Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Uso de álgebras modernas para seguridad y criptografía

Ingeniería en Ciencias de Datos y Matemáticas

Documentación

Profesores
Dr. Alberto Francisco Martínez Herrera, Dr. Salvador Mancilla Hernández

Socio Formador
LiCore

Equipo 1

José Andrés Meyer Crabtree	A01366785
Diego Paasche Portillo	A01028103
Luis Leopoldo Jiménez Pérez	A01275004
Federico Medina García Corral	A01721441
Eugenio Andrade Lozano	A01721296
César Guillermo Vázquez Álvarez	A01197857

Monterrey, Nuevo León. 7 de febrero de 2023

Índice general

1. Requisitos técnicos

Primero se va a analizar la topología con la que se trabajó. Se tiene 2 auditores, un Centro de Control y conexión vía Wi-Fi. Para poder utilizar este servicio, los 2 auditores serán 2 Raspberry Pi 3 Modelo B, el cual está compuesto por un procesador ARMv8 con 4 puertos USB, conexión HDMI, ethernet, Wi-Fi, Bluetooth entre otras cosas. Su procesador de 64 bits soporta el rango completo de distribución ARM GNU/Linux y Microsoft Windows 10, de igual manera, soporta herramientas de software como Python, Gnu, y otros.

Se recomienda utilizar el Raspberry Pi 3 Modelo B como mínimo requisito y de ser posible, utilizar Raspberry de nuevas generaciones así como lo es la Raspberry Modelo B+. El Centro de Control es el encargado de tener la base de datos, en este caso, lo recomendado sería una computadora portátil o de escritorio con los siguientes componentes (para poder utilizar este servicio sin ningún problema): un SSD y memoria RAM de 8 GB (la ventaja de estos 2 elementos es que se pueden encontrar a precios bajos). El requerimiento mínimo del procesador para poder correr el servicio es un Intel Core i3. En caso de que el Centro de Control se encuentra en una computadora de escritorio, de igual manera se necesitará tener un monitor. Realmente cualquier monitor logra funcionar para este servicio, el requerimiento mínimo sería que fuera al menos de 60 Hz, que es lo normal en el mercado ya que, para esto, tener un monitor de más de 60 Hz pueda ser que sea un gasto innecesario.

Una tarjeta de video no es necesario ya que con el procesador i3 es más que suficiente. Por otra parte, es necesario y un requisito indispensable el utilizar el lenguaje de programación Python. Como recomendación se puede decir que siempre se utilice la versión más nueva que, en este caso, es la 3.11, lanzada en enero de 2023. Dentro de esto, también es indispensable utilizar Django REST, el cual es un framework de desarrollo web de código abierto. Cabe recalcar que estos son los requerimientos mínimos para poder utilizar este servicio, lo recomendable es estar alrededor de los requisitos mencionados anteriormente, sin embargo, no es indispensable que sean exactamente iguales.

2. Licencias

En el sistema se utilizaron solamente librerías y software de uso público. A pesar de que sean de uso público, las librerías y el software utilizado tienen un excelente uso y entregan muy buenos resultados. Es por esto que se decidió utilizarlos para la elaboración de este servicio de encriptamiento mediante curvas elípticas.

- **Python 3**
- **Numpy 1.22.3**: Numpy es una librería de Python para el cálculo numérico y el análisis de datos.
- **Pandas 1.4.3**: Esta sirve para el manejo y análisis de estructuras de datos.
- **Matplotlib.pyplot 3.5.1**: Pyplot es un módulo Matplotlib que propone varias funciones sencillas para añadir elementos tales como líneas, imágenes o textos a los ejes de un gráfico.
- **Datetime**: Esta es una librería la cual permite el manejo y manipulación de fechas y horas.
- **Hashlib 1.5**: Hashlib sirve para hashear, como bien el mismo nombre lo dice.
- **Socket 3.3**: Al importar socket en Python se puede generar un enlace entre 2 aplicaciones para que de esta manera tengan comunicación.
- **Time default**: Time, no es lo mismo que datetime, la librería time sirve para retornar el valor en fracciones de la suma del sistema y el tiempo de CPU del usuario del proceso actual.
- **Json 1.6.2**: Json es una librería que sirve para transferir información a través de la web y para almacenar ajustes y configuración en formato JSON. Este, de igual forma, es un formato ligero de intercambio de datos.

- **Django 4.0**: Django es una de la librerías que más destacan dentro del servicio. Esta sirve para construir cualquier tipo de sitio web de manera eficiente mediante una API.
- **Pathlib 1.0**: Esta sirve para las clases que representan rutas del sistema de archivos con semántica apropiada para diferentes sistemas operativos.
- **Ecdsa 0.18.0**: La librería Ecdsa ayuda a la creación y verificación de firmas.
- **GetMac 0.9.0**: GetMac nos proporciona la MAC Address.
- **Os 3.4**: Os es una librería que provee una manera versátil de usar funcionalidades dependientes del sistema operativo.

3. Instalación, compatibilidad y dependencias

Para poder utilizar este servicio de encriptación mediante curvas elípticas, se instaló Python 3 el cual es el lenguaje de programación con el que se trabajó este proyecto. Este es de uso público así es que es muy probable que no se tenga ningún problema en descargarlo ni en utilizarlo. De igual manera, las librerías mencionadas en la Sección 2 tienen versiones donde sí son compatibles y otras que no lo son con Python 3. A continuación se muestran las versiones con las que sí son compatibles en caso de que quein repita el proceso conllevado por el equipo lo logre sin problemas:

- **Numpy (importada como np)**: Compatibilidad de versión 1.3.0 a 1.5.3
- **Pandas (importada como pd)**: Compatibiliad de versión 1.16.0 a 1.24.0
- **Matplotlib.pylot (importada como plt)**: Compatibilidad de versión 3.0 a 3.6
- **Datetime**: Versión viene por default
- **Hashlib (importada como sha256)**: Compatibilidad de versión 1.5 a 1.5.2
- **Socket**: Compatibilidad de versión 3.3 a 3.7
- **Time**: Versión viene por default
- **Json**: Compatibilidad de versión 1.1.1 a 1.6.3
- **Django**: Compatibilidad de versión 3.1 a 4.1
- **Pathlib (importada como path)**: Compatibilidad de versión 0.6 a 1.0.1
- **Ecdsa**: Compatibilidad de versión 0.18.0
- **Getmac**: Compatibilidad de versión 0.9.1

Por la parte de Django, se importaron librerías como Django Paranoid Model en la versión 1.0.7, Django-RestFramework, DjangoHexadecimalField en la versión 0.0.3 y Django Extensions en la versión 3.2.1. El funcionamiento de cada librería viene explicado en la Sección 2, de igual forma se muestran las versiones de cada una de estas librerías con un rango de versiones anteriores a versiones más nuevas que tienen compatibilidad con el servicio de encriptación.

Es necesario que al descargar estas librerías, la computadora que se esté utilizando esté conectadas a internet para poder importar las librerías dentro de Python. Así mismo, para poder utilizar este servicio, se necesita una buena conexión a internet ya que de esta manera es como se están mandando los archivos encriptados del Centro de Control hacia los auditores.

Para la instalación de las Raspberry, es necesario meter las trazas a estas para que, de esta forma, se pueda compartir la información de los auditores al Centro de Control. Cabe recalcar que las trazas (base de datos) tuvieron un pre-procesamiento antes de agregarlas a las Raspberry para darles el formato correcto para el envío de información y se añaden al utilizar Python dentro de las Raspberry. Este nuevo formato se encuentra claramente explicado en el Reporte Técnico al igual que se muestra en las Figuras 12 y 13 dando como resultado lo mostrado en la Tabla 9 de dicho documento.

4. Documentación de la Arquitectura del Código

Dentro del código se encuentran diferentes archivos y carpetas que de manera unida forman el API y la arquitectura que se decidió hacer. En la parte principal se encuentra la carpeta "Archivos trazas", donde se encuentran los archivos .csv que se utilizaron para hacer pruebas y saber cómo se envía la información. También, está la carpeta Auditores, en donde se encuentra el código que se encarga de mandar las solicitudes a la API, verificar y tomar la acción correspondiente en base a esto, ya sea guardar la información o no aceptarla.

La carpeta "Exploracion" contiene los archivos que se utilizaron para entender mejor la información que se va a recibir, incluye gráficas y pruebas JSON, hashing y ECDSA. Los archivos .gitignore, README.md y requirements.txt son archivos que ayudan a facilitar en hacer los archivos más limpios. El archivo .gitignore incluye todos los tipos de archivos que no se buscan guardar, por ejemplo, la base de datos local. El archivo README.md es justo el archivo donde toda esta información se está mostrando, finalmente el requirements.txt guarda todas las librerías que se tienen que instalar para que el código se pueda correr.

Finalmente, la carpeta más importante es la de "main", aquí se encuentran todos los modelos y la comunicación entre ellos. Cada carpeta representa un modelo específico y todos tienen los mismos archivos. Estos archivos son:

- **Carpeta "migrations"**: Contiene las migraciones (cambios en los modelos) que se han hecho, esto ayuda a que la base de datos cambie al mismo tiempo que se cambian los modelos.
- **init.py**: Permite guardar la información dentro de la carpeta para importarla como si fuera un paquete de Python.
- **admin.py**: Registra el modelo en el sitio del administrador junto con la información que se quiere mostrar.
- **models.py**: Están los modelos (tablas) que se están creando. Incluye los campos del modelo y su representación como texto.
- **serializers.py**: Se encarga de convertir la información que se recibe (en formato JSON) al formato que usa Django y vice versa. Esto para poder guardar la información en la base de datos y si se busca visualizarla que también se pueda.
- **viewsets.py**: Se encarga de permitir hacer las solicitudes, las más comunes siendo POST para crear información, GET para obtener información, PUT o PATCH para actualizar información y DELETE para borrar. Dentro también incluye los permisos para las acciones.
- **urls.py**: Contiene los URLs que se van a agregar de su respectivo modelo para poder mandar la solicitud.

4.1. Modelos y sus campos:

- **Auditor (Auditores)**
  - Nombre (name)
  - MAC Address (mac_address)
- **Certificate (Certificados)**
  - ID del Auditor (auditor)
  - ID del Centro de Control (control_center)
  - ID de la Llave Pública (public_key)
  - Sí está autorizado (is_authorized)
  - Fecha de Expiración (expiring_date)

- **ControlCenter (Centro de Control)**
  - Nombre (name)
  - Dominio (domain)
- **Entry (Entradas)**
  - ID del Auditor (auditor)
  - Fecha y hora (date)
  - Produciendo o Consumiendo (is_producing) de tipo booleano (True = produciendo, False = consumiendo)
  - Cantidad (quantity)
  - Emisor MAC (mac_emisor)
  - IP del Receptor (ip_receptor)
  - Firma (signature) de tipo hexadecimal
- **Output (Salida o Respuesta)**
  - ID del auditor
  - Mensaje (message), ya sea Upload, Shutdown o Disconnect
- **PublicKey (Llave Pública)**
  - Algorithm. En este caso ECDSA (Se deja para futuras implementaciones de otros posibles algoritmos)
  - Public key (public_key) de tipo hexadecimal
- **User (Usuarios)**:
  - Mail (email)
  - Contraseña (password)

Para poder correr el servidor son necesarios los siguientes comandos:

- **pip install -r requirements.txt**
- **cd main**
- **python3 manage.py migrate**
- **python3 manage.py runserver**

5. Características principales y Ejemplo

5.1. Generación de un certificado SSL autofirmado para el desarrollo local de Django

Los siguientes pasos son obtenidos y referenciados del sitio timonweb.com que se adjunta en anexos.

5.1.1. Paso 1 - Generación de un certificado SSL local

1. En primer lugar, vamos a instalar mkcert en la máquina la cual será el Centro de Control.

```
choco install mkcert
```

2. A continuación, vamos a hacer que tu Sistema Operativo confíe en los certificados locales que vamos a generar. Para ello, debes instalar una autoridad de certificación (CA) local en el almacén de confianza del sistema. Ejecute el siguiente comando:

```
mkcert -install
```

3. A continuación, necesitas generar un certificado para el dominio localhost.

   En el terminal, ve a la raíz de tu proyecto Django. A continuación, ejecute el siguiente comando de terminal para generar un certificado para las redes que utilizará:

```
mkcert -cert-file cert.pem -key-file key.pem localhost 127.0.0.1 10.0.0.1
```

   Esto es solo un ejemplo de posibles redes, si todo está bien, se le generará dos archivos cert.pem y key.pem.

5.1.2. Paso 2 - Configurar el servidor Django para trabajar con HTTPS

El comando por defecto de Django manage.py runserver no soporta SSL; por lo tanto, necesitamos usar el comando alternativo manage.py runserver_plus, que es parte del paquete Django Extensions.

1. Ejecuta el siguiente comando para instalar las extensiones Django junto con el servidor Wekzeug:

```
pip install django-extensions Werkzeug
```

2. A continuación, dentro del archivo settings.py en el editor de código y se añade django_extensions a la lista INSTALLED_APPS:

```
INSTALLED_APPS = [
    # other apps
    "django_extensions",
]
```

3. Por último, iniciar el servidor de desarrollo local en modo HTTPS ejecutando el comando

```
python manage.py runserver_plus --cert-file cert.pem --key-file key.pem
```

   Y finalmente se debería ver el servidor de desarrollo local ejecutándose en la dirección https://localhost:8000 por defecto.

5.2. ECDSA realizado por nosotros (versión no-óptima)

```
#Sacar el grupo E
def E(a, b, p):

    puntos = set()

    for x in range(p):
        for y in range(p):
            y_2 = (x**3 + a*x + b) % p
            if (y**2 % p) == y_2:
                puntos.add((x, y))
                if y != 0:
                    puntos.add((x, -y % p))

    return puntos
```

Figura 1: Función para encontrar el grupo E bajo los parámetros a, b y p

```
#Encontrar el orden de un grupo (q)
def orden(Grupo):

    orden_puntos = len(Grupo) + 1

    return orden_puntos
```

Figura 2: Función para encontrar orden del grupo ingresado, equivalente a la variable q

```
# inverso aditivo
def inversoAditivo(x, p):

    if x <= (p-1):
        return (p - x)

    else:
        return (p - (x%p))
```

Figura 3: Función para calcular el Inverso Aditivo

```
#inverso multiplicativo
def inversoMultiplicativo(x, p):

    resultado = 0
    for y in range(p):
        if (x * y) % p == 1:
            resultado = y
    return resultado
```

Figura 4: Función para calcular el Inverso Multiplicativo

```
#pendiente entre dos puntos en la curva eliptica
def slope(x1, y1, x2, y2, a, p):
    if (x1 == x2 and y1 == y2):
        return ((3 * x1 ** 2) + a) * inversoMultiplicativo((2 * y1), p) % p

    else:
        return ( (y2 + inversoAditivo(y1, p)) * inversoMultiplicativo(x2 + inversoAditivo(x1, p), p)) % p
```

Figura 5: Función para calcular la pendiente entre dos puntos dentro de una Curva Elíptica

```python
#suma de curva
def sumaCurva(x1, y1, x2, y2, p, s):

    x3 = ((s**2) + inversoAditivo(x1,p) + inversoAditivo(x2, p)) % p
    y3 = (s * (x1 + inversoAditivo(x3, p)) + inversoAditivo(y1, p)) % p

    return x3, y3
```

Figura 6: Función para calcular la suma de la curva

```python
#exponenciacion binaria para curvas elipticas
def expbin(x1, y1, k, p, a):

    k = bin(k)[3:]
    x2 = x1
    y2 = y1

    for i in k:
        if i == "1":
            s = slope(x2, y2, x2, y2, a, p)
            x2, y2 = sumaCurva(x2, y2, x2, y2, p, s)
            s = slope(x2, y2, x1, y1, a, p)
            x2, y2 = sumaCurva(x1, y1, x2, y2, p, s)

        else:
            s = slope(x2, y2, x2, y2, a, p)
            x2, y2 = sumaCurva(x2, y2, x2, y2, p, s)

    return x2, y2
```

Figura 7: Función para poder utilizar el método de la Exponenciación Binaria

```python
#definir la inversa de un grupo para valores no primos
def Euler(Grupo, p):

    inversa = []
    phi_n = len(Grupo)

    for i in Grupo:
        inversa.append((i ** (phi_n - 1)) % p)

    if len(set(inversa)) != phi_n:
        print("No todos tienen inversa, no es un grupo")
    else:
        print("Todas tienen inversa")
    return inversa
```

Figura 8: Función para encontrar las inversas de los elementos dentro de un grupo utilizando el método de Euler

```python
#definir la inversa de un grupo para valores no primos
#grupo de primos relativos
def U_n(n, inferior, superior):

    U = []
    for i in range(inferior, superior + 1):
        if rec_mcd(i, n) == 1:
            U.append(i)

    return U
```

Figura 9: Función para crear el grupo de los primos relativos de la variable n

```python
#residuo del maximo comun divisor
def rec_mcd(a, b):
    res = b % a
    if res > 0:
        return rec_mcd(res, a)
    if res == 0:
        return a
```

Figura 10: Función poder realizar la operación del residuo del máximo común divisor

Teniendo las funciones descritas, se mostrará el siguiente ejemplo

- $h(x) = 12$
- $k_E = 11$

Primeramente, se tuvo que establecer la función de la curva elíptica, la cual es $y^2 \equiv x^3 + 2x + 2 \; mod \; 17$ (se tomó esta curva de manera arbitraria únicamente para mostrarla como ejemplo de cómo funciona el código). Después, se tuvieron que encontrar los puntos por los que pasa la curva al igual que el orden. Para esto se utilizaron las funciones de la Figura 1 y de la Figura 2.

```python
# Numero de elementos de E

q = orden(E(2, 2, 17))
```

Figura 11: Elementos de la curva elíptica y su orden, la cual es q = 19

Después, se tiene que encontrar el valor de B, el cual está dado por $B = dA$. Este se encontró con A y d y utilizando la función expbin() (exponenciación binaria) de la siguiente forma.

```python
# B = dA -> 10*(5,1)

A = [5,1]
d = 10
p = 17
a = 2

B = expbin(A[0], A[1], d, p, a)
```

Figura 12: El punto B da la coordenada (7, 11)

En seguida, se debe encontrar el valor de R, el cual se calcula utilizando la exponenciación binaria. Para este se utilizó el siguiente código.

```python
# R = (K_E)*A -> 11*(5,1)

K_E = 11

R = expbin(A[0], A[1], K_E, p, a)
```

Figura 13: La coordenada generada R es (13,10)

Utilizando R se puede encontrar el valor de x_R, el cual es la coordenada en x de R. Con el resultado de R, se puede observar que el valor $x_R = 13$, la cual se representará con la variable r. Después, para poder calcular $s = (h(x) + d \cdot r) \cdot k_E^{-1} \; mod \; q$, falta el valor de k_E^{-1}, el cual es la inversa en el grupo de primas relativas del k_E que se asignó para este ejercicio. Para este, se creó el grupo U_{19} donde el 19 representa el valor de q, es decir, el orden de la Curva Elíptica, y utilizando la función de Euler se encontró el inverso del valor $k_E = 11$ el cual es $k_E^{-1} = 7$

```
# Inverso de K_E en el grupo U19 (orden de E = q = 19)

Grupo_U19 = U_n(19, 1, 19)

Inversa_U19 = Euler(Grupo_U19, 19)
K_E_Inversa = Inversa_U19[K_E-1]
```

Figura 14: El valor de la inversa de 11 en el grupo U_{19} es 7

Ya teniendo todos los valores necesarios, ahora sí se puede realizar la firma, la cual sigue la función de s mostrada anteriormente y al final de la Sección 1.1. Ingresando todos los valores encontrados esta sería $s = (12 + 10 \cdot 13) \cdot 7 mod 19$ la cual es igual a 6. Con esto, ya se completó lo que es la Generación de Firmas.

```
# s = (h(x) + d*r) * inversa(K_E) mod q

h_x = 12
d = 10

s = (h_x + d*r) * K_E_Inversa % q
```

Figura 15: Generación de la Firma ($s = 6$)

Ahora, para poder hacer la Verificación de la Firma, se debe de seguir los pasos que se encuentran en la Sección 1.2. Primeramente, se debe de calcular el valor auxiliar w, el cual se muestra a continuación.

```
# Calcular el auxiliar w = intersa(s) mod q

s_Inversa = Inversa_U19[s-1]
s_Inversa

w = s_Inversa % p
```

Figura 16: Cálculo del auxiliar $w = 16$

Después, se debe encontrar el otro valor auxiliar tanto para u_1 como para u_2. En el siguiente código se muestra cómo se calcularon dichos valores.

```
#Calcular u1 = w * h(x) mod q

u1 = w * h_x % q
```

Figura 17: Cálculo del auxiliar $u_1 = 2$

```
# Calcular u2 = w * r mod q

u2 = w * r % q
```

Figura 18: Cálculo del auxiliar $u_2 = 18$

Como último cálculo, se debe encontrar el valor de P, el cual está dado por $u_1 \cdot A + u_2 \cdot B$. Este cálculo se da uniendo las coordenadas de A y B junto con sus respectivas constantes, lo cual requiere calcularse cada una con exponenciación binaria para después sumarlas usando la función de Suma de la Curva, encontrada en la Figura 6. El siguiente código muestra cómo se elaboró dicho cálculo.

```
# Calcular P = u1*A + u2*B y sacar x_P

A_u1 = expbin(A[0], A[1], u1, p, a)
B_u2 = expbin(B[0], B[1], u2, p, a)

pen = slope(A_u1[0], A_u1[1], B_u2[0], B_u2[1], a, p)

#P = A_u1 + B_u2
P = sumaCurva(A_u1[0], A_u1[1], B_u2[0], B_u2[1], p, pen)
```

Figura 19: Cálculo de P, el cual arroja $P = (13, 10)$

Finalmente, se debe verificar si la coordenada en x de P (x_P) es congruente con $r \, mod \, q$, para poder saber si la firma es válida o no. Para esto, se creó el siguiente código, con lo que se encontró que efectivamente la firma es válida.

```
# x_p es congruente con r mod q?

xp = P[0]

if xp == r % q:
    print("Si es valido")
else:
    print("No es valido")
```

Figura 20: Validación de la Firma, la cual dice que la creada evidentemente es válida

5.3. ECDSA utilizado (versión más eficiente)

Debido a que se realizó personalmente el algoritmo de ECDSA solamente faltó optimizar el algoritmo de firmado ya que tardaba alrededor de una hora en realizar el firmado de los datos debido a la magnitud de los números. Es importante tener en cuenta que la optimización de un algoritmo de cifrado como ECDSA puede ser un proceso complejo y requiere una comprensión profunda de los detalles técnicos del algoritmo. Por lo tanto, es recomendable trabajar con un especialista en seguridad criptográfica para lograr los mejores resultados. Es por eso que utilizamos la biblioteca ECDSA en Python.

Para el código de la realización del firmado de datos, primero se tienen que empaquetar y hashear los datos que se van a mandar en el formato de envío para que desde el Centro de Control puedan volverlos a hashear y realizar la verificación.

```
# creando las llaves de firmado
sk = SigningKey.generate(curve=NIST256p)

# private key
signing_key_hex_string = sk.to_string().hex()
```

Figura 21: Creación de la llave pública y privada

```
# creando la firma
sk = SigningKey.from_string(bytearray.fromhex(signing_key_hex_string), curve=NIST256p)
signature = sk.sign(hashed.encode('utf-8'))
signature_hex_string = signature.hex()
```

Figura 22: Generación de la Firma

En la verificación de la firma se necesita que antes el Centro de Control se haya comunicado con el auditor para establecer una clave pública mutua entre ambos. Con la clave pública establecida y la firma se puede hacer la validación.

```
def hash_dict(d):
    hash_string = ""
    for key, value in sorted(d.items()):
        hash_string += str(key) + str(value)
    return hashlib.sha256(hash_string.encode()).hexdigest()
```

Figura 23: Función para poder hacer el hashing

```python
class EntryViewSet(viewsets.ModelViewSet):
    # Asignar un queryset con todos los objetos Entry de la base de datos
    queryset = Entry.objects.all()
    # Asignar la clase EntrySerializer como el serializador utilizado por la clase
    serializer_class = EntrySerializer
    # Definir que los permisos necesarios para acceder a los métodos de la clase son estar autenticado
    permission_classes = [permissions.IsAuthenticated]

    # Método que se ejecuta antes de crear un nuevo objeto Entry
    def perform_create(self, serializer):
        # Obtener el nombre del auditor a partir de los datos enviados en la solicitud
        auditor = self.request.data.get('auditor')
        date = self.request.data.get('date')
        is_producing = self.request.data.get('is_producing')
        mac_emisor = self.request.data.get('mac_emisor')
        ip_receptor = self.request.data.get('ip_receptor')
        quantity = self.request.data.get('quantity')
        signa = self.request.data.get('signature')

        json_package = {"auditor": auditor, "date": date, "is_producing": is_producing, "quantity":
        ↪  quantity,
                        "mac_emisor": mac_emisor, "ip_receptor": ip_receptor}
        hashed = hash_dict(json_package)

        # Buscar el primer certificado asociado al auditor
        certificate = Certificate.objects.filter(auditor__id=auditor).first()
        pubObj = PublicKey.objects.get(id=certificate.public_key.pk)
        publiKey = pubObj.public_key

        verifyingkey = VerifyingKey.from_string(bytearray.fromhex(publiKey), curve=NIST256p)
        signature = bytearray.fromhex(signa)
        try:
            print('Verify transmited data', verifyingkey.verify(signature, hashed.encode('utf-8')))
        except:
            print(" ")

        # Comprobar si existe un certificado asociado al auditor
        if certificate:
            # Comprobar si el certificado ha caducado
            if certificate.check_expiry():
                # Guardar el nuevo objeto Entry
                serializer.save()
            else:
                # Lanzar una excepción ValidationError si el certificado ha caducado
                raise ValidationError("Certificate has expired.")
        else:
            # Lanzar una excepción ValidationError si no se encuentra un certificado asociado al auditor
            raise ValidationError("No matching certificate found for the given auditor_id")
```

Figura 24: Verificación de la firma

5.4. Ejemplo video

El video mostrado en la Sección 7 del documento es el ejemplo del funcionamiento de la red creada. Al inicio de dicho video, se observa en la pantalla de televisión las 2 tarjetas Raspberry Pi 3 que representan los auditores. Cuando se intenta iniciar la comunicación desde el auditor, esta falla ya que el Centro de Control no ha creado los certificados para ambas. Al crear el certificado e iniciar la comunicación nuevamente, se recibe el mensaje 201 que indica que se establece la comunicación con el Centro de Control, pero la comunicación está apagada desde el CC, por lo que no se enciende ningún foco indicador de comunicación. Esto se comprueba con ambos auditores.

Tras comprobar que la comunicación se puede establecer y que el permiso del CC sí es esencial para recibir los datos, se cambia el permiso para autorizar la comunicación. Una vez cambiado el permiso, los focos de cada Raspberry se encienden y esto confirma la comunicación exitosa. El permiso desde el Centro de

Control se activa y desactiva múltiples veces durante la comunicación para verificar que los certificados y la creación de la red estén correctos.

6. Ejecución del proyecto en un eterno local propio (Paso a paso).

Para poder correr el siguiente proyecto necesita realizar los siguientes pasos, además de tener registradas las llaves de acceso personales en Github.

6.1. Centro de control

- **git clone git@github.com:cesarxyz/Equipo1_MA2006B.git**
  Para futuras actualizaciones si es que ya tiene el repositorio en su sistema, realice **git pull**
- **pip install -r requirements.txt**
- **cd main**
- **python3 manage.py migrate**
- Sin certificado SSL **python3 manage.py runserver**
  Con certificado SSL **python manage.py runserver_plus –cert-file cert.pem –key-file key.pem**
- Para crear el usuario administrador: **python3 manage.py createsuperuser**
- Llenar el modelo de Centro de Control, con los datos de la organización que autorizará el certificado al auditor.
- Esperar que un Auditor se registre.
- Crear un certificado con el auditor, la clave pública y la organización que autoriza.

6.2. Auditor

- **git clone git@github.com:cesarxyz/Equipo1_MA2006B.git**
  Para futuras actualizaciones si es que ya tiene el repositorio en su sistema, realice **git pull**
- **pip install -r requirements.txt**
- **cd Auditores**
- Entrar al archivo post_auditor_0.py con su editor de texto favorito o en su defecto **nano post_auditor_0.py** y se guardan cambios con **ctrl + o** y salir con **ctrl + x**.
- Cambiar el dominio o IP del Centro de Control y nombrar al auditor en las siguientes variables:

```
# Puede ser el dominio, ejemplo tec.mx
ip_receptor = "10.22.207.160"

# El nombre que se le dara al dispositivo
auditor_name = "Auditor 0"
```

Figura 25: Variables a cambiar

- De igual forma si gusta eliminar o editar los puertos de salida de la Raspberry Pi, editar o eliminar las siguientes lineas:

```
# Librería de los pines de la Raspberry
import RPi.GPIO as GPIO

# Puerto donte está conectado el led
LED_PIN = 11
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)
```

Figura 26: Variables de salidas GPIO

Si se requiere cambiar las acciones de subprocesos solo remplazar las acciones después de cada condicional de tipo if.

```
if data["auditor"] == auditor_pk:
    if data["message"] == '1':
        print("Upload")
        # Salida para encender el LED.
        GPIO.output(LED_PIN, GPIO.HIGH)
    elif data["message"] == '2':
        print("Shutdown")
        # Salida para apagar el LED.
        GPIO.output(LED_PIN, GPIO.LOW)
    elif data["message"] == '3':
        print("Disconnect")
    else:
        print("Message null")
```

Figura 27: Variables de opción para la comunicación bidireccional

- Finalmente correr **python3 post_auditor_0.py**

6.3. Recomendaciones

- Si encuentra algún problema, revisar los registros de errores y depurar el código si es necesario.
- Realizar un seguimiento constante de los cambios en el repositorio y actualizar su copia local con regularidad.
- Realizar pruebas regulares para asegurarse de que el proyecto sigue funcionando como se esperaba.
- Documentar sus avances y cualquier problema que encuentre para hacer un seguimiento de su progreso y poder ayudar a otros desarrolladores que puedan trabajar en el proyecto en el futuro.

7. FAQs.

- **¿Es necesario utilizar alguna computadora con componentes de baja gama?** Lo que se recomienda es siempre utilizar componentes como los que recomendamos en el primer apartado de requisitos técnicos.
- **¿Alguien que no tiene conocimiento dentro de la programación podrá utilizar este servicio?** Lo mas recomendable es que este servicio sea utilizado por gente ya con algún conocimiento previo ya que, si en un futuro necesita hacer cambios, este pueda realizarlos sin ningun problema.
- **¿Se puede utilizar algún otro lenguaje de programación** La respuesta corta es sí se puede, pero se tendría que transcribir todo el código al nuevo lenguaje de programación.
- **¿Cúanto tiempo me tardaría en entender todo el código si tengo conocimientos básicos de programación** Lo que se recomienda es tener ya conocimiento previo a programación dado a que dentro del código hay funciones, librerías que ya necesitan conocimiento previo.

- **¿Alguien sin conocimiento podrá utilizar este servicio?** Sí, dentro de la documentación y dentro del reporte técnico se explica a detalle cómo funciona y un ejemplo para que el ususario se pueda ir guiando sin ningún problema.
- **¡No dudes en ponerte en contacto con nosotros!** Para mas detalles del código puedes comunciarte con nosotros al correo A01197857@tec.mx.

8. Anexos.

Video del funcionamiento de la red creada:

https://drive.google.com/file/d/1H4de6kfVGiRmpFXF1Xv-7k4XLfE4U7hd/view?usp=sharing

Cómo ejecutar un servidor local de desarrollo Django sobre HTTPS con un certificado SSL autofirmado de confianza

https://timonweb.com/django/https-django-development-server-ssl-certificate/