Algoritmos de encriptación para IoT

Ciberseguridad (CS)

Autor: Antonio Gómez Giménez
Email: i72gogia@uco.es

Córdoba
(2022/2023)

UNIVERSIDAD D CÓRDOBA

Índice:

1. Introducción:

Hoy en día, en pleno siglo XXI, podemos apreciar la continua evolución de las tecnologías que nos rodean y el continuo desarrollo que estas llevan a pasos agigantados. Desde ordenadores cuánticos hasta pequeñas placas con cierta capacidad de cómputo.

Todos los tipos de tecnologías tienen su relevancia, pero el concepto de IoT(internet de las cosas) ha obtenido bastante relevancia debido a las mejoras que este ha introducido, siendo muy importante para ciertos conceptos como automatización de empresas, el uso casi necesario en big data, etc.

IoT no es un concepto definido como tal, pero se puede llegar a entender cómo la digitalización de cosas y su respectiva conexión a internet. Por ejemplo una serie de sensores que reciben información y la traspasan entre otros dispositivos o internet para tomar una serie de decisiones automatizadas y sin intervención humana, todo ello teniendo en cuenta que los datos se generan en tiempo real. IoT se aplica en distintos ámbitos como transporte, minería de datos, domótica, automatización, etc.

En este trabajo, nos centraremos en la seguridad aplicada al IoT, concretando más a la integridad de los datos. Como se ha descrito anteriormente, IoT está en continuo crecimiento y por tanto, es necesario también aumentar y mejorar la seguridad de la misma. Este campo es muy extenso, por ello nos centraremos en la integridad de los datos, evitando así la modificación de los mismos aplicando distintos tipos de algoritmos de encriptación.

Lógicamente, añadir esta seguridad implica aumentar la capacidad de cómputo en nuestros dispositivos, cosa que en muchos casos no es posible. Por ello, se motivó a hacer dicho trabajo, para implementar una api que se pueda ejecutar en un dispositivo permitiendo así centralizar el cómputo en un único dispositivo con mayor potencia que gestione la encriptación de los datos.

Para ello, se realizará sobre docker, para que no importe el lugar donde se deseé implementar. Aparte, se realizará con docker-compose, por si en un futuro se desea añadir otro o varios contenedores con otros servicios.

2. Implementación:

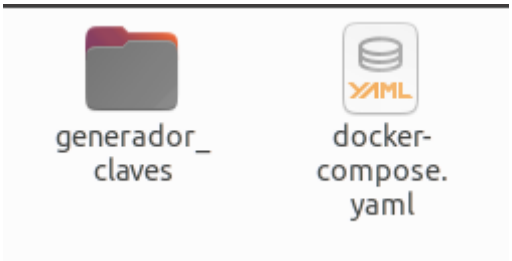
Para la realización de dicho propósito, se creó el siguiente docker compose, donde se encontrarán todos los servicios (incluyendo el docker creado con el servicio de encriptación). Se puede apreciar en la siguiente imagen:

```yaml
version: "3.3"


services:

  generador_claves:
      build: ./generador_claves/app
      image: generador_claves
      ports:
        - 8000:8000
      stdin_open: true
      tty: true
      volumes:
        - ./generador_claves/app:/usr/src/app

```

Esto permitirá, que con solo abrir una terminal donde se encuentre dicho fichero, al iniciar con el siguiente comando, empiece a funcionar el servicio:

-> sudo docker-compose up

El directorio sería el siguiente:



Cabe destacar, que para que funcione correctamente el docker-compose, se creó previamente el docker con el servicio de encriptación, y se especificó los requisitos necesarios, para que funcionara correctamente la api. En la siguiente imagen se puede apreciar el contenido del docker creado.

```
FROM python:3.8
COPY . usr/src/app
WORKDIR /usr/src/app

RUN pip3 install -r requirements.txt


ENTRYPOINT uvicorn --host 0.0.0.0 main:app --reload --port 8000
```

Una vez preparado el entorno, se creó el código que contiene la encriptación, este es un código sencillo, donde se aprovecha de los parámetros de entrada en la petición de la api para, a partir de la cadena con el dato a encriptar y el tipo de encriptación, genera dicha encriptación. Asumiendo así el cómputo de la encriptación. El código es el siguiente:

```python
#paquetes necesarios para el funcionamiento del programa
from fastapi import FastAPI
from typing import Optional
import hashlib


app = FastAPI()


@app.get("/")#cuando se realiza un get en la api no se realiza ninguna accion
def read_root():
    return {"No se realiza ninguna acción"}


@app.get("/encriptar/")#funcion para insertar datos en la bd
def read_root(tipo_de_cifrado: str, cadena_a_cifrar: str, hexadecimal: Optional[bool] = True):

    try:
        #print(cadena_a_cifrar)
        h = hashlib.new(tipo_de_cifrado)
        h.update(cadena_a_cifrar.encode())
        #print(h.digest(), h.hexdigest())
        if not hexadecimal:
            return {str(h.digest())}
        else:
            return {str(h.hexdigest())}
    except:
        return {"Ha ocurrido un error inesperado, revise el tipo de cifrado introducido (SHA1, SHA224, SHA256, SHA384, SHA512 o MD5)"}
```

Para mayor entendimiento, se explican los parámetros de entrada de la api:

- **tipo_de_cifrado**: es una cadena que especifica el tipo de algoritmo de cifrado que se pretende aplicar, pudiendo ser SHA1, SHA224, SHA256, SHA384, SHA512 o MD5.
- **cadena_a_cifrar**: es una cadena que contienen los datos que se pretenden cifrar.
- **hexadecimal**: es un parámetro opcional que por defecto es true, indica si se prefiere recibir los datos en formato hexadecimal o binario.

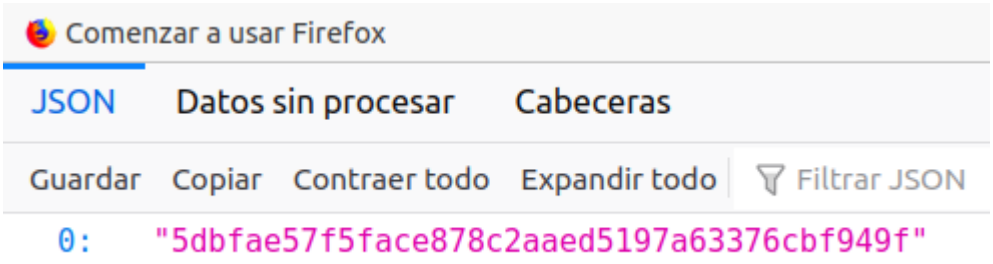
Dichos parámetros se aplican para la petición get encriptar, si no se especifica dicha petición, no se realizará ninguna acción.

3. Resultados:

Para probar el funcionamiento de la api, se realizaron distintas peticiones donde a la cadena "*esto es una prueba*" se le aplicaban distintos algoritmos de encriptación. Los ejemplos son los siguientes:

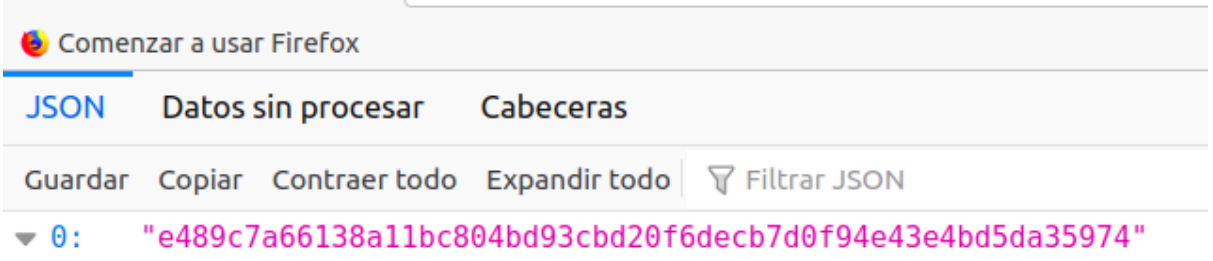
Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA1&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->



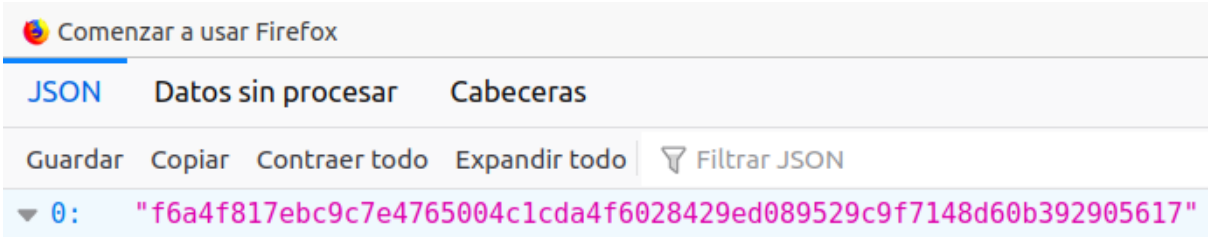
Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA224&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->



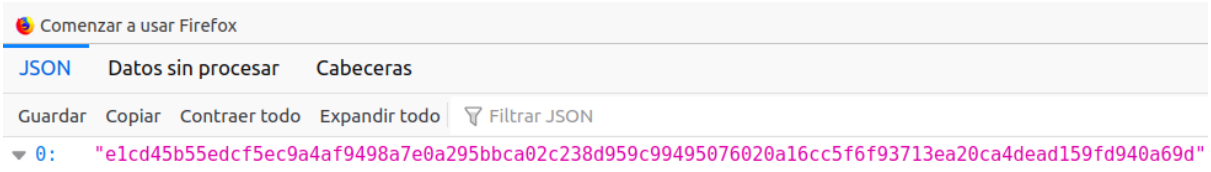
Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA256&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->



Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA384&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->

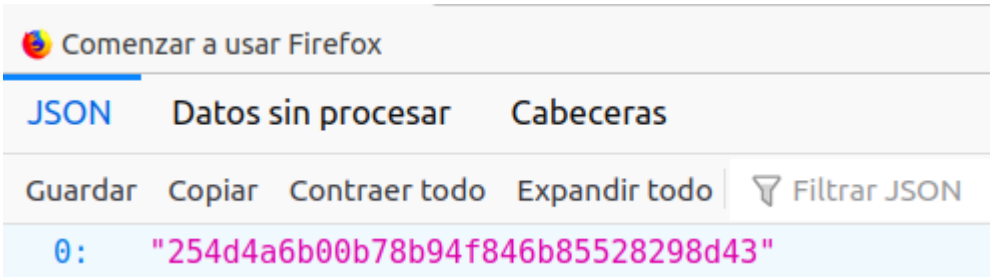


Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA512&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->

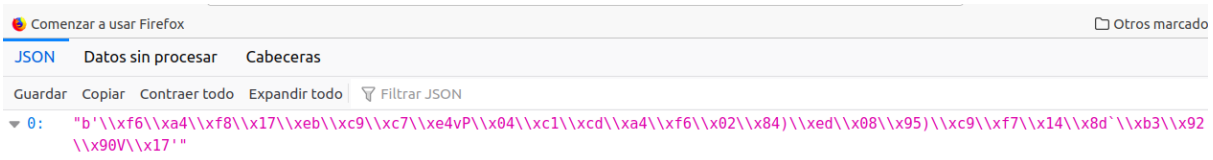
Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=MD5&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=true

Resultado->



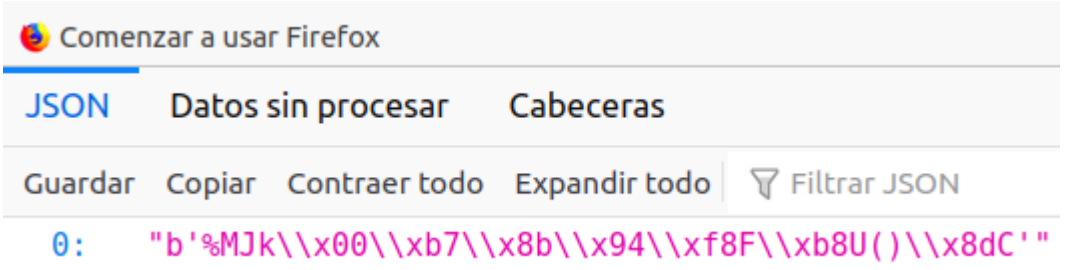
Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=SHA256&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=false

Resultado->



Petición->
http://localhost:8000/encriptar/?tipo_de_cifrado=MD5&cadena_a_cifrar=esto%20es%20una%20prueba&hexadecimal=false

Resultado->



4. Conclusiones:

Como podemos comprobar en este trabajo, se ha podido lograr una api que se encargue de la encriptación, delegando el cómputo a un dispositivo especializado para ello, evitando incrementar los costos de un proyecto, teniendo que realizar gastos en cada dispositivo para que cada uno de ellos tenga una capacidad de cómputo mejor. Además, gracias a docker, obviamos la estructura del dispositivo final, gracias a la encapsulación de docker.

Finalmente, por la propia estructura de la api, si se desea cambiar la encriptación de los dispositivos IoT, únicamente, sería necesario modificar la api, añadiendo una nuevo algoritmo de encriptación y especificando en los dispositivos la nueva llamada.