

Get the slides!

# Implementing HTTPS In Python

Thierry Sans

## 5 servers - **Only one can be trusted**

*"There are **5 servers but only one can be trusted**. Your mission, if you choose to accept it, is to **implement a proper TLS client to find the legitimate server** and uncover the 4 imposters"*

Your HTTPS client must:

- Exchange a symmetric key
- Verify the server's identify (signature + certificate)
- Use the symmetric key to decrypt the data  
if and only if the server is trusted

# Step by Step Process

**Step 0:** Setting-up the work environment

**Step 1:** Key Exchange and Data Decryption

**Step 2:** Certificate and Server's Signature Verifications

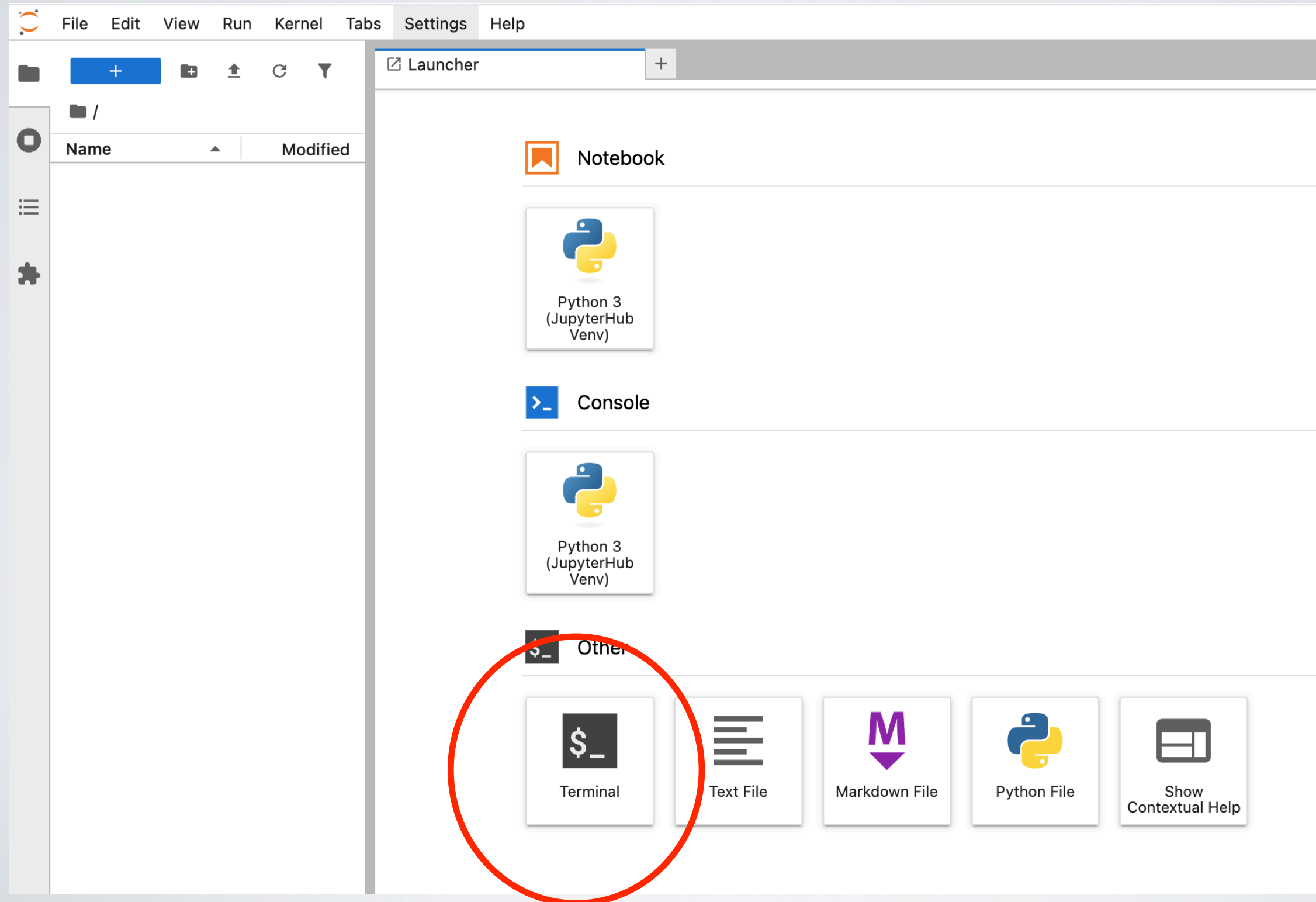
**Final Step:** Find the trusted server among the imposters

Step 0

**Setup the Work Environment**



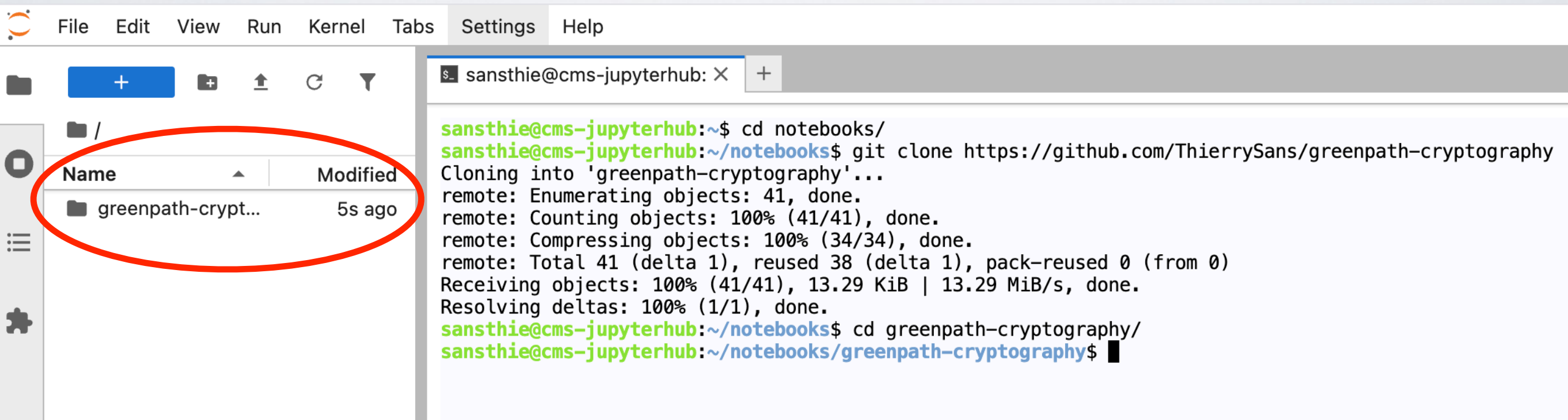
# Starting the Console



# Fetching the Code

In the console, type these three commands:

1. `cd notebooks`
2. `git clone https://github.com/ThierrySans/greenpath-cryptography`
3. `cd greenpath-cryptography`



The screenshot shows a JupyterLab interface. On the left is a file browser with a table of files. The file 'greenpath-crypt...' is highlighted with a red circle. On the right is a terminal window showing the execution of the commands listed in the previous block.

Name	Modified
greenpath-crypt...	5s ago

```
sansthe@cms-jupyterhub: X +
sansthe@cms-jupyterhub:~$ cd notebooks/
sansthe@cms-jupyterhub:~/notebooks$ git clone https://github.com/ThierrySans/greenpath-cryptography
Cloning into 'greenpath-cryptography'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 41 (delta 1), reused 38 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (41/41), 13.29 KiB | 13.29 MiB/s, done.
Resolving deltas: 100% (1/1), done.
sansthe@cms-jupyterhub:~/notebooks$ cd greenpath-cryptography/
sansthe@cms-jupyterhub:~/notebooks/greenpath-cryptography$
```

# Working with the Code

The screenshot displays the JupyterLab interface. On the left is a file explorer showing the directory structure of a project named 'greenpath-cryptography'. The files listed are 'config', 'cryptobox', 'client.py' (highlighted), 'README.md', 'requirements.txt', 'run.py', and 'server.py'. All files were modified '3m ago'. The main area contains a code editor for 'client.py' with the following Python code:

```
1 from cryptobox import *
2
3 class Client():
4     def __init__(self, url: bytes, trusted_certs: list[bytes]):
5         self.url = url
6         self.trusted_certs = trusted_certs
7
8     def send_client_hello(self) -> bytes:
9         print(self.url)
10        print(self.trusted_certs)
11
12    def send_client_ready(self, server_hello: bytes) -> bytes:
13        None
14
15    def receive_data(self, data: bytes) -> bytes:
16        None
17
```

At the bottom is a terminal window with the following output:

```
sansthie@cms-jupyterhub: ~$ cd notebooks/
sansthie@cms-jupyterhub: ~/notebooks$ git clone https://github.com/ThierrySans/greenpath-cryptography
Cloning into 'greenpath-cryptography'...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 41 (delta 1), reused 38 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (41/41), 13.29 KiB | 13.29 MiB/s, done.
Resolving deltas: 100% (1/1), done.
sansthie@cms-jupyterhub: ~/notebooks$ cd greenpath-cryptography/
sansthie@cms-jupyterhub: ~/notebooks/greenpath-cryptography$
```



# Running the Code

The server id from  
**server1 to server5**

In the console, run the command:

```
python3 run.py greenpath.program server2
```

The domain name from of the url  
you would type in the browser

```
sansthie@cms-jupyterhub: X +
sansthie@cms-jupyterhub:~/notebooks/greenpath-cryptography$ python3 run.py greenpath.program server2
Traceback (most recent call last):
  File "/home/sansthie/notebooks/greenpath-cryptography/run.py", line 69, in <module>
    run(url, trusted_cert_dir, server_dir)
  File "/home/sansthie/notebooks/greenpath-cryptography/run.py", line 23, in run
    server_hello = server.send_server_hello(client_hello)
                   ~~~~~
  File "/home/sansthie/notebooks/greenpath-cryptography/server.py", line 12, in send_server_hello
    self.shared_key = key_exchange(priv, peer_pub)
                     ~~~~~
  File "/home/sansthie/notebooks/greenpath-cryptography/cryptobox/cryptobox.py", line 70, in key_exchange
    peer_public_key = serialization.load_der_public_key(peer_public_key)
                     ~~~~~
TypeError: argument 'data': Cannot convert "<class 'NoneType'>" instance to a buffer.
sansthie@cms-jupyterhub:~/notebooks/greenpath-cryptography$
```

# Understanding `client.py`

Server

```
client.py
1 from cryptobox import *
2
3 class Client():
4     def __init__(self, url: bytes, trusted_certs: list[bytes]):
5         self.url = url
6         self.trusted_certs = trusted_certs
7
8     def send_client_hello(self) -> bytes:
9         None
10
11
12     def send_client_ready(self, server_hello: bytes) -> bytes:
13         None
14
15
16     def receive_data(self, data: bytes) -> bytes:
17         None
18
19
20
```

client\_hello

server\_hello

client\_ready

data

Should return the  
decrypted data

Step 1

**Key Exchange  
and  
Data Decryption**

**send\_client\_hello:**

1. **Generate**  $K_{S1}, K_{P1}$

client\_hello:  $K_{P1}$

server\_hello:  $K_{P2}$

**send\_server\_hello:**

1. **Generate**  $K_{S2}, K_{P2}$

2. **Derive**  $k = \text{kex}(K_{S2}, K_{P1})$

**send\_client\_ready:**

1. **Derive**  $k = \text{kex}(K_{S1}, K_{P2})$

2. **Encrypt**  $\text{readyc} = \text{enc}(k, \text{b"ready"})$

client\_ready:  $\text{readyc}$

data:  $mc$

**send\_data:**

1. **Decrypt**  $\text{b"ready"}$  using  $k$

2. **Encrypt**  $mc = \text{enc}(k, m)$

**receive\_data:**

1. **Decrypt**  $m$  using  $k$

# Useful Functions from `cryptobox`

## Key Exchange


---

```
from cryptobox import generate_asymmetric_key, key_exchange

priv1, pub1 = generate_asymmetric_key()
priv2, pub2 = generate_asymmetric_key()

shared1 = key_exchange(priv1, pub2)
shared2 = key_exchange(priv2, pub1)

assert shared1 == shared2
```




## Symmetric Encryption

---

```
from cryptobox import generate_symmetric_key, encrypt, decrypt

key = generate_symmetric_key()
message = b"Hello World!"
ciphertext = encrypt(key, message)
plaintext = decrypt(key, ciphertext)

assert plaintext == message
```





# What we have so far

Let's test all 5 servers :

```
python3 run.py greenpath.program server1
```

```
python3 run.py greenpath.program server2
```

```
python3 run.py greenpath.program server3
```

```
python3 run.py greenpath.program server4
```

```
python3 run.py greenpath.program server5
```

Each of these servers give you some content **but which one to trust?**

Step 2

**Certificate Verification  
and  
Server's Signature Verifications**

## send\_client\_hello:

1. Generate  $K_{s1}, K_{p1}$

client\_hello:  $K_{p1}$

## send\_server\_hello:

1. Generate  $K_{s2}, K_{p2}$
2. Derive  $k = \text{kex}(K_{s2}, K_{p1})$
3. **Sign**  $\text{sig}_B = \text{sign}(K_{sB}, K_{p1} + K_{p2} + \text{cert})$
4. **Encrypt**  $\text{sigc} = \text{enc}(k, \text{sig}_B + \text{cert})$

server\_hello:  $K_{p2} + \text{sigc}$

## send\_client\_ready:

1. Derive  $k = \text{kex}(K_{s1}, K_{p2})$
2. **Decrypt**  $\text{sig}_B + \text{cert}$  using  $k$
3. **Check** cert using trusted certs
4. **Check**  $\text{verify}(K_{p1} + K_{p2} + \text{cert}, \text{sig}_B, K_{pB})$
5. Encrypt  $\text{readyc} = \text{enc}(k, \text{b"ready"})$

client\_ready:  $\text{readyc}$

## send\_data:

1. Decrypt  $\text{b"ready"}$  using  $k$
2. Encrypt  $\text{mc} = \text{enc}(k, m)$

data:  $\text{mc}$

## receive\_data:

1. Decrypt  $m$  using  $k$

# Useful Functions from `cryptobox`

## Digital Signature

---

```
from cryptobox import generate_assymmetric_key, sign, verify

priv, pub = generate_assymmetric_key()
msg = b"Hello World!"
signature = sign(priv, msg)

assert verify(msg, signature, pub)
```



## Certificate

---

```
from cryptobox import generate_assymmetric_key, create_certificate_authority, certificate_signing_request, generate_certificate

ca_priv, ca_pub = generate_assymmetric_key()
ca_cert = create_certificate_authority(ca_priv)

priv, pub = generate_assymmetric_key()
csr = certificate_signing_request(priv, "example.com")

cert = generate_certificate(ca_cert, ca_priv, csr)

print(certificate_info(cert))
assert verify_certificate(cert, [ca_cert])
```



Final Step

**Find the Trusted Server  
among the Imposters**



# Only one can be trusted

Only one command does not fail, which one?

```
python3 run.py greenpath.program server1
```

```
python3 run.py greenpath.program server2
```

```
python3 run.py greenpath.program server3
```

```
python3 run.py greenpath.program server4
```

```
python3 run.py greenpath.program server5
```

**[Bonus]** can you explain what is wrong with the four other servers?  
What makes them imposters?