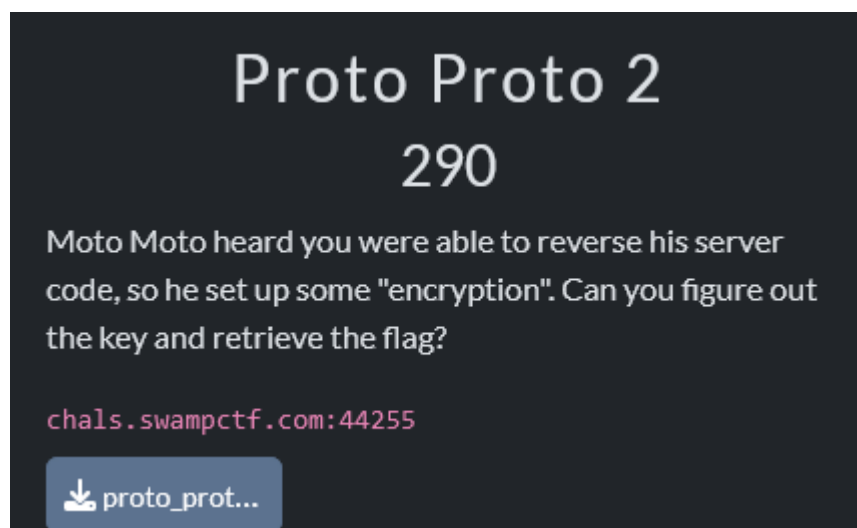
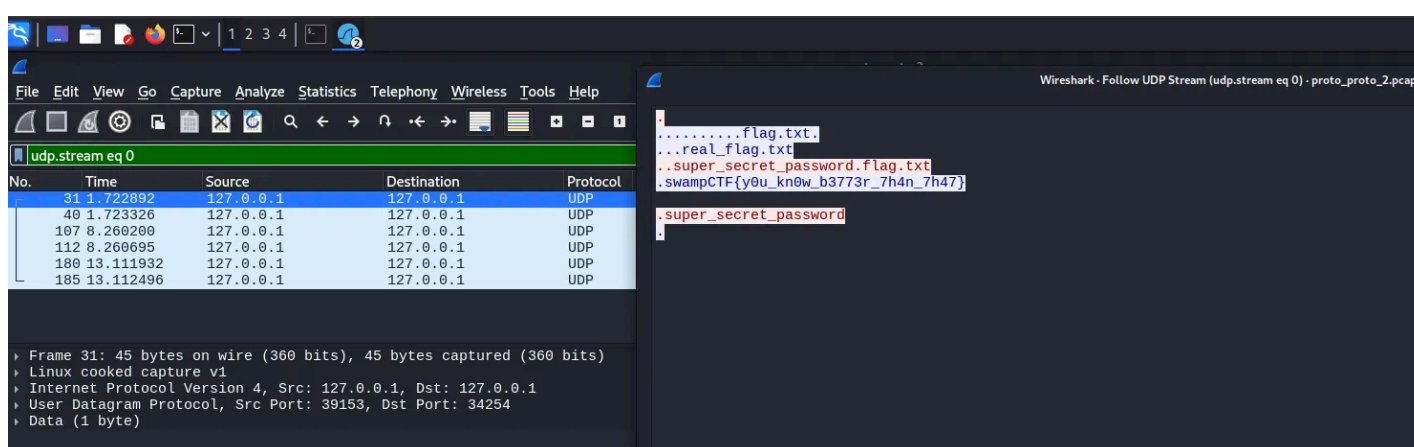


# Forensics - Proto Proto 2



The task description hints at some encryption. Let's look at the UDP traffic in the new pcap file:



We can see that the client gets the flag using: **super\_secret\_password.flag.txt**. We need to somehow get the value of super\_secret\_password.

What if we try to pass the value of a fake flag (**swampCTF{y0u\_kn0w\_b3773r\_7h4n\_7h47}**) from a pcap file as super\_secret\_password, and then add **.flag.txt**? Let's try it.

Our script (the logic is the same as in the previous Proto Proto challenge):

```
import socket # Importing the socket library to create a network connection

# Define the server's address and port number
server = "chals.swampctf.com"
port = 44255

# Construct the payload that will be sent to the server
payload = (
    b"\x02" # The first byte is likely a protocol identifier or header (starting byte)
    + len(b"swampCTF{y0u_kn0w_b3773r_7h4n_7h47}").to_bytes(1, "big") # Length of the first string ("swampCTF{y0u_k
    + b"swampCTF{y0u_kn0w_b3773r_7h4n_7h47}" # The first string, which is the value to be sent to the server
    + len(b".flag.txt").to_bytes(1, "big") # Length of the second string ("flag.txt") in a single byte
    + b".flag.txt" # The second string, which is the value to be sent to the server
)

# Create a UDP socket (AF_INET for IPv4, SOCK_DGRAM for UDP)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Send the constructed payload to the server at the specified address and port
```

```
s.sendto(payload, (server, port))

# Receive the response from the server, up to 4096 bytes
response, addr = s.recvfrom(4096)

# Close the socket after receiving the response to free up resources
s.close()

# Print the response in hexadecimal format for inspection (useful for debugging)
print("HEX response:", response.hex())

# Print the decoded response (decoding may ignore non-UTF characters)
print("Decoded response:", response.decode(errors='ignore'))
```

Execute it:

```
root@kali: /home/kali/Desktop
File Actions Edit View Help
(root@kali)-[/home/kali/Desktop]
# python3 for16.py
HEX response: 07695f646f5f7265616c4b652c0c467a70340652686a11624278757f0a33350d32082b345c6d387f14
Decoded response: i_do_realKe, Fzp4RhjbBxu
+4\m8
```

In Decoded response we got **part of the key**! You might think that "Ke" is part of the key, but by changing the key with the fake flag to just "swampCTF", we get exactly **i\_do\_real**. So, part of the key: **i\_do\_real...**

Next, you actually need to guess the rest of the key a little bit... Let's look at the challenge description again:



Why not use the word "encryption" as the rest of the key? Let's try this option and our key will be: **i\_do\_real\_encryption**  
 //A bit of a guess, I know :)

Let's change the key in the previous code to our new key:

```
import socket

server = "chals.swampctf.com"
port = 44255

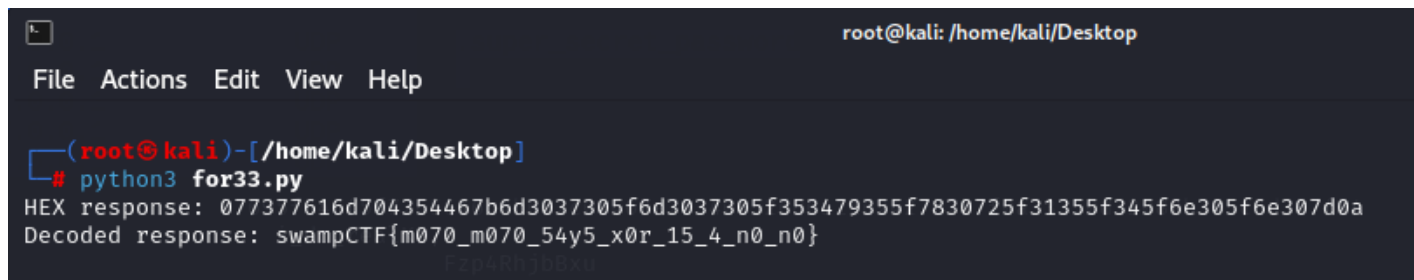
payload = (
    b"\x02"
    + len(b"i_do_real_encryption").to_bytes(1, "big")
    + b"i_do_real_encryption"
    + len(b"flag.txt").to_bytes(1, "big")
    + b"flag.txt"
)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(payload, (server, port))
response, addr = s.recvfrom(4096)
```

```
s.close()

print("HEX response:", response.hex())
print("Decoded response:", response.decode(errors='ignore'))
```

Execute it:

A terminal window with a dark background and light text. The title bar shows 'root@kali: /home/kali/Desktop'. The menu bar includes 'File', 'Actions', 'Edit', 'View', and 'Help'. The prompt is '(root@kali)-[/home/kali/Desktop]'. The user has entered '# python3 for33.py'. The output shows 'HEX response: 077377616d704354467b6d3037305f6d3037305f353479355f7830725f31355f345f6e305f6e307d0a' and 'Decoded response: swampCTF{m070\_m070\_54y5\_x0r\_15\_4\_n0\_n0}'.

```
root@kali: /home/kali/Desktop
File Actions Edit View Help
(root@kali)-[/home/kali/Desktop]
# python3 for33.py
HEX response: 077377616d704354467b6d3037305f6d3037305f353479355f7830725f31355f345f6e305f6e307d0a
Decoded response: swampCTF{m070_m070_54y5_x0r_15_4_n0_n0}
```

We got the flag!

Flag: **swampCTF{m070\_m070\_54y5\_x0r\_15\_4\_n0\_n0}**