

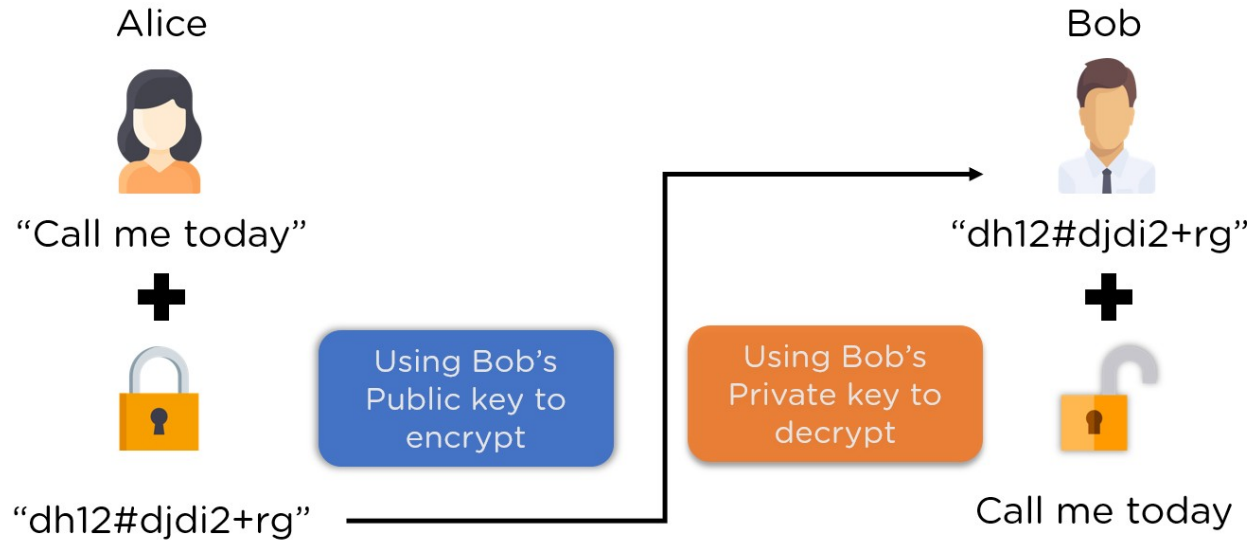
Server and Client Communication with RSA

PowerPoint Presentation in Information and Cybersecurity
By
Krishnaraj Thadesar, PA20, Batch A1

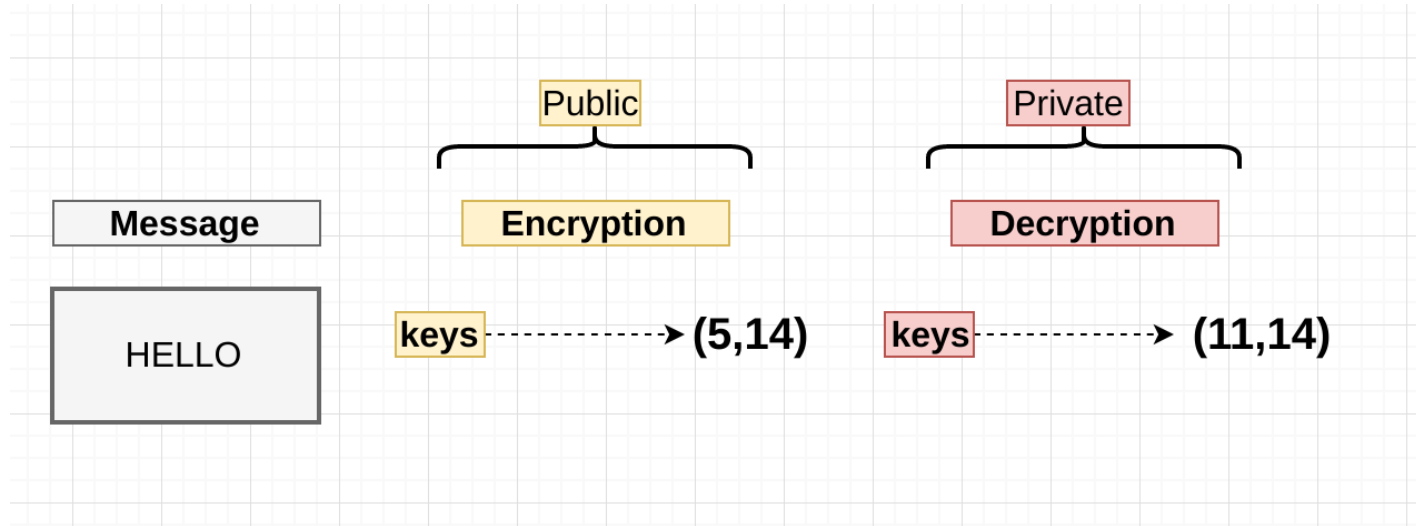
Contents

1. Introduction to Encryption on Client and Server - 3
2. Simple working of RSA - 4
3. Case study of how Whatsapp Encryption works - 7
4. Explanation of My Project - 12
5. Limitations - 13
6. Code and Outputs - 14

Basic Working of Encrypted Chats with Client and Server



Simple Working of RSA



How the Keys looks like



```
1  ———BEGIN PGP PRIVATE KEY BLOCK———
2
3  lQIGBGQ3mhsBBAC00HioVa0XhVb0Po7Gw9/LLM4K9Lvmi0djRaQdJs8kTagJGJKP
4  yQk2Dpt0oxN3FY4/HeutrKnn/EYDnuTmglXvo7Vbessfbn9h6NQ9ZyCJmln/SAQT
5  Vz14PtAXymiii+puG7QC0rIYA4acNc2kWiGoC0kyqtljAL0+5K8qxrGKhWARAQAB
6  /gcDAnqziXIUqkxp+emZF2o8uw7JRWCyLUedfNztqU+Jy8XcQJeSEhqobqM3p8X6
7  z0LJ9QHm6lr6BduvzulxBnkjuqPyCOUmJz2C7368uq2mNgYwb7w9tfYhI608wWVs
8  LCBa/f0mLVhAqCBhj26HNYkPYkt1PcKgYU/Qx2R0bDhMavNMwWxmE9IAkYSqpQjb
9  m0tcYva53lj3jnj2LmWinvvD88P2uHvd4fXGmomqIzeRK2rADICy4z8s4o72BLVL
10 zCGITg4YLChjJttxULJSobApla//ckJFNdnuJbsjNCoLrN+J8XYPmSI/C/aTwxR
11 X21wJcwf7tVn1Ps9BXY3IGZurrbGbwg9I8V/091hb5od3J5IP3CJrs0i8CTwu7/L
12 FaKWTDsL3aIpFbqdeCLReJF4fMBbe1WwECdAgYEZDeaGwEEALeH73HJ8lyR6pN1
13 I7pT+FNgutYL49S50XPyPh4jcQ3EqV4Sm96Xy6YBpFEShVokWHSBrmiMe30EGkdY
14 go8g0tKBMG1sByiPSmE/ktLChWae2EV/taWkyhNwIsrAonLRVCVo/FRR/PIBaT8
15 qWC8P6w4XIRUpKIi2QEhd+1sIGvbYUKUbkeadnDunDgyg/7gb/H20TXaL6XvMyzT
16 QjWlWBy+GMuL1fSCB3Kh1cRVHVsjdjj1V2c1zx0lilTo08PNV2X6NvDNI/6yfuRX
17 NeNUq//u3WJd1Ayq/2SE91b0M0LLKg5/42ZUoQetc6VEIcjl7c2/XQ9rH6vgY/BP
18 6A378Y1ZZo3Z0zvCyVh5NMhgKYi2BBgBCAAGFiEEseSJ53V8cfAaI9i3JiEHouMD
19 /UIFamQ3mhsCGwwACgkQJiEHouMD/UImEgP8D/hMV0JBLSWERGUGqthbsslrWtKB
20 uFnGobSJuHCjRtULzsRLcKixuLMXyJiuTmtqpjEl1i2BqhPzqz6IDojFn8Q0KsWw
21 g9Hw/cFGwTPNPgujchRWZFk0qu+BBzzgl/bS1+EpbkAD7oxkJZxZI7yjqGNTrJJ5
22 3rJ6wrgXJLQyuJs=
23 =hg+7
24  ———END PGP PRIVATE KEY BLOCK———
25
```

Server Clients usually work with End to End Encryption

A good example of that is Signal, and Whatsapp, telegram etc.



WhatsApp's E2E Encryption Protocol

WhatsApp uses open source *Signal Protocol* developed by Open Whisper Systems (They have their own messaging application, Signal). Signal Protocol uses primitives like Double Ratchet Algorithm, prekeys, Triple Diffie Hellman, Curve25519, AES and HMAC_SHA256.

What goes on in whatsapp - A simple case study

1. Session setup by Ankita using x3dh.
2. Ankita calculates master secret and Using DH Ratchet step, derives a root key and a chain key to be used by Hashing Ratchet.
3. Ankita derives a message key and next chain key using the chain key using Hashing Ratchet.
4. She encrypts her message using message key (AES256 in CBC mode).
5. Every time she sends a message her Hashing Ratchet moves forward.
6. Every time she receives a response from Bud, which includes a new public key in header, she advances her DH Ratchet, calculates a new root key and a new chain key.



45276 72562 44885 91807
54725 76248 71361 88249
12423 27573 49547 11977

To verify that messages and calls with Parth are end-to-end encrypted, scan or upload this code on their device. You can also compare the number above instead. [Learn more](#)

Scan Code

Upload Code

The QR code contains:

1. A version.
2. The user identifier for both parties.
3. The full 32-byte public Identity Key for all devices of both parties.

From Whatsapp Documentation

When either device scans the other's QR code, the keys are compared to ensure that what is in the QR code matches the Identity Key as retrieved from the server. The 60-digit number is computed by concatenating the two 30-digit numeric fingerprints for each user's device Identity Keys. To calculate a 30-digit numeric fingerprint:

1. Lexicographically sort public Identity Keys for all of the user's devices and concatenate them.
2. Iteratively SHA-512 hash the sorted Identity Keys and user identifier 5200 times.
3. Take the first 30 bytes of the final hash output.
4. Split the 30-byte result into six 5-byte chunks.
5. Convert each 5-byte chunk into 5 digits by interpreting each 5-byte chunk as a big-endian unsigned integer and reducing it modulo 100000.
6. Concatenate the six groups of five digits into thirty digits.

Our Simple Setup

- We have used RSA to encrypt and decrypt messages between the client and the server.
- Their keys are shared using Diffie Hellman learnt and executed in a previous Assignment.
- They communicate with TCP/IP protocol using the socket inbuilt library in python.
- Communication is possible on any network, as long as host IP and port are known.

Our setup limitations

1. Only 1 client can talk to the server at one time. This can be improved by tweaking the code a little bit.
2. There is no means of reconnecting.
3. We can't know if the server or client are both online or not once they go offline, so things like blue ticks and gray ticks would not work.

Code and Output

Transferring Video, or any media files is also possible this way.



```
1 # code for sending video file to client
2
3 # echo-server.py
4
5 import socket
6 import os
7
8 HOST = "127.0.0.1" # Standard loopback interface address (localhost)
9 PORT = 65432 # Port to listen on (non-privileged ports are > 1023)
10
11 # Define the file to be sent
12 file_name = '../Assignment_3/sunset.mp4'
13
14 # Create a socket object
15 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
16
17     # Bind the socket to a specific address and port number
18     s.bind((HOST, PORT))
19
20     # Listen for incoming connections
21     s.listen()
22
23     # Wait for a client to connect
24     conn, addr = s.accept()
25
26     # Send the file to the client
27     with open(file_name, 'rb') as f:
28         data = f.read(1024)
29         while data:
30             conn.send(data)
31             data = f.read(1024)
32
33     # Close the connection
34     conn.close()
```




```
1  # Code for receiving video from server
2  import socket
3
4  HOST = "127.0.0.1" # The server's hostname or IP address
5  PORT = 65432 # The port used by the server
6  # Create a socket object
7  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
8
9      # Connect to the server
10     s.connect((HOST, PORT))
11
12     # Receive the file from the server
13     with open('received_video.mp4', 'wb') as f:
14         while True:
15             data = s.recv(1024)
16             if not data:
17                 break
18             f.write(data)
19
20     # Close the connection
21     s.close()
```




```
1 def encrypt_alg(plain_text):
2     # print("encrypting: ", plain_text, type(plain_text))
3     cipher_texts = []
4     plain_texts = []
5
6     for i in plain_text:
7         # print(ord(i))
8         cipher_text = rsa_encryption(ord(i), key=server_public_key)
9         cipher_texts.append(cipher_text)
10        # print(cipher_text)
11
12    cipher_texts = [chr(i) for i in cipher_texts]
13    cipher_text = "".join(cipher_texts)
14
15    time.sleep(0.01)
16    return cipher_text
```



```
1 def decrypt_alg(cipher_text):
2     plain_texts = []
3
4     for i in cipher_text:
5         # print(ord(i))
6         plain_text = rsa_decryption(ord(i), key=private_key)
7         plain_texts.append(plain_text)
8
9     plain_texts = [chr(i) for i in plain_texts]
10    plain_text = "".join(plain_texts)
11
12    time.sleep(0.01)
13    return plain_text
```



```
1  # BEGIN CONVERSATION
2
3
4  input_thread = threading.Thread(target=send_data, args=(s,))
5  input_thread.start()
6
7  encrypt_thread = threading.Thread(target=encrypt)
8  encrypt_thread.start()
9
10 decrypt_thread = threading.Thread(target=decrypt, args=(server_name,))
11 decrypt_thread.start()
12
13
14 while texting:
15     data = s.recv(1024)
16     print(time.ctime(time.time()), ", From ", server_name)
17     print(
18         ">",
19         data.decode(encoding="utf-32").strip(" "),
20     )
21     messages_received.insert(0, data.decode(encoding="utf-32").strip(" "))
22
23 s.close()
24
```

```

1  =====>> SERVER <<=====
2  Server Started, Listening for connections ...
3  Connected by ('127.0.0.1', 55016)
4  Identify Yourself for the client: parth
5  Generating Keys
6  Our key Information
7  Private Key [77, 42593]
8  Public Key [29033, 42593]
9  Servers data {'name': 'parth', 'e': 29033, 'n': 42593}
10 Clients data {'name': 'krish', 'e': 4349, 'n': 36391}
11 Client's Key Information [4349, 36391]
12 Messages to this chat are now end to end encrypted. No one outside of this chat, Not even Mark Zuckerberg can read or listen to them.
13
14
15 > hello!
16 > Sun Apr 16 20:59:04 2023 , From krish
17 *> 你好
18 Sun Apr 16 20:59:04 2023 , From krish
19 > Hi!
20 Sun Apr 16 20:59:14 2023 , From krish
21 *> 你好 很高兴认识你 很高兴认识你 很高兴认识你
22 Sun Apr 16 20:59:14 2023 , From krish
23 > client is sending some stuff
24 server is receiving some stuff as well
25 > bye
26 Sun Apr 16 21:00:04 2023 , From krish
27 *>
28 we are done texting

```

```

1  =====>> CLIENT <<=====
2  Identify Yourself for the server: krish
3  Our key Information
4  Private Key [149, 36391]
5  Public Key [4349, 36391]
6  Servers data {'name': 'parth', 'e': 29033, 'n': 42593}
7  Server's Public Key [29033, 42593]
8  Clients data {'name': 'krish', 'e': 4349, 'n': 36391}
9  Messages to this chat are now end to end encrypted. No one outside of this chat, Not even Mark Zuckerberg can read or listen to them.
10
11
12 > Sun Apr 16 20:58:57 2023 , From parth
13 *> 你好
14 Sun Apr 16 20:58:57 2023 , From parth
15 > hello!
16 Hi!
17 > client is sending some stuff
18 > Sun Apr 16 20:59:22 2023 , From parth
19 *> 你好 很高兴认识你 很高兴认识你 很高兴认识你
20 Sun Apr 16 20:59:22 2023 , From parth
21 > server is receiving some stuff as well
22 bye

```

Thank You!