Instituto Tecnológico y de Estudios

Superiores de Occidente.

**Firmar archivo enviado y cifrado por TCP**

|  |  |
| --- | --- |
| Materia | Seguridad en redes |
| Profesor | Luis Julián Domínguez Pérez |
| Fecha | 07/05/2022 |
| Autor(es) | Rodolfo Luthe Narváez |

# Explicación del funcionamiento

## Cliente

|  |
| --- |
| import requests  files = {'file': open('file.txt', 'rb')}  r = requests.post('http://localhost:8080',files = files)  print(r.text) |

Requests es una librería muy utilizada para enviar cosas por http y de esta manera interactuar con recursos web. Se crea un json con el archivo y se manda por post al servidor corriendo en el puerto 8080 de localhost. Al final se imprime la respuesta del servidor. Esto no cambia de la tarea pasada.

## Servidor

|  |
| --- |
| from http.server import BaseHTTPRequestHandler, HTTPServer  import logging  import nacl.secret  import nacl.utils  import nacl.pwhash  from nacl.signing import SigningKey  class S(BaseHTTPRequestHandler):      def \_set\_response(self):          self.send\_response(200)          self.send\_header('Content-type', 'text/html')          self.end\_headers()      def do\_GET(self):          logging.info("GET request,\nPath: %s\nHeaders:\n%s\n", str(self.path), str(self.headers))          self.\_set\_response()          self.wfile.write("GET request for {}".format(self.path).encode('utf-8'))      def do\_POST(self):          content\_length = int(self.headers['Content-Length']) # <--- Gets the size of data          post\_data = self.rfile.read(content\_length) # <--- Gets the data itself          key = nacl.utils.random(nacl.secret.SecretBox.KEY\_SIZE)          box = nacl.secret.SecretBox(key)          encrypted = box.encrypt(bytes(post\_data.decode('utf-8'), 'utf-8'))          f = open("encrypted.txt", "w")          signing\_key = SigningKey.generate()          signed = signing\_key.sign(encrypted.ciphertext)          # f.write(bytes(signed, 'utf-8'))          f.write(str(signed))          f.close()            verify\_key = signing\_key.verify\_key          verify\_key\_bytes = verify\_key.encode()          self.\_set\_response()          self.wfile.write(verify\_key\_bytes)          logging.info("POST request,\nPath: %s\nHeaders:\n%s\n\nBody:\n%s\n\nEncrypted Body:\n%s\n\Signed Body:\n%s\nVerify Key:\n%s\n",              str(self.path), str(self.headers), post\_data.decode('utf-8'), encrypted.ciphertext, signed, verify\_key\_bytes)  def run(server\_class=HTTPServer, handler\_class=S, port=8080):      logging.basicConfig(level=logging.INFO)      server\_address = ('', port)      httpd = server\_class(server\_address, handler\_class)      logging.info('Starting httpd...\n')      try:          httpd.serve\_forever()      except KeyboardInterrupt:          pass      httpd.server\_close()      logging.info('Stopping httpd...\n')  if \_\_name\_\_ == '\_\_main\_\_':      from sys import argv      if len(argv) == 2:          run(port=int(argv[1]))      else:          run() |

El servidor es el que viene incluido en serpy pero desambiguado, puede recibir GET y POST. No le hace nada a las peticiones y solo contesta que tipo de petición fue y a que ruta. Lo que hace es hacer registro de cada petición con la librería de logging y esto lo usamos para evidenciar que el cliente envió el archivo. Es el mismo que la tarea pasada.

El server usa http como su método de recibir peticiones, por default usa el puerto de http, que es el 8080, pero se puede especificar uno distinto si es necesario.

La encripción se hace con PyNaCl, primero creamos una llave y la usamos para obtener una “caja” que luego nos permite encriptar el archivo que nos envió el cliente. Al último enviamos el mensaje encriptado de regreso. Esto es igual a la tarea pasada.

Finalmente usamos la librería de firmado encontrada en PyNaCl para firmar el contenido de un archivo antes de escribirlo y generar una llave verificadora que le corresponde al archivo. Esta llave verificadora se le regresa al cliente.

# Código a repositorio

<https://github.com/Rluthen/serverEncrypt>

# Bibliografía

|  |  |
| --- | --- |
| [1] | O. Smith, «Python Http Request Tutorial,» DataCamp, 19 09 2019. [En línea]. Available: https://www.datacamp.com/community/tutorials/making-http-requests-in-python. [Último acceso: 15 04 2022]. |
| [2] | mdonkers, «Simple Python 3 HTTP server for logging all GET and POST requests,» GitHub, 17 12 2019. [En línea]. Available: https://gist.github.com/mdonkers/63e115cc0c79b4f6b8b3a6b797e485c7. [Último acceso: 15 04 2022]. |
| [3] | «http.server — HTTP servers,» Python, [En línea]. Available: https://docs.python.org/3/library/http.server.html#module-http.server. [Último acceso: 15 04 2022]. |
| [4] | scichris, «How to upload file with python requests?,» Stack Overflow, 21 03 2014. [En línea]. Available: https://stackoverflow.com/questions/22567306/how-to-upload-file-with-python-requests. [Último acceso: 20 04 2022]. |
| [5] | «Secret Key Encryption- PyNaCl 1.6.0.dev1 documentation,» PyNaCl , [En línea]. Available: https://pynacl.readthedocs.io/en/latest/secret/. [Último acceso: 25 04 2022]. |