Immagine che contiene testo

Descrizione generata automaticamente 

*Grado en Ingeniería Informática-Ingeniería del Software*

*2021-2022*

*Curso de Seguridad en Sistemas Informáticos y en Internet*

**SECURITY TEAM 11**

*Matilde Ghidini*

*Matteo Halilaga*

*Gabriele Petroni*

**PAI 2.**

**VERIFICADORES DE INTEGRIDAD EN LA TRANSMISIÓN PUNTO-PUNTO PARA ENTIDAD FINANCIERA**

**INDICE**

[**INTRODUCCIÓN** 3](#_Toc99026193)

[**Objetivos del proyecto** 3](#_Toc99026194)

[**TECNOLOGIAS Y LIBRERÍAS UTILIZADAS** 3](#_Toc99026195)

[**Lenguaje de programación: Python** 3](#_Toc99026196)

[**Sqlite3** 3](#_Toc99026197)

[**Algoritmo SHA256** 3](#_Toc99026198)

[**JSON** 3](#_Toc99026199)

[**Logging** 3](#_Toc99026200)

[**Socket** 3](#_Toc99026201)

[**Uuid** 4](#_Toc99026202)

[**Hamachi** 4](#_Toc99026203)

[**SOLUCION** 4](#_Toc99026204)

[**PRUEBAS REALIZADAS** 5](#_Toc99026205)

[**Ejecucion si fallas de integridad** 5](#_Toc99026206)

[**Simulacion ataque de man-in-the-middle** 5](#_Toc99026207)

[**Simulacion replay-attack** 5](#_Toc99026208)

[**ESTABLECIMIENTO DE CLAVES** 5](#_Toc99026209)

# **INTRODUCCIÓN**

Existen numerosas API en los lenguajes de programación, comandos en los sistemas operativos y herramientas comerciales/software libre que se pueden utilizar para comprobar la integridad de los datos en el almacenamiento mediante la correspondiente generación de resúmenes de mensajes (message digests o checksums). En este Proyecto de Aseguramiento de la Información usaremos técnicas para poder verificar la integridad de datos en la transmisión por redes públicas como Internet y evitar los diferentes tipos de posibles ataques. En este PAI, la Política de Seguridad sobre la Integridad de las Transmisiones de un entidad financera establece que:

*“En todas las transferencias bancarias por medios electrónicos no seguros se debe conservar la integridad y confidencialidad de las comunicaciones”*

## **Objetivos del proyecto**

1. Desarrollar/Seleccionar el verificador de integridad para los mensajes de transferencia bancaria que se transmiten a través de las redes públicas evitando los ataques de man-in-the-middle y de replay (tanto en el servidor como en el cliente).

2. Desplegar un verificador de integridad en los sistemas cliente/servidor para llevar a cabo la realización de la verificación de forma práctica de los mensajes transmitidos entre un servidor y un cliente.

# **TECNOLOGIAS Y LIBRERÍAS UTILIZADAS**

## **Lenguaje de programación: Python**

Hemos decidido desarrollar nuestro proyecto utilizando el lenguaje Python.

## **Sqlite3**

Para crear una base de datos donde almacenar los NONCEs de los mensajes hemos utilizado la librería Python Sqlite3, que es un base de datos SQL self-contained y file-based.

## **Algoritmo SHA256**

Para generar el hash de los mensajes hemos decidido utilizar el algoritmo SHA256.

La implementación ha sido realizada a través de las librerías Python hashlib (sha256) y hmac. Hemos utilizados, con los cambios necesarios, el algoritmo utilizado en el PAI1.

## **JSON**

Hemos utilizado la librería Python json, para la serialización y deserialización e los datos enviados del client hasta el server.

## **Logging**

Hemos utilizado la librería Python logging, para

## **Socket**

Hemos utilizado la librería Python socket, para implementar construir nuestra arquitectura cliente-servidor.

## **Uuid**

Hemos utilizado la librería Python uuid, para la generación de los nonces.

## **Hamachi**

Hamachi es un software de virtualización de redes que permite emular una red local (LAN) a los dispositivos conectados por WAN. Con Hamachi se puede generar una red local aunque los dispositivos se encuentren en distintos lugares repartidos por el mundo. Para ello, Hamachi hace uso de redes privadas virtuales ([VPN](https://www.geeknetic.es/VPN/que-es-y-para-que-sirve)). Hemos utilizado. Hamachi para simular el funcionamiento de nuestro programa en diferentes dispositivos.

Immagine che contiene testo

Descrizione generata automaticamente

# **SOLUCION**

El proyecto se basa en un arquitectura cliente-servidor, en la que el client (el cliente) envía un mensaje al server (organización financiera). El mensaje que se envía entre el cliente y el servidor de la organización financiera tiene el siguiente formato: “Cuenta origen, Cuenta destino, Cantidad transferida”.

**Input del sistema**

El mensaje (Cuenta Origen, Cuenta Destino, Cantidad) a verificar la integridad en su transmisión, nombre del algoritmo que se usará para verificar la integridad, clave utilizada por el cliente y el servidor.

**Output del sistema**

Indicación en el cliente y servidor si se ha conservado la integridad o no se ha conservado. La salida podría ser presentada en una ventana al emisor del mensaje y en el servidor dejar constancia en un fichero de logs de los mensajes que no han llegado de forma íntegra.

**Funcionamiento**

El cliente envía un mensaje al servidor. El usuario solo tiene que inserir la cantidad de dinero que desea transferir; para el desarrollo de este proyecto hemos decidido tener la cuenta de origen y la cuenta de destino como constantes fijas.

A este mensaje se le añade un nonce, generado casualmente y luego se genera un código hash del mensaje + nonce.

A este punto, lo que se envia al servidor es un objeto en formato JSON en el que se encuentran el mensaje, el nonce y el hash.

El servidor primero verifica la integridad del mensaje, generando nuevamente el hash del mensaje recibido, para averiguar que los dos sean iguales. Si no lo son, hay una falla de integridad y el servidor lo escribe en el log.

Luego el servidore hace una verificación del nonce, para verificar que no se haya utilizado ya otra vez. Hacemos esta comprobación, almacenando en una base de datos todos los nonce. De esta manera, el servidor confronta el nonce del mensaje recibido con los que ya están en la base de datos. Si no lo encuentra significa que el control del mensaje ha ido bien; por otro lado, si el nonce ya está en la base de datos, significa que este código ya ha sido utilizado y puede tratarse de un replay attack; entonces el servidor rechaza la transacción y escribe un aviso en el log.

Al final, en el fichero de log se pueden ver todas la informaciones sobre la ejecución del programa.

# **PRUEBAS REALIZADAS**

## **Ejecucion si fallas de integridad**

## **Simulacion ataque de man-in-the-middle**

## **Simulacion replay-attack**

# **ESTABLECIMIENTO DE CLAVES**

Para que el cliente y servidor tengan la misma clave para hacer la comprobación de la integridad ha sido aplicado el siguiente razonamiento.

La criptografía simétrica es más insegura ya que el hecho de pasar la clave es una gran vulnerabilidad, pero se puede cifrar y descifrar en menor tiempo del que tarda la criptografía asimétrica, que es el principal inconveniente y es la razón por la que existe la criptografía híbrida.

Los sistemas de criptografía asimétricos pueden ser combinados con proceso simétricos. En este caso, las claves se comparten primero con criptografía asimétrica, pero la siguiente comunicación es criptografía de manera simétrica. Este sistema de criptografía hibrido se utiliza cuando los usuarios quieren tener la velocidad de la criptografía simétrica y al mismo tiempo la seguridad de la criptografía asimétrica.

## **Criptografía híbrida**

Este sistema es la unión de las ventajas de los dos, debemos de partir que el problema de ambos sistemas criptográficos es que el simétrico es inseguro y el asimétrico es lento.

El proceso para usar un sistema criptográfico híbrido es el siguiente (para enviar un archivo):

* Generar una clave pública y otra privada (en el receptor).
* Cifrar un archivo de forma síncrona.
* El receptor nos envía su clave pública.
* Ciframos la clave que hemos usado para encriptar el archivo con la clave pública del receptor.
* Enviamos el archivo cifrado (síncronamente) y la clave del archivo cifrada (asíncronamente y solo puede ver el receptor).