



TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACION

Presenta:

Julissa Migdalia José Cruz	21620153
Luz María Mendoza cortes	21620044
Ángeles Gonzales Martínez	20620293
Elisahadad Mayte León de Jesús	19620299

Tema:

Practica 1: CONTRASEÑAS Y CERTIFICADOS

Docente:

Ing. Edward Osorio Salinas.

Carrera:

Ingeniera en Sistemas Computacionales

Grupo: 7US

Tlaxiaco. Oaxaca.. A 30 de Agosto de 2024.

"Educación, Ciencia y Tecnología, Progresos día con día"®

Boulevard Tecnológico Km. 2.5, Llano Yosovee C.P. 69800. Tlaxiaco. Oax. México.

Tels. Dir. (953) 55 20788, (953) 55 21322, (953) 55 20405 e-mail:

dir_tlaxiaco@tecnm.mx | www.tlaxiaco.tecnm.mx



Índice

Introducción.....	3
Objetivo	3
Materiales	3
Desarrollo	4
Contraseña	17
Certificado digital:.....	17
Firma digital:	17
Cifrado simétrico:.....	17
Cifrado asimétrico:.....	17
Picadillo (también conocido como Hashing):	18
Encriptación:.....	18
AES (Advanced Encryption Standard):	18
RSA (Rivest-Shamir-Adleman):	18
SHA-256 (Secure Hash Algorithm 256-bit):.....	19
SSL (Secure Sockets Layer):	19
TLS (Transport Layer Security):	19
HTTPS (Hypertext Transfer Protocol Secure):	20
SFTP (Secure File Transfer Protocol):.....	20
SSH (Secure Shell):.....	20
Conclusión	21
Bibliografía.....	22

Introducción

En seguridad y virtualización, las contraseñas y los certificados son esenciales para proteger los sistemas y los datos. Las contraseñas, si se gestionan adecuadamente, proporcionan una primera línea de defensa sólida, mientras que los certificados permiten una autenticación y cifrado robustos en las comunicaciones y transacciones digitales. Ambos deben implementarse y gestionarse con las mejores prácticas de seguridad para garantizar la integridad de los sistemas y la protección de la información.

Objetivo

El objetivo de esta práctica es que el alumno conozca y aplique los conceptos de contraseñas y certificados digitales en la seguridad de la información.

Materiales

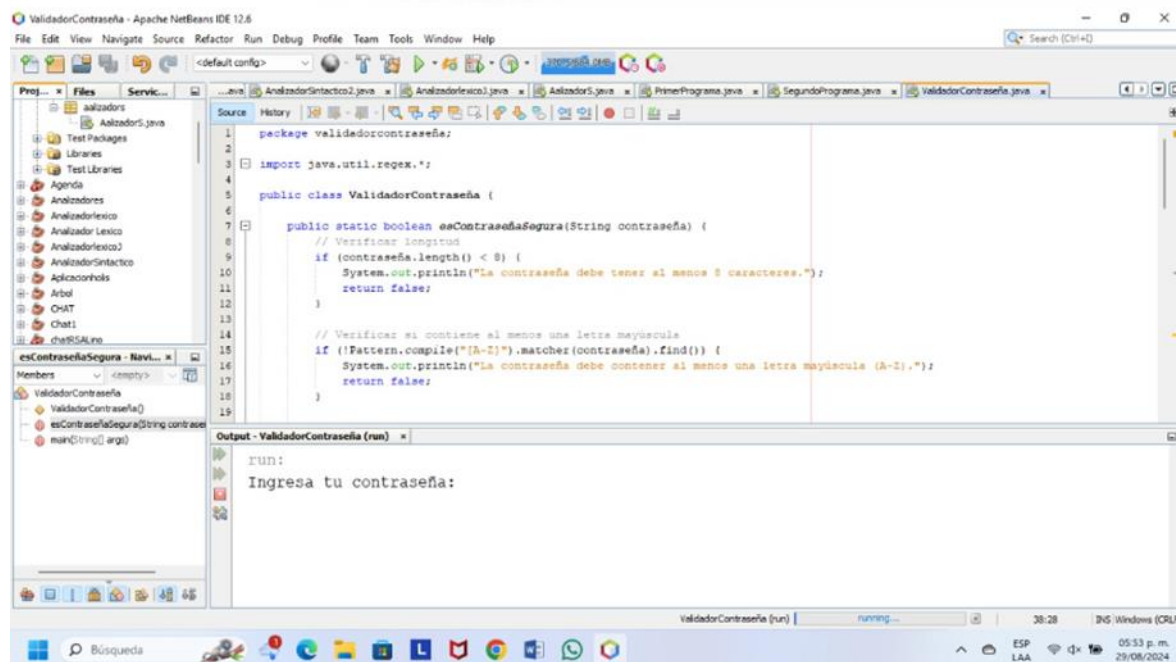
- Computadora
- Internet
- Lenguaje de programación (Java, Python o HTML5)

Desarrollo

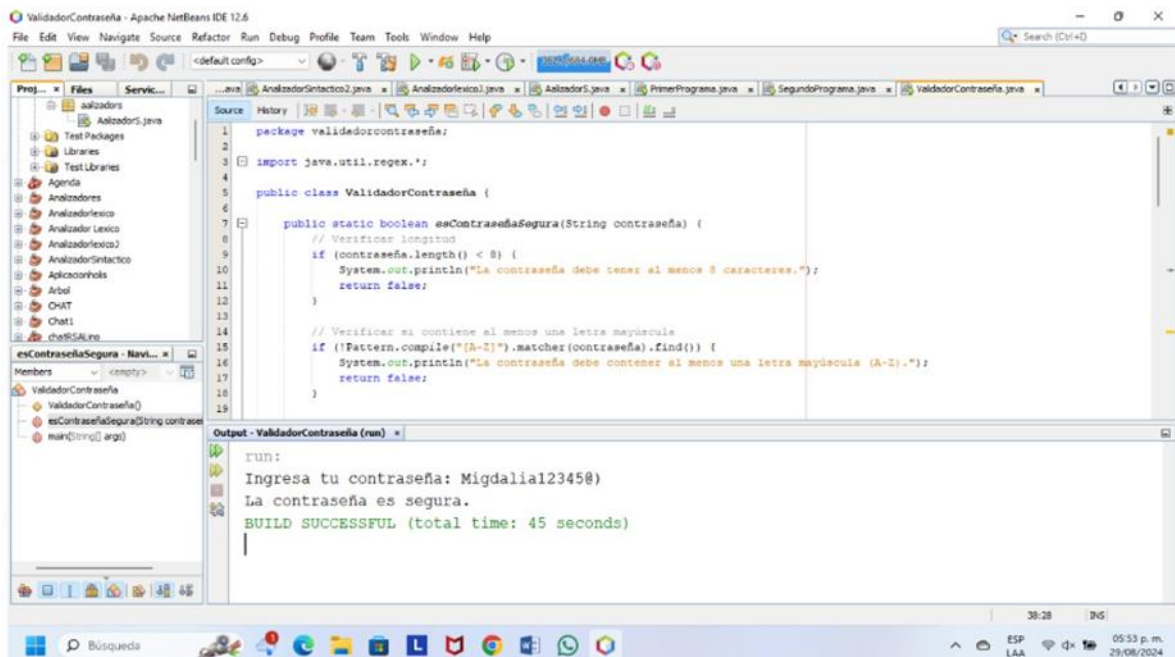
1.- Crea un programa en Python que permita al usuario ingresar una contraseña y que valide si la contraseña es segura o no. Una contraseña segura debe cumplir con los siguientes criterios basados en las recomendaciones de Google:

- Tener al menos 8 caracteres.
- Tener al menos una letra mayúscula (A-Z).
- Tener al menos una letra minúscula (a-z).
- Tener al menos un número (0-9).
- Tener al menos un carácter especial (!, @, #, \$, %, ^, &, *, (,), -, _, =, +, [,], {, }, |, \, ;, :, ', ", ,, <, >, /, ?, ~, `).
- No debe contener espacios en blanco.
- No debe tener más de 2 caracteres iguales consecutivos.
- Si la contraseña cumple con los criterios, el programa deberá mostrar un mensaje indicando que la contraseña es segura, de lo contrario, deberá mostrar un mensaje indicando que la contraseña no es segura.

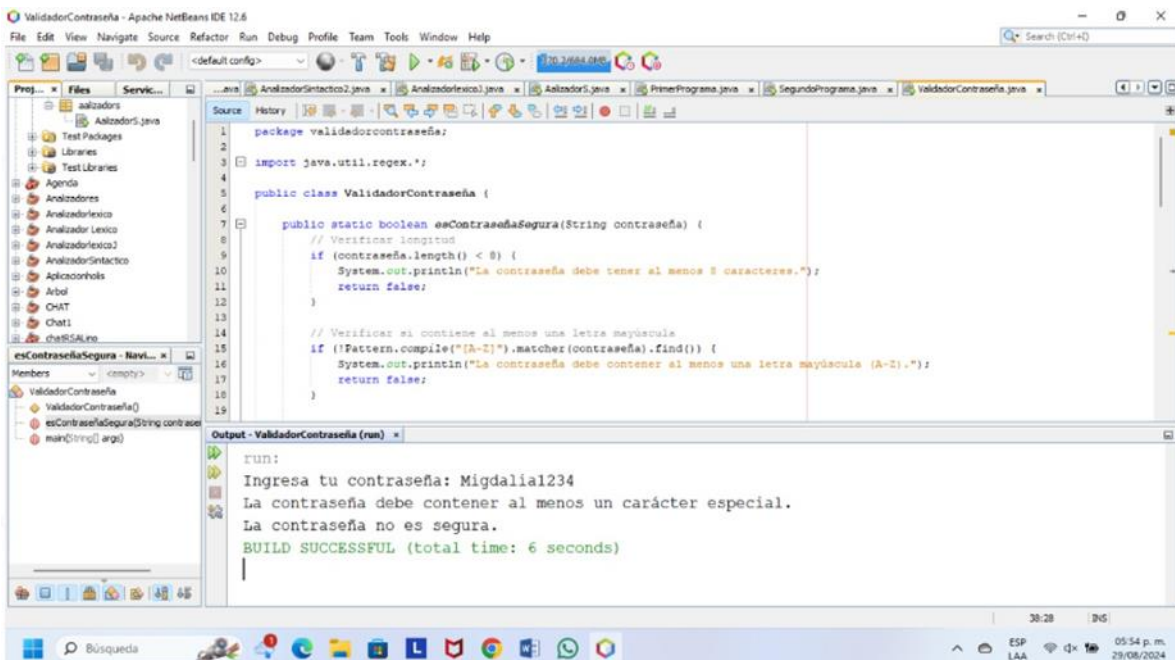
Para el desarrollo de este punto Utilizamos el lenguaje de programación de Java, quedando un código que principalmente al imprimir nos muestra un mensaje donde dice “Ingresa tu contraseña”, como se puede ver a continuación en la siguiente imagen:



Para comprobar si el código funciona de manera correcta, procederemos a correr el programa y escribiremos una contraseña que cumpla con todos y cada uno de los puntos señalados por la práctica, dándonos el siguiente resultado



Ahora aremos el mismo procedimiento, pero para una contraseña que no cumpla con alguno de los puntos señalados para esta práctica, quedando de la siguiente manera.



Código implementado

```
package validadorcontraseña;

import java.util.regex.*;

public class ValidadorContraseña {

    public static boolean esContraseñaSegura(String contraseña) {

        // Verificar longitud
        if (contraseña.length() < 8) {

            System.out.println("La contraseña debe tener al menos 8 caracteres.");

            return false;

        }

        // Verificar si contiene al menos una letra mayúscula
        if (!Pattern.compile("[A-Z]").matcher(contraseña).find()) {

            System.out.println("La contraseña debe contener al menos una letra mayúscula (A-Z).");

            return false;

        }

        // Verificar si contiene al menos una letra minúscula
        if (!Pattern.compile("[a-z]").matcher(contraseña).find()) {

            System.out.println("La contraseña debe contener al menos una letra minúscula (a-z).");

            return false;

        }

        // Verificar si contiene al menos un número
        if (!Pattern.compile("[0-9]").matcher(contraseña).find()) {

            System.out.println("La contraseña debe contener al menos un número (0-9).");

            return false;

        }

        // Verificar si contiene al menos un carácter especial
        if (!Pattern.compile("[!@#$%^&*()\|_-+=\\[\\]{}|\\\\\\\\;:\\\".,<>/?~`]").matcher(contraseña).find()) {
```

```

        System.out.println("La contraseña debe contener al menos un carácter
especial.");
        return false;
    }
    // Verificar si contiene espacios en blanco
    if (Pattern.compile("\\s").matcher(contraseña).find()) {
        System.out.println("La contraseña no debe contener espacios en blanco.");
        return false;
    }
    // Verificar si hay más de 2 caracteres iguales consecutivos
    if (Pattern.compile("(.)\\1{2,}").matcher(contraseña).find()) {
        System.out.println("La contraseña no debe tener más de 2 caracteres iguales
consecutivos.");
        return false;
    }
    return true;
}

public static void main(String[] args) {
    // Solicitar la contraseña al usuario
    java.util.Scanner scanner = new java.util.Scanner(System.in);
    System.out.print("Ingresa tu contraseña: ");
    String contraseña = scanner.nextLine();
    // Validar la contraseña
    if (esContraseñaSegura(contraseña)) {
        System.out.println("La contraseña es segura.");
    } else {
        System.out.println("La contraseña no es segura.");
    }
    scanner.close();
}
}

```

2.- Crea un programa que me recomiende una contraseña segura. La contraseña debe cumplir con los criterios de la instrucción anterior.

Explicación del código

Se comienza definiendo constantes que contienen los diferentes tipos de caracteres que se pueden usar en la contraseña

MAYÚSCULAS: Letras mayúsculas (A-Z).

MINÚSCULAS: Letras minúsculas (a-z).

DÍGITOS: Números (0-9).

SPECIAL_CHARACTERS: Caracteres especiales que aumentan la complejidad.

ALL_CHARACTERS: Combinación de todos los caracteres anteriores.

```
package codigo;

import java.security.SecureRandom;

public class PasswordGenerator {

    // Definir los caracteres que se pueden usar en la contraseña
    private static final String UPPERCASE = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String LOWERCASE = "abcdefghijklmnopqrstuvwxyz";
    private static final String DIGITS = "0123456789";
    private static final String SPECIAL_CHARACTERS = "!@#$%^&*()-_+={}|\\:;'\",.<>/?~`";
    private static final String ALL_CHARACTERS = UPPERCASE + LOWERCASE + DIGITS + SPECIAL_CHARACTERS;

    public static void main(String[] args) {
        // Generar y mostrar una contraseña segura
        String password = generatePassword(12); // Cambia el número si necesitas una contraseña más larga
    }
}
```

Método main()

En este es el punto de entrada del programa. Se llama al método generatePassword para generar una contraseña de 12 caracteres, que luego se imprime en la consola.

```
public static void main(String[] args) {
    // Generar y mostrar una contraseña segura
    String password = generatePassword(12); // Cambia el número si necesitas una contraseña más larga
    System.out.println("Contraseña generada: " + password);
}
```

Generación de Contraseña

Validación de Longitud: Verifica que la longitud de la contraseña sea al menos 8 caracteres. Si no se cumple, se lanza una excepción.

```
public static String generatePassword(int length) {
    if (length < 8) {
        throw new IllegalArgumentException("La longitud mínima de la contraseña debe ser 8 caracteres.");
    }
}
```


Generación aleatoria: Utiliza SecureRandom para generar números aleatorios de manera segura.

```
SecureRandom random = new SecureRandom();
StringBuilder password = new StringBuilder(length);

// Asegurar que la contraseña tenga al menos una letra mayúscula, una minúscula, un número y un c
password.append(UPPERCASE.charAt(random.nextInt(UPPERCASE.length())));
password.append(LOWERCASE.charAt(random.nextInt(LOWERCASE.length())));
password.append(DIGITS.charAt(random.nextInt(DIGITS.length())));
password.append(SPECIAL_CHARACTERS.charAt(random.nextInt(SPECIAL_CHARACTERS.length())));
```

Construcción de Contraseña: Asegura que la contraseña contiene al menos una letra mayúscula, una letra minúscula, un dígito y un carácter especial.

Mezcla de Caracteres (shuffleString)

Este método toma la contraseña generada y mezcla sus caracteres para evitar patrones predecibles.

```
// Método para mezclar los caracteres de la cadena
private static String shuffleString(String string) {
    char[] characters = string.toCharArray();
    SecureRandom random = new SecureRandom();
    for (int i = 0; i < characters.length; i++) {
        int randomIndex = random.nextInt(characters.length);
        char temp = characters[i];
        characters[i] = characters[randomIndex];
        characters[randomIndex] = temp;
    }
    return new String(characters);
}
```

Código en java

package codigo;

import java.security.SecureRandom;

public class PasswordGenerator {

 // Definir los caracteres que se pueden usar en la contraseña

 private static final String UPPERCASE =
 "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

 private static final String LOWERCASE = "abcdefghijklmnopqrstuvwxyz";

 private static final String DIGITS = "0123456789";

 private static final String SPECIAL_CHARACTERS = "!@#\$%^&*()-
 _+=[]{}|\\;:\",.<>/?~";

 private static final String ALL_CHARACTERS = UPPERCASE + LOWERCASE
 + DIGITS + SPECIAL_CHARACTERS;

 public static void main(String[] args) {

 // Generar y mostrar una contraseña segura

 String password = generatePassword(12); // Cambia el número si necesitas
 una contraseña más larga

```

        System.out.println("Contraseña generada: " + password);
    }

    public static String generatePassword(int length) {
        if (length < 8) {
            throw new IllegalArgumentException("La longitud mínima de la
            contraseña debe ser 8 caracteres.");
        }

        SecureRandom random = new SecureRandom();

        StringBuilder password = new StringBuilder(length);

        // Asegurar que la contraseña tenga al menos una letra mayúscula, una
        minúscula, un número y un carácter especial

        password.append(UPPERCASE.charAt(random.nextInt(UPPERCASE.length())));

        password.append(LOWERCASE.charAt(random.nextInt(LOWERCASE.length())));

        password.append(DIGITS.charAt(random.nextInt(DIGITS.length())));

        password.append(SPECIAL_CHARACTERS.charAt(random.nextInt(SPECIAL_CHARACTERS.length())));

        // Completar el resto de la contraseña con caracteres aleatorios

        for (int i = 4; i < length; i++) {
            password.append(ALL_CHARACTERS.charAt(random.nextInt(ALL_CHARACTERS.length())));
        }

        // Mezclar los caracteres para evitar cualquier patrón predecible

        return shuffleString(password.toString());
    }

    // Método para mezclar los caracteres de la cadena

    private static String shuffleString(String string) {
        char[] characters = string.toCharArray();

        SecureRandom random = new SecureRandom();

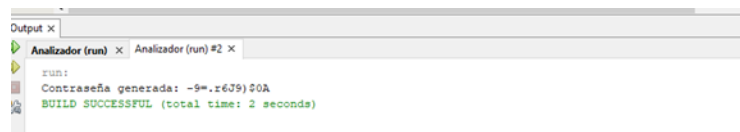
```

```

    for (int i = 0; i < characters.length; i++) {
        int randomIndex = random.nextInt(characters.length);
        char temp = characters[i];
        characters[i] = characters[randomIndex];
        characters[randomIndex] = temp;
    }
    return new String(characters);
}
}

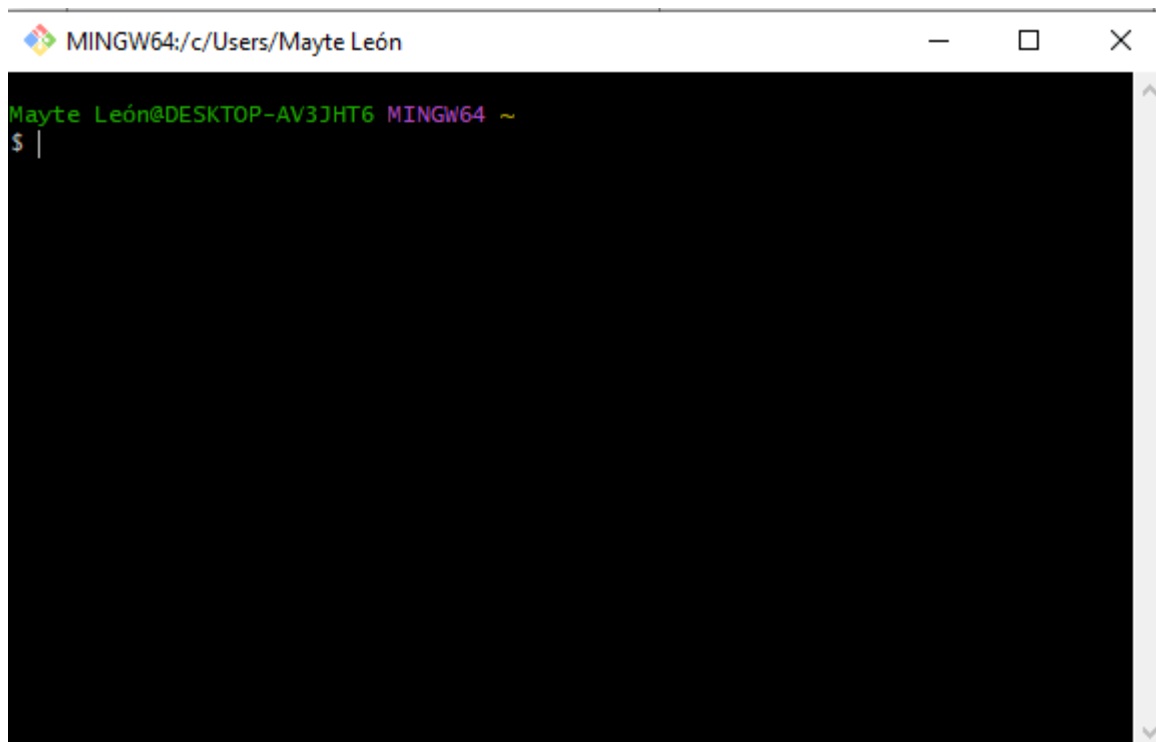
```

Resultado

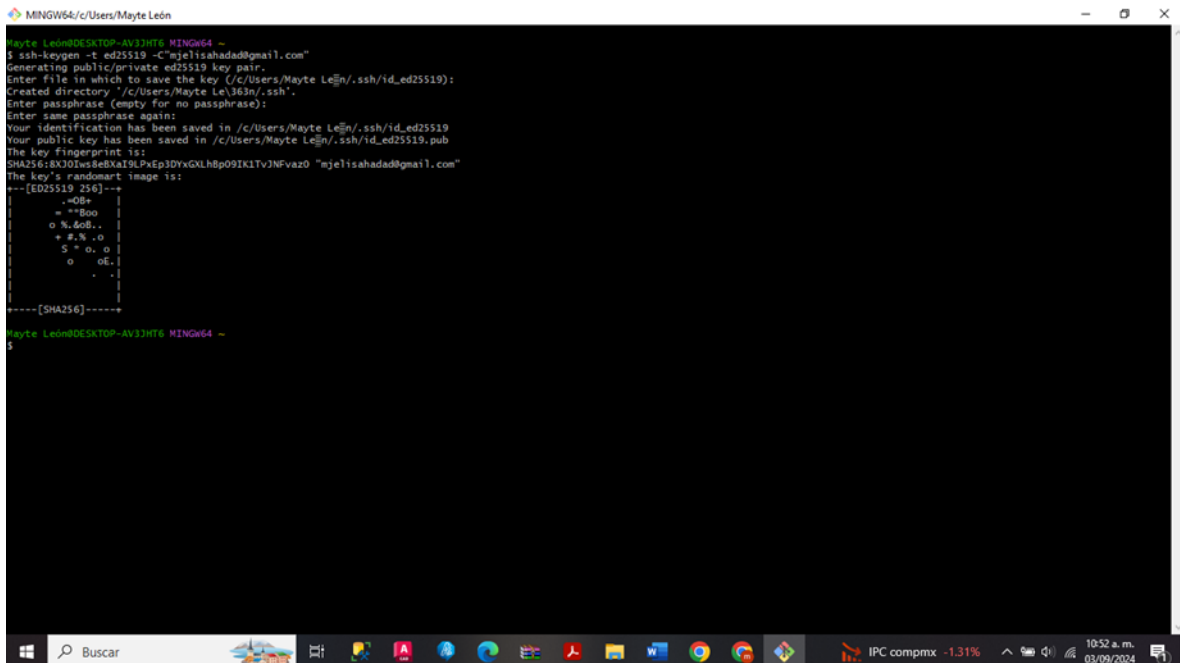


3.- Crea un certificado SSH, clave pública y clave privada, añade el certificado SSH a tu cuenta de GitHub y realiza un git clone de un repositorio nuevo utilizando la ruta SSH del repositorio.

a.- Lo primero que aremos será abrir nuestro git bash.



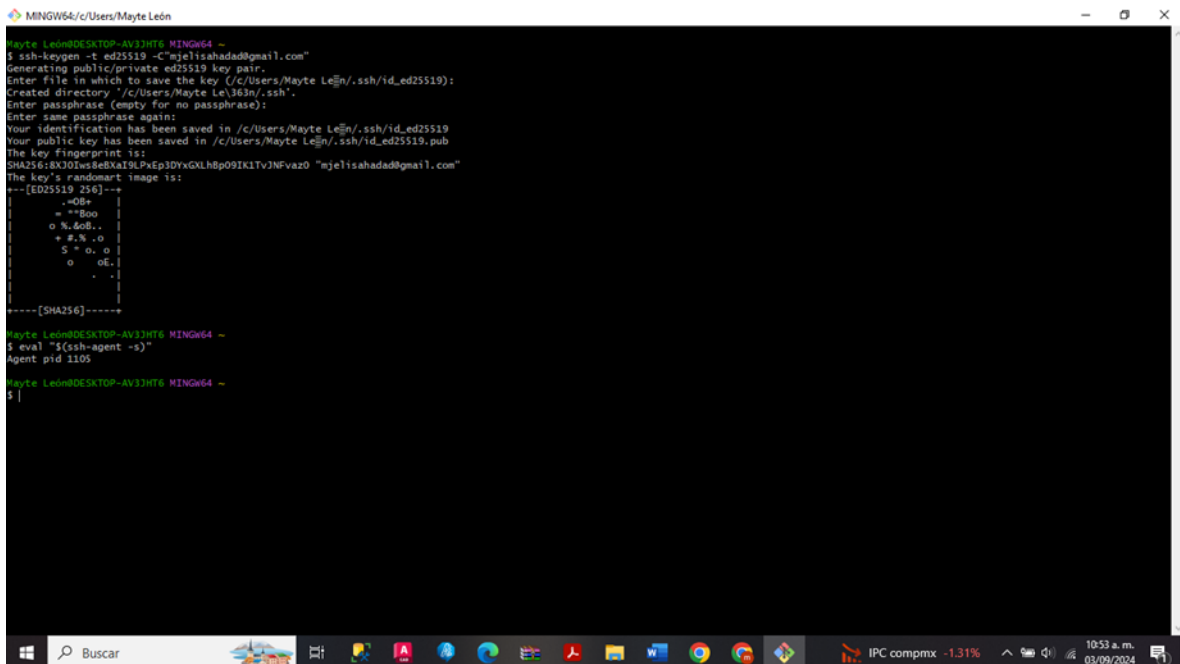
b.- Una vez abierta nuestra ventana agregaremos la siguiente instrucción `ssh-keygen -t ed25519 -Cmjelisahadad@gmail.com`



```
MINGW64/c/Users/Mayte León
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-keygen -t ed25519 -C"mjelisahadad@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Mayte León/.ssh/id_ed25519):
Created directory "/c/Users/Mayte León/.ssh".
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Mayte León/.ssh/id_ed25519
Your public key has been saved in /c/Users/Mayte León/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:8XJ0Iw8e8XaI9LpEp3DvXGLH8p09IK1TvJNFvaz0 "mjelisahadad@gmail.com"
The key's randomart image is:
---[ED25519 256]---
|.O=+ |
| = +Boo |
| o %&B.. |
| + #.% .o |
| S * o. o |
| o oE. |
| . . |
|-----[SHA256]-----

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$
```

c.- Posteriormente agregaremos la siguiente instrucción para continuar con el proceso eval `"$(ssh-agent -s)"`



```
MINGW64/c/Users/Mayte León
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-keygen -t ed25519 -C"mjelisahadad@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Mayte León/.ssh/id_ed25519):
Created directory "/c/Users/Mayte León/.ssh".
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Mayte León/.ssh/id_ed25519
Your public key has been saved in /c/Users/Mayte León/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:8XJ0Iw8e8XaI9LpEp3DvXGLH8p09IK1TvJNFvaz0 "mjelisahadad@gmail.com"
The key's randomart image is:
---[ED25519 256]---
|.O=+ |
| = +Boo |
| o %&B.. |
| + #.% .o |
| S * o. o |
| o oE. |
| . . |
|-----[SHA256]-----

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 1105

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$
```

d.- Siguiente instrucción par el proceso ssh-add ~/.ssh/id_ed25519

```
MINGW64~/c/Users/Mayte León
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-keygen -t ed25519 -C "mjelisahad@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/Mayte León/.ssh/id_ed25519):
Created directory '/c/Users/Mayte León/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Mayte León/.ssh/id_ed25519
Your public key has been saved in /c/Users/Mayte León/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:8XJOIw8eBxaI9LPxepJDYGLHbp09IKLTVjNFvaz0 "mjelisahad@gmail.com"
The key's randomart image is:
---[ED25519 256]---
      =+O= |
      o==Boo|
      o %doB..|
      + #.% o |
      S * o. o |
      o oE. |
      . . |
      |
-----[SHA256]-----

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 1105

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-add ~/.ssh/id_ed25519
Identity added: /c/Users/Mayte León/.ssh/id_ed25519 ("mjelisahad@gmail.com")

Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$
```

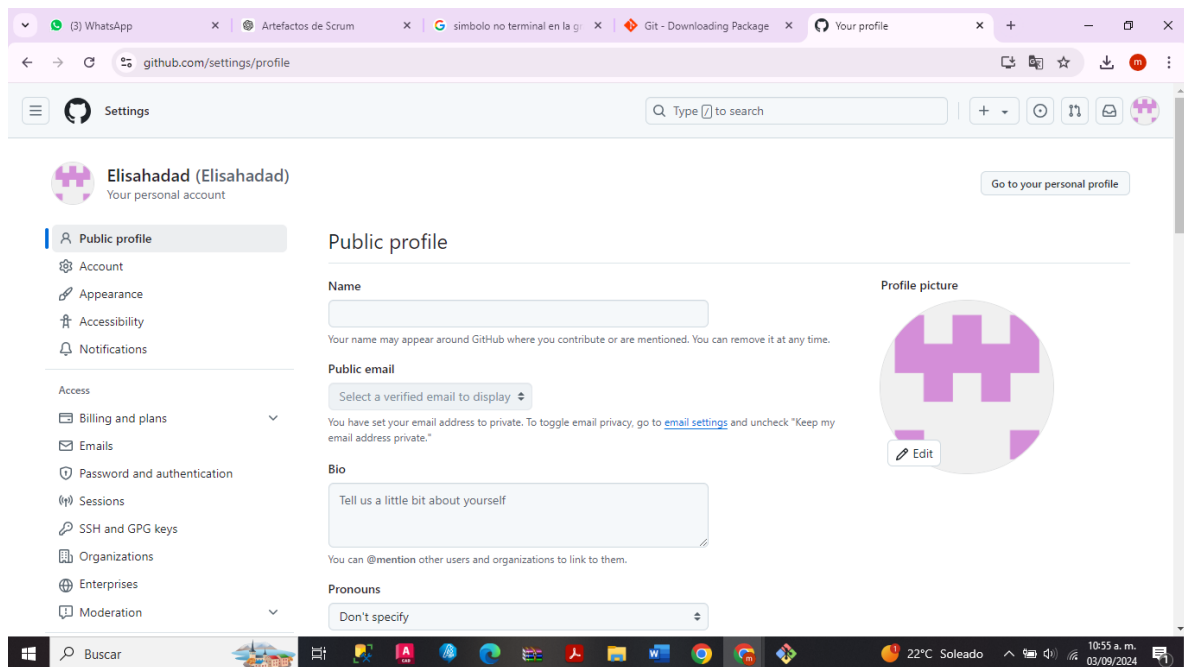
e.- La siguiente instrucción será cat ~/.ssh/id_ed25519.pub

```

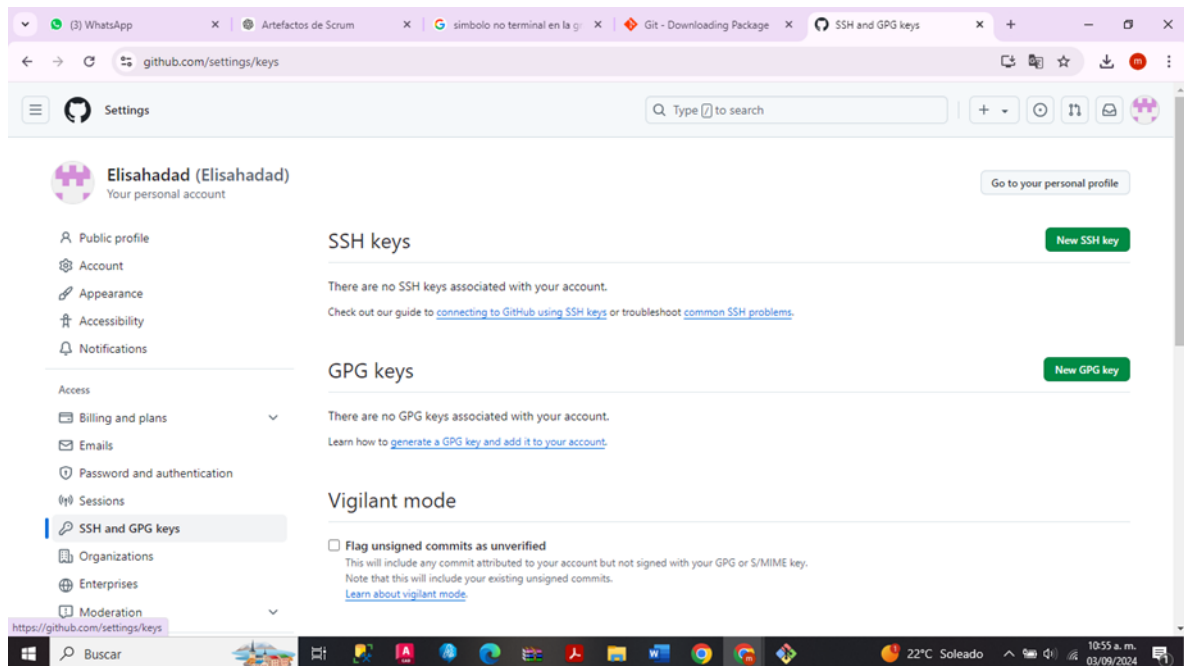
MINGW64:~\c\Users\Mayte León
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-keygen -t ed25519 -C "mjelishahad@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c:/Users/Mayte León/.ssh/id_ed25519):
Created directory '/c:/Users/Mayte León/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c:/Users/Mayte León/.ssh/id_ed25519
Your public key has been saved in /c:/Users/Mayte León/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:8XJOIws8e8XaI9LPxEPjDyXGLh8p0IKLTvJNFvaz0 "mjelishahad@gmail.com"
The key's randomart image is:
+--[ED25519 256]--+
|
|o=
|o  = "Boo
|o %&oB..
|+ #.% .o
|S " o. o |
|o oE|
|o .|
+-----+
|
|-----[SHA256]-----+
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 1105
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ ssh-add ~/.ssh/id_ed25519
Identity added: /c:/Users/Mayte León/.ssh/id_ed25519 ("mjelishahad@gmail.com")
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZD01ZTE5AAAAIB4WA6ia0B6YQZdIIA44F00HZ0VR0/kbPocBG8qR9kc "mjelishahad@gmail.com"
Mayte León@DESKTOP-AV3JHT6 MINGW64 ~
$

```

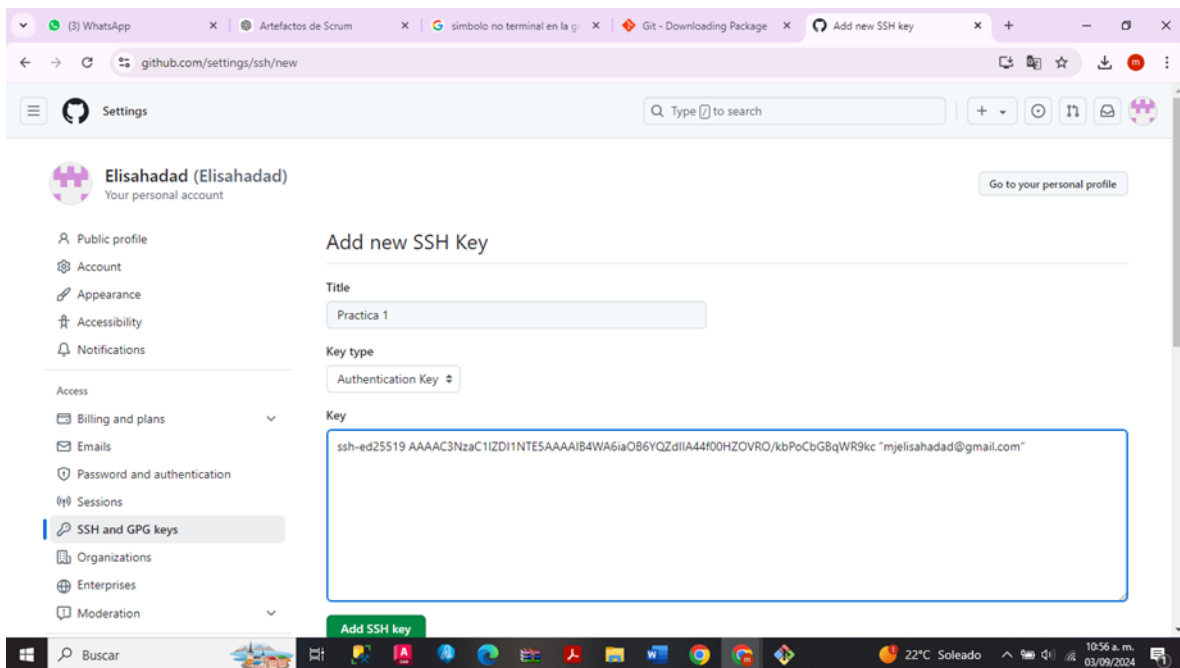
f.- Posteriormente abriremos la plataforma github.



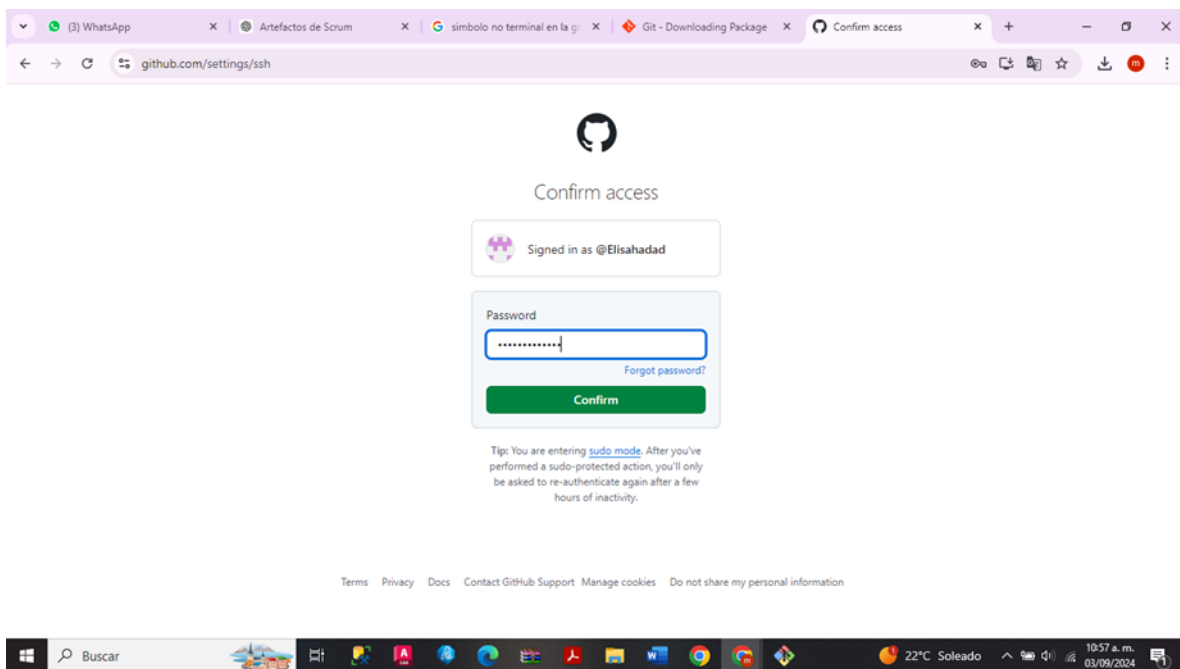
g.- Ya en nuestro perfil nos vamos a la opción que dice SSH and GPG keys.



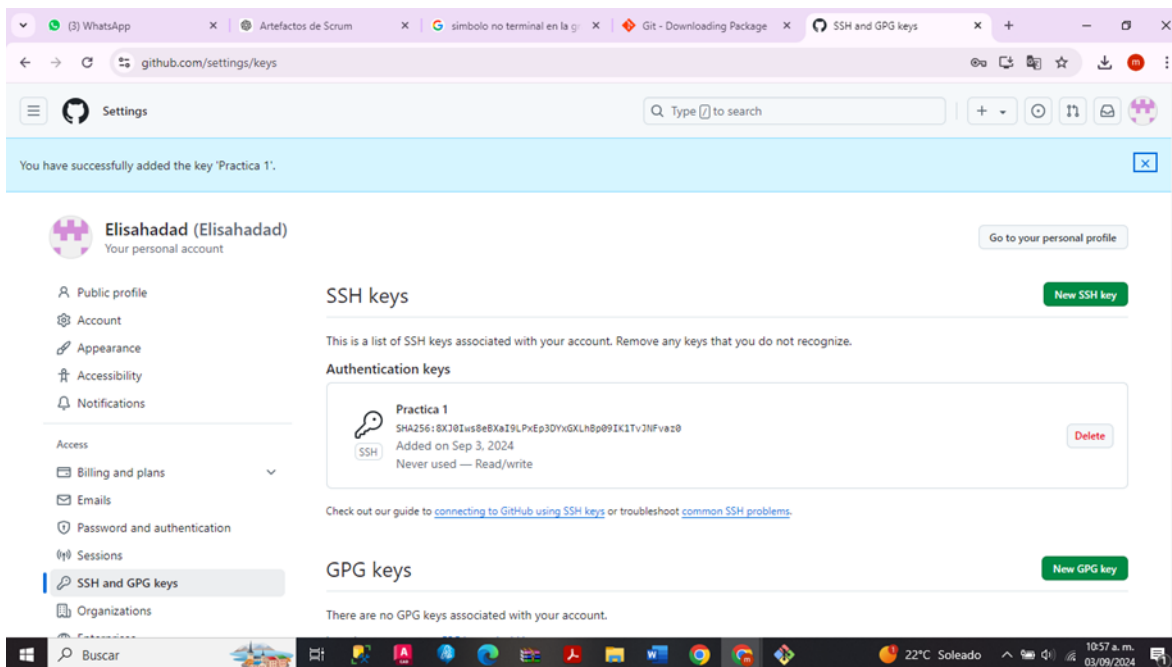
h.- Dentro de esta ventana damos clic al botón de New SSH key y nos dirigirá a la siguiente ventana.



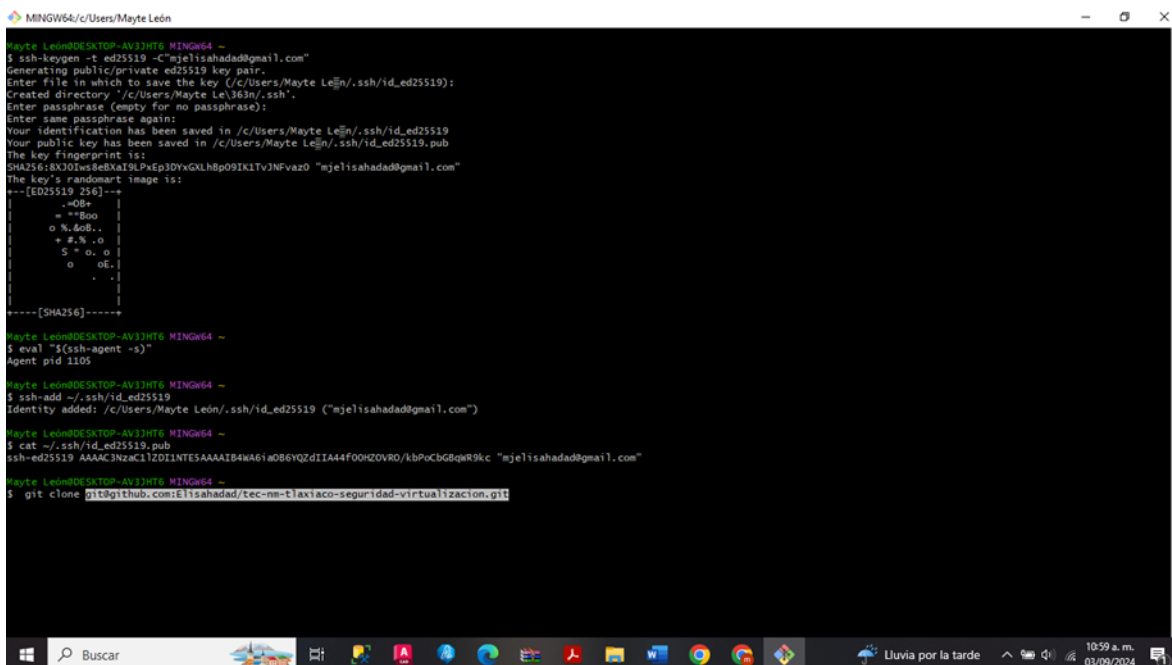
i.- Agregaremos el título del SSH y agregaremos el protocolo extensivo que nos proporcionó Git bash y daremos clic en la opción Add SSH key.



j.- Confirmamos el acceso para que se nos proporcione el SSH key.



k.- Una vez teniendo la llave, copiaremos el repositorio que queremos clonar y agregaremos lo siguiente git clone junto el URL.



Cifrado asimétrico: El cifrado asimétrico utiliza un par de claves relacionadas: una clave pública y una clave privada. La clave pública se utiliza para cifrar el mensaje, y solo la

clave privada correspondiente puede descifrarlo. Este tipo de cifrado es fundamental para la seguridad en Internet, ya que permite intercambiar información de manera segura sin necesidad de compartir la clave privada. Un ejemplo de cifrado asimétrico es el algoritmo RSA.

Picadillo (también conocido como Hashing): El picadillo, o función hash, es un proceso que convierte una entrada de datos en una cadena de longitud fija, generalmente representada por un valor hexadecimal. Las funciones hash se utilizan para verificar la integridad de los datos, como en la verificación de contraseñas o la firma digital. Ejemplos de algoritmos hash son SHA-256 y MD5.

Encriptación: Encriptación es el proceso de convertir datos en un formato codificado, llamado texto cifrado, para evitar que personas no autorizadas puedan acceder a la información. La encriptación puede ser simétrica o asimétrica, y es una práctica fundamental en la protección de la privacidad y la seguridad de la información.

6.- Investiga y describe los siguientes algoritmos de cifrado:

AES (Advanced Encryption Standard):

Descripción: AES es un algoritmo de cifrado simétrico ampliamente utilizado para proteger datos. Fue adoptado como estándar de cifrado por el gobierno de los Estados Unidos en 2001 y es reconocido por su seguridad y eficiencia. AES utiliza bloques de datos de 128 bits y admite claves de 128, 192 o 256 bits, lo que lo hace muy flexible y seguro.

Funcionamiento: AES cifra los datos en bloques de 128 bits utilizando una serie de transformaciones que incluyen sustitución, permutación, y mezclado de bits. Dependiendo de la longitud de la clave (128, 192 o 256 bits), el algoritmo realiza 10, 12 o 14 rondas de estas transformaciones para cifrar o descifrar la información.

Uso: Es ampliamente utilizado en diversas aplicaciones, incluidas las comunicaciones seguras, el almacenamiento de datos, y la protección de contraseñas.

RSA (Rivest-Shamir-Adleman):

Descripción: RSA es un algoritmo de cifrado asimétrico desarrollado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman. Es uno de los primeros algoritmos de criptografía de clave pública y sigue siendo ampliamente utilizado para asegurar datos, especialmente en el intercambio seguro de claves.

Funcionamiento: RSA se basa en la dificultad de factorizar grandes números enteros. Utiliza un par de claves, una clave pública para cifrar datos y una clave privada para descifrarlos. La seguridad de RSA radica en la longitud de las claves, que suele ser de 2048 bits o más en implementaciones modernas. Este método asegura que solo el poseedor de la clave privada pueda descifrar la información cifrada con la clave pública.

Uso: RSA es comúnmente utilizado en protocolos como HTTPS para asegurar la comunicación en Internet, en la firma digital, y en la encriptación de correos electrónicos.

SHA-256 (Secure Hash Algorithm 256-bit):

Descripción: SHA-256 es una función hash criptográfica que genera un valor hash de 256 bits (32 bytes) a partir de una entrada de datos de cualquier tamaño. Es parte de la familia de algoritmos SHA-2, desarrollados por la Agencia de Seguridad Nacional de los Estados Unidos (NSA).

Funcionamiento: SHA-256 toma una entrada y la procesa mediante una serie de operaciones bit a bit, mezclado y sustitución para producir un valor hash fijo de 256 bits. Este valor es único para cada entrada y cambiará drásticamente incluso si la entrada se modifica ligeramente, lo que garantiza la integridad de los datos.

Uso: SHA-256 se utiliza ampliamente en la verificación de integridad de archivos, en la generación de direcciones de criptomonedas como Bitcoin, y en la firma digital.

8.- Investiga y describe los siguientes estándares de cifrado:

SSL (Secure Sockets Layer):

Descripción: SSL es un protocolo criptográfico desarrollado por Netscape en la década de 1990 para asegurar la comunicación en la web. SSL permite que los datos transmitidos entre un servidor web y un navegador sean cifrados, asegurando así la privacidad y la integridad de la información.

Funcionamiento: SSL utiliza tanto cifrado simétrico como asimétrico para proteger los datos. Cuando un navegador se conecta a un servidor, se inicia un proceso llamado "handshake" donde se autentica el servidor (y opcionalmente el cliente), se acuerdan los algoritmos de cifrado a utilizar y se intercambian las claves para la sesión. Una vez completado el handshake, los datos se transmiten cifrados utilizando la clave simétrica acordada.

Uso: SSL fue ampliamente utilizado para proteger transacciones en línea, como compras y banca en línea. Sin embargo, debido a vulnerabilidades y problemas de seguridad, SSL ha sido reemplazado en gran medida por TLS.

TLS (Transport Layer Security):

Descripción: TLS es el sucesor de SSL y fue desarrollado para mejorar la seguridad del protocolo SSL. TLS proporciona cifrado, autenticación y la integridad de los datos transmitidos entre dos aplicaciones, como un navegador y un servidor web.

Funcionamiento: Similar a SSL, TLS comienza con un handshake que autentica las partes y acuerda los parámetros de cifrado. TLS ha mejorado en cuanto a seguridad y eficiencia en comparación con SSL, abordando varias vulnerabilidades que existían en SSL. TLS soporta versiones más avanzadas de algoritmos de cifrado y proporciona una seguridad más robusta.

Versiones: La primera versión de TLS (TLS 1.0) se lanzó en 1999, y desde entonces ha habido varias actualizaciones, siendo TLS 1.3 (publicada en 2018) la versión más reciente y

segura. Las versiones anteriores, como TLS 1.0 y 1.1, han sido descontinuadas debido a vulnerabilidades conocidas.

Uso: TLS es ahora el estándar de facto para asegurar la comunicación en la web. Se utiliza en HTTPS (HTTP Secure), correo electrónico seguro (como SMTPS), y en otros protocolos de red que requieren seguridad.

8.- Investiga y describe los siguientes protocolos de seguridad:

HTTPS (Hypertext Transfer Protocol Secure):

Descripción: HTTPS es una versión segura del protocolo HTTP (Hypertext Transfer Protocol), utilizado para la transferencia de datos en la web. HTTPS agrega una capa de seguridad mediante el uso de SSL (Secure Sockets Layer) o TLS (Transport Layer Security) para cifrar los datos que se envían y reciben entre un navegador y un servidor web.

Funcionamiento: Cuando un usuario accede a un sitio web mediante HTTPS, se establece una conexión segura entre el navegador y el servidor a través de un proceso conocido como "handshake", donde se autentican las partes y se acuerdan las claves de cifrado. Una vez establecida la conexión segura, los datos transmitidos, como formularios de entrada, cookies y otros contenidos, son cifrados, lo que protege la información contra interceptaciones y manipulaciones.

Uso: HTTPS es esencial para la seguridad en la web, especialmente en sitios que manejan información sensible como contraseñas, datos personales y detalles de tarjetas de crédito. Los navegadores modernos muestran un candado en la barra de direcciones para indicar que una conexión HTTPS es segura.

SFTP (Secure File Transfer Protocol):

Descripción: SFTP es un protocolo de red que permite la transferencia segura de archivos entre un cliente y un servidor. SFTP se basa en el protocolo SSH (Secure Shell) y proporciona capacidades de transferencia de archivos con autenticación segura y cifrado.

Funcionamiento: A diferencia de FTP (File Transfer Protocol), que transmite datos sin cifrar, SFTP cifra tanto los datos como las credenciales de usuario, garantizando la seguridad durante la transferencia. SFTP también permite operaciones de gestión de archivos, como renombrar, eliminar y cambiar permisos, de manera segura.

Uso: SFTP se utiliza comúnmente en entornos donde la seguridad es crucial, como en la transferencia de datos financieros, documentos legales y cualquier otra información confidencial. Es preferido sobre FTP y FTPS (FTP Secure) debido a su integración con SSH y su seguridad mejorada.

SSH (Secure Shell):

Descripción: SSH es un protocolo de red criptográfico diseñado para operar de manera segura sobre una red insegura. SSH proporciona una capa segura para la administración

remota de servidores y la transferencia de archivos, reemplazando herramientas más antiguas como Telnet y rlogin, que transmitían datos sin cifrar.

Funcionamiento: SSH utiliza cifrado asimétrico para autenticar la conexión entre el cliente y el servidor, y luego establece un canal seguro mediante cifrado simétrico para la comunicación. SSH permite a los usuarios ejecutar comandos en un servidor remoto, transferir archivos, y realizar túneles de puertos para encapsular otros protocolos de red dentro de un canal SSH seguro.

Uso: SSH es fundamental en la administración de sistemas y redes, especialmente en entornos Linux y Unix. Los administradores de sistemas lo utilizan para acceder de forma segura a servidores remotos, transferir archivos de manera segura, y realizar tareas de mantenimiento y configuración remotas.

Conclusión

Las contraseñas y los certificados son pilares fundamentales para la protección de los sistemas y la información. Las contraseñas, cuando se gestionan adecuadamente,

constituyen la primera barrera contra accesos no autorizados, mientras que los certificados digitales aseguran la autenticación y el cifrado de las comunicaciones, garantizando la integridad y confidencialidad de los datos. En un entorno virtualizado, donde la seguridad es crítica, la correcta implementación y gestión de contraseñas y certificados es esencial para mantener la confianza, proteger recursos sensibles y asegurar el acceso controlado a los sistemas.

Bibliografía

[Firma digital, ¿qué es y para qué sirve? \(ciberseguridad.com\)](#)

[¿Qué es el cifrado? Definición de cifrado de datos | IBM](#)

[¿Qué son los certificados digitales? | Fortinet](#)