



INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

ESCUELA DE INGENIERÍA Y CIENCIAS

INGENIERÍA EN CIENCIAS DE DATOS Y MATEMÁTICAS

CAMPUS MONTERREY, NUEVO LEÓN

**Andrea Bravo Avila A01028579**

**Renata Vargas Caballero A01025281**

**Natalia Monserrat González De León A00831090**

**Fernando González Rosas A01253694**

**Alberto Lozano Cárdenas A01067141**

**Fernanda Sherlin Calderón López A01275430**

**Gil Herzberg Alperon A00827347**

6 DE JUNIO DEL 2023

## **Documnetación**

USO DE ÁLGEBRAS MODERNAS PARA SEGURIDAD Y CRIPTOGRAFÍA

GRUPO 602

DR. ALBERTO F. MARTÍNEZ

OSF: LiCORE

## 1. Condiciones de Uso

La topología utilizada contiene dos auditores, una Raspberry Pi 3, un centro de control en un servidor de AWS, y conexión a WiFi. La Raspberry tiene un procesador Quad Core 1.2GHz Broadcom BCM2837 de 64 bits, con RAM de 1 Gb, y capacidad de programar en Python y C/C++. El *centro de control* es donde se decriptan los datos y es capaz de recibir información y almacenarla. En este caso el centro de control es un servidor de Amazon Web Services, por lo que se necesita únicamente una computadora con acceso a internet para poder acceder a él. En necesario tener la posibilidad de correr códigos de Python para poder utilizar el servicio con el objetivo de que se puedan encriptar y decriptar los datos que se recompilan en los auditores.

## 2. Requisitos de Instalación

Para poder utilizar el sistema se debe de contar con Python 3.10.11, y tener instaladas las librerías:

- **hashlib** versión 1.5
- **pyOpenSSL** versión 23.2.0
- **json** versión por default
- **flask** versión 2.3.2
- **pymysql** versión 1.0.3
- **requests** versión 2.31.0
- **Cryptography** versión 40.0.2
- **time** versión por default
- **pandas** importado como pd, versión 2.0.2
- **csv** versión por default
- **paramiko** versión 3.2.0

Además se debe de contar con un servidor AWS, para este proyecto se utilizó la versión gratuita de este servidor, el cual permite su uso por un tiempo limitado; en este se deben de configurar los certificados, así como la base de datos S2. En el computador también se debe tener instalado MySQL para poder conectarse a la base de datos RDS desde la línea de comandos.

## 3. Arquitectura del sistema

En el github se encuentran los archivos requeridos para utilizar el sistema, así como los datos de prueba y otra información relevante para el uso apropiado de este.

A continuación se enlistan los archivos que pueden encontrarse en el github, y un poco de información de lo que estos contienen:

- **Documentacion\_Cripto.pdf** Cuenta con la documentación de todo el sistema.
- **apache-selfsigned.crt** Certificado de autenticidad obtenido con OpenSSL.
- **app.py** Crea la aplicación de flask como endpoint para HTTPS.
- **borrar\_registros.py** Limpia la base de datos después de hacer pruebas.
- **certificados.txt** Contiene los comandos a seguir para generar los certificados.
- **genera\_llaves.py** Código de Python que genera las llaves publicas y privadas.
- **mandado\_hash\_firma.py** Código que genera el hash y la firma.
- **preprocesado\_de\_datos.ipynb** Procesado de la base de datos de prueba para que funcione en conjunto con los códigos.
- **prueba.py** Prueba de generar la conexión de MySQL a la base de datos
- **query.py** Código de automatización con ssh de la raspberry y la RDS en MySQL.
- **requirements.txt** Contiene todas las librerías y módulos que se deben instalar.

## 4. Documentación

### 4.1. Creación del servidor AWS

El centro de control es un servidor AWS, para ellos se crea una cuenta, y se establece un servidor. El cual se configura más adelante para utilizar como centro de control.

### 4.2. Configuración de la base de datos AWS RDS

Se configura una base de datos del tipo RDS (Relational Data Service) en el servidor de AWS. Para ello se elije el motor MySQL para la base de datos, la instancia es EC2, con clase db.t3.micro. Aquí también se le da un nombre a la base de datos. Los demás parametros se dejaron en el modo default.

### 4.3. Configuración del servidor AWS EC2

Se crea una instancia de EC2 en AWS, en la cual se elige una AMI (Amazon Machine Image) en este caso es la plataforma de Amazon Linux, con un tipo de instancia t2.micro. Los demás valores se dejaron en default. Con la EC2 creada se puede uno conectar usando SSH, como resulta en la figura 2. Para esto se usa el comando:

```
1 ssh -i <archivo_claves>.pem <instancia de EC2 de AWS>
2 chmod <archivo_claves>.pem
```

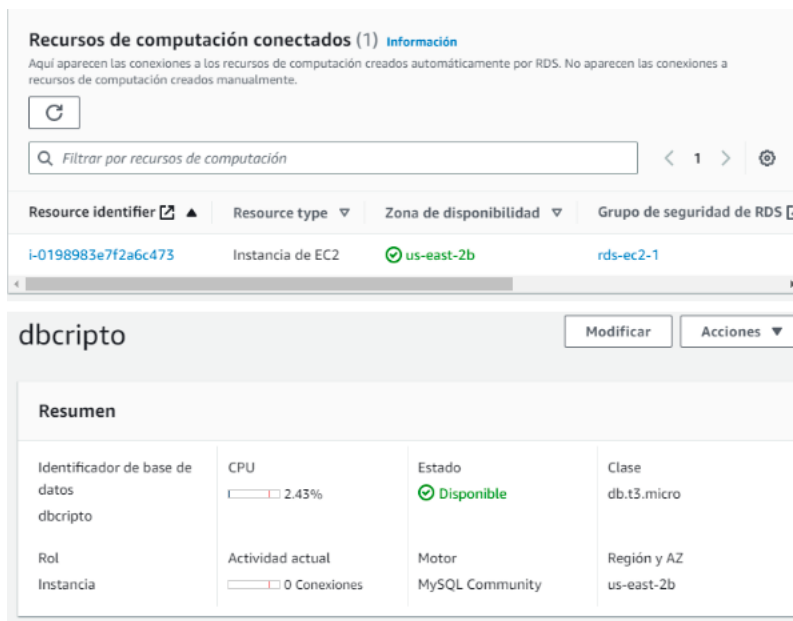


Figura 1: Configuración de la base de datos AWS RDS

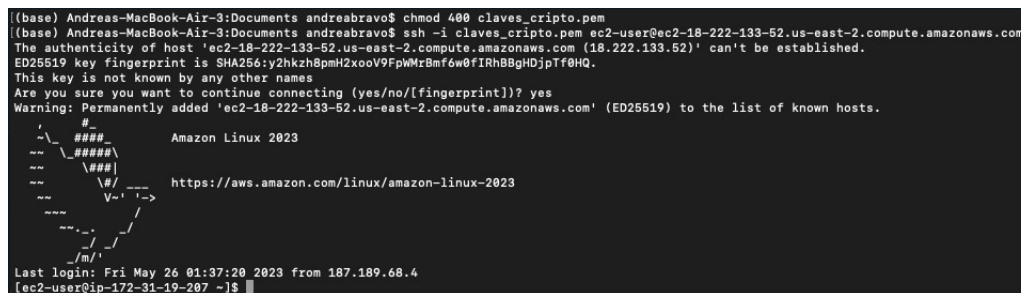


Figura 2: Resultado del comando de ssh

#### 4.4. Aplicación Flask

Se crea una aplicación Flask que recibe datos a través de un punto final POST, que es una dirección endpoint en una aplicación web que acepta solicitudes HTTP enviadas al servidor y los mete en la base de datos RDS de AWS. En este código se verifica el hash y la firma; y corre el app para establecer la conexión. La aplicación toma la clave pública que le enviará la raspberry, se conecta a la base de datos en AWS, define una ruta para recibir las solicitudes POST donde al ejecutarse se verifica la firma y el hash y si son correctos los inserta con SQL al RDS.

```
1 def receive_data():
2     data = request.get_json()
3
4     # Hash
5     data_string = json.dumps({k: v for k, v in data.items() if k not in ['hash', 'signature']})
6     data_hash = hashlib.sha256(data_string.encode()).hexdigest()
7     if data_hash != data['hash']:
8         return 'Hash verification failed', 400
```

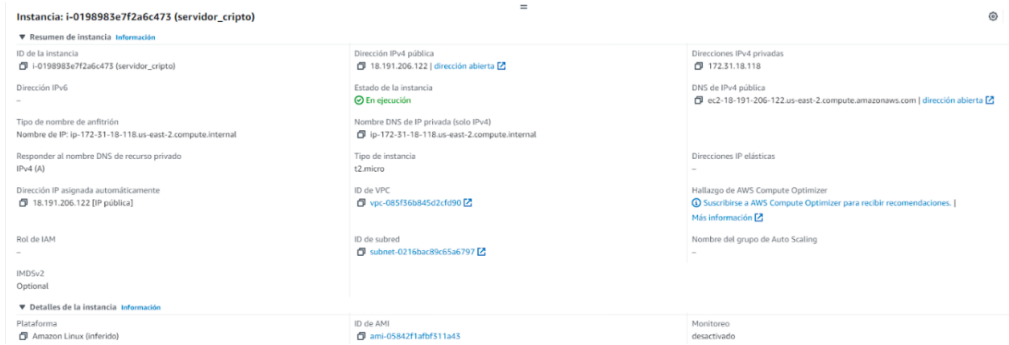


Figura 3: Configuración del servidor AWS EC2

```

9
10 # Firma
11 try:
12     public_key.verify(
13         bytes.fromhex(data['signature']),
14         data_string.encode(),
15         padding.PSS(
16             mgf=padding.MGF1(hashes.SHA256()),
17             salt_length=padding.PSS.MAX_LENGTH
18         ),
19         hashes.SHA256()
20     )
21 except Exception as e:
22     return 'Signature verification failed', 400

```

Listing 1: Sección de app.py

## 4.5. Instalación y configuración de los certificados SSL en la instancia EC2

Se genera un certificado SSL autofirmado, los certificados son usualmente firmados por instituciones, pero en este caso se hace un proceso de autofirmado. Con el certificado es posible establecer conexiones seguras HTTPS en la aplicación Flask establecida en 4.4. Estos se crean con los comandos:

```

1 sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache
-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt

```

EL certificado se guarda de manera local y puede ser accesado con los siguientes comandos:

```

1 scp -i "MyKeyPair.pem" ubuntu@ec2-18-191-206-122.us-east-2.compute.amazonaws.com:/etc/
ssl/certs/apache-selfsigned.crt

```

## 4.6. Conexión de MySQL a RDS

Desde la terminal de comandos se ingresan los comandos a continuación para conectarse a la base de datos RDS de AWS, y después ahí se ejecutan comandos de SQL, que permiten visualizar los registros.

```

1 mysql -h <host> -u <user> -p <database>

```

donde  $o$  que esta entre  $j_i$ , se sustituye por los datos de la base de datos definida anteriormente. Con la conexión establecida, se puede crear la tabla e ingresar los datos a RDS desde a consola.

```
1 USE dbcrypto; CREATE TABLE MeterReadings ( EntryID INT, ID VARCHAR(255),
ConsumptionOrProduction INT, Day INT, Month INT, Year INT, Time TIME, Reading FLOAT,
PRIMARY KEY (EntryID) );
```

## 4.7. Raspberry Pi

En la Raspberry se debe de primero instalar el lenguaje de Python para de ahí poder correr los códigos que permiten encriptar los datos.

### 4.7.1. Claves

El código de `genera_llaves.py` genera un par de claves publica y privada con el algoritmo de RSA y los guarda en un archivo local.

```
1 from cryptography.hazmat.primitives import serialization
2 from cryptography.hazmat.primitives.asymmetric import rsa
3 private_key = rsa.generate_private_key(
4     public_exponent=65537,
5     key_size=2048,
6 )
7 public_key = private_key.public_key()
```

Listing 2: Sección de `genera_llaves.py`

La clave pública guardada es enviada a la instancia de EC2 con el siguiente comando

```
1 scp -i /path/to/your-key.pem /path/to/public_key.pem ec2-user@your-ec2-ip:/home/ec2-user
/
```

### 4.7.2. Envío de datos

El código realiza varias tareas para enviar datos de un archivo CSV a un servidor de manera segura. Toma los datos y la clave privada, define a donde va a enviar los datos. Tomando una porción de los datos crea un diccionario y un hash para verificar su integridad usando SHA256, firma los datos con la clave privada de RSA. Después con la firma y el hash se envían los datos con un POST al servidor, y después repite el proceso con las demás porciones de los datos.

```
1 data = {
2     'EntryID': row['EntryID'],
3     'ID': row['ID'],
4     'ConsumptionOrProduction': row['ConsumptionOrProduction'],
5     'Day': row['Day'],
6     'Month': row['Month'],
7     'Year': row['Year'],
8     'Time': row['Time'],
```

```

9         'Reading': row['Reading']
10     }
11     #HASH
12     data_string = json.dumps(data)
13     data_hash = hashlib.sha256(data_string.encode()).hexdigest()
14
15     #Firma
16     signature = private_key.sign(
17         data_string.encode(),
18         padding.PSS(
19             mgf=padding.MGF1(hashes.SHA256()),
20             salt_length=padding.PSS.MAX_LENGTH
21         ),
22         hashes.SHA256()
23     )
24
25     data['hash'] = data_hash
26     data['signature'] = signature.hex()
27
28     response = requests.post(url, data=json.dumps(data), headers={'Content-Type': '
    application/json'}, verify='C:\\Users\\gilhe\\claves\\apache-selfsigned.crt')

```

Listing 3: Sección de mandado.hash.firma.py

## 4.8. Automatización de conexión

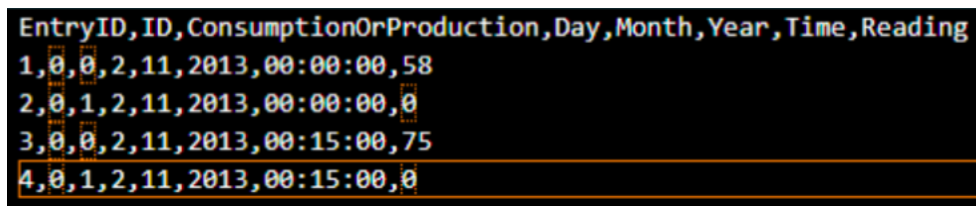
Se establece una conexión SSH con el servidor EC2 con la llave generada en la Raspberry. Con esto se ejecuta un comando en MySQL, el cual pide la contraseña, usuario y host de la RDS para poder llevar a cabo cualquier acción. Con esto se puede visualizar y filtrar los datos de la RDS de manera segura y automatizada.

```

1     stdin, stdout, stderr = ssh.exec_command("mysql -h dbcripto.ct1hupiz0cef.us-east-2.rds.
    amazonaws.com -u usuariomaestro -p'usuariomaestro' dbcripto -e 'SELECT * FROM
    AllMeterReadings WHERE EntryID < 5;'")

```

Listing 4: ejemplo de comando de MySQL en query.py



EntryID	ID	ConsumptionOrProduction	Day	Month	Year	Time	Reading
1	0	0	2	11	2013	00:00:00	58
2	0	1	2	11	2013	00:00:00	0
3	0	0	2	11	2013	00:15:00	75
4	0	1	2	11	2013	00:15:00	0

Figura 4: CSV resultante del query.py

## 5. Licencia

Para utilizar el servicio se requieren librerías de Python, las cuales todas son de uso público, haciendo que el servicio sea más barato.

- **secrets** Genera números pseudo-aleatorios criptográficamente fuertes, aptos para el manejo de información confidencial tal como autenticación, tokens y similares. Con una licencia PSF. [8]
- **hashlib 1.5** Es una librería para generar y utilizar hashes y sus funciones. Con una licencia Python Software Foundation License. [6]
- **csv** Librería para leer y escribir archivos csv. Es un módulo del python standard Library. [9]
- **csv** Librería para manejo de archivos csv Python. Es un módulo del python standard Library. [9]
- **pyOpenSSL 23.2.0** Librería con módulos para el uso de SSL, que permite conexiones seguras en redes computacionales. De esta librería se ocupan los módulos SSL y Crypto. Con una licencia Apache Software License (Apache License, Version 2.0).[1]
- **json** Es un módulo del Python Standard Library inspirado en JavaScript que permite intercambio de datos. [9]
- **flask 2.3.2** Librería para WSGI, de aplicaciones web. Con licencia BSD License (BSD-3-Clause). Se ocuparon los módulos Flask y request. [5]
- **pymysql 1.0.3** librería basada en PEP 249 para el uso de MySQL como cliente. Se utiliza el módulo cursors. Con licencia MIT License. [3]
- **requests 2.31.0** Es una librería para usar HTTP. Con una licencia Apache Software License (Apache 2.0). [4]
- **cryptography 40.0.2** Es una librería que tiene recetas y primitivas criptográficas para desarrolladores en python. Con una licencia Apache Software License, BSD License (Apache-2.0 OR BSD-3-Clause). De esta librería se utilizan los módulos de serialization, hashing, padding y rsa. [7]
- **time** Es un módulo del Python Standard Library con diferentes funciones relacionadas al tiempo. [9]
- **pandas 2.0.2** Es una librería para manejo de estructuras de datos. Con una licencia BSD License (BSD 3-Clause License Copyright (c) 2008-2011, AQR Capital Management, LLC, Lambda Foundry, Inc. and...).
- **paramiko 3.2.0** Librería para la implementación de SSHv2 para cliente y servidor. Con una licencia GNU Library or Lesser General Public License (LGPL) (LGPL). [2]



## 6. Contactos

Los autores y encargados del desarrollo del proyecto son:

- Andrea Bravo Avila a01028579@tec.mx
- Renata Vargas Caballero a01025281@tec.mx
- Natalia Monserrat González de León a00831090@tec.mx
- Fernando González Rosas a01253694@tec.mx
- Alberto Lozano Cárdenas a01067141@tec.mx
- Fernanda Sherlin Calderón López a01275430@tec.mx
- Gil Herzberg Alperon a00827347@tec.mx

## Referencias

- Caswell, M, et al. (2023). PyOpenSSL [Apache Software License (Apache License, Version 2.0)]. <https://pypi.org/project/pyOpenSSL/#data>
- Forcier, J. (2023). paramiko [GNU Library or Lesser General Public License (LGPL) (LGPL)]. <https://pypi.org/project/paramiko/>
- Naoki, I. (2023). PyMySQL [MIT License]. <https://pypi.org/project/pymysql/#data>
- Reitz, K. (2023). Requests [Apache Software License (Apache 2.0)]. <https://pypi.org/project/requests/#data>
- Ronacher, A. (2023). Flask [BSD License (BSD-3-Clause)]. <https://pypi.org/project/Flask/#data>
- Smith, G. P. (2009). Hashlib [Python Software Foundation License]. <https://pypi.org/project/hashlib/#data>
- The Python Cryptographic Authority and individual contributors. (2023). Cryptography [Apache Software License, BSD License (Apache-2.0 OR BSD-3-Clause)]. <https://pypi.org/project/cryptography/>
- Tuohela, I. (2012). Secrets [PSF]. <https://pypi.org/project/secrets/#data>
- Various Authors. (2023). Python Standard Library. <https://docs.python.org/3/library/>