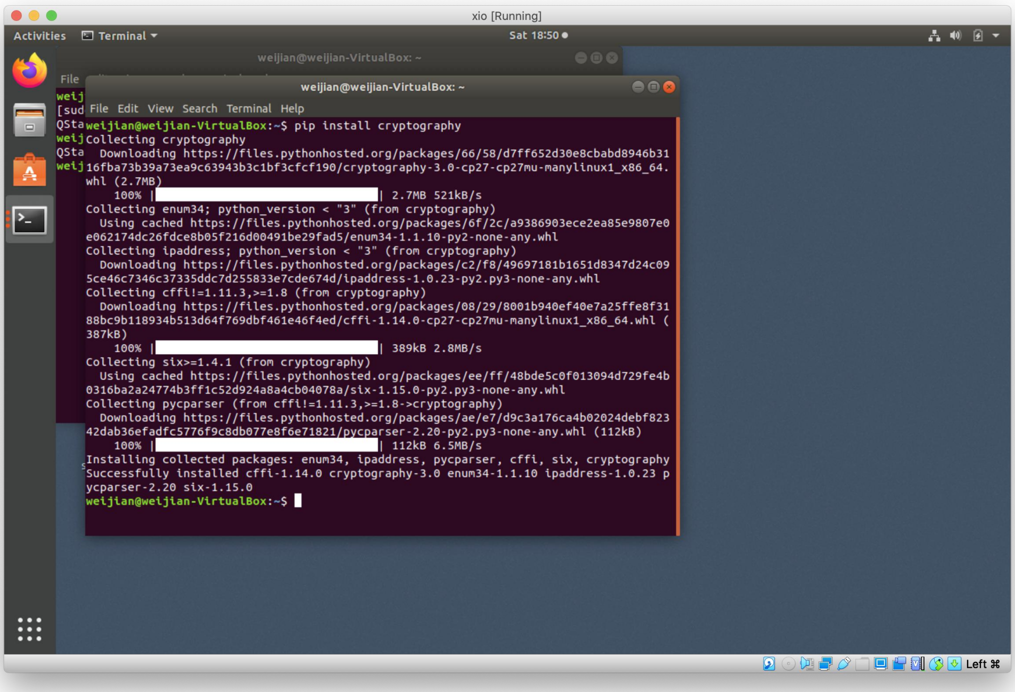
How Does Cryptography Help?

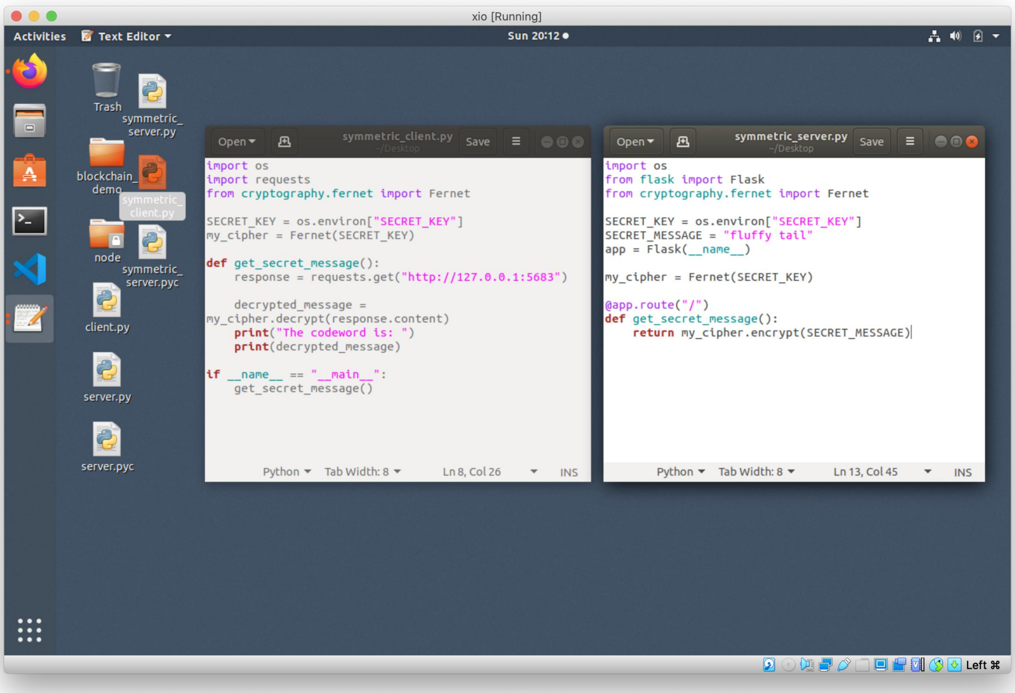
In this part, you'll learn one way to keep your data safe by creating your own cryptography keys and using them on both your server and your client. While this won't be your final step, it will help you get a solid foundation for how to build Python HTTPS applications.

We need install cryptography first:


xio [Running]
Activities
Terminal
Sat 18:50
weijian@weijian-VirtualBox: ~
File Edit View Search Terminal Help

```
weijian@weijian-VirtualBox:~$ pip install cryptography
Collecting cryptography
  Downloading https://files.pythonhosted.org/packages/66/58/d7ff652d30e8cbabd8946b31
16fba73b39a73ea9c63943b3c1bf3cfcf190/cryptography-3.0-cp27-cp27mu-manylinux1_x86_64.
whl (2.7MB)
    100% |████████████████████████████████| 2.7MB 521kB/s
Collecting enum34; python_version < "3" (from cryptography)
  Using cached https://files.pythonhosted.org/packages/6f/2c/a9386903ece2ea85e9807e0
e062174dc26fdce8b05f216d00491be29fad5/enum34-1.1.10-py2-none-any.whl
Collecting ipaddress; python_version < "3" (from cryptography)
  Downloading https://files.pythonhosted.org/packages/c2/f8/49697181b1651d8347d24c09
5ce46c7346c37335ddc7d255833e7cde674d/ipaddress-1.0.23-py2.py3-none-any.whl
Collecting cffi!=1.11.3,>=1.8 (from cryptography)
  Downloading https://files.pythonhosted.org/packages/08/29/8001b940ef40e7a25ffe8f31
88bc9b118934b513d64f769dbf461e46f4ed/cffi-1.14.0-cp27-cp27mu-manylinux1_x86_64.whl (
387kB)
    100% |████████████████████████████████| 389kB 2.8MB/s
Collecting six>=1.4.1 (from cryptography)
  Using cached https://files.pythonhosted.org/packages/ee/ff/48bde5c0f013094d729fe4b
0316ba2a24774b3ff1c52d924a8a4cb04078a/six-1.15.0-py2.py3-none-any.whl
Collecting pycparser (from cffi!=1.11.3,>=1.8->cryptography)
  Downloading https://files.pythonhosted.org/packages/ae/e7/d9c3a176ca4b02024debf823
42dab36efadfc5776f9c8db077e8f6e71821/pycparser-2.20-py2.py3-none-any.whl (112kB)
    100% |████████████████████████████████| 112kB 6.5MB/s
Installing collected packages: enum34, ipaddress, pycparser, cffi, six, cryptography
Successfully installed cffi-1.14.0 cryptography-3.0 enum34-1.1.10 ipaddress-1.0.23 p
ycparser-2.20 six-1.15.0
weijian@weijian-VirtualBox:~$ █
```

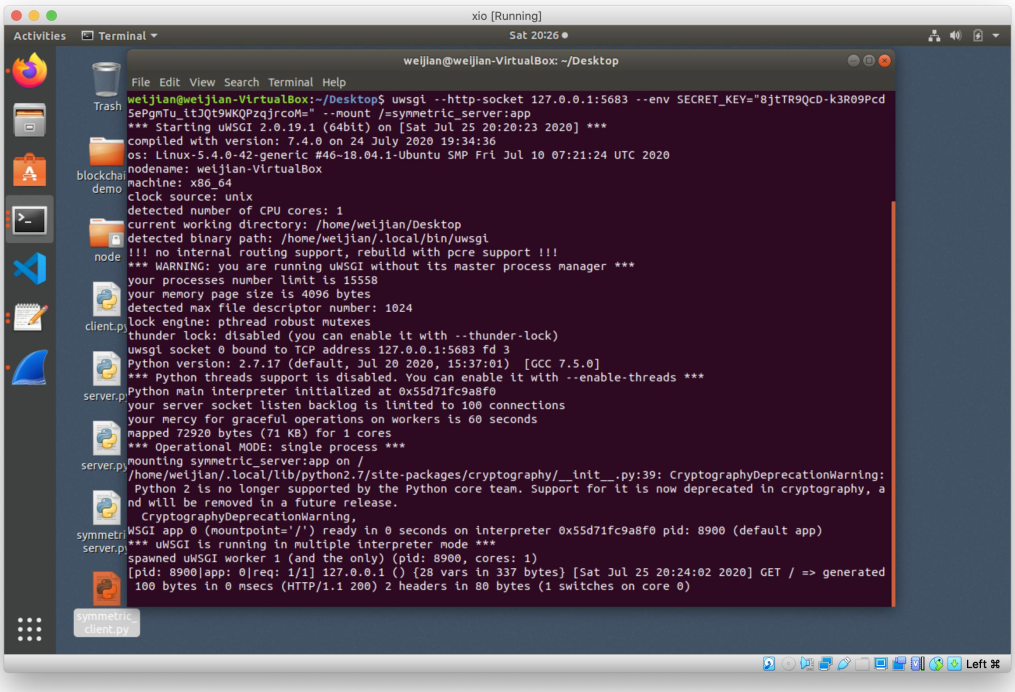
Then we create new server.py and client.py files:



```
import os
import requests
from cryptography.fernet import Fernet

SECRET_KEY = os.environ["SECRET_KEY"]
my_cipher = Fernet(SECRET_KEY)

def get_secret_message():
    response = requests.get("http://127.0.0.1:5683")

    decrypted_message =
my_cipher.decrypt(response.content)
    print("The codeword is: ")
    print(decrypted_message)

if __name__ == "__main__":
    get_secret_message()
```

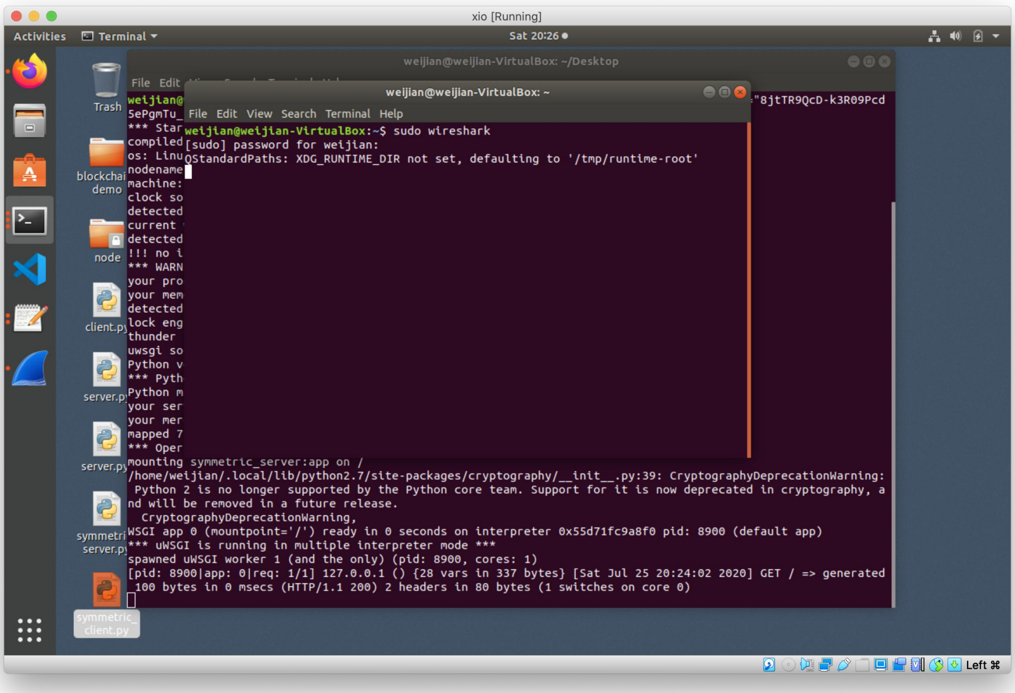
```
import os
from flask import Flask
from cryptography.fernet import Fernet

SECRET_KEY = os.environ["SECRET_KEY"]
SECRET_MESSAGE = "fluffy tail"
app = Flask(__name__)

my_cipher = Fernet(SECRET_KEY)

@app.route("/")
def get_secret_message():
    return my_cipher.encrypt(SECRET_MESSAGE)
```

Run the server application with SECRET_KEY:


xio [Running]
Activities
Terminal
Sat 20:26
weijian@weijian-VirtualBox: ~/Desktop
File Edit View Search Terminal Help

```
weijian@weijian-VirtualBox:~/Desktop$ uwsgi --http-socket 127.0.0.1:5683 --env SECRET_KEY="8jtTR9QcD-k3R09Pcd
5ePgmTu_itJQt9WKQPzqjrcoM=" --mount /=symmetric_server:app
*** Starting uWSGI 2.0.19.1 (64bit) on [Sat Jul 25 20:20:23 2020] ***
compiled with version: 7.4.0 on 24 July 2020 19:34:36
os: Linux-5.4.0-42-generic #46~18.04.1-Ubuntu SMP Fri Jul 10 07:21:24 UTC 2020
nodename: weijian-VirtualBox
machine: x86_64
clock source: unix
detected number of CPU cores: 1
current working directory: /home/weijian/Desktop
detected binary path: /home/weijian/.local/bin/uwsgi
!!! no internal routing support, rebuild with pcre support !!!
*** WARNING: you are running uWSGI without its master process manager ***
your processes number limit is 15558
your memory page size is 4096 bytes
detected max file descriptor number: 1024
lock engine: pthread robust mutexes
thunder lock: disabled (you can enable it with --thunder-lock)
uwsgi socket 0 bound to TCP address 127.0.0.1:5683 fd 3
Python version: 2.7.17 (default, Jul 20 2020, 15:37:01)  [GCC 7.5.0]
*** Python threads support is disabled. You can enable it with --enable-threads ***
Python main interpreter initialized at 0x55d71fc9a8f0
your server socket listen backlog is limited to 100 connections
your mercy for graceful operations on workers is 60 seconds
mapped 72920 bytes (71 KB) for 1 cores
*** Operational MODE: single process ***
mounting symmetric_server:app on /
/home/weijian/.local/lib/python2.7/site-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning:
 Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, a
nd will be removed in a future release.
  CryptographyDeprecationWarning,
WSGI app 0 (mountpoint='/') ready in 0 seconds on interpreter 0x55d71fc9a8f0 pid: 8900 (default app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 8900, cores: 1)
[pid: 8900|app: 0|req: 1/1] 127.0.0.1 () {28 vars in 337 bytes} [Sat Jul 25 20:24:02 2020] GET / => generated
 100 bytes in 0 msecs (HTTP/1.1 200) 2 headers in 80 bytes (1 switches on core 0)
```

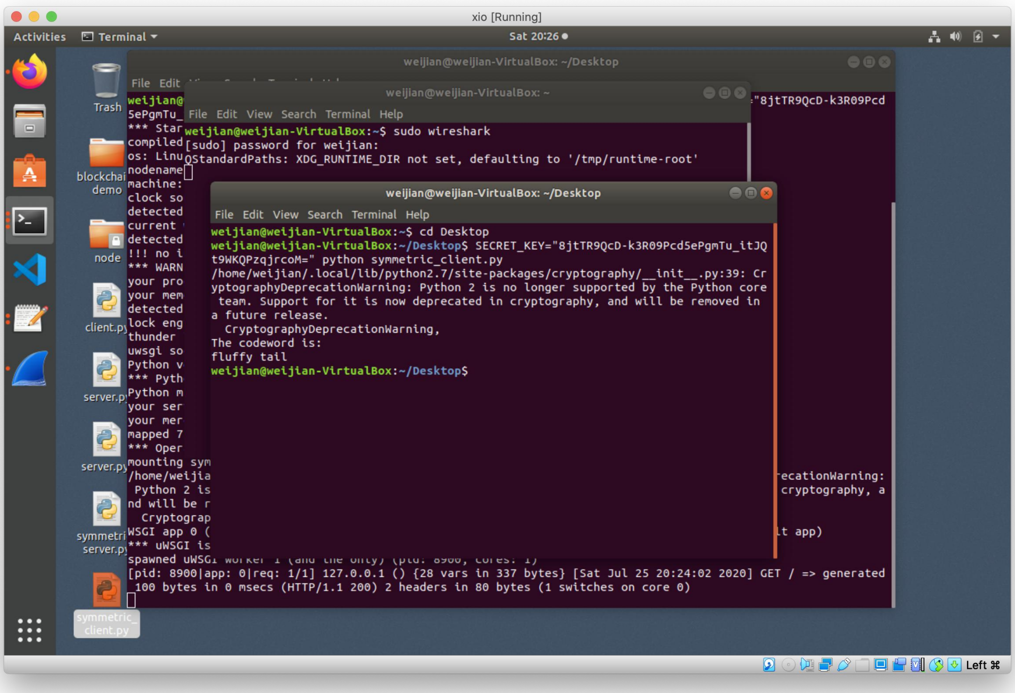
Open wireshark:



```
weijian@weijian-VirtualBox:~$ sudo wireshark
[sudo] password for weijian:
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

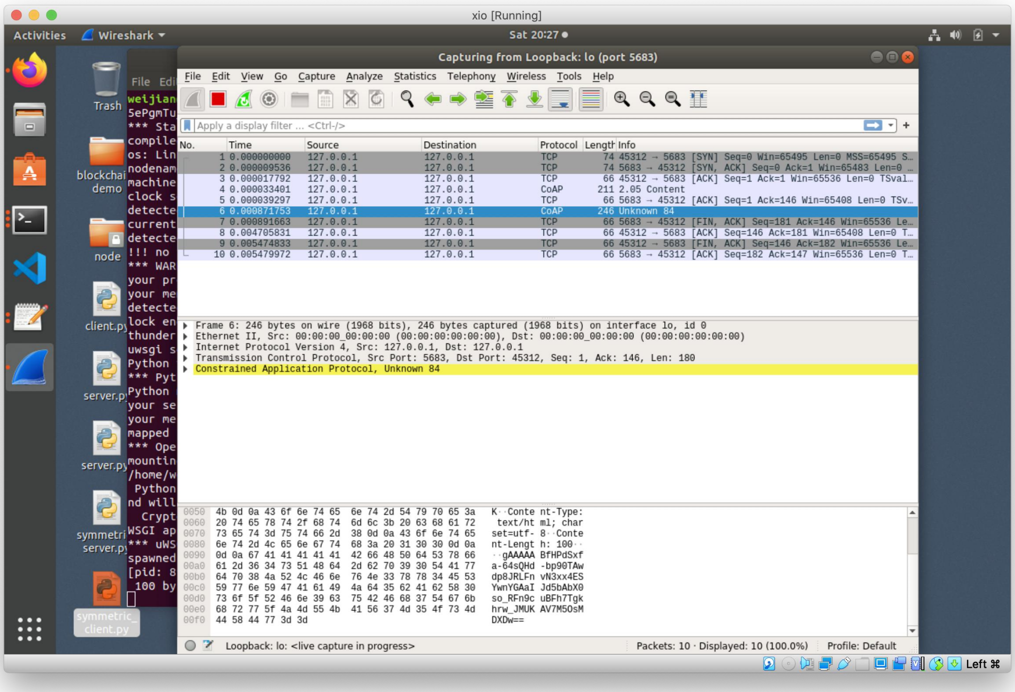
```
mounting symmetric_server:app on /
/home/weijian/.local/lib/python2.7/site-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning:
 Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, a
nd will be removed in a future release.
  CryptographyDeprecationWarning,
WSGI app 0 (mountpoint='/') ready in 0 seconds on interpreter 0x55d71fc9a8f0 pid: 8900 (default app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 8900, cores: 1)
[pid: 8900|app: 0|req: 1/1] 127.0.0.1 () {28 vars in 337 bytes} [Sat Jul 25 20:24:02 2020] GET / => generated
 100 bytes in 0 msecs (HTTP/1.1 200) 2 headers in 80 bytes (1 switches on core 0)
```

Run client.py:



```
weijian@weijian-VirtualBox:~$ cd Desktop
weijian@weijian-VirtualBox:~/Desktop$ SECRET_KEY="8jtTR9QcD-k3R09Pcd5ePgmTu_itJQ
t9WKQPzqjrcoM=" python symmetric_client.py
/home/weijian/.local/lib/python2.7/site-packages/cryptography/__init__.py:39: Cr
yptographyDeprecationWarning: Python 2 is no longer supported by the Python core
 team. Support for it is now deprecated in cryptography, and will be removed in
a future release.
  CryptographyDeprecationWarning,
The codeword is:
fluffy tail
weijian@weijian-VirtualBox:~/Desktop$
```

We still can get secret message, now let’s see what we got in wireshark:


xio [Running]
Activities
Wireshark
Sat 20:27
Capturing from Loopback: lo (port 5683)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Trash
blockchain demo
node
client.py
server.py
server.py
symmetric server.py
symmetric client.py

```
No.  Time         Source      Destination  Protocol Length Info
   1 0.000000000  127.0.0.1   127.0.0.1    TCP        74 45312 → 5683 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 S…
   2 0.000009536  127.0.0.1   127.0.0.1    TCP        74 5683 → 45312 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 …
   3 0.000017792  127.0.0.1   127.0.0.1    TCP        66 45312 → 5683 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval…
   4 0.000033401  127.0.0.1   127.0.0.1    CoAP      211 2.05 Content
   5 0.000039297  127.0.0.1   127.0.0.1    TCP        66 5683 → 45312 [ACK] Seq=1 Ack=146 Win=65408 Len=0 TSv…
   6 0.000871753  127.0.0.1   127.0.0.1    CoAP      246 Unknown 84
   7 0.000891663  127.0.0.1   127.0.0.1    TCP        66 5683 → 45312 [FIN, ACK] Seq=181 Ack=146 Win=65536 Le…
   8 0.004705831  127.0.0.1   127.0.0.1    TCP        66 45312 → 5683 [ACK] Seq=146 Ack=181 Win=65408 Len=0 T…
   9 0.005474833  127.0.0.1   127.0.0.1    TCP        66 45312 → 5683 [FIN, ACK] Seq=146 Ack=182 Win=65536 Le…
  10 0.005479972  127.0.0.1   127.0.0.1    TCP        66 5683 → 45312 [ACK] Seq=182 Ack=147 Win=65536 Len=0 T…

▸ Frame 6: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 5683, Dst Port: 45312, Seq: 1, Ack: 146, Len: 180
▸ Constrained Application Protocol, Unknown 84

0050  4b 0d 0a 43 6f 6e 74 65  6e 74 2d 54 79 70 65 3a   K··Conte nt-Type:
0060  20 74 65 78 74 2f 68 74  6d 6c 3b 20 63 68 61 72    text/ht ml; char
0070  73 65 74 3d 75 74 66 2d  38 0d 0a 43 6f 6e 74 65   set=utf- 8··Conte
0080  6e 74 2d 4c 65 6e 67 74  68 3a 20 31 30 30 0d 0a   nt-Lengt h: 100··
0090  0d 0a 67 41 41 41 41 41  42 66 48 50 64 53 78 66   ··gAAAAA BfHPdSxf
00a0  61 2d 36 34 73 51 48 64  2d 62 70 39 30 54 41 77   a-64sQHd -bp90TAw
00b0  64 70 38 4a 52 4c 46 6e  76 4e 33 78 78 34 45 53   dp8JRLFn vN3xx4ES
00c0  59 77 6e 59 47 41 61 49  4a 64 35 62 41 62 58 30   YwnYGAaI Jd5bAbX0
00d0  73 6f 5f 52 46 6e 39 63  75 42 46 68 37 54 67 6b   so_RFn9c uBFh7Tgk
00e0  68 72 77 5f 4a 4d 55 4b  41 56 37 4d 35 4f 73 4d   hrw_JMUK AV7M5OsM
00f0  44 58 44 77 3d 3d                                  DXDw==

Loopback: lo: <live capture in progress>        Packets: 10 · Displayed: 10 (100.0%)        Profile: Default
```

Awesome! This means that the data was encrypted and that eavesdroppers have no clue what the message content actually is. Not only that, but it also means that they could spend an insanely long amount of time trying to brute-force crack this data, and they would almost never be successful.

Your data is safe! But wait a minute—you never had to know anything about a key when you were using Python HTTPS applications before. That's because HTTPS doesn't use symmetric encryption exclusively. As it turns out, sharing secrets is a hard problem.

To prove this concept, navigate to http://127.0.0.1:5683 in your browser, and you'll see the encrypted response text. This is because your browser doesn't know anything about your secret cipher key. So how do Python HTTPS applications really work? That's where asymmetric encryption comes into play.