Ciências ULisboa | Informática

Transport Layer Security em Python

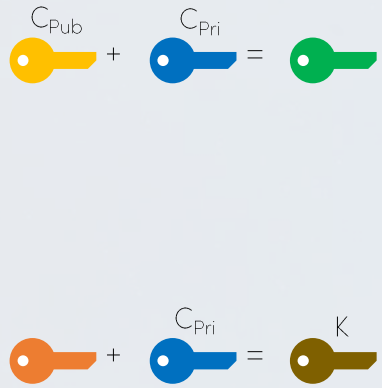
Pedro Ferreira, Vinicius Cogo

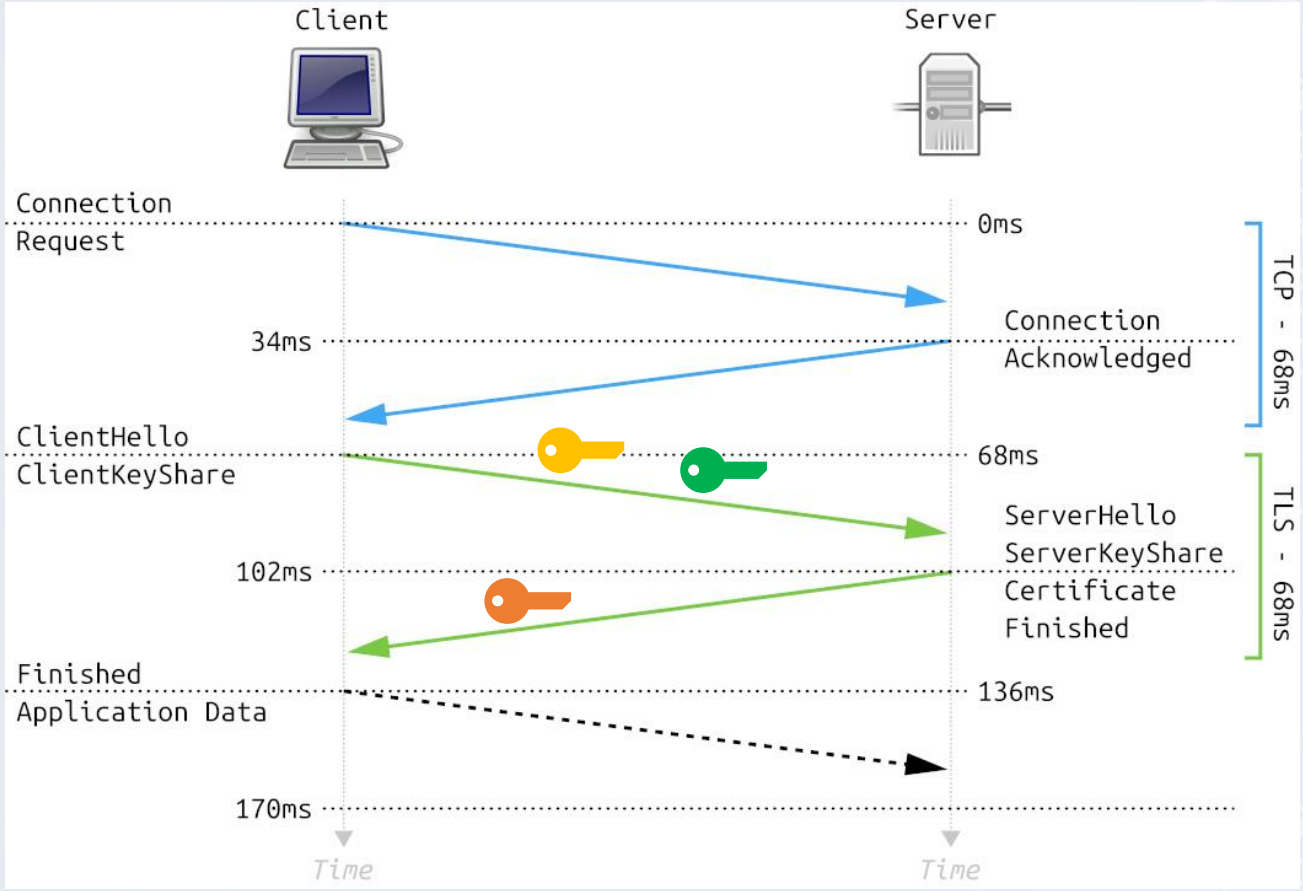
Tópicos

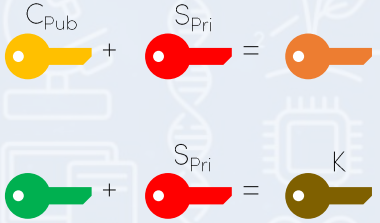
Ciências ULisboa

1. Protocolo TLS
2. TLS em Python
3. Criação de Chaves e Certificados
4. Autenticação mútua TLS com Flask
5. Criação de chaves e certificados para o navegador

Protocolo TLS







Criação de chaves e certificados

- Criar uma chave RSA com 2048 bits

```
$ openssl genrsa -out name.key 2048
```

- Criar um certificado auto-assinado:

```
$ openssl req -x509 -new -nodes -key ca.key -sha256 -days 365 -out ca.pem
```

- Emitir pedido de assinatura de certificado

```
$ openssl req -new -nodes -key name.key -sha256 -days 365 -out name.csr
```

- Gerar um certificado assinado pelo CA (a partir do pedido do passo anterior)

```
$ openssl x509 -req -in name.csr -CA ca.pem -CAkey ca.key \
-CAcreateserial -out name.crt -days 365 -sha256
```

TLS em Python

- Disponibilizado no **módulo `ssl`**
  - Usa a biblioteca OpenSSL
  - Existe em SOs com OpenSSL disponível

- Modo de operação
  - A **`socket`** é **criada** da forma habitual
  - É criado e **configurado** um **`SSLContext`**
  - É criado um objeto **`SSLSocket`** a partir da **`socket`** e do **`SSLContext`**
    Este objeto é um *wrapper* da `socket` que cifra e decifra a informação enviada/recebida pela `socket`

TLS em Python

- A função `wrap_socket()`
  - Recebe uma socket e um `SSLContext` com os parâmetros relacionados à autenticação, a cifra, e o protocolo

```
import ssl
...
# código que cria uma socket (sock)e faz a ligação
...
context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
...
# código com a configuração do SSLContext (próximos slides)
...
sslsock = context.wrap_socket(sock, server_hostname=HOST)
```

TLS em Python

- A função `wrap_socket()`
  - Recebe uma socket e parâmetros relacionados com a autenticação, a cifra, e o protocolo

Cliente não se identifica, mas pede a identificação do servidor

```
import ssl
...
# código que cria uma socket (sock) e faz a ligação
...
context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
context.verify_mode = ssl.CERT_REQUIRED
context.check_hostname = True
context.load_verify_locations(cafile='ca.pem')

sslsock = context.wrap_socket(sock, server_hostname=HOST)
```

TLS em Python

- A função `wrap_socket()`
  - Recebe uma socket e parâmetros relacionados com a autenticação, a cifra, e o protocolo

Servidor identifica-se, mas não pede a identificação do cliente

```
import ssl
...
# código que cria uma socket (sock) e faz a ligação
...
context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_SERVER)
context.verify_mode = ssl.CERT_NONE
context.load_cert_chain(certfile='serv.crt', keyfile='serv.key')

sslsock = context.wrap_socket(conn_sock, server_side=True)
```

Autenticação mútua TLS com Flask

```
from flask import Flask
import ssl
app = Flask(__name__)
@app.route('/ola', methods = ["GET"])
def login():
    return 'Olá Mundo'
if __name__ == '__main__':
    context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_SERVER)
    context.verify_mode = ssl.CERT_REQUIRED
    context.load_verify_locations(cafile='ca.pem')
    context.load_cert_chain(certfile='server.crt',keyfile='server.key')
    app.run('localhost', ssl_context=context, debug = True)
```

Autenticação mútua TLS com Flask

```
import requests

r=requests.get(
    'https://localhost:5000/ola',
    verify='ca.pem',
    cert=('client.crt','client.key'))

print (r.status_code)
print (r.content.decode())
print (r.headers)
print ('***')
```

Criação de chaves e certificados para o navegador

- Criar um ficheiro PKCS12 (um "tar" para chaves e certificados):

```
$ openssl pkcs12 -export -inkey cli.key -in cli.crt -out cli.p12
```

- Este **`cli.p12`** pode ser utilizado no navegador
  para permitir o cliente fazer aceder ao recurso /ola através dele

Bibliografia

- Brandon Rhodes and John Goerzen. Foundations of Python Network Programing, second edition, Apress.

- Python online documentation:ssl —TLS/SSL wrapper for socket objects. (https://docs.python.org/3/library/ssl.html)