The University of the West Indies, Mona Campus

Faculty of Science and Technology, Physics Department

Net-Centric Computing (COMP2190), Computing Department

Project 2 Documentation

Name: Abegail McCalla (620157646)

Lecturer: Daniel T. Fokum, Ph.D.

Due Date: November 20, 2024

PROJECT DESCRIPTION

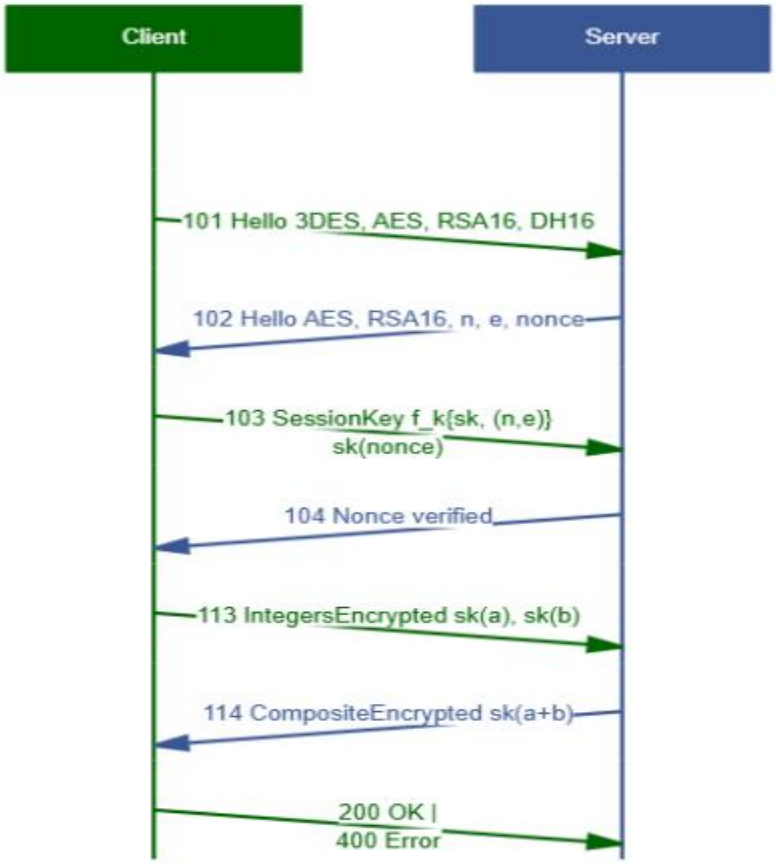

Figure 1: Messages exchanged for cryptography project

RSA encryption is computationally expensive and so it is typically used to distribute session keys for a symmetric encryption algorithm. In this assignment, the existence of two parties, a client and a server, that use public key encryption to exchange a session key that will be used to encrypt data using the simplified AES algorithm was simulated. Initially the server prompts the user to enter two prime numbers, one should be between 211 and 281 and the other between 229 and 307. The product of your numbers should be less than 65536. The server then verifies if this condition is met. If the prime numbers are not valid, the user is prompted to try again. Otherwise, the modulus n, φ(n), the private exponent d, and the public exponent e are calculated and the server is ready to accept client connection. Afterwards, the client sends a hello message to the server and indicates a list of symmetric and asymmetric algorithms that it can support. The server then responds with its own hello message, which includes one symmetric, and asymmetric algorithm that it can support from the list supplied by the client. This hello message also includes the server's public key as well as a 16-bit

pseudorandom string. The client then sends a session key message that comprises of the “103 Session Key ” string followed by a symmetric key that is encrypted with the server’s public key, this is in turn followed by the nonce encrypted with the symmetric key algorithm. The server then verifies that the received nonce matches what was sent. If there is no match, the server sends the “400 Error” message to the client and closes the connection to the client. If there is a match, the server sends the “104 Nonce Verified” message to the client. Upon receiving the “104 Nonce Verified” message, the client then prompts the user to enter two integers and sends a “113 Integers Encrypted…” message to the server, which contains the two encrypted integers. These integers are encrypted using the simplified AES algorithm, and the secret key that was exchanged using AES. The server then decrypts the two integers from the client, compute their sum, and include their symmetric key encrypted sum in the “114 Composite Encrypted…” message. The client then decrypts this encrypted sum and compares the plaintext sum with its locally computed sum. If they match, then the client returns the “200 OK” message to the server. If they do not, the client sends the “400 Error” message. Afterwards, the client computes the server’s private key and verifies this computed private key with the server. The server then compares the client’s computed server private key and the server’s actual private key and responds with the result of the comparison. Whether the keys match or not, the client then closes the socket to the server once the final message has been sent.

DESIGN TRADE-OFFS

1. **RSA Key Size:**
   - **Trade-off:** Larger RSA keys mean greater security but also greater computational overhead. The trade-off in choosing RSA key size balances security with performance. Larger keys, for example, 2048-bit or higher, can give better security and make it even more difficult to break the encryption by factorization, which is critical for protecting sensitive data against advanced attacks. This, however, results in higher computational overhead due to the larger key sizes used; it makes the encryption and decryption processes slower, which is the possible disadvantage when used for applications that demand high-speed data processing. Conversely, smaller keys such as 1024-bit remain faster and efficient, though they are far less secure, hence vulnerable against modern cryptographic attacks. The selection of appropriate RSA key size is thus a trade-off between strong security and realistic system performance and resource availability.
   - **Decision:** Balance between security and performance, using key sizes that are secure but not overly burdensome.
2. **Symmetric Key Algorithm:**
   - **Trade-off:** Being faced by the need to select a symmetric key algorithm: AES, 3DES. The selection of a symmetric key algorithm is essentially a trade-off between security, performance, and ease of implementation. While algorithms such as AES are secure and trusted by many, they are suited for very high-security applications. The drawback about AES is that it can be computational in nature, thus making systems with low processing power very slow. Less complex algorithms, or at least those requiring less computational power, might offer faster performance, but at the expense of reduced security with respect to vulnerabilities and key lengths, such as 3DES. Therefore, it would be a question of choice depending on what the application requires: better security with future proofing through AES or faster performance, probably, if a simpler algorithm is used, given the threat landscape and resource constraints.
   - **Decision:** Simplified AES is chosen for better performance and security.
3. **Message Formats:**

- **Trade-off:** Designing message formats that are easy to parse and secure. The trade-off in the design of message formats lies between clarity, security, and efficiency. Formats with structured and verbose representations, such as the usage of descriptive tags and extensive metadata, enhance clarity and have their debugging facilitated since detailed information concerning the content of a message is available. However, these increase the size of a message and processing overhead that can delay the communication process. While shorter and more compact formats enhance efficiency by lessening the amount of data that needs to be transmitted and processed, compact formats can reduce readability or ease of debugging, not to mention limit the amount of contextual information available for security checks. That means message format selection is a trade-off: clear, debuggable, and secure communications versus fast and efficient data transmission.
- **Decision:** Use clear and structured message formats like "103 Session Key" to ensure clarity and maintain security.

IMPROVEMENTS AND EXTENSIONS

1. **Enhanced Security:**
   - **Improvement:** Stronger key generation mechanisms and larger keys should be used
   - **Extension:** More advanced cryptographic techniques like Elliptic Curve Cryptography (ECC) should be used for key exchange.
2. **Error Handling:**
   - **Improvement:** Implement robust error handling and logging mechanisms. The values entered by the user on the client side could be checked to ensure that they are integers.
   - **Extension:** Develop a more detailed protocol for error messages and retries.
3. **Session Management:**
   - **Improvement:** Enhance session management by maintaining session states and handling multiple clients.
   - **Extension:** Implement session tokens and timeouts to manage active sessions securely.
4. **Integration with TLS/SSL:**
   - **Improvement:** Integrate the current protocol with existing TLS/SSL libraries to leverage their robustness.
   - **Extension:** Develop a full-fledged implementation that mimics real-world TLS/SSL handshakes and encryption.

PREVENTING ADVERSARIES FROM OBTAINING THE PRIVATE KEY

To prevent an adversary from calculating the private key given the public key, the following measures can be implemented:

- **Use Large Prime Numbers:** Ensure p and q are sufficiently large (e.g., 2048-bit primes or larger). This makes factorizing $n = p \cdot q$ computationally infeasible.
- **Use Secure Key Generation:** Generate keys using cryptographically secure random number generators to avoid predictability.
- **Regular Key Rotation:** Periodically change the keys and update the key pairs to limit the impact of any potential key compromise.
- **Secure Transmission and Storage:** Encrypt the private key and ensure it is transmitted and stored securely to prevent unauthorized access.
- **Multi-Factor Authentication:** Implement multi-factor authentication for accessing private keys, adding an extra layer of security.

Applying these methods can significantly enhance the security of the RSA algorithm and protect against adversaries attempting to derive the private key from the public key.

OPERATIONAL CODE AND COMMAND PROMPTS

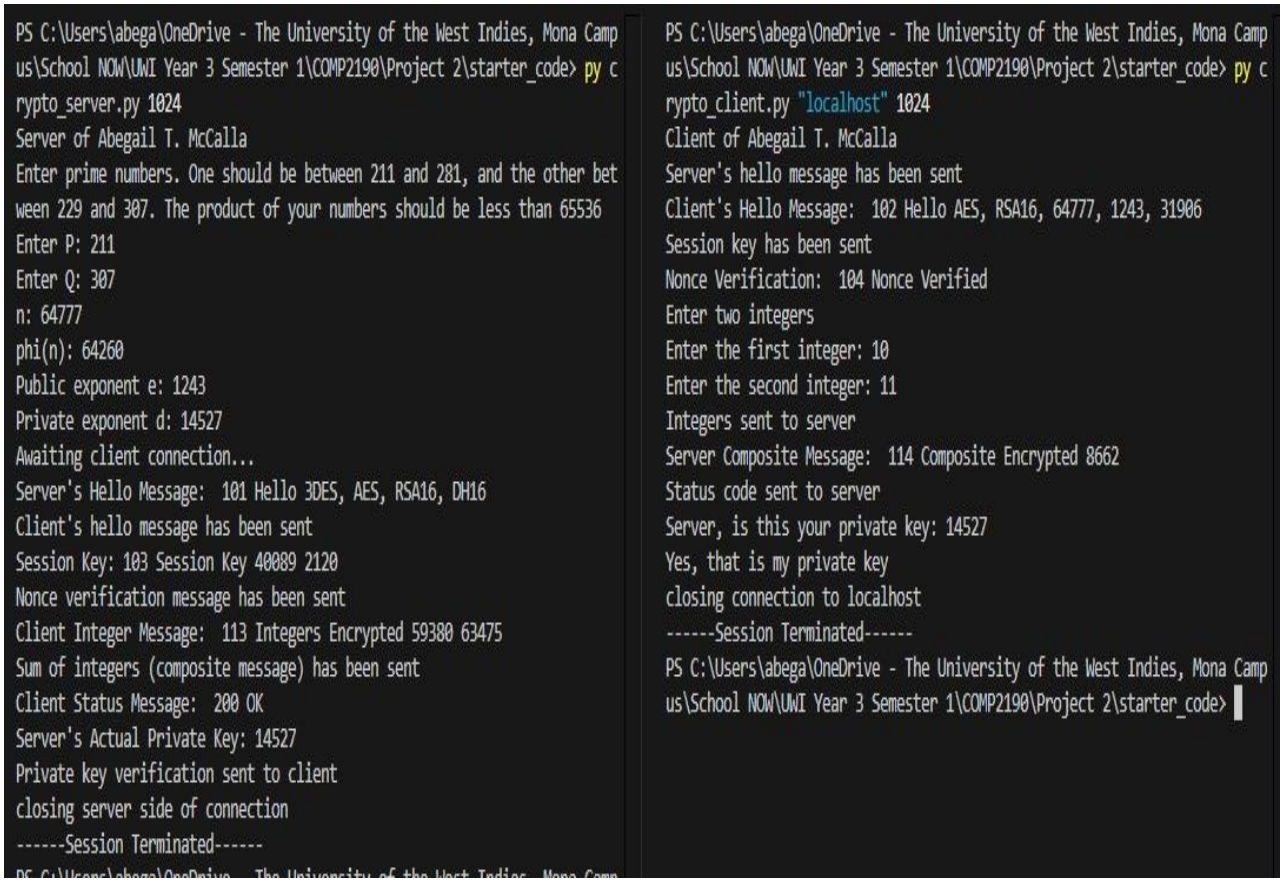

Figure 2. Completed code for part A and part C.

For part B, the command "gpg --gen-key" was ran in a Linux terminal and was used to generate the keys. I was then prompted to enter my real name, email adress and passphrase. The command "gpg --armor --export abegail.mccalla@mymona.uwi.edu > my_pubkey.asc" was used to store the generated public key as an asc file. The command "gpg --armor --export-secret-keys abegail.mccalla@mymona.uwi.edu > my_prikey.asc" was used to store the generated private key as an asc file. I then uploaded my public key to Ubuntu keyserver so that others may sign and verify it.

```
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --keyserver keyserver.ubuntu.com --send-key A5123ECB5C39D1F903B8CDAE2597E07DF3D1CD04
gpg: sending key 2597E07DF3D1CD04 to hkp://keyserver.ubuntu.com
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --keyserver keyserver.ubuntu.com --recv-key F1470F68
gpg: key EF4F9C83F1470F68: public key "Shamari McPherson <mcphersonshamari1011@gmail.com>" imported
gpg: Total number processed: 1
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --sign-key F1470F68
abegailmccalla@LAPTOP-QURKDVJQ:~$
pub  rsa4096/EF4F9C83F1470F68
     created: 2024-11-17  expires: 2028-11-17  usage: SC
     trust: unknown       validity: unknown
sub  rsa4096/24162FD9468B3DF7
     created: 2024-11-17  expires: 2028-11-17  usage: E
[ unknown] (1). Shamari McPherson <mcphersonshamari1011@gmail.com>


pub  rsa4096/EF4F9C83F1470F68
     created: 2024-11-17  expires: 2028-11-17  usage: SC
     trust: unknown       validity: unknown
 Primary key fingerprint: 21D1 8095 1A16 3C45 A1B7  A177 EF4F 9C83 F147 0F68

     Shamari McPherson <mcphersonshamari1011@gmail.com>

This key is due to expire on 2028-11-17.
Are you sure that you want to sign this key with your
key "Abegail McCalla <abegailtatyanamccalla77@gmail.com>" (175BA6D0E4D6F562)

Really sign? (y/N) y
gpg: signing failed: No secret key
gpg: signing failed: No secret key

Key not changed so no update needed.
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --check-sigs F1470F68
pub   rsa4096 2024-11-17 [SC] [expires: 2028-11-17]
      21D180951A163C45A1B7A177EF4F9C83F1470F68
uid           [ unknown] Shamari McPherson <mcphersonshamari1011@gmail.com>
sig!3        EF4F9C83F1470F68 2024-11-17  Shamari McPherson <mcphersonshamari1011@gmail.com>
sub   rsa4096 2024-11-17 [E] [expires: 2028-11-17]
sig!         EF4F9C83F1470F68 2024-11-17  Shamari McPherson <mcphersonshamari1011@gmail.com>
```

Figure 3. Part B where I uploaded my public key to the Ubuntu key server (gpg --keyserver keyserver.ubuntu.com --send-key F3D1CD04), obtained Shamari McPherson's public key (gpg --keyserver keyserver.ubuntu.com --recv-key F1470F68), signed her public key (gpg --sign-key F1470F68) then checked the signature (gpg --check-sigs F1470F68).

```
MACOWNERs-MacBook-Pro:~ macowner$ gpg --check-sigs F3D1CD04
pub   rsa3072 2024-11-19 [SC] [expires: 2026-11-19]
      A5123ECB5C39D1F903B8CDAE2597E07DF3D1CD04
uid           [  full  ] Abegail McCalla <abegail.mccalla@mymona.uwi.edu>
sig!3        2597E07DF3D1CD04 2024-11-19  Abegail McCalla <abegail.mccalla@mymona.uwi.edu>
sig!         EF4F9C83F1470F68 2024-11-19  Shamari McPherson <mcphersonshamari1011@gmail.com>
sub   rsa3072 2024-11-19 [E] [expires: 2026-11-19]
sig!         2597E07DF3D1CD04 2024-11-19  Abegail McCalla <abegail.mccalla@mymona.uwi.edu>
```

Figure 4. Part B where Shamari McPherson signed my public key.

```
gpg: 2 good signatures
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --keyserver keyserver.ubuntu.com --recv-key 7D8D0471
gpg: key A93CFCCB7D8D0471: public key "fabari williams <iamfabari2003@gmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --sign-key 7D8D0471

pub  rsa3072/A93CFCCB7D8D0471
     created: 2024-11-16  expires: 2026-11-16  usage: SC
     trust: unknown       validity: unknown
sub  rsa3072/A2E171793B317219
     created: 2024-11-16  expires: 2026-11-16  usage: E
[ unknown] (1). fabari williams <iamfabari2003@gmail.com>


pub  rsa3072/A93CFCCB7D8D0471
     created: 2024-11-16  expires: 2026-11-16  usage: SC
     trust: unknown       validity: unknown
 Primary key fingerprint: BFEF 7ECF 9BAD 291A 92E1  7406 A93C FCCB 7D8D 0471

     fabari williams <iamfabari2003@gmail.com>

This key is due to expire on 2026-11-16.
Are you sure that you want to sign this key with your
key "Abegail McCalla <abegailtatyanamccalla77@gmail.com>" (175BA6D0E4D6F562)

Really sign? (y/N) y
gpg: signing failed: No secret key
gpg: signing failed: No secret key

Key not changed so no update needed.
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --check-sigs 7D8D0471
pub   rsa3072 2024-11-16 [SC] [expires: 2026-11-16]
      BFEF7ECF9BAD291A92E17406A93CFCCB7D8D0471
uid           [ unknown] fabari williams <iamfabari2003@gmail.com>
sig!3        A93CFCCB7D8D0471 2024-11-16  fabari williams <iamfabari2003@gmail.com>
sub   rsa3072 2024-11-16 [E] [expires: 2026-11-16]
sig!         A93CFCCB7D8D0471 2024-11-16  fabari williams <iamfabari2003@gmail.com>
```

Figure 5.

```
Key not changed so no update needed.
abegailmccalla@LAPTOP-QURKDVJQ:~$ gpg --check-sigs 7D8D0471
pub   rsa3072 2024-11-16 [SC] [expires: 2026-11-16]
      BFEF7ECF9BAD291A92E17406A93CFCCB7D8D0471
uid           [ unknown] fabari williams <iamfabari2003@gmail.com>
sig!3        A93CFCCB7D8D0471 2024-11-16  fabari williams <iamfabari2003@gmail.com>
sub   rsa3072 2024-11-16 [E] [expires: 2026-11-16]
sig!         A93CFCCB7D8D0471 2024-11-16  fabari williams <iamfabari2003@gmail.com>

gpg: 2 good signatures
abegailmccalla@LAPTOP-QURKDVJQ:~$
```

Figure 6.

Figures 5 and 6. Part B where I obtained Fabari Williams' public key (gpg --keyserver keyserver.ubuntu.com --recv-key 7D8D0471), signed his public key (gpg --sign-key 7D8D0471) then checked the signature (gpg --check-sigs 7D8D0471).

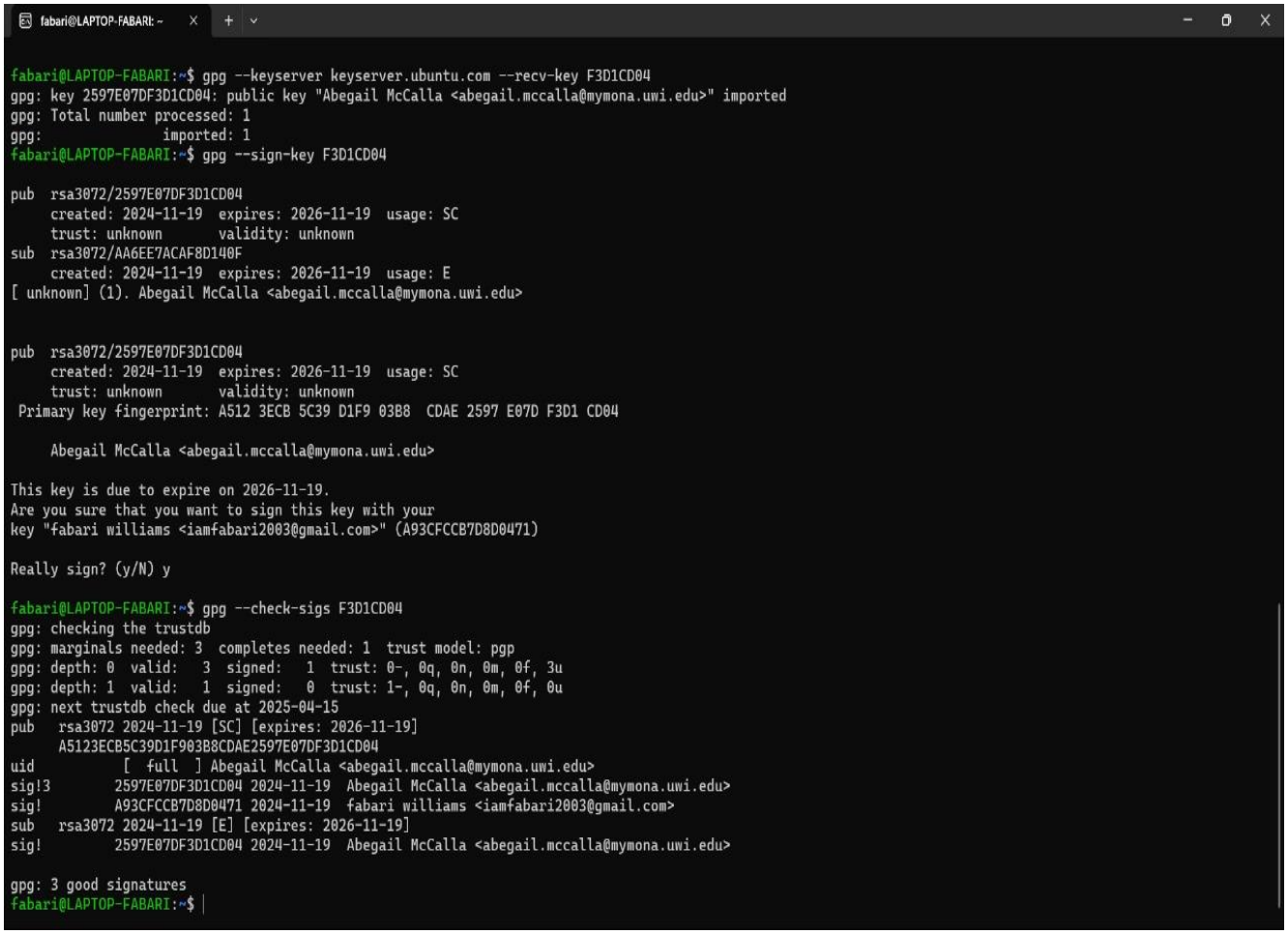

Figure 7. Part B where Fabari Williams signed my public key

Figure 8. Part B where signed keys were uploaded to Ubuntu key server (gpg --send-keys --keyserver keyserver.ubuntu.com [key id])

Search results for '0xA5123ECB5C39D1F903B8CDAE2597E07DF3D1CD04'

```
Type bits/keyID          cr. time   exp time   key expir

pub (4)rsa3072/a5123ecb5c39d1f903b8cdae2597e07df3d1cd04 2024-11-19T17:31:38Z

uid Abegail McCalla <abegail.mccalla@mymona.uwi.edu>
sig cert  2597e07df3d1cd04 2024-11-19T17:31:38Z 2026-11-19T17:31:38Z _______________ [selfsig]
sig cert  ef4f9c83f1470f68 2024-11-19T18:29:41Z _______________ _______________ ef4f9c83f1470f68
sig cert  a93cfccb7d8d0471 2024-11-19T18:50:00Z _______________ _______________ a93cfccb7d8d0471
sig cert  9541f09cac8e28cc 2024-11-19T22:09:09Z _______________ _______________ 9541f09cac8e28cc
sig cert  51de9b6623a4ad3f 2024-11-19T22:28:58Z _______________ _______________ 51de9b6623a4ad3f

sub (4)rsa3072/97393102faed02696d4c6469aa6ee7acaf8d140f 2024-11-19T17:31:38Z
sig sbind  2597e07df3d1cd04 2024-11-19T17:31:38Z _______________ 2026-11-19T17:31:38Z []
```

Figure 9. Part B where individuals who signed my public key can be seen and verified

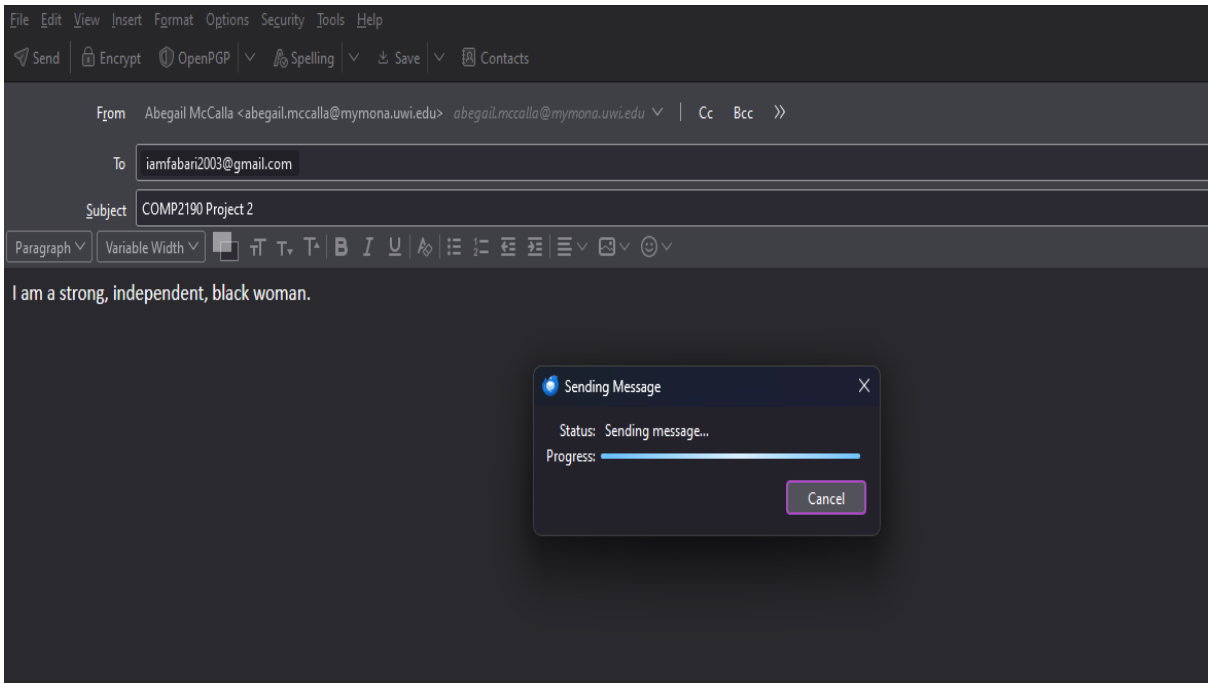

Figure 10. Part B where plain text message has been encrypted and then sent to Fabari Williams

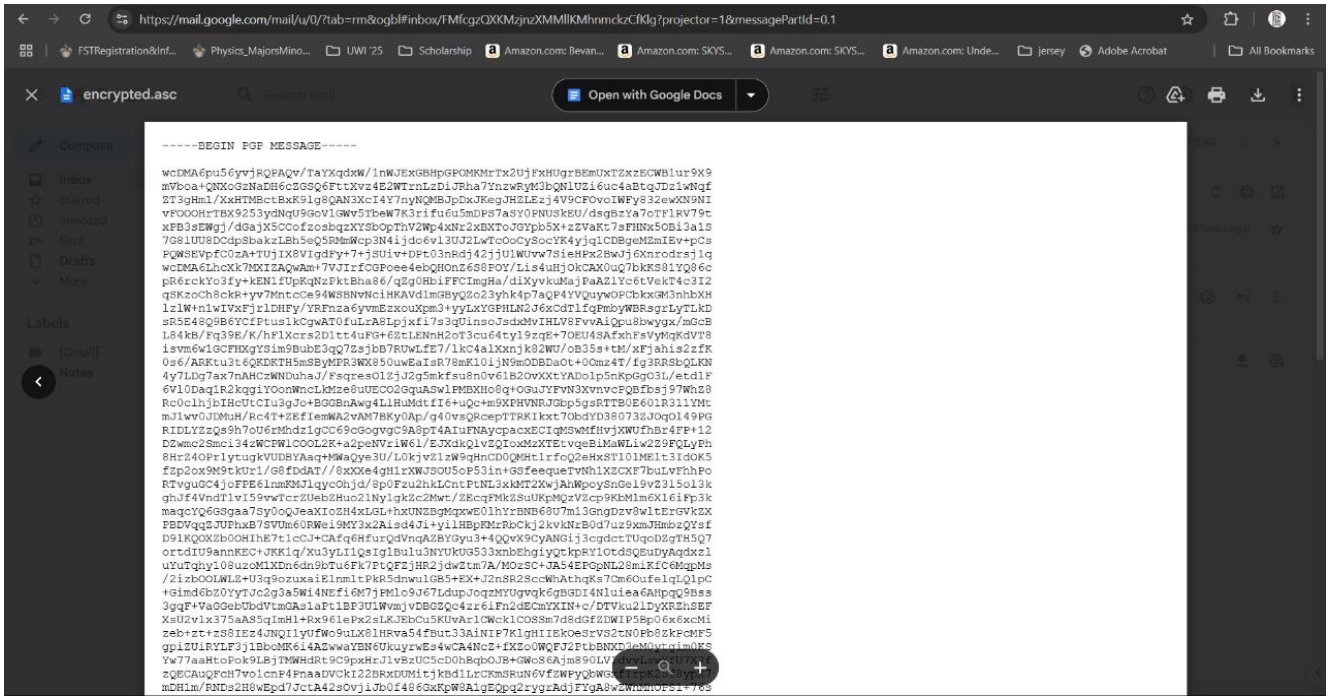

Figure 11. Part B where encrypted message was received by Fabari Williams

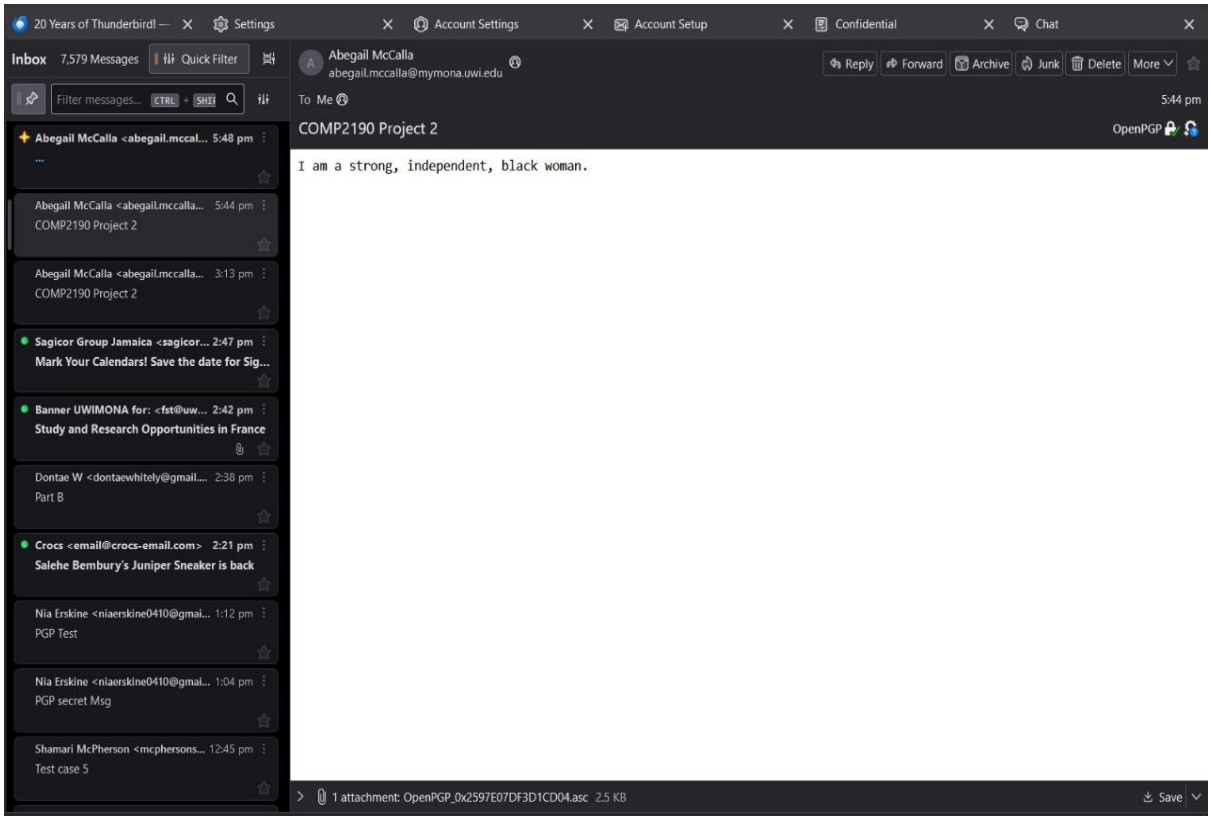

Figure 12. Part B where encrypted message has been decrypted by Fabari Williams to produce plain text

TEST CASES

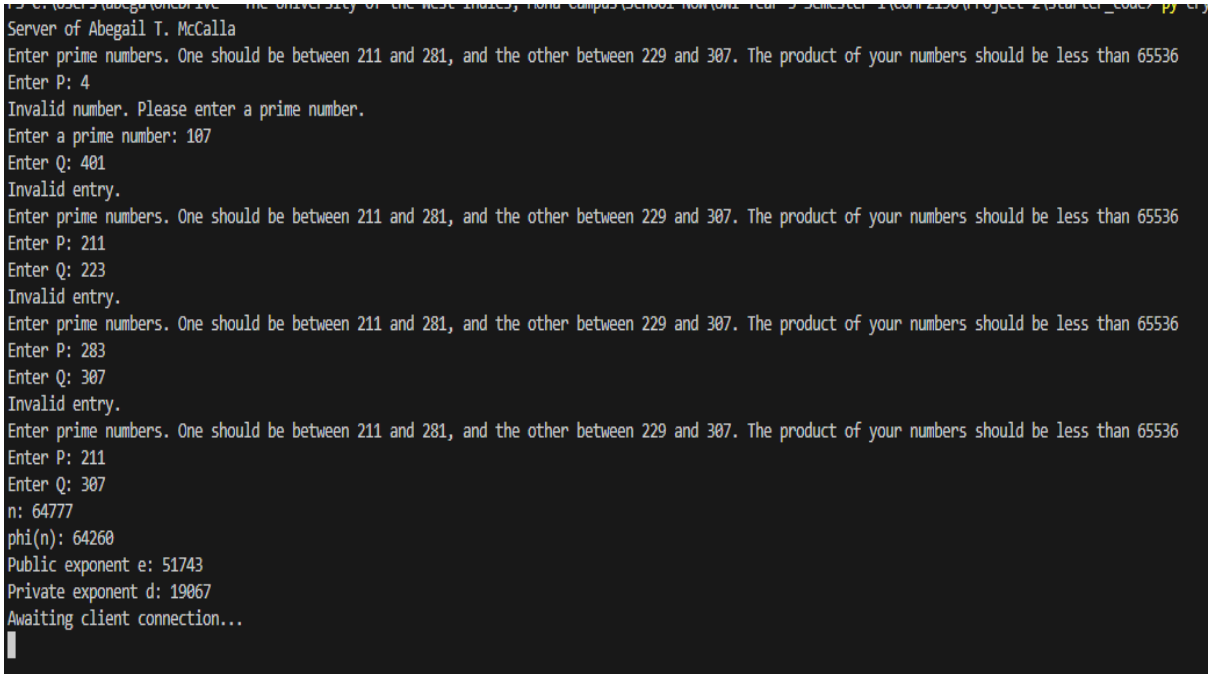

Figure 13.

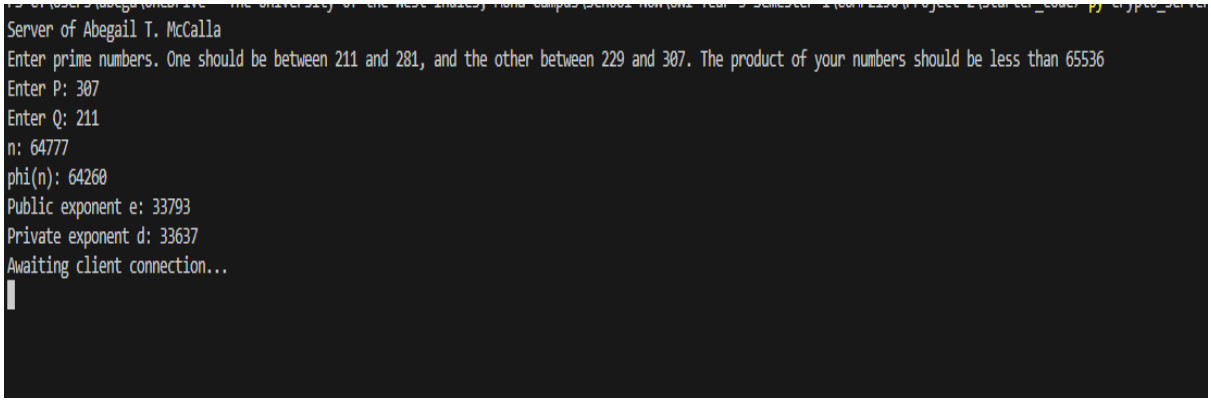

Figure 14.

Figures 13 and 14. Part A where the server was tested to ensure that values that were entered that did not meet necessary conditions were handled. If a value that is not a prime number was entered, the user is prompted to enter a prime number. If prime numbers were entered but they were not within the specified ranges that were set for the respective entries, the user is prompted to enter prime numbers that meet the specified conditions.

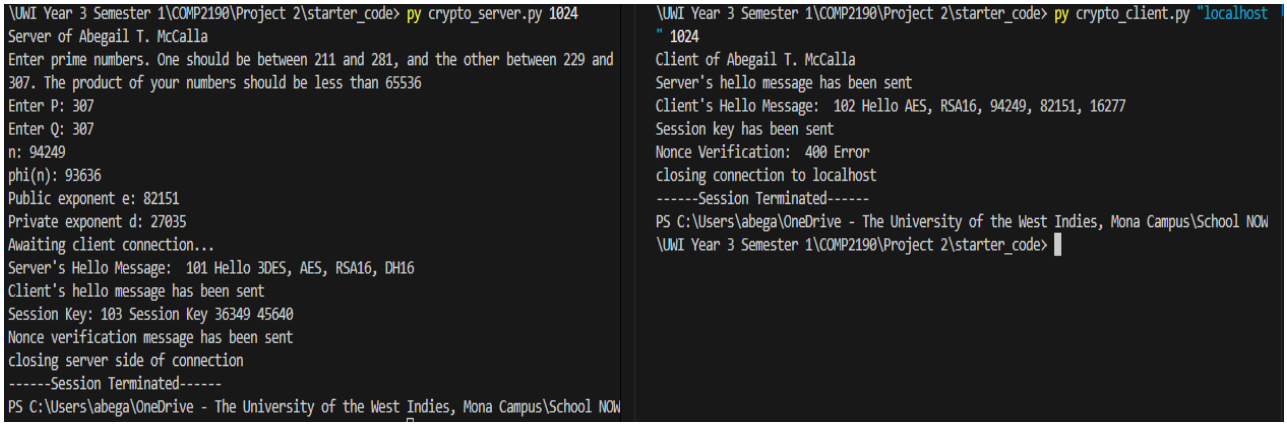

Figure 15. Part A where the nonce received by the server did not match what was sent and the "400 Error" message was sent to the client

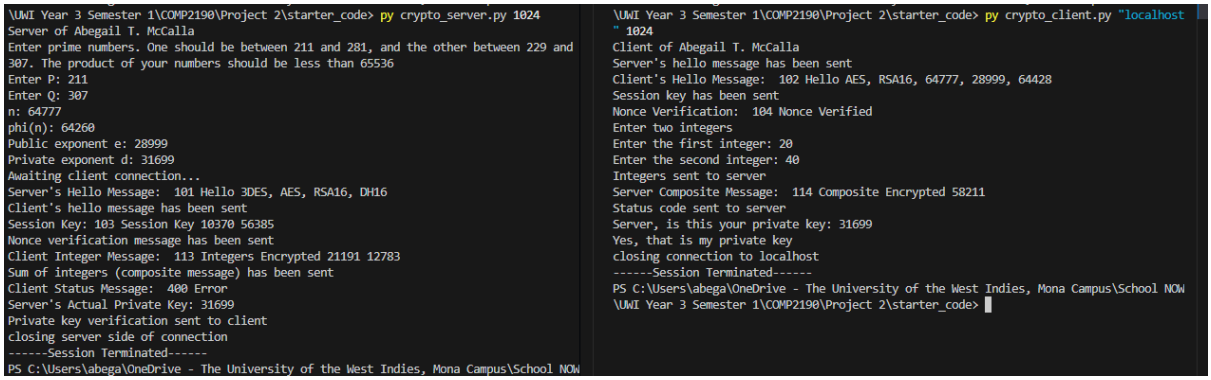

Figure 16. Part A where the decrypted composite message sum sent by the server did not match the client's calculated sum and the "400 Error" message was sent to the server

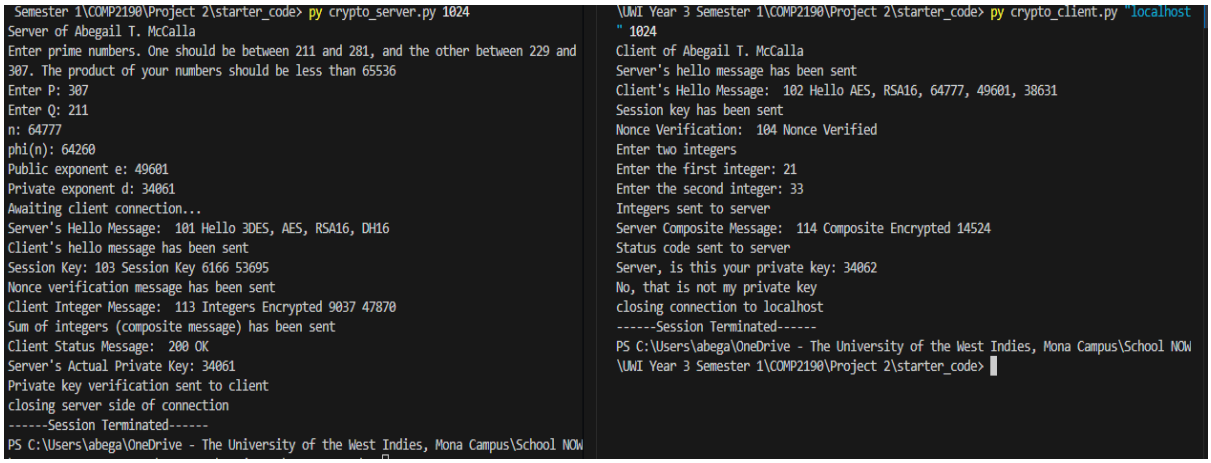

Figure 17. Part C where server's actual private key did not match client's calculated server private key and "No, that is not my private key" message was sent to client

CASES WHERE CODE IS KNOWN NOT TO WORK CORRECTLY AND ASSUMPTIONS THAT WERE MADE

There was no case that was identified where the code does not work properly as all conditions that were specified in the instructions were implemented and errors that could occur were handled accordingly, as seen in the test cases section above. It was assumed that the ranges of values that the user needs to enter are vice versa for p and q in the server. Therefore, if p is between 211 and 281, then q needs to be between 229 and 307 and if q is between 211 and 281, then p needs to be between 229 and 307.