Proyecto de Encriptado - EAX

Br. Franco Gil

FaCyT – Departamento de Computación, Universidad de Carabobo
Naguanagua - Edo. Carabobo.

***Resumen*— Este artefacto hace énfasis en publicar una documentación acerca de cómo se encuentra estructurado el proyecto de encriptado de datos, que hace referencia al complemento práctico del curso de Redes del Computador II.**

***Palabras Claves*— Encriptado, EAX, algoritmos simétricos, MAC, python.**

I. Definición de la aplicación.

El proyecto es un aplicativo vía web que permite encriptar información dada una cadena de texto que es ingresada por el usuario. A lo largo del proceso de encriptado el usuario podrá ver que el texto es convertido en secuencias de caracteres irreconocibles que formarán parte del contenido encriptado.

La aplicación proveerá de una clave "privada" que es generada por el algoritmo simétrico de encriptación, éste será cedido a la aplicación y posteriormente al usuario, con ella se puede restaurar el contenido textual a como estaba antes de activar el servicio de encriptado.

Esta aplicación hará uso de un algoritmo de encriptado simétrico llamado EAX, este método de encriptado forma parte del conjunto de métodos modernos sobre cifrados de bloques simétricos.

Este algoritmo fue desarrollado dentro del NIST (The National Institute of Standards and Technology) en Octubre del 2003 por Bellare, Rogaway y Wagner. Está dentro de un conjunto de algoritmos que no solo proveen confidencialidad al cifrar y descifrar información, sino que además, provee integridad en los datos y autenticación. Es así como este algoritmo posee un modo de operación denominado AEAD (Authenticated Encryption with Associated Data) y lo que buscan no es solo llevar un texto entendible a uno sin sentido a través de cifrados sino que además permiten asegurar autenticidad en el texto o contenido que es devuelto.

A continuación se mostrará un pequeño diagrama de flujo que está asociado al modo de cifrado de bloques simétricos.

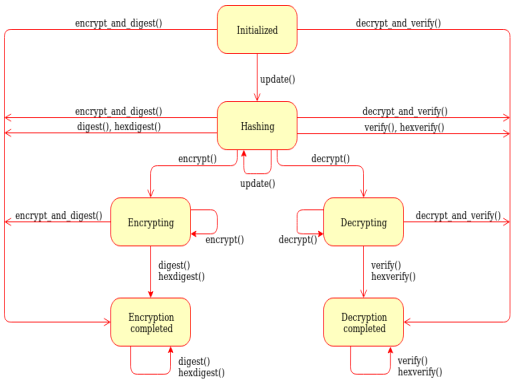

Fig. 1 Diagrama de estado genérico para cifrados dentro del modo AEAD.

Usando este método de encriptado se liberan elementos que son de carácter públicos y al menos uno de carácter privado, como la llave que permitirá establecer el contenido encriptado a su estado inicial.

Uno de los elementos públicos que emiten estos algoritmos (AEAD) se denominada MAC (message authentication code) o "*tag*", esto permite confirmar principalmente en el lado del destinatario si este contenido a desencriptar corresponde al contenido de origen, y que no ha sido modificado durante la entrega (algo bastante similar a la funciones hash). Luego podrá utilizarse la llave privada o el contenido privado para restaurar el contenido a su estado inicial.

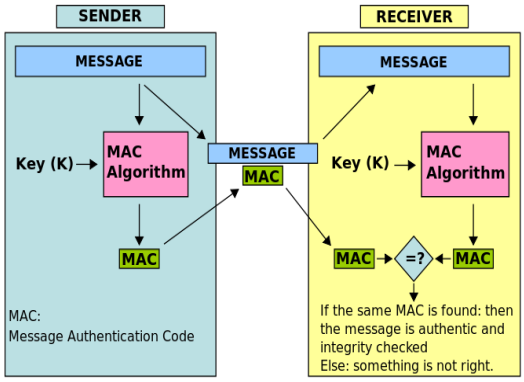

Fig. 2 Intercambio de parámetros entre emisor y receptor del contenido a encriptar.

II. Artefactos de UML.

Otros artefactos necesarios serán mostrados para ampliar la documentación, ambos están hechos en el lenguaje de Modelado UML.

1. *Diagrama de Casos de Uso.*

Este diagrama muestra un secuencia lineal de los estados que debe cumplir el producto de software para cifrar la información.

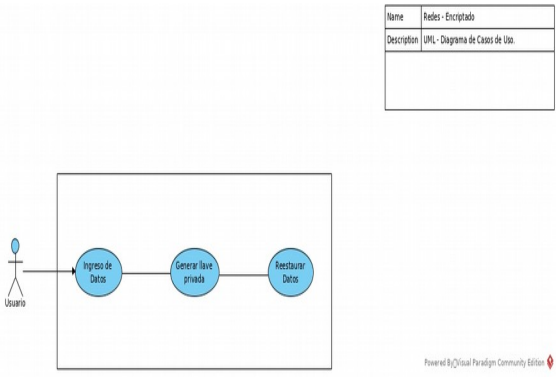

Fig. 3 Diagrama de casos de uso.

A. Definición de los Casos de Uso:

- ***Nombre de la Actividad**: Ingresar Datos.*
- **Definición**: Aquí el actor deberá introducir al menos una cadena de texto como recurso para iniciar el proceso de encriptación.

- **Nombre de la Actividad**: Generar llave privada.
- **Definición**: El modo de encriptado a utilizar es un modelo de encriptado simétrico, por lo cual se deberá generar una secuencia de caracteres irreconocibles y sin un patrón determinado, que formará parte del componente para establecer un conjunto de datos irreconocibles al recurso a inicial.

- **Nombre de la Actividad**: Restaurar Datos.
- **Definición**: Esta actividad se centrará en activar los módulos necesario para desencriptar la información necesaria, una vez que se le provean la llave privada como argumento principal y algunos otros parámetros secundarios de carácter público.

B. Definición de los Actores:

- **Usuario**: Suministra la cadena de texto y un parámetro de autenticidad, es el activador del flujo de encriptado y desencriptado.

2. *Diagrama de Actividades.*

Este diagrama simula a un diagrama de flujo, es una colección de arcos y vértices, con un punto de ejecución y otro de finalización. Cada columna con su título representa los diferentes módulos que componen el sistema, que para este caso en específico es el aplicativo de encriptación.

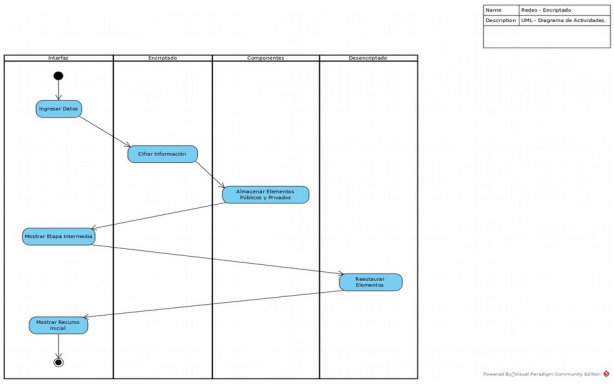

Fig. 4 Diagrama de actividades.

III. Arquitectura del proyecto.

Para llevar a cabo la etapa de desarrollo de esta aplicación se decidió usar el siguiente conjunto de tecnologías:

A. Sección de Frontend.

- HTML.

B. Sección de Backend:

- python.
- Falsk.

C. Librerías utilizadas:

- codecs
- base64
- Crypto
- flask_cors

Procederemos a definir solo la lista de librerías de terceros utilizadas sobre el lenguaje python:

- **base64**: Conjunto de módulos internos del lenguaje python. Se compone de una clase que permite una variante de codificación para caracteres ASCII imprimibles.

- **codecs**: Conjunto de módulos que forman parte del conjunto de librerías asociadas el núcleo de python, esta

viene por defecto tras la utilización de lenguaje. Compone una clase que hace el manejo de codificación y encodeo de los tipos de datos textuales o representaciones simbólicos, como cadenas, bytes, caracteres.

- **Crypto**: Es una librerías de terceros, es decir, no viene por defecto de los módulos necesarios para el lenguaje python. Está compuesta de una colección algoritmos de encriptado y de hashing, dentro de este conjunto está el algoritmo de encriptado EAX que hace uso esta aplicación.

- **flask_cors**: Es una librerías de terceros, es decir, no viene por defecto de los módulos necesarios para el lenguaje python. Conjunto de clases y decoradores que permiten el un intercambio de recursos entre clientes y servidores por medio de módulos AJAX. Útil como extensión del microframework flaks para el manejo de CORS (Cross Origin Resource Sharing).

IV. Explicación del Código Fuente.

Esta sección de código posee un decorador que no es más que incluir una función dentro de otra. Aquí está definido un endpoint de la aplicación que es activado cuando se genera una solicitud de tipo POST sobre la ruta '/encryption'.

La definición de la función llamada encrypt() podrá ser llamada cuando se active el endpoint, de esto se encarga el decorador.

Esta función debe recibir dos parámetros, ambos viene encapsulados dentro de los argumentos de la solicitud, llevarán por nombre, *header* y *data*. Estos parámetros deben ser convertidos a bytes para ser usados por los algorimos de encriptado.

En la línea nro. 39, el método encrypt_and_digest() usará la cadena de texto en bytes para proceder con su cifrado y posterior ocultamiento. Como se había nombrado anteriormente, el método de encriptado simétrico EAX es un tipo de algoritmo AEAD (Advanced Encryption Standard) y por lo tanto generará un *tag* o *MAC* (Message Authentication Code), que será público. De esa manera el destinatario no tendría problemas con determinar la autenticidad y veracidad del recurso a encriptar.

Luego de que todos los parámetros son encodeados sobre el alfabeto ASCII esos podrán ser enviados como respuesta desde el backend. De esto se encarga el método jsonify() de la clase Flask, dicha respuesta retorna un código de estado 200 indicando que esta etapa se ha cumplido de manera exitosa.

```
29  @encryption.route('/encryption', methods=['POST'])
30  def encrypt():
31
32      header = str.encode(request.args['header'])
33      data = str.encode(request.args['data'])
34
35      key = get_random_bytes(16)
36
37      cipher = AES.new(key, AES.MODE_EAX)
38      cipher.update(header)
39      ciphertext, tag = cipher.encrypt_and_digest(data)
40
41      json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
42      json_v = [ b64encode(x).decode('utf-8') for x in (cipher.nonce, h
43      result = dict(zip(json_k, json_v))
44
45      # Encode the key into unidocde to pass it through flask's request
46      q_key = str(key).encode('utf-8')[2:-1]
47
48      result['q_key'] = q_key
49
50      return jsonify({'result': result}), 200
51
```

Fig. 5 Sección del código fuente.

Esta sección es la última a mostrar sobre el código fuente (ver imagen 6) que soporta a la aplicación de encriptado.

Las primeras líneas de esta función, captan los argumentos necesarios para llevar a cabo la ejecución del descifrado.

Por un lado están los 4 primeros atributos:

- **ciphertext**: Texto cifrado e irreconocible. Es una cadena de texto proporcionada por el usuario.
- **header**: Parámetro opcional, útil para ofrecer mayor seguridad en el proceso de encriptado. Debe ser una cadena de texto proporcionada por el usuario.
- **nonce**: Parámetro opcional. Autentifica el mensaje y clave a desencriptar.
- **tag**: También llamado MAC (Message Authentication Code) permite verificar la integridad del mensaje encriptado.

Por último, la clave privada:

- **key**: Secuencia de bytes, clave que formará parte del proceso de descifrado, debe ser una secuencia única y que debe ser proporcionada por el usuario para descifrar la información. En este caso la secuencia de bytes es generada aleatoriamente (lín. 35 del cód. fuente), pero podría darse el caso de que el cliente genere una. Siempre y cuando tenga un peso de 16 bytes.

La clave privada viene encodeada en una cadena de texto, para poder ser enviada de manera exitosa por endpoint anterior (/encryption, POST), es por ello que en la lín. 61 del código fuente

es convertida a bytes, necesaria en ese tipo de dato para el proceso de descifrado.

Sobre la lín. 73 del código fuente se crea el objeto de la clase que cifrará los datos, necesita principalmente la clave privada, el modo de desencriptado, por que si recordamos la clase AES, posee un conjunto de algoritmos de encriptado y para seleccionar el algoritmo AEX debemos usar un número de bytes que lo caracterizan a través de la constante AES.MODE_EAX (16 bytes), luego de eso se añade otro parámetro llamado *nonce* que hace la función de autenticar el mensaje y clave a desencriptar.

En la lín. 75 la función decrypt_and_verify() usará el texto cifrado, es decir, el texto encriptado e irreconocible y el *tag* o *MAC* que debe ser idéntico al generado en el proceso encriptado encrypt_and_digest(), esto retorna una cadena de texto que debe ser idéntica a la ofrecida por el usuario durante el proceso de encriptado.

En caso de que algunos de los parámetros de autenticación como MAC o la clave privada sean incorrectos se alcanzará un error, que se será atajado y mostrado como error ante la respuesta del endpoint.

Si los parámetros son los correctos, entonces, el usuario podrá ver el texto descifrado y en su estado previo.

```
53  @encryption.route('/decryption', methods=['POST'])
54  def app_decryption():
55      # We assume that the key was securely shared beforehand
56      result = request.args.to_dict()
57
58      q_key = result.pop('q_key')
59
60      # Decocode the key into byte to pass it through the EAX cipher.
61      key = codecs.escape_decode(q_key)[0]
62
63      try:
64          # Static
65          json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
66
67          result = str(result).replace('\'', '"')
68
69          b64 = json.loads(result)
70          b64 = { k:b64[k].replace(' ', '+') for k in json_k }
71          jv = { k:b64decode(b64[k]) for k in json_k }
72
73          cipher = AES.new(key, AES.MODE_EAX, nonce=jv['nonce'])
74          cipher.update(jv['header'])
75          plaintext = cipher.decrypt_and_verify(jv['ciphertext'], jv['tag'])
76
77          output = plaintext.decode()
78
79      except (ValueError, KeyError) as err:
80          output = err
81
82      return jsonify({
83          'message': 'Decrypt Data',
84          'result': output
85      }), 200
86
```

Fig. 6 Sección #2 del código fuente.

V. Interfaz de usuario.

La aplicación muestra un conjunto de cuadros de texto. A partir de los cuadros de texto superiores, el usuario podrá intercambiar información entre la interfaz y el algoritmo de encriptación.

Posee tres botones:

- **Encrypt**: Con él, una vez rellenado los cuadros de texto, podrá enviar una solicitud al backend y por lo tanto, activar el endpoint para procesar el texto, generando los datos intermedios, como los son el texto cifrado, el atributo *nonce* y el atributo *tag* (MAC).
- **Put the Key**: Este botón permite copiar el contenido de la clave privada de un cuadro de texto a otro, para mayor comodidad al momento de enviar la clave privada durante el paso de desencriptado.
- **Reverse**: Con este botón se logrará, en caso de no alterar los atributos intermedios ni la clave privada, desencriptar el contenido y volver al estado original.

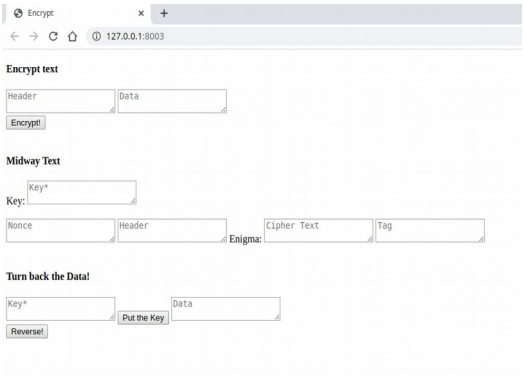

Fig. 7 Interfaz de la aplicación.

VI. Resultados obtenidos.

A continuación se añadirá una prueba generada por el algoritmo de encriptación EAX, con la secuencia de argumentos que ejemplifica la encriptación, resultados intermedios y desencriptación de una cadena de texto:

1) Cadena de Texto:

> Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500, cuando un impresor desconocido usó una galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen.

2) Resultados intermedios:

- **Key**: \xb5^\xca`"\xff\x0f\xb0\xed{Pia-Hu
- **Nonce**: 9Gg3cUINRExs0s4VQ1kqpw==
- **Cipher** Text: aW+EfFm …
- **Tag**: 12FAhYvWG6qc2G85zvsuZg==

Fig. 8 Resultados de la aplicación tras encriptar una cadena de texto.

REFERENCIAS

[1] The EAX mode of Operation (A Two-Pass Authenticated-Encryption Scheme Optimized for Simplicity and Efficiency), M. Bellare, P. Rogaway, D. Wagner, 18 de Enero del 2004, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/eax/eax-spec.pdf , accedido el 14 de Marzo del 2020.

[2] Crypto.Cipher packeage, https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html , accedido el 14 de Marzo del 2020.

[3] Message authentication code, Wikipedia - The Free Encyclopedia, https://en.wikipedia.org/wiki/Message_authentication_code , accedido el 14 de Marzo del 2020.

[4] Symmetric-key algorithm, Wikipedia - The Free Encyclopedia, https://en.wikipedia.org/wiki/Symmetric-key_algorithm , accedido el 14 de Marzo del 2020.

[5] AES - PyCryptodome, https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html, accedido el 14 de Marzo del 2020.

[6] NIST (The National Institute of Standards and Technology), https://www.nist.gov/about-nist, accedido el 14 de Marzo del 2020.