es vulnerable a este tipo de ataques.

4.3 Ataque de texto plano escogido

Para desarrollar este ataque, se calculará el efecto avalancha. Formalmente, una función f satisface el Criterio Avalancha cuando cambia, en promedio, la mitad de los bits de salida cada vez que se cambia un bit de entrada. Para que dicho valor cumpla el Criterio Avalancha debe ser cercano a $1/2$. Por lo tanto, se emplearon distintos mensajes de los cuales se modificaron ligeramente la entrada para obtener su texto cifrado con la misma clave y calcular su efecto avalancha. Para los cálculos se empleó la siguiente clave:

```
GTATTTTCTCAGCGGGGTCAAGCGGCATTAACCTTACAACGCTCTTCAGGCGCTTGCCAGAACAACTAGAG
AGTCTCAAACTCATCATAATGAGTCAAATCATTCTTTCATATAGCCGGAAACATCCAAGTGCGAGCTATCT
TTATACGGACGGACCACGTGTCAGTTGCCGCGGGTTCGGCCTAAGAGGCGGAGTCCTCCTCCATTGGAGGC
TGACTGTTCACATTACAAAGTATTACACTAGGTTGTAACATAG
```

Se probaron con diferentes mensajes y de diferente longitud. Los resultados se muestran en la **Tabla 20**

Tabla 20
Prueba del Criterio Avalancha con diferentes mensajes

Ítem	Mensaje	Texto cifrado	Longitud en bits	Bits diferentes	Criterio avalancha
1	a	8c0034	24	-	-
2	b	bb00c0	24	10	0.5833

CONTINÚA

3	gato loco	00b4f30c96137c8d294610	88	-	-
4	Gato loco	c824a5c61921ade2fb65a7	88	44	0.5000
5	gato locO	4e753f89e8a44510371e85	88	46	0.4772
6	gato l0co	1f554c756baa64c6f0ef23	88	52	0.4090
7	mi gato esta loco	429146a297ff3f3b508f0d5a624d18cb992bfd54b7	168	-	-
8	mi gato 3sta loco	0a41bcddad38689f3f16ecff2590d5d36c48bc4ec4	168	90	0.4642
9	m1 g4t0 3st4 l0c0	bf65281c7c5595ae18d164f2e03fd767daefbe4bed	168	89	0.4702
10	Mi gato esta loco	79e5d358fd79197d1cca166c973011f202e6885450	168	85	0.4940
11	11223344 55667788	af74206b3bf06190b7e9aa52a652c4365b68	144	-	-
12	11223344 55667789	daf3c8d2089700ca631efe7cf7f002b0c9e6	144	72	0.5000
13	00223344 55667788	21e73383e2b6e1a1b8702d81d4d6e583c515	144	68	0.5277
14	12345678 12345678	5aec89c31eae34e15b5f29d0b5b30d4fb230	144	71	0.5069

Como se puede observar en la **Tabla 20**, un pequeño cambio en el mensaje involucra una avalancha de cambios en el texto cifrado. Por ejemplo, para el texto “a”, su equivalente ASCII en binario es 01100001. Al cifrarlo, su texto cifrado en hexadecimal es 8c0034, y en binario es 100011000000000000110100 que tiene una longitud de 24 bits. Cuando se modifican los dos últimos bits del texto original para obtener “b” (01100010 en binario). Al cifrar el nuevo texto se obtiene bb00c0, que en binario es 101110110000000011000000. Al comparar los dos textos cifrados en binario, se observa que, de los 24 bits, 10 son diferentes. Por lo tanto, el criterio avalancha para este ejemplo es $1 - 10/24 = 0.5833$.

En promedio, cambia un 49.3% de los bits del texto cifrado de ejemplo al cambiar entre 1 y 8 bits del mensaje. Como dicho valor es muy cercano a $1/2$, el criptosistema propuesto tiene una fuerte propiedad avalancha.

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

El presente trabajo se originó con el fin de implementar un criptosistema basado en redes neuronales y en ADN en una aplicación móvil de chat, que permita una comunicación segura entre dos usuarios autorizados. Para lograrlo, se emplearon diversas herramientas y metodologías que permitan un desarrollo sostenible y resistente a cambios.

Los actuales sistemas de cifrado están bajo un constante ataque por parte de criptoanalistas y personas mal intencionadas. Por ello, se investigó nuevos sistemas de cifrado, como la criptografía neuronal y la criptografía basada en ADN, los cuales permiten la generación de una clave criptográfica, cifrado y descifrado de la información, características fundamentales de cualquier criptosistema.

Para lograr dicho análisis, se empleó la metodología investigación-acción, que permitió realizar un estudio del estado del arte para conocer los dos algoritmos criptográficos. Los resultados iniciales del estudio permitieron diseñar y probar el algoritmo progresivamente. Fue importante el uso de Python, ya que permitió una implementación ágil con un nivel de abstracción alto. Además, Python tiene librerías matemáticas que permiten el análisis estadístico de los datos de manera gráfica. Esto resultó en una deducción del comportamiento de los datos mientras se realizaban las simulaciones. Gracias a ello, se logró realizar 14’861,100 simulaciones para buscar y probar la estructura óptima de una red TPM.

Otro aspecto importante dentro de la investigación realizada fue la validación de los algoritmos criptográficos. Para ello fue importante investigar y sintetizar los modelos de ataques realizados por varios autores para los dos algoritmos de cifrado. Los resultados

preliminares de dichos modelos permitieron escoger los más relevantes y que arrojen resultados apreciables sobre la seguridad de los dos algoritmos.

Para realizar la descripción de los datos de las simulaciones de la criptografía neuronal, fue útil el empleo de las librerías matplotlib de Python, que permitió la graficar un histograma con los datos generados, y fitdistrplus de R, que permitió describir y ajustar la distribución de los datos. Adicionalmente, se empleó GNU Octave que permitió generar una gráfica que describa el comportamiento de los datos conforme se modifican los valores iniciales.

Los resultados preliminares muestran que la estructura óptima de una red TPM para los valores de K, N y L son 8, 16 y 8 respectivamente a partir de 14’861,100 simulaciones. Con dichos valores, la red TPM puede generar una clave criptográfica de 512 bits, que, al transformarla a secuencias de ADN, resulta en una clave de 256 nucleótidos. El algoritmo criptográfico basado en ADN emplea dicha clave para cifrar los mensajes de texto dividiéndolos en bloques de longitud fija.

Ambos criptosistemas fueron sometidos a criptoanálisis bajo distintos modelos de ataque. A pesar de que el algoritmo de criptografía neuronal y el algoritmo de criptografía basada en ADN son distintos, realizan funciones complementarias dentro del criptosistema, por lo que se realizaron pruebas de criptoanálisis por separado en cada uno de ellos. Los resultados muestran que los dos algoritmos son resistentes frente a los ataques de sincronización pasiva (con una probabilidad de 0.00004% de éxito a partir de 2’361,100 simulaciones), y ataque de fuerza bruta (con una duración máxima aproximada de 3.12×10^{140} años). Además, las pruebas realizadas arrojaron como resultado que el criptosistema tiene efecto avalancha, al modificar aproximadamente un 49.3% de sus bits de salida con sólo modificar pocos bits de entrada.

Por otro lado, para el diseño e implementación de la aplicación móvil de chat, se incluyeron aspectos básicos que permitan la comunicación. Esto permitió centrar el desarrollo en el criptosistema y acoplarlo a la aplicación de forma rápida. Fue vital el uso

de la metodología XP, ya que permitió identificar claramente los requerimientos de la aplicación móvil. Además, debido a que se tenía una incertidumbre inicial sobre el algoritmo de cifrado, dicha metodología permitió gestionar los cambios que surgieron durante el desarrollo.

Como aporte de este estudio, se pudo determinar la estructura óptima de una red TPM que priorice la seguridad frente a una red atacante. Además, se logró abstraer el funcionamiento biológico del ADN y plasmarlo en un criptosistema, adaptando dos de sus principales funciones (transcripción y traducción) a los procesos de cifrado y descifrado de un mensaje.

5.2 Recomendaciones

Se recomienda realizar un estudio para investigar el comportamiento del algoritmo al cifrar mensajes cuyo tamaño se superior a 10,000 caracteres. En los resultados se observa un incremento de 23.8 veces el tiempo de cifrado al incrementar 5 veces la longitud del mensaje.

Durante el presente estudio, existieron valores de L que no fueron considerados debido a limitaciones de hardware y software. Se recomienda realizar un análisis con los valores 2^{63}, 2^{127}, 2^{255} y 2^{511}, los cuales generan también claves de 512 bits.

Para futuros proyectos se recomienda realizar pruebas exhaustivas de criptoanálisis a los dos criptosistemas. Para el algoritmo de criptografía neuronal se recomienda los ataques propuestos por (Ruttor, 2006), como son: un ataque geométrico (Geometric Attack), un ataque mayoritario (Majority Attack) y un ataque genético (Genetic Attack).

Para el algoritmo de criptografía basado en ADN se recomienda realizar los ataques que son comunes para criptosistemas que realizan el cifrado por bloques, como lo menciona (Knudsen, 2011). Entre los más comunes están: ataques de sólo texto cifrado (Ciphertext-only attack, COA), ataques de texto plano conocido (Known plaintext attack, KPA) en el cual puede ser un criptoanálisis lineal, ataques de texto plano escogido

(Chosen plaintext attack, CPA) en el cual puede ser un criptoanálisis diferencial, ataques de texto plano escogido adaptativamente (Adaptively chosen plaintext attack, CPA2), ataques de texto cifrado escogido (Chosen ciphertext attacks, CCA). También figuran dentro de los ataques los ataques por tablas (Table attack), ataques de diccionario (Dictionary attack), ataques por coincidencia de texto cifrado (Matching ciphertext attack) y ataques de compromiso tiempo-memoria (Time-memory trade-off attack).

Se recomienda además generalizar el algoritmo propuesto para que pueda ser empleado por distintas aplicaciones en varias plataformas.

REFERENCIAS BIBLIOGRÁFICAS

Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. Science, 266(5187).

Al Hasib, A., & Haque, A. M. (Noviembre de 2008). A comparative study of the performance and security issues of AES and RSA cryptography. *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference, 2*, 505-510. IEEE.

Allam, A. M., Abbas, H. M., & El-Kharashi, M. W. (2013). Security analysis of neural cryptography implementation. *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, 195-199. IEEE.

Al-Mahdi, H., Shahin, O. R., Fouad, Y., & Alkhaldi, K. (2018). Design and analysis of DNA Binary Cryptography Algorithm for Plaintext. *10*(3), págs. 699-706.

Anikin, I., Makhmutova, A. Z., & Gadelshin, O. E. (2016). Symmetric encryption with key distribution based on neural networks. 1-4. IEEE.

Ayala, W., Fuertes, W., Galárraga, F., Aules, H., & Toulkeridis, T. (2017). *Software Application to Evaluate the Complexity Theory of the RSA and Elliptic Curves Asymmetric Algorithms.* IEEE.

Barker, E. (2016). Recommendation for Key Management Part 1: General. *NIST Special Publication 800-57 Part 1*(Revision 4).

Barrett, M. P. (2018). *Framework for Improving Critical Infrastructure Cybersecurity Version 1.1.*

Bauer, F. L. (2011). Cryptanalysis. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 277-281). Boston, MA: Springer US.

Bauer, F. L. (2011). Cryptology. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 283-284). Boston, MA: Springer US.

Bauer, F. L. (2011). Cryptosystem. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 284-285). Boston, MA: Springer US.

Bickman, L., & Rog, D. J. (2009). Applied Research Design: A Practical Approach. En L. Bickman, & D. J. Rog (Edits.), *Applied Social Research Methods* (Segunda ed., págs. 3-43). Estados Unidos de América: SAGE Publications, Inc.

Biryukov, A. (2011). Chosen Plaintext Attack. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 205-206). Boston, MA: Springer US.

Canós, J. H., Letelier, P., & Penadés, M. C. (2003). Metodologías ágiles en el desarrollo de software. *Metodologías Ágiles en el Desarrollo de Software, 1(10)*, 1-8.

Canteaut, A. (2011). Stream Cipher. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 1263-1265). Boston, MA: Springer US.

Committee on National Security Systems. (6 de Abril de 2015). Committee on National Security Systems (CNSS) Glossary.

Daemen, J., & Rijmen, V. (2011). Rijndael. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 1046-1049). Boston, MA: Springer US.

Gehani, A., LaBean, T., & Reif, J. (2003). DNA-based cryptography. En *Aspects of Molecular Computing* (págs. 167-188). Berlin, Heidelberg: Springer.

Google. (18 de Septiembre de 2018). *Build an Android App Using Firebase and the App Engine Flexible Environment.* Obtenido de Google Cloud: https://cloud.google.com/solutions/mobile/mobile-firebase-app-engine-flexible?hl=en

Gupta, S., Nanda, N., Chhikara, N., Gupta, N., & Jain, S. (2018). MUTUAL LEARNING IN TREE PARITY MACHINES USING CUCKOO SEARCH ALGORITHM FOR SECURE PUBLIC KEY EXCHANGE.

Hoyle, D. (2017). *ISO 9000 Quality Systems Handbook* (Séptima ed.). Nueva York: Routledge.

Jain, S., & Bhatnagar, V. (2014). Analogy of various DNA based security algorithms using cryptography and steganography. *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, (págs. 285-291).

Kaliski, B. (2011). Asymmetric Cryptosystem. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 49-50). Boston, MA: Springer US.

Kaliski, B. (2011). Symmetric Cryptosystem. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (pág. 1271). Boston, MA: Springer US.

Kalsi, S., Kaur, H., & Chang, V. (2018). DNA Cryptography and Deep Learning using Genetic Algorithm with NW algorithm for Key Generation. *Journal of medical systems, 42*(1).

Kanter, I., Kinzel, W., & Kanter, E. (2002). Secure exchange of information by synchronization of neural networks. *EPL (Europhysics Letters), 57(1)*, 141.

Katz, J., & Lindell, Y. (2014). *Introduction to modern cryptography.* Chapman and Hall/CRC.

Klimov, A., Mityagin, A., & Shamir, A. (2002). Analysis of neural cryptography. *International Conference on the Theory and Application of Cryptology and Information Security*, 288-298. Berlin Heidelberg: Springer.

Knudsen, L. R. (2011). Block Ciphers. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 152-157). Boston, MA: Springer US.

Lloyd, S., & Adams, C. (2011). Key Management. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 683-688). Boston, MA: Springer US.

Lodish, H., Berk, A., Matsudaira, P., Kaiser, C. A., & Scott, M. P. (2005). *Biología celular y molecular* (Quinta ed.). (A. Méndez, S. Rondinone, & O. Giovanniello, Trads.) Buenos Aires: Médica Panamericana.

Mendes Calo, K., Estevez, E. C., & Fillottrani, P. R. (2010). *A quantitative framework for the evaluation of Agile Methodologies.* Journal of Computer Science & Technology, 10.

Meneses, F., Fuertes, W., Sancho, J., Salvador, S., Flores, D., Aules, H., . . . Nuela, D. (2016). RSA Encryption Algorithm Optimization to Improve Performance and Security Level of Network Messages. *IJCSNS, 16*(8), pág. 55.

Microsoft. (30 de Julio de 2018). *Firebase Cloud Messaging.* Obtenido de Microsoft Docs: https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging

Niazi, S. K., & Brown, J. L. (2015). *Fundamentals of Modern Bioprocessing.* CRC Press.

Nieles, M., Dempsey, K., & Pillitteri, V. Y. (2017). An Introduction to Information Security. *NIST Special Publication 800-12*(Revision 1).

Padilla, J. M., Meyer-Baese, U., & Foo, S. (2018). Security evaluation of Tree Parity Re-keying Machine implementations utilizing side-channel emissions. *EURASIP Journal on Information Security, 2018*(1), pág. 3.

Pattanayak, S., & Ludwig, S. A. (2017). Encryption Based on Neural Cryptography. 321-330. Springer.

Pelosi, G. (2011). Secure Index. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 1116-1119). Boston, MA: Springer US.

Preneel, B. (2011). Modes of Operation of a Block Cipher. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 789-794). Bostom, MA: Springer US.

Pressman, R. S., & Maxim, B. R. (2015). *Software Engineering: A practitioner's approach* (Octava ed.). New York, Estados Unidos de América: McGraw-Hill Education.

Ravikant, A., & Umphress, D. (2008). *Extreme programming for a single person team.* (ACM, Ed.) Proceedings of the 46th Annual Southeast Regional Conference on XX.

Ruttor, A. (2006). *Neural synchronization and cryptography.* Würzburg: arXiv preprint arXiv:0711.2411.

Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell system technical journal, 28(4)*, 656-715.

Singh, S., Maakar, S. K., & Kumar, S. (2013). A Performance Analysis of DES and RSA Cryptography. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS) Web Site: www. ijettcs. org, 2(3)*.

Sommerville, I. (2016). *Software Engineering* (Décima ed.). Harlow, Reino Unido: Pearson.

Upadhyaya, S. (2015). Secure Communication Using DNA Cryptography with Secure Socket Layer (SSL) Protocol in Wireless Sensor Networks. *Procedia Computer Science, 70*, 808-813.

van Tilborg, H. C. (2011). Shannon's Model. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 1194-1195). Boston, MA: Springer US.

Vergili, I., & Yucel, M. D. (2001). Avalanche and Bit Independence Properties for the Ensembles of Randomly Chosen nxn S-Boxes. EE dept of METU.

Wiener, M. J. (2011). Exhaustive Key Search. En H. C. van Tilborg, & S. Jajodia (Edits.), *Encyclopedia of Cryptography and Security* (págs. 431-433). Boston, MA: Springer US.

Zhang, Y., Liu, X., Ma, Y., & Cheng, L.-C. (2017). An optimized DNA based encryption scheme with enforced secure key distribution. *Cluster Computing, 20*(4), págs. 3119-3130.