CS631 ASSIGNMENT-3
Introducing cryptography to the modbus communication protocol to have communication security in the interaction between modbus simulator and the client.

Gargi Sarkar
Roll no: 21111263

October 2021

Contents

1 Introduction

The Modbus communication protocol specification does not include any authentication mechanism for validating communication between the master and slave. For this reason, any unauthenticated remote attacker can easily breach into any slave via a Modbus master. The goal of this asked assignment is to implement some cryptographic measure to achieve a sort of security. To do that we are taking help of some proxy servers to avoid direct communication between the client and the Modbus device.

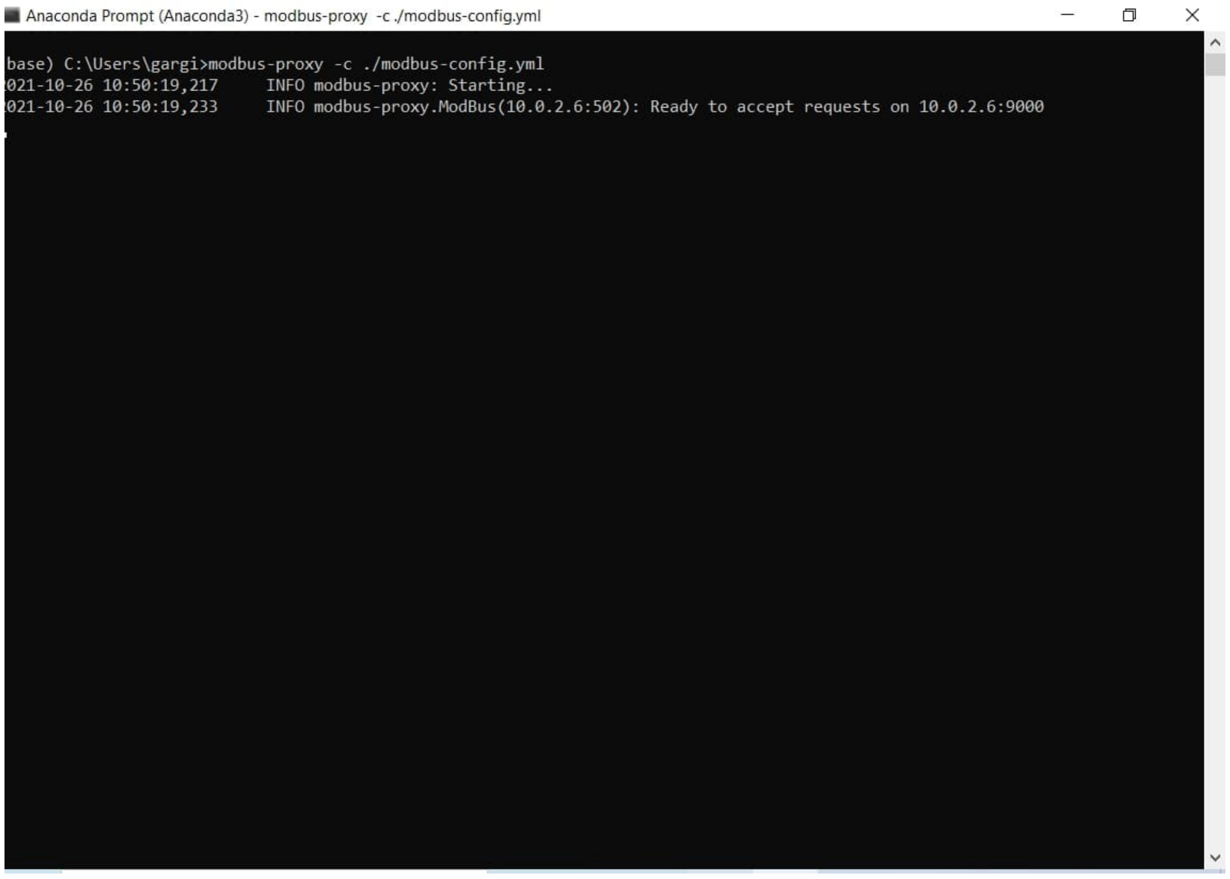
2 Motivation

The Modbus communication protocol first came in use in 1979 to communicate specially between PLCs, but later for its simplicity and robustness it has become defacto standard communication for Industrial control Systems. But due to being this old it is lacking security as in that old arena Cyber Security was not of concern. The time has changed now. Cyber attacks in ICS/OT system is very common, even leading to nation's security and safety crisis, thereby, forcing the implementation of cryptographic protocols to secure safety to the systems. One of the very common and easy attack on modbus is sniffing the traffic and finding the particular modbus device (its easy because the communication is not encrypted)and then manipulation of commands. Here in this task we are going to encrypt the communication to avoid the mentioned.

3 Proxy-server and its Configuration

The proxy server acts as a bridge between client and Modbus device. It allows multiple clients to communicate with the same Modbus device. To run the proxy server, a configuration file in YAML specifying the Modbus connection i.e. the Modbus device url and the listen interface i.e. to which url the clients connect should be written at first. Using this configuration, file the Modbus proxy and the client connection will be established.

Figure 1: Building connection with the proxy server



Anaconda Prompt (Anaconda3) - modbus-proxy -c ./modbus-config.yml

```
base) C:\Users\gargi>modbus-proxy -c ./modbus-config.yml
021-10-26 10:50:19,217     INFO modbus-proxy: Starting...
021-10-26 10:50:19,233     INFO modbus-proxy.ModBus(10.0.2.6:502): Ready to accept requests on 10.0.2.6:9000
```

4 Key Exchange between Proxy server and Client

Symmetric encryption is faster than asymmetric encryption . And as availability is the primary concern in ICS/OT system, so, choosing symmetric encryption is the smarter way. But, as the proxy server and client never met before, they do not have the pre-exchanged same key. To do the exchange securely, we will be needing the help of asymmetric cryptography. As according to asymmetric cryptography protocol (here we are using RSA) the receiver (here the proxy) will be generating public, private key pairs, and by keeping his private key secret, it will send the public key to the client. Now the client will generate some symmetric key and after receiving the public key from the proxy, it will be encrypting the symmetric key using the rsa public key and send it back to the Modbus proxy. The proxy will decrypt the message using its private key, and therefore, that will be leading the proxy and the client to have the same symmetric key to communicate between them.

4.1 Generation of asymmetric key pair

We will be generating it on the very beginning of modbus-proxy.py file using rsa protocol as:
$publickey, privatekey = rsa.newkeys()$

4.2 Sending the public key to the client

To send the above generated public key to the client, use the code:
$client.writer.write()$ in the modbus - proxy.py file's $class\ Modbus\ (connection)$, $async\ def\ handle\ client$ section

4.3 Receiving the public key in client from the proxy

So in the modbus_client file, receive the proxy public key using:
$rsa.Publickey.\ load\_pkcs1()$

4.4 Generation of symmetric key

In the modbus client file, generate the encryption key using $Fernet.generate\_key()$

4.5 Encrypting the symmetric key using proxy public key

In the modbus client, encrypt the symmetric key using the function:
$rsa.encrypt(symmetrickey, publicKey)$.

4.6 Send the encrypted key to the proxy

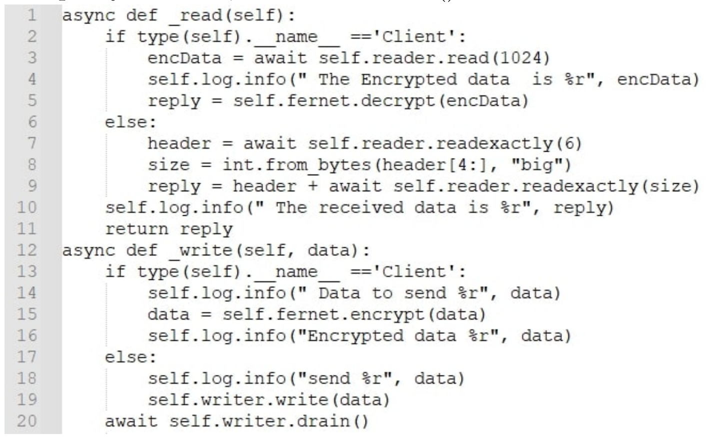
Use the function call $sock.sendall()$to send the encrypted key to the proxy.

4.7 Receiving encrypted message from client in proxy and decrypting it

In the modbus proxy file $class\ Modbus(Connection) async\ def handle\_client()$, use the function $client.reader.read()$ to receive the encrypted symmetric key and after receiving it, use $rsa.decrypt(encrypted\ message, privatekey)$ to decrypt.

5 Encrypted data communication using symmetric key cryptography

As both parties are having the same key now, to start further communication of data, the first thing we have to do is to transfer the ADU. To encrypt the ADU using the symmetric key, call the function $fernet.encrypt(adu)$ in tcp.py(client) file and send the encrypt data to the proxy using $sock.sendall()$function. To receive the data from proxy to the client, use the function $sock.recv()$ and decrypt the data using $fernet.decrypt()$ in the same tcp.py(client) file. Similarly, in order to receive the data from client to proxy in the modbusproxy file's $class\ Modbus(Connection)$, we have to use $reader.read()$ function and to send from proxy to client, use $writer.write()$ function.

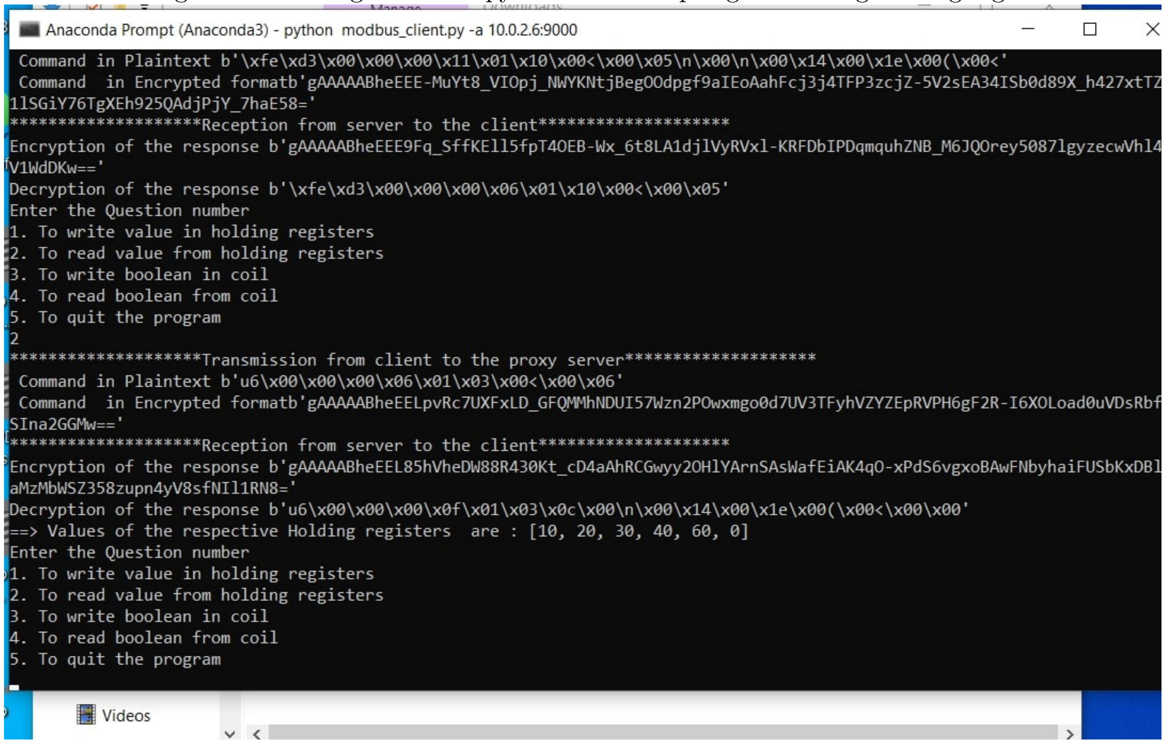


```
async def _read(self):
    if type(self).__name__ =='Client':
        encData = await self.reader.read(1024)
        self.log.info(" The Encrypted data  is %r", encData)
        reply = self.fernet.decrypt(encData)
    else:
        header = await self.reader.readexactly(6)
        size = int.from_bytes(header[4:], "big")
        reply = header + await self.reader.readexactly(size)
    self.log.info(" The received data is %r", reply)
    return reply
async def _write(self, data):
    if type(self).__name__ =='Client':
        self.log.info(" Data to send %r", data)
        data = self.fernet.encrypt(data)
        self.log.info("Encrypted data %r", data)
    else:
        self.log.info("send %r", data)
        self.writer.write(data)
    await self.writer.drain()
```

The _read() function in proxy file is having a two sided connection, it receives commands from both sides of the bridge, i.e. from the client and the server. The function $if\ type(self).\_name\_ ==' Client'$ is used to specify from whom to read, because if the message is coming from the server, it can simply read that, whereas, if the message is coming from the client, it has to apply some decryption mechanism to read that as it will be encrypted.
Similarly, the _write() function in proxy file is also having a two sided connection, it sends modbus commands to both side of the bridge. Using the same $if\ type(self).\_name\_ ==' Client'$ function, we have to specify to whom to send. To send to the client, the proxy has to encrypt before sending the data, whereas, it will directly send command to the modbus device.

The detailed changes made by me are in the corresponding attached files. To keep the PDF short(as advised), I am ending my report here. The snaps of my work with the simulator are given below:

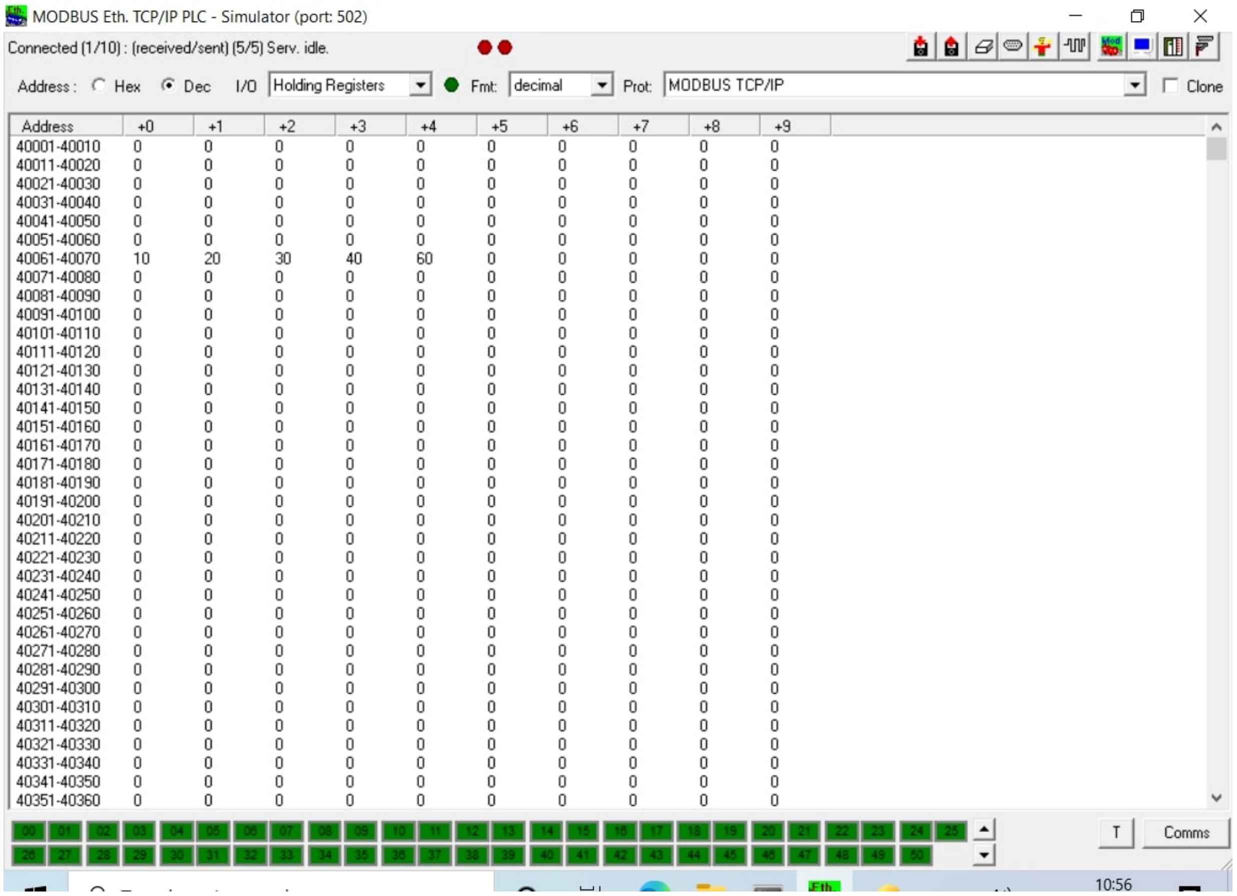
Figure 2: Running the client python file and opting for writing holding registers



Anaconda Prompt (Anaconda3) - python modbus_client.py -a 10.0.2.6:9000

```
 Command in Plaintext b'\xfe\xd3\x00\x00\x00\x11\x01\x10\x00<\x00\x05\n\x00\n\x00\x14\x00\x1e\x00(\x00<'
 Command  in Encrypted formatb'gAAAAABheEEE-MuYt8_VIOpj_NWYKNtjBegOOdpgf9aIEoAahFcj3j4TFP3zcjZ-5V2sEA34ISb0d89X_h427xtTZ
1lSGiY76TgXEh925QAdjPjY_7haE58='
*******************Reception from server to the client*******************
Encryption of the response b'gAAAAABheEEE9Fq_SffKEll5fpT4OEB-Wx_6t8LA1djlVyRVxl-KRFDbIPDqmquhZNB_M6JQOrey5087lgyzecwVhl4
V1WdDKw=='
Decryption of the response b'\xfe\xd3\x00\x00\x00\x06\x01\x10\x00<\x00\x05'
Enter the Question number
1. To write value in holding registers
2. To read value from holding registers
3. To write boolean in coil
4. To read boolean from coil
5. To quit the program
2
*******************Transmission from client to the proxy server*******************
 Command in Plaintext b'u6\x00\x00\x00\x06\x01\x03\x00<\x00\x06'
 Command  in Encrypted formatb'gAAAAABheEELpvRc7UXFxLD_GFQMMhNDUI57Wzn2POwxmgo0d7UV3TFyhVZYZEpRVPH6gF2R-I6XOLoad0uVDsRbf
SIna2GGMw=='
*******************Reception from server to the client*******************
Encryption of the response b'gAAAAABheEEL85hVheDW88R430Kt_cD4aAhRCGwyy2OHlYArnSAsWafEiAK4qO-xPdS6vgxoBAwFNbyhaiFUSbKxDBl
aMzMbWSZ358zupn4yV8sfNIl1RN8='
Decryption of the response b'u6\x00\x00\x00\x0f\x01\x03\x0c\x00\n\x00\x14\x00\x1e\x00(\x00<\x00\x00'
==> Values of the respective Holding registers  are : [10, 20, 30, 40, 60, 0]
Enter the Question number
1. To write value in holding registers
2. To read value from holding registers
3. To write boolean in coil
4. To read boolean from coil
5. To quit the program
```

Figure 3: Corresponding result in the simulator for writing holding registers



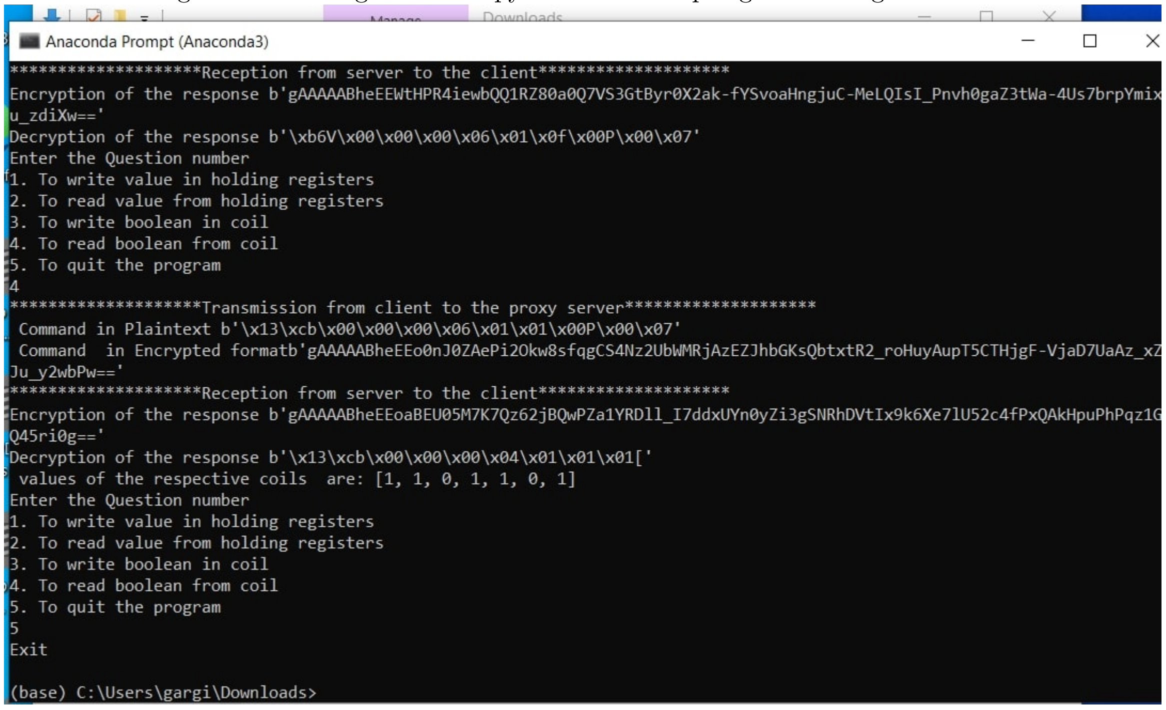
MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (1/10) : (received/sent) (5/5) Serv. idle.

Address : ○ Hex ● Dec I/O Holding Registers Fmt: decimal Prot: MODBUS TCP/IP ☐ Clone

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
40001-40010	0	0	0	0	0	0	0	0	0	0
40011-40020	0	0	0	0	0	0	0	0	0	0
40021-40030	0	0	0	0	0	0	0	0	0	0
40031-40040	0	0	0	0	0	0	0	0	0	0
40041-40050	0	0	0	0	0	0	0	0	0	0
40051-40060	0	0	0	0	0	0	0	0	0	0
40061-40070	10	20	30	40	60	0	0	0	0	0
40071-40080	0	0	0	0	0	0	0	0	0	0
40081-40090	0	0	0	0	0	0	0	0	0	0
40091-40100	0	0	0	0	0	0	0	0	0	0
40101-40110	0	0	0	0	0	0	0	0	0	0
40111-40120	0	0	0	0	0	0	0	0	0	0
40121-40130	0	0	0	0	0	0	0	0	0	0
40131-40140	0	0	0	0	0	0	0	0	0	0
40141-40150	0	0	0	0	0	0	0	0	0	0
40151-40160	0	0	0	0	0	0	0	0	0	0
40161-40170	0	0	0	0	0	0	0	0	0	0
40171-40180	0	0	0	0	0	0	0	0	0	0
40181-40190	0	0	0	0	0	0	0	0	0	0
40191-40200	0	0	0	0	0	0	0	0	0	0
40201-40210	0	0	0	0	0	0	0	0	0	0
40211-40220	0	0	0	0	0	0	0	0	0	0
40221-40230	0	0	0	0	0	0	0	0	0	0
40231-40240	0	0	0	0	0	0	0	0	0	0
40241-40250	0	0	0	0	0	0	0	0	0	0
40251-40260	0	0	0	0	0	0	0	0	0	0
40261-40270	0	0	0	0	0	0	0	0	0	0
40271-40280	0	0	0	0	0	0	0	0	0	0
40281-40290	0	0	0	0	0	0	0	0	0	0
40291-40300	0	0	0	0	0	0	0	0	0	0
40301-40310	0	0	0	0	0	0	0	0	0	0
40311-40320	0	0	0	0	0	0	0	0	0	0
40321-40330	0	0	0	0	0	0	0	0	0	0
40331-40340	0	0	0	0	0	0	0	0	0	0
40341-40350	0	0	0	0	0	0	0	0	0	0
40351-40360	0	0	0	0	0	0	0	0	0	0

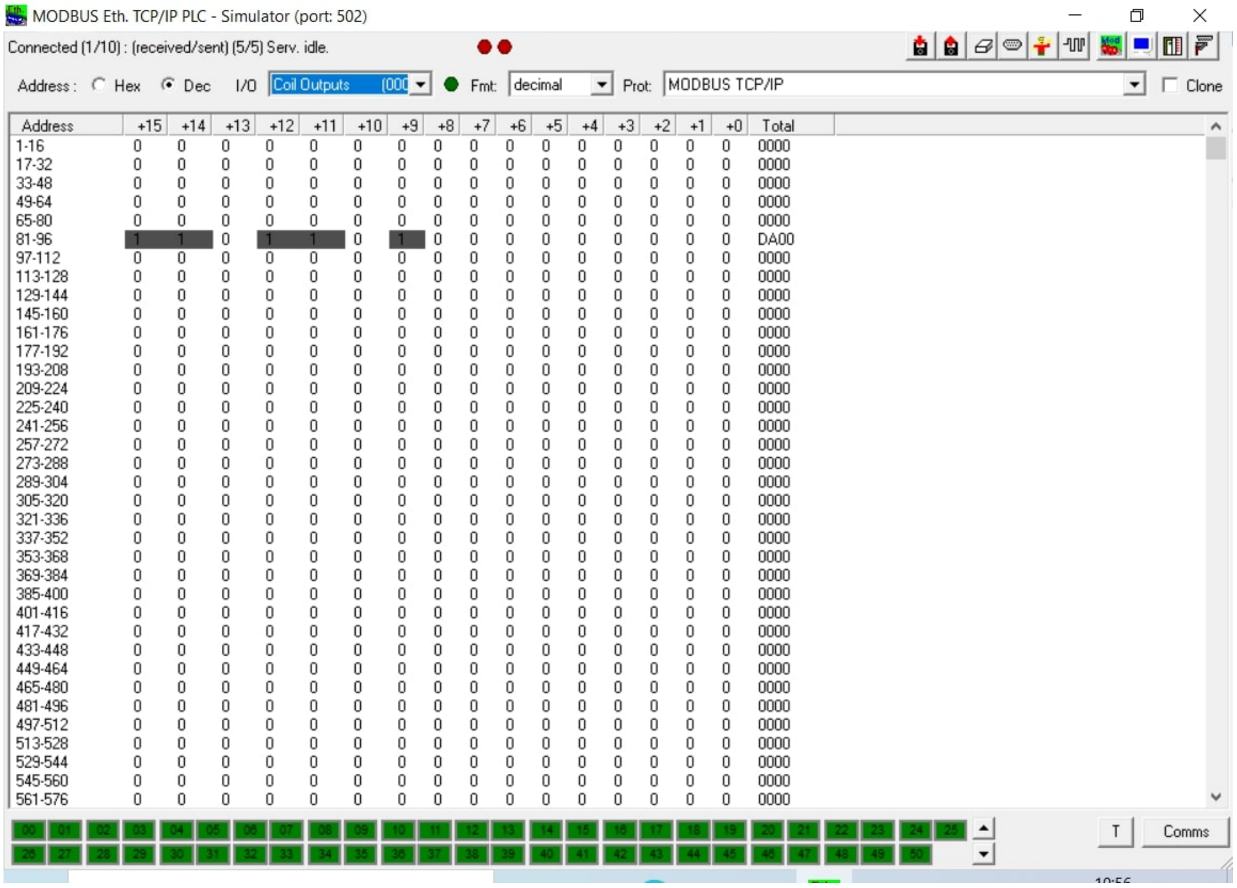
Figure 4: Running the client python file and opting for writing boolean in coil



```
*******************Reception from server to the client*******************
Encryption of the response b'gAAAAABheEEWtHPR4iewbQQ1RZ80a0Q7VS3GtByr0X2ak-fYSvoaHngjuC-MeLQIsI_Pnvh0gaZ3tWa-4Us7brpYmix
u_zdiXw=='
Decryption of the response b'\xb6V\x00\x00\x00\x06\x01\x0f\x00P\x00\x07'
Enter the Question number
1. To write value in holding registers
2. To read value from holding registers
3. To write boolean in coil
4. To read boolean from coil
5. To quit the program
4
*******************Transmission from client to the proxy server*******************
 Command in Plaintext b'\x13\xcb\x00\x00\x00\x06\x01\x01\x00P\x00\x07'
 Command  in Encrypted formatb'gAAAAABheEEo0nJ0ZAePi2Okw8sfqgCS4Nz2UbWMRjAzEZJhbGKsQbtxtR2_roHuyAupT5CTHjgF-VjaD7UaAz_xZ
Ju_y2wbPw=='
*******************Reception from server to the client*******************
Encryption of the response b'gAAAAABheEEoaBEU05M7K7Qz62jBQwPZa1YRDll_I7ddxUYn0yZi3gSNRhDVtIx9k6Xe7lU52c4fPxQAkHpuPhPqz1G
Q45ri0g=='
Decryption of the response b'\x13\xcb\x00\x00\x00\x04\x01\x01\x01['
 values of the respective coils  are: [1, 1, 0, 1, 1, 0, 1]
Enter the Question number
1. To write value in holding registers
2. To read value from holding registers
3. To write boolean in coil
4. To read boolean from coil
5. To quit the program
5
Exit

(base) C:\Users\gargi\Downloads>
```

Figure 5: Corresponding result in the simulator for writing coil



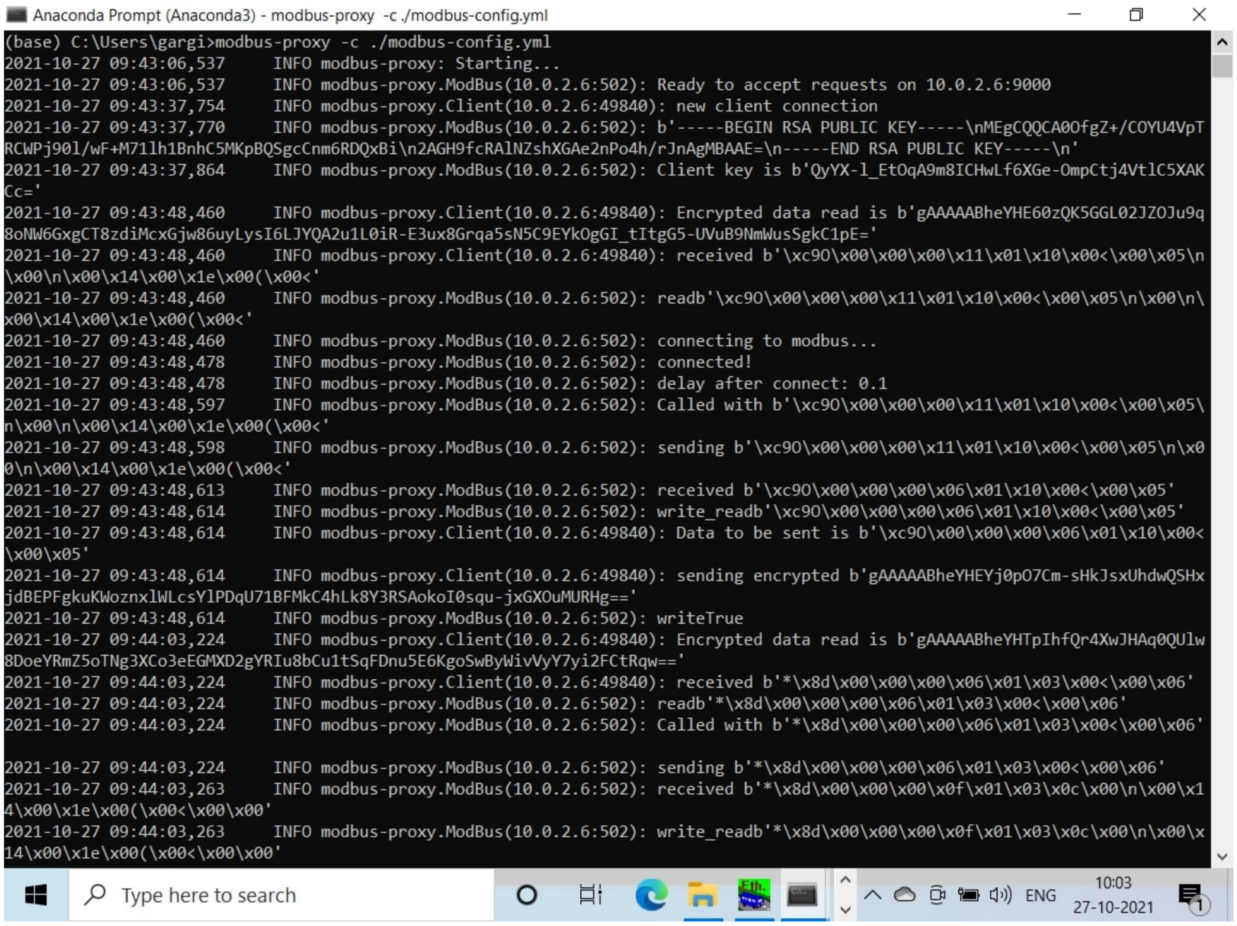
MODBUS Eth. TCP/IP PLC - Simulator (port: 502)

Connected (1/10) : (received/sent) (5/5) Serv. idle.

Address : Hex [✓ Dec] I/O Coil Outputs (000 Fmt: decimal Prot: MODBUS TCP/IP Clone

Address	+15	+14	+13	+12	+11	+10	+9	+8	+7	+6	+5	+4	+3	+2	+1	+0	Total
1-16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
17-32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
33-48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
49-64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
65-80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
81-96	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	DA00
97-112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
113-128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
129-144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
145-160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
161-176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
177-192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
193-208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
209-224	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
225-240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
241-256	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
257-272	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
273-288	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
289-304	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
305-320	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
321-336	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
337-352	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
353-368	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
369-384	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
385-400	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
401-416	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
417-432	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
433-448	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
449-464	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
465-480	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
481-496	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
497-512	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
513-528	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
529-544	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
545-560	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
561-576	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

Figure 6: Interaction among Client, Proxy-Server and the Modbus Device


Anaconda Prompt (Anaconda3) - modbus-proxy -c ./modbus-config.yml
Type here to search
ENG
10:03
27-10-2021

```
(base) C:\Users\gargi>modbus-proxy -c ./modbus-config.yml
2021-10-27 09:43:06,537     INFO modbus-proxy: Starting...
2021-10-27 09:43:06,537     INFO modbus-proxy.ModBus(10.0.2.6:502): Ready to accept requests on 10.0.2.6:9000
2021-10-27 09:43:37,754     INFO modbus-proxy.Client(10.0.2.6:49840): new client connection
2021-10-27 09:43:37,770     INFO modbus-proxy.ModBus(10.0.2.6:502): b'-----BEGIN RSA PUBLIC KEY-----\nMEgCQQCA0OfgZ+/COYU4VpT
RCWPj90l/wF+M71lh1BnhC5MKpBQSgcCnm6RDQxBi\n2AGH9fcRAlNZshXGAe2nPo4h/rJnAgMBAAE=\n-----END RSA PUBLIC KEY-----\n'
2021-10-27 09:43:37,864     INFO modbus-proxy.ModBus(10.0.2.6:502): Client key is b'QyYX-l_EtOqA9m8ICHwLf6XGe-OmpCtj4VtlC5XAK
Cc='
2021-10-27 09:43:48,460     INFO modbus-proxy.Client(10.0.2.6:49840): Encrypted data read is b'gAAAAABheYHE60zQK5GGL02JZOJu9q
8oNW6GxgCT8zdiMcxGjw86uyLysI6LJYQA2u1L0iR-E3ux8Grqa5sN5C9EYkOgGI_tItgG5-UVuB9NmWusSgkC1pE='
2021-10-27 09:43:48,460     INFO modbus-proxy.Client(10.0.2.6:49840): received b'\xc9O\x00\x00\x00\x11\x01\x10\x00<\x00\x05\n
\x00\n\x00\x14\x00\x1e\x00(\x00<'
2021-10-27 09:43:48,460     INFO modbus-proxy.ModBus(10.0.2.6:502): readb'\xc9O\x00\x00\x00\x11\x01\x10\x00<\x00\x05\n\x00\n\
x00\x14\x00\x1e\x00(\x00<'
2021-10-27 09:43:48,460     INFO modbus-proxy.ModBus(10.0.2.6:502): connecting to modbus...
2021-10-27 09:43:48,478     INFO modbus-proxy.ModBus(10.0.2.6:502): connected!
2021-10-27 09:43:48,478     INFO modbus-proxy.ModBus(10.0.2.6:502): delay after connect: 0.1
2021-10-27 09:43:48,597     INFO modbus-proxy.ModBus(10.0.2.6:502): Called with b'\xc9O\x00\x00\x00\x11\x01\x10\x00<\x00\x05\
n\x00\n\x00\x14\x00\x1e\x00(\x00<'
2021-10-27 09:43:48,598     INFO modbus-proxy.ModBus(10.0.2.6:502): sending b'\xc9O\x00\x00\x00\x11\x01\x10\x00<\x00\x05\n\x0
0\n\x00\x14\x00\x1e\x00(\x00<'
2021-10-27 09:43:48,613     INFO modbus-proxy.ModBus(10.0.2.6:502): received b'\xc9O\x00\x00\x00\x06\x01\x10\x00<\x00\x05'
2021-10-27 09:43:48,614     INFO modbus-proxy.ModBus(10.0.2.6:502): write_readb'\xc9O\x00\x00\x00\x06\x01\x10\x00<\x00\x05'
2021-10-27 09:43:48,614     INFO modbus-proxy.Client(10.0.2.6:49840): Data to be sent is b'\xc9O\x00\x00\x00\x06\x01\x10\x00<
\x00\x05'
2021-10-27 09:43:48,614     INFO modbus-proxy.Client(10.0.2.6:49840): sending encrypted b'gAAAAABheYHEYj0pO7Cm-sHkJsxUhdwQSHx
jdBEPFgkuKWoznxlWLcsYlPDqU71BFMkC4hLk8Y3RSAokoI0squ-jxGXOuMURHg=='
2021-10-27 09:43:48,614     INFO modbus-proxy.ModBus(10.0.2.6:502): writeTrue
2021-10-27 09:44:03,224     INFO modbus-proxy.Client(10.0.2.6:49840): Encrypted data read is b'gAAAAABheYHTpIhfQr4XwJHAq0QUlw
8DoeYRmZ5oTNg3XCo3eEGMXD2gYRIu8bCu1tSqFDnu5E6KgoSwByWivVyY7yi2FCtRqw=='
2021-10-27 09:44:03,224     INFO modbus-proxy.Client(10.0.2.6:49840): received b'*\x8d\x00\x00\x00\x06\x01\x03\x00<\x00\x06'
2021-10-27 09:44:03,224     INFO modbus-proxy.ModBus(10.0.2.6:502): readb'*\x8d\x00\x00\x00\x06\x01\x03\x00<\x00\x06'
2021-10-27 09:44:03,224     INFO modbus-proxy.ModBus(10.0.2.6:502): Called with b'*\x8d\x00\x00\x00\x06\x01\x03\x00<\x00\x06'

2021-10-27 09:44:03,224     INFO modbus-proxy.ModBus(10.0.2.6:502): sending b'*\x8d\x00\x00\x00\x06\x01\x03\x00<\x00\x06'
2021-10-27 09:44:03,263     INFO modbus-proxy.ModBus(10.0.2.6:502): received b'*\x8d\x00\x00\x00\x0f\x01\x03\x0c\x00\n\x00\x1
4\x00\x1e\x00(\x00<\x00\x00'
2021-10-27 09:44:03,263     INFO modbus-proxy.ModBus(10.0.2.6:502): write_readb'*\x8d\x00\x00\x00\x0f\x01\x03\x0c\x00\n\x00\x
14\x00\x1e\x00(\x00<\x00\x00'
```