Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-1

Actualizar de 2023
Actividad

Actividad 4

Serialización y manejo de *bytes*

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/A4/`
- **Hora del *push*:** 16:40

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción: Protocolo Kame-iSEKAi

Hace unos meses se abrió un portal entre nuestro mundo y el reino de las Tortugas. Desde dicho evento, han llegado muchos mensajes de ese mundo, pero todos se encuentran encriptados y no se logra ver su contenido. Luego de mucho investigar, expertos y expertas en el área de criptografía lograron entender el protocolo de la encriptación, el cual decidieron llamar: "Protocolo Kame-iSEKAI". Con este descubrimiento, ahora te piden a ti crear un programa capaz de encriptar y desencriptar diferentes mensajes utilizando este protocolo. De este modo entender qué nos quieren decir las Tortugas de dicho mundo.

Archivos

En el directorio de la actividad encontrarás los siguientes archivos:

- Modificar `desencriptar.py`: Contiene las funciones necesarias para desencriptar un mensaje.
- Modificar `encriptar.py`: Contiene las funciones necesarias para encriptar un mensaje.
- No modificar `errors.py`: Contiene las clases de Errores personalizados a levantar durante el proceso de encriptación y desencriptación.
- No modificar `test_desencriptar.py`: Contiene el código necesario para ejecutar los *tests* relacionados con la desencriptación de mensajes.
- No modificar `test_encriptar.py`: Contiene el código necesario para ejecutar los *tests* relacionados con la encriptación de mensajes.

Estructura del programa

Esta actividad consta de dos partes, en las cuales se te pedirá que implementes funciones que permitan encriptar y desencriptar los mensajes siguiendo un protocolo específico.

Protocolo Kame-iSEKAI

Encriptación

El protocolo de encriptación KAme-iSEKAI se basa en 2 elementos: (1) el mensaje a encriptar y (2) una secuencia de números. Luego, este protocolo presenta 3 pasos: separar el mensaje original en 2 nuevos mensajes, codificar la secuencia de números en un *bytearray*, y finalmente concatenar los elementos para construir el mensaje encriptado.

Paso 1: Separar el mensaje

El primer paso de este protocolo es extraer los *bytes* indicados en la secuencia, para esto, se deben asegurar ciertas condiciones:

- El número más grande de la secuencia debe ser menor al largo del mensaje a encriptar.
- No pueden existir números repetidos en la secuencia, es decir, todos son únicos.

Si la secuencia cumple con ambas condiciones, se deben formar 2 *bytearray*: uno con el mensaje original, pero sin los *bytes* indicados en la secuencia de números; y otro con solo los *bytes* indicados en la secuencia de números, **respetando el mismo orden que están en la secuencia**, es decir, si la secuencia es `[4, 0]`, este segundo *bytearray* estará formado por el *byte* de la posición 4, seguido del *byte* de la posición 0.

Por ejemplo, llamemos a `mensaje` la secuencia de *bytes* que se desea encriptar y `ARRAY` a la lista con la secuencia de números. El resultado de este paso sería la creación de `m_reducido` y `m_bytes_secuencia`, las dos nuevas secuencias de *bytes* generadas:

$$\texttt{mensaje} = b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_9\ b_{10}$$
$$\texttt{ARRAY} = [\ 0,\ 10,\ 2\ ]$$
$$\texttt{m\_reducido} = b_1\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_9$$
$$\texttt{m\_bytes\_secuencia} = b_0\ b_{10}\ b_2$$

Paso 2: Codificar secuencia de números

El segundo paso consiste en codificar la secuencia de número en un *bytearray*. Para esto, se debe transformar cada número de la secuencia en un mensaje de 2 *bytes* con formato *big endian*.

Por ejemplo, llamemos `ARRAY` a la lista con la secuencia de números. El resultado de este paso sería la creación de `secuencia_codificada`:

$$\texttt{ARRAY} = [\ 0,\ 10,\ 2\ ]$$
$$\texttt{secuencia\_codificada} = \texttt{b'\textbackslash x00\textbackslash x00\textbackslash x00\textbackslash x0A\textbackslash x00\textbackslash x02'}$$

Paso 3: Concatenar mensaje final

El último paso consiste en concatenar todos los *bytearray* generados para tener un único mensaje encriptado. La estructura del mensaje encriptado es:

1. El primer fragmento del mensaje corresponde al largo de la secuencia de números. Para esto, se debe transformar este número en un *bytearray* compuesto por 4 *bytes* en formato *big endian*. Usando el ejemplo mostrado en los pasos anteriores con `ARRAY`, este fragmento corresponde a transformar el número 3 en un mensaje de 4 *bytes*: `b'\x00\x00\x00\x03'`
2. El segundo fragmento corresponde al *bytearray* con los *bytes* extraídos utilizando la secuencia de números. Usando el ejemplo mostrado en los pasos anteriores, el segundo fragmento corresponde al `m_bytes_secuencia`.
3. El tercer fragmento corresponde al *bytearray* con el mensaje original sin los *bytes* extraídos. Usando el ejemplo mostrado en los pasos anteriores, el tercer fragmento corresponde al `m_reducido`.
4. El último fragmento corresponde al *bytearray* con la secuencia de número codificados siguiendo las instrucciones del paso 2. Usando el ejemplo mostrado en los pasos anteriores, el último fragmento corresponde a la `secuencia_codificada`.

A modo de resumen, el formato del mensaje a encriptar es:

`largo_secuencia` + `m_bytes_secuencia` + `m_reducido` + `secuencia_codificada`

Ejemplo de encriptación

Para ayudarte a entender de mejor forma el método de encriptación que debes implementar, utilizaremos como ejemplo el siguiente mensaje de 13 *bytes* y secuencia:

`b'\x05\x08\x03\x02\x04\x03\x05\x09\x05\x09\x01\x0A\xFF'`

$$[\,11,\ 0,\ 2,\ 4\,]$$

1. El primer paso es separar el mensaje utilizando la secuencia. En este caso, `m_reducido` y `m_bytes_secuencia` quedarán del siguiente modo:

`m_reducido` = `b'\x08\x02\x03\x05\x09\x05\x09\x01\xFF'`

`m_bytes_secuencia` = `b'\x0A\x05\x03\x04'`

2. El segundo paso es transformar la secuencia de números en un mensaje de *bytes*. En este caso, `secuencia_codificada` quedará del siguiente modo:

`secuencia_codificada` = `b'\x00\x0B\x00\x00\x00\x02\x00\x04'`

3. El último paso consiste en concatenar las secuencias de *bytes* generadas y agregar, al inicio del mensaje, el largo de la secuencia de números (4 en este caso). El mensaje encriptado final será:

`b'\x00\x00\x00\x04\'` + `m_bytes_secuencia` + `m_reducido` + `secuencia_codificada`

`b'\x00\x00\x00\x04\'` + `b'\x0A\x05\x03\x04'` +
`b'\x08\x02\x03\x05\x09\x05\x09\x01\xFF'` + `b'\x00\x0B\x00\x00\x00\x02\x00\x04'`

Desencriptación

En el caso de la desencriptación, se aplicarán los mismos cambios de la encriptación, pero de forma invertida para obtener el mensaje original.

Ejemplo de desencriptación

Finalmente, para ayudarte a entender el proceso de desencriptación, utilizaremos el siguiente mensaje

```
b'\x00\x00\x00\x04\x0A\x05\x03\x04
\x08\x02\x03\x05\x09\x05\x09\x01\xFF
\x00\x0B\x00\x00\x00\x02\x00\x04'
```

1. El primer paso es separar el mensaje en los 4 fragmentos. Para esto, necesitamos obtener el largo de la secuencia de números. Este valor siempre estará contenido en los 4 primeros *bytes* del mensaje, en este caso: `b'\x00\x00\x00\x04'`. Luego, se transforman esos *bytes* en el `int` correspondiente: 4.

   Ya conocido el valor del largo de la secuencia, se puede extraer la `secuencia_codificada`, el `m_reducido` y `m_bytes_secuencia`. En este caso:

   - Dado que el largo es 4, `m_bytes_secuencia` serán los 4 *bytes* posteriores a los *bytes* asociadas al largo. Es decir, los *bytes* 4, 5, 6 y 7. [1]
   - Luego, cómo el largo de la secuencia codificada es 4 y cada número de dicha secuencia se transformó en 2 *bytes*, la `secuencia_codificada` tendrá un largo de 8 *bytes*, los cuales estarán posicionados al final del mensaje.
   - Finalmente, el resto del mensaje corresponda el mensaje original sin los *bytes* extraídos por la secuencia de números, es decir, `m_reducido`.

   ```
   m_bytes_secuencia = b'\x0A\x05\x03\x04'
   secuencia_codificada = b'\x00\x0B\x00\x00\x00\x02\x00\x04'
   m_reducido = b'\x08\x02\x03\x05\x09\x05\x09\x01\xFF'
   ```

2. El segundo paso es decodificar `secuencia_codificada`. En otras palabras, transformar los fragmentos de 2 *bytes* en el número que corresponde.

   ```
   secuencia_codificada = b'\x00\x0B \x00\x00 \x00\x02 \x00\x04'
   secuencia_decodificada = [ 11, 0, 2, 4 ]
   ```

3. El último paso consiste en unificar `m_bytes_secuencia` y `m_reducido` asegurando que los *bytes* de `m_bytes_secuencia` estén en las posiciones indicadas por `secuencia_decodificada`, es decir, el primer *byte* de `m_bytes_secuencia` estará en la posición 11 del mensaje desencriptado, el segundo *byte* de `m_bytes_secuencia` estaré en la posición 0 del mensaje desencriptado y así sucesivamente. El resultado final sería:

   ```
   b'\x05\x08\x03\x02\x04\x03\x05\x09\x05\x09\x01\x0A\xFF'
   ```

[1] Recordar que un *bytearray* es al final una lista cuyo primer valor está en la posición 0. Así que los *bytes* asociadas al largo estarían en los *bytes* 0, 1, 2 y 3.

Parte 1 - Encriptación

En esta parte, debes completar todas las funciones necesarias para poder encriptar un mensaje correctamente. Para esto se entrega el archivo `encriptar.py` con las siguientes funciones a completar.

- Modificar `def serializar_diccionario(dictionary: dict) -> bytearray:`
  Esta función transforma el diccionario en un `string`, luego lo codifica con UTF-8 y retorna un *bytearray* de este. En caso que ocurra algún error del tipo `TypeError` durante todo este proceso, se deberá atrapar dicha excepción y levantar otra excepción del tipo `JsonError`.
- Modificar `def verificar_secuencia(mensaje: bytearray, secuencia: List[int]) -> None:`
  Esta función verifica que el mensaje y la secuencia cumplan con las 2 condiciones indicadas en el "Paso 1: Separar el mensaje" del protocolo. En caso que no se cumpla alguna de estas condiciones, se debe levantar una excepción del tipo `SequenceError`. En otro caso, de no levantar ninguna excepción, esta función retorna `None`.
- Modificar `def codificar_secuencia(secuencia: List[int]) -> bytearray:`
  Esta función se encarga de realizar el "Paso 2: Codificar secuencia de números" del protocolo, es decir, transformar la secuencia de números en un *bytearray* donde cada número corresponderá a 2 *bytes* con formato *big endian*. Esta función debe retornar el *bytearray* resultante de esta transformación.
- Modificar `def codificar_largo(largo: int) -> bytearray:`
  Esta función se encarga de realizar la primera parte del "Paso 3: Concatenar mensaje final" del protocolo, es decir, transformar un número en un *bytearray* compuesto por 4 *bytes* con formato *big endian*. Debe retornar el *bytearray* con el número transformado.
- Modificar `def separar_msg(mensaje: bytearray, secuencia: List[int]) -> List[bytearray]:`
  Esta función se encarga de aplicar el "Paso 1: Separar el mensaje" del protocolo, es decir, segmentar el `mensaje` en 2 *bytearray*.
  - El primer *bytearray* corresponde a `m_reducido`, es decir, al mensaje original sin los *bytes* según las posiciones indicadas en la secuencia de números.
  - El segundo *bytearray* corresponde a `m_bytes_secuencia`, es decir, los *bytes* correspondiente a las posiciones indicadas en la secuencia de número.

  Esta función retorna una lista en donde el primer elemento corresponde a `m_reducido` y el segundo elemento a `m_bytes_secuencia`.
- No modificar `def encriptar(mensaje: dict, secuencia: List[int]) -> bytearray:` Esta función se encarga de encriptar un diccionario utilizando las funciones definidas previamente.

Parte 2 - Desencriptación

En esta parte, debes completar todas las funciones necesarias para poder desencriptar un mensaje correctamente. Para esto se entrega el archivo `desencriptar.py` con las siguientes funciones a completar.

- Modificar `def deserializar_diccionario(mensaje_codificado: bytearray) -> dict:`
  Esta función transforma un `string` en el diccionario que corresponde. Para esto, primero decodifica el `mensaje_codificado` con UTF-8 y luego lo transforma en un diccionario. En caso que durante este proceso ocurra algún error del tipo `JSONDecodeError`, se deberá atrapar dicha excepción y levantar otra excepción del tipo `JsonError`. En otro caso, esta función debe retornar el diccionario obtenido de la deserialización.

- **Modificar** `def decodificar_largo(mensaje: bytearray) -> int:`
  Esta función se encarga de obtener los primeros 4 *bytes* del mensaje y lo convierte en el número correspondiente aplicando una transformación del tipo *big endian*. Finalmente, retorna el número obtenido de esta transformación.

- **Modificar** `def separar_msg_encriptado(mensaje: bytearray) -> List[bytearray]:`
  Esta función utiliza a `decodificar_largo` para obtener el largo de la secuencia de números. Luego, con dicho valor se encarga de separar el mensaje en las 3 fragmentos restantes: `secuencia_codificada`, `m_bytes_secuencia` y `m_reducido`. Finalmente retorna una lista donde el primer elemento corresponde al `m_bytes_secuencia`, el segundo elemento corresponde a `m_reducido` y el último elemento corresponde a la `secuencia_codificada`.

- **Modificar** `def decodificar_secuencia(secuencia_codificada: bytearray) -> List[int]:`
  Se encarga de transformar la `secuencia_codificada` en la lista de números. Para esto, se deben tomar de a 2 *bytes* y convertirlo en el número correspondiente aplicando una transformación del tipo *big endian*. Debe retornar la lista de los números transformados.

- **Modificar** `def desencriptar(mensaje: bytearray) -> bytearray:`
  Esta función se encarga de utilizar las funciones definidas anteriormente (`decodificar_largo`, `separar_msm_encriptado` y `decodificar_secuencia`) para obtener el mensaje original a partir del mensaje encriptado. Debe retornar un *bytearray* con el mensaje original.