

**CST-407 Activity 2 Data Encryption Tutorial**

Alex M. Frear

College of Science, Engineering, and Technology, Grand Canyon University

Course Number: CST-407

Professor: Dr. Melody White

08/02/2025

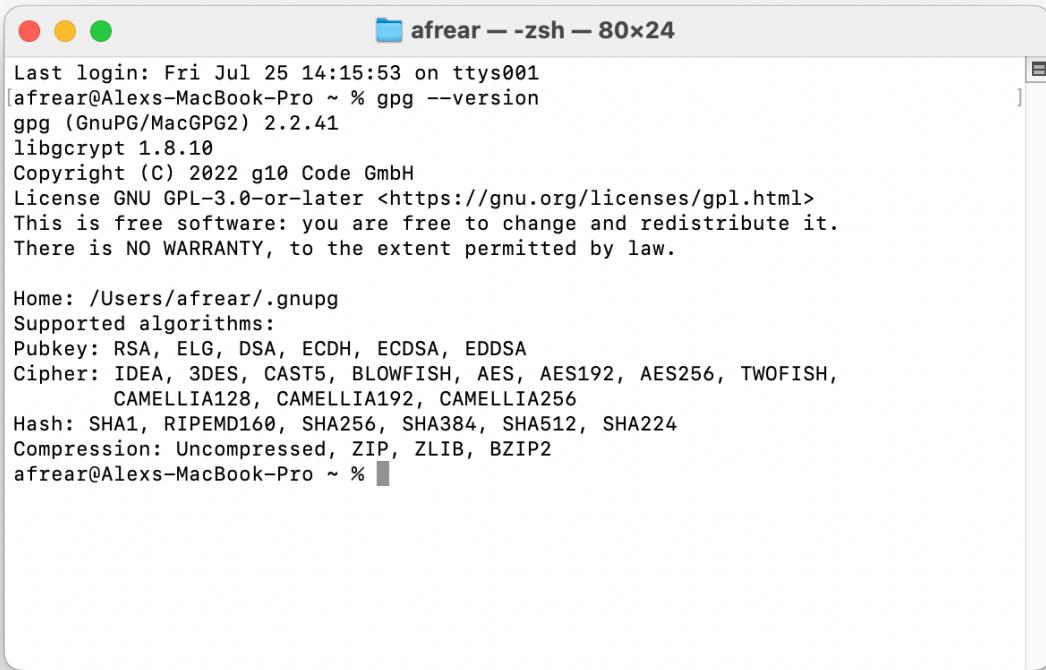
## Table of Contents

<b>Part 1: Generating and Managing GPG Keys .....</b>	<b>3</b>
<b>Screenshots .....</b>	<b>3</b>
Verified GPG Installation .....	3
GPG Keychain Generated Key .....	4
Symmetric Encryption via GPG GUI.....	5
Symmetric Encryption via CLI (ASCII Output) .....	6
ASCII Encrypted File in TextEdit.....	7
Decrypting the .asc File with GPG in Terminal .....	8
Decrypted File Confirmed in TextEdit .....	9
<b>Part 2: Signing and Decryption Verification.....</b>	<b>10</b>
<b>Screenshots .....</b>	<b>10</b>
Exporting Public Key.....	10
Importing Public Key .....	11
Encrypting File with Asymmetric Key.....	12
Decryption and Signature Verification .....	13
<b>Part 3: Asymmetric Encryption and Digital Signatures via Command Line .....</b>	<b>14</b>
<b>Screenshots .....</b>	<b>14</b>
Encrypting a Message with a Public Key .....	14
Decrypting the Encrypted Message .....	15
Confirming Decryption Output.....	16
Verifying a Digitally Signed Message.....	17
Creating a Signed and Encrypted Message .....	18
Decrypting and Verifying a Signed-Encrypted Message.....	19
<b>Part 4: Symmetric Encryption .....</b>	<b>20</b>
<b>Screenshots .....</b>	<b>20</b>
Creating the Symmetric Message File .....	20
Encrypting the Message with a Passphrase .....	21
Decrypted File Output and File List View .....	22
Verifying the Decrypted Content Matches Original .....	23
<b>Part 5: RSA Java Encryption Program .....</b>	<b>24</b>
<b>Screenshots .....</b>	<b>24</b>
Running the RSA Encryption Application .....	24
<b>Summary and Reflection .....</b>	<b>25</b>

# Part 1: Generating and Managing GPG Keys

## Screenshots

### Verified GPG Installation



The screenshot shows a terminal window titled "afrear -- zsh -- 80x24". The window contains the following text output from the "gpg --version" command:

```
Last login: Fri Jul 25 14:15:53 on ttys001
[afrear@Alexs-MacBook-Pro ~ % gpg --version
gpg (GnuPG/MacGPG2) 2.2.41
libgcrypt 1.8.10
Copyright (C) 2022 g10 Code GmbH
License GNU GPL-3.0-or-later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /Users/afrear/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
afrear@Alexs-MacBook-Pro ~ %
```

Figure 1 Verified GPG is installed and functioning on macOS.

## GPG Keychain Generated Key

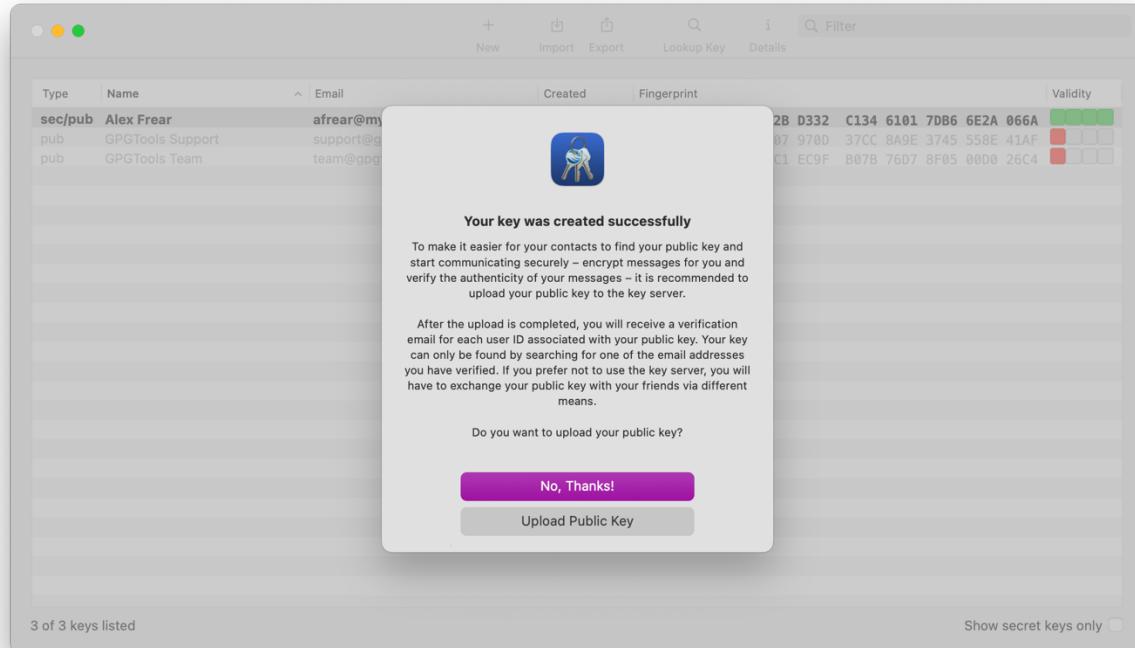


Figure 2 GPG Keychain showing newly created RSA key pair.

## Symmetric Encryption via GPG GUI

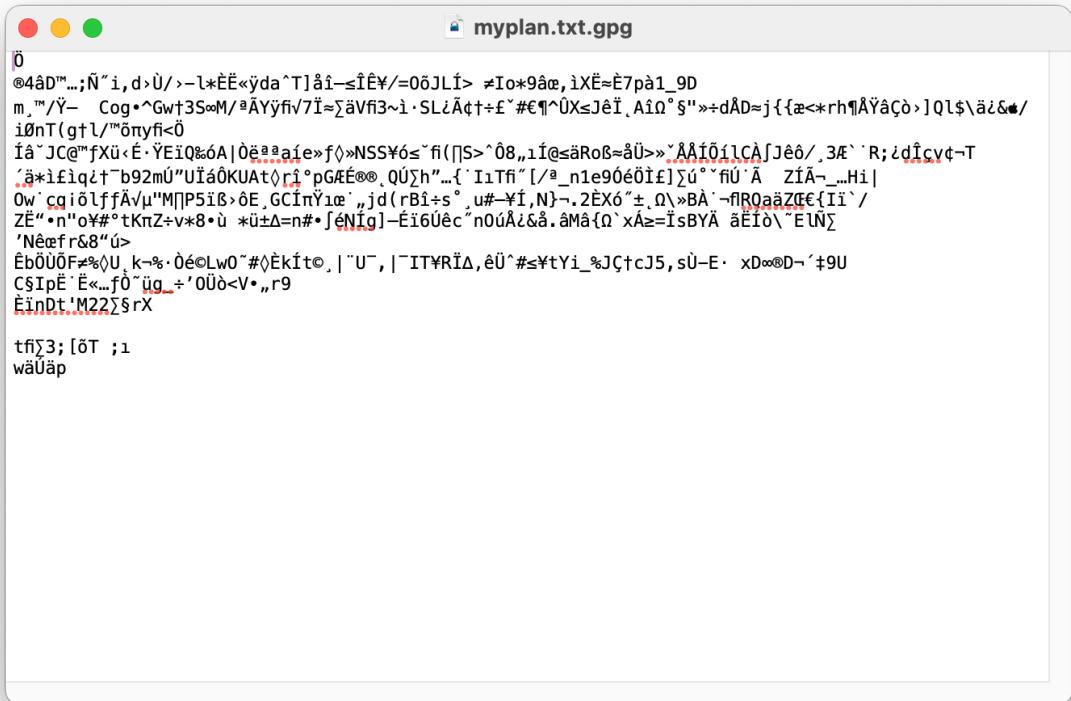
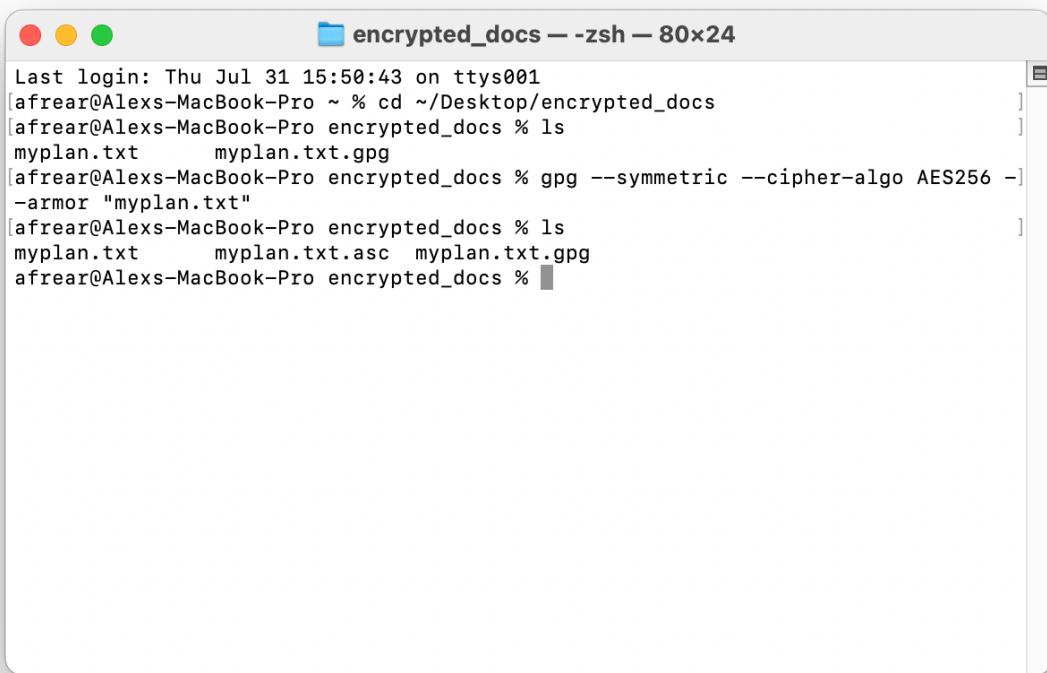


Figure 3 Encrypted file `myplan.txt.gpg` opened in TextEdit showing unreadable encrypted data.

## Symmetric Encryption via CLI (ASCII Output)



A screenshot of a macOS terminal window titled "encrypted\_docs -- zsh -- 80x24". The window shows the following command-line session:

```
Last login: Thu Jul 31 15:50:43 on ttys001
[afrear@Alexs-MacBook-Pro ~ % cd ~/Desktop/encrypted_docs
[afrear@Alexs-MacBook-Pro encrypted_docs % ls
myplan.txt      myplan.txt.gpg
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --symmetric --cipher-algo AES256 -
-armor "myplan.txt"
[afrear@Alexs-MacBook-Pro encrypted_docs % ls
myplan.txt      myplan.txt.asc  myplan.txt.gpg
afrear@Alexs-MacBook-Pro encrypted_docs %
```

Figure 4 Terminal showing encryption command and resulting `asc` file using `--armor`.

## ASCII Encrypted File in TextEdit

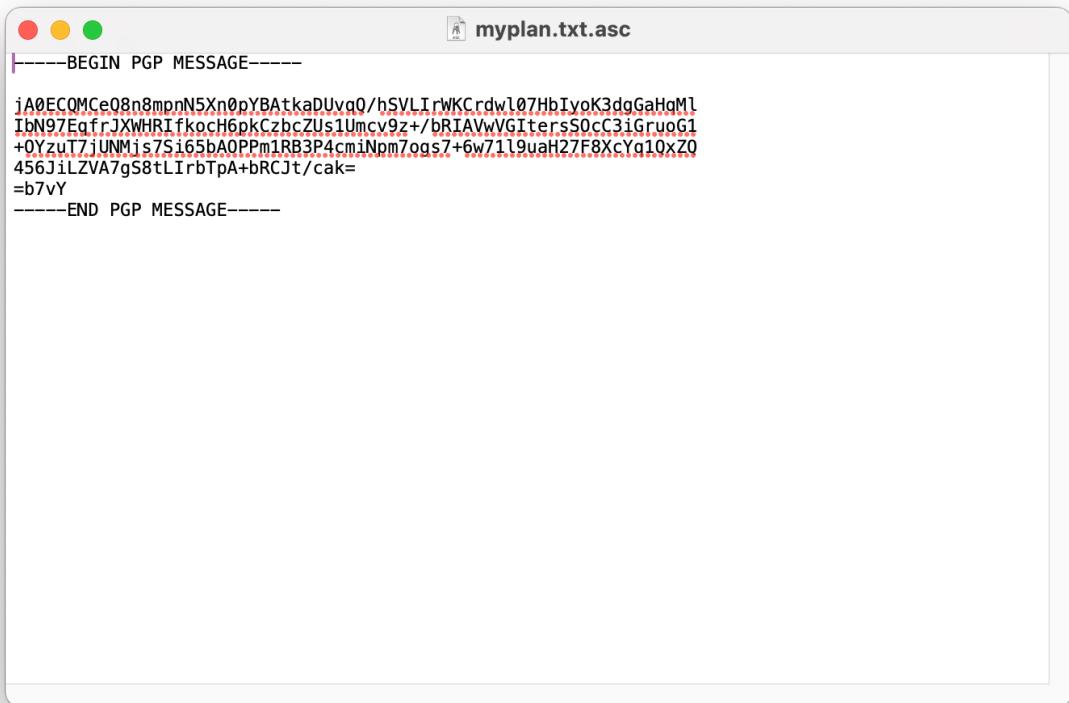
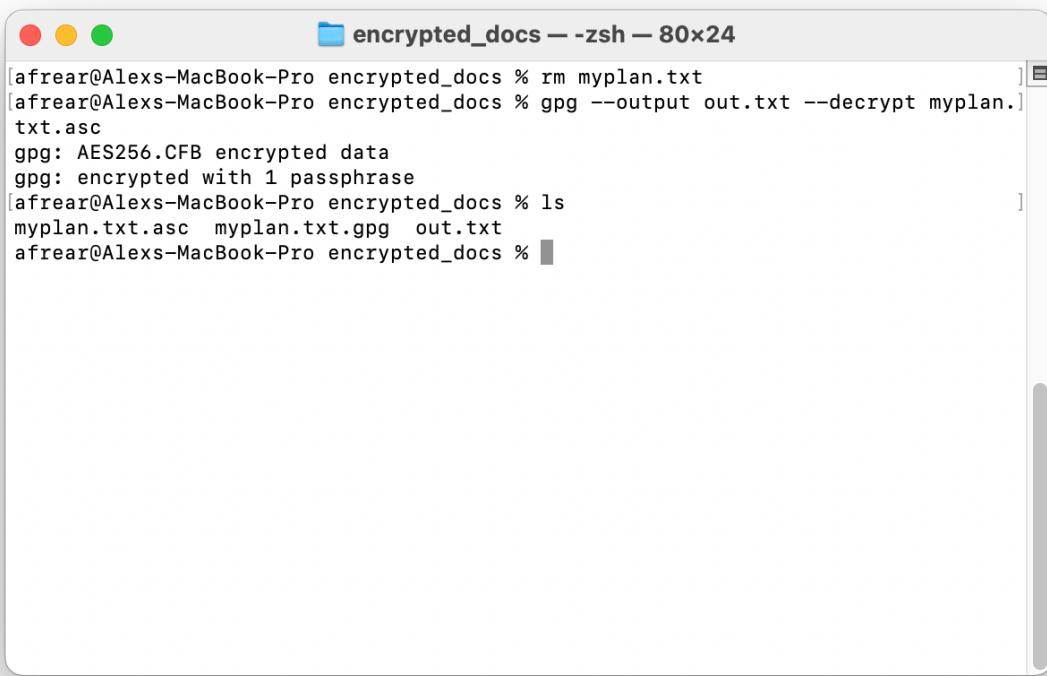


Figure 5 Encrypted `asc` file opened inTextEdit showing ASCII armored content.

## Decrypting the .asc File with GPG in Terminal



The screenshot shows a macOS Terminal window with the title bar "encrypted\_docs -- zsh -- 80x24". The terminal session is as follows:

```
[afrear@Alexs-MacBook-Pro encrypted_docs % rm myplan.txt
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --output out.txt --decrypt myplan.]
txt.asc
gpg: AES256.CFB encrypted data
gpg: encrypted with 1 passphrase
[afrear@Alexs-MacBook-Pro encrypted_docs % ls
myplan.txt.asc myplan.txt.gpg out.txt
afrear@Alexs-MacBook-Pro encrypted_docs % ]
```

Figure 6 Terminal decrypting `myplan.txt.asc` into `out.txt` using GPG.

## Decrypted File Confirmed in TextEdit



Figure 7 Decrypted `out.txt` opened in TextEdit showing restored original message.

## Part 2: Signing and Decryption Verification

### Screenshots

#### Exporting Public Key

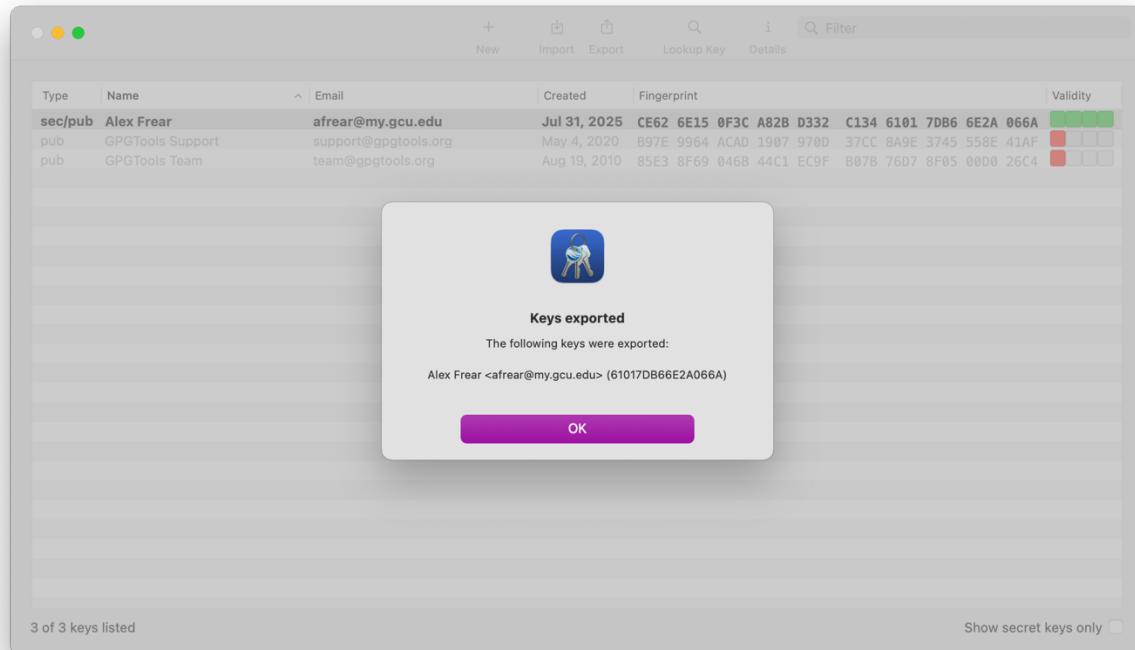


Figure 8 Export confirmation for the newly created GPG public key.

## Importing Public Key

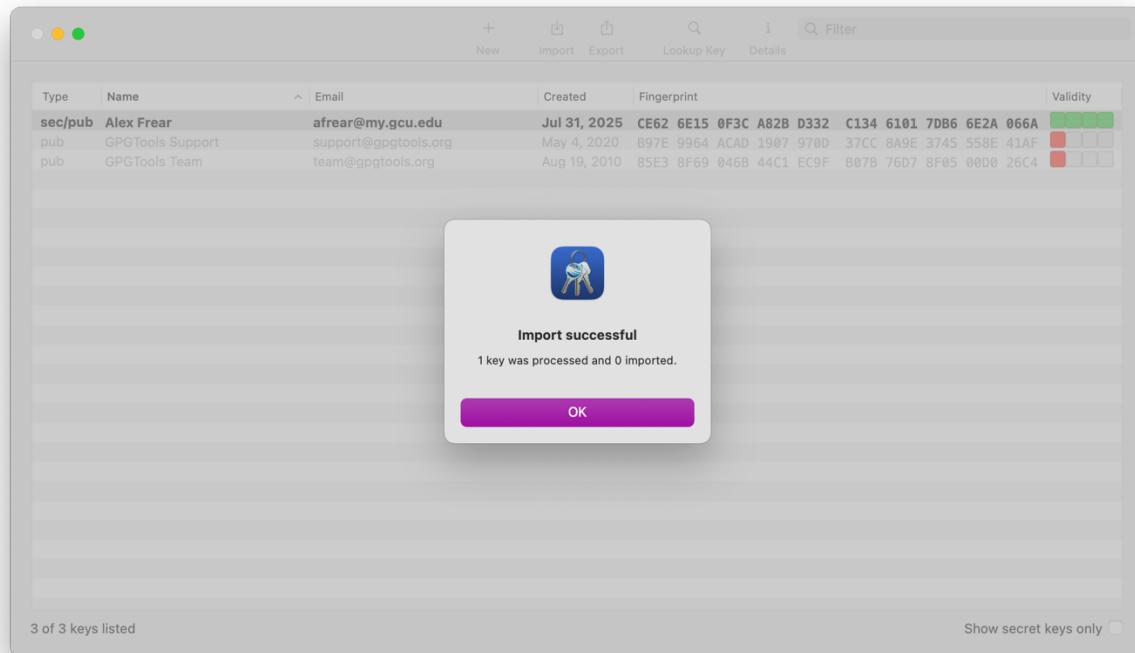


Figure 9 Successful import confirmation of a public key into GPG Keychain.

## Encrypting File with Asymmetric Key

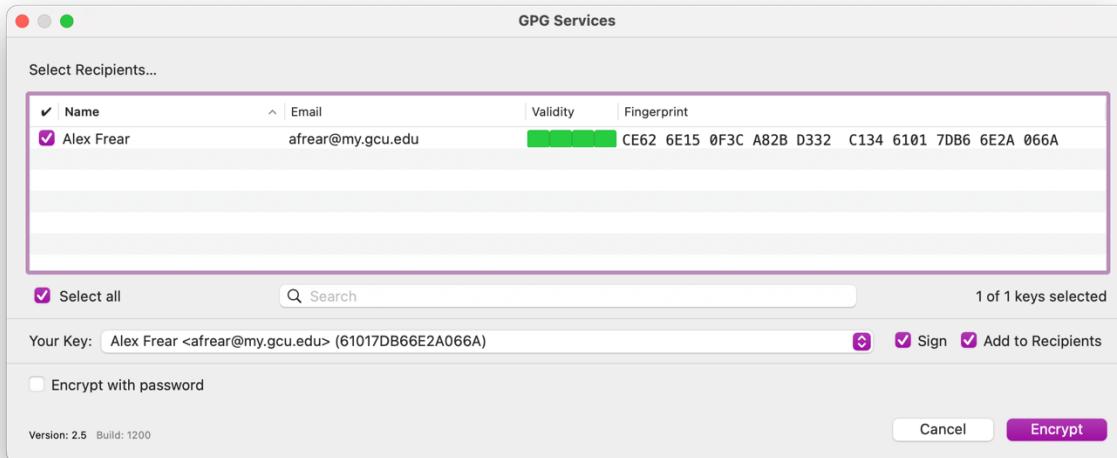


Figure 10 Encrypting a file using GPG with recipient selected and signing enabled.

## Decryption and Signature Verification

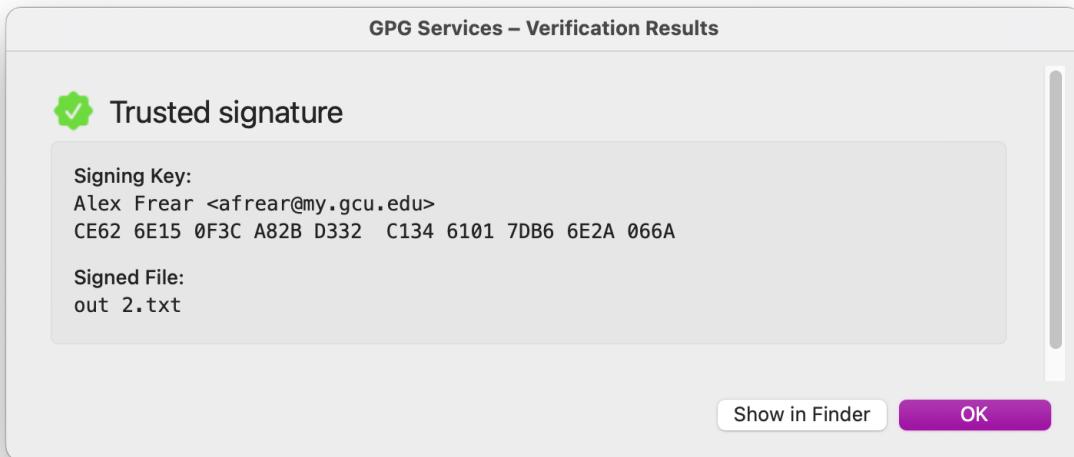


Figure 11 GPG confirms the file was decrypted and signed by a trusted key.

## Part 3: Asymmetric Encryption and Digital Signatures via Command Line

### Screenshots

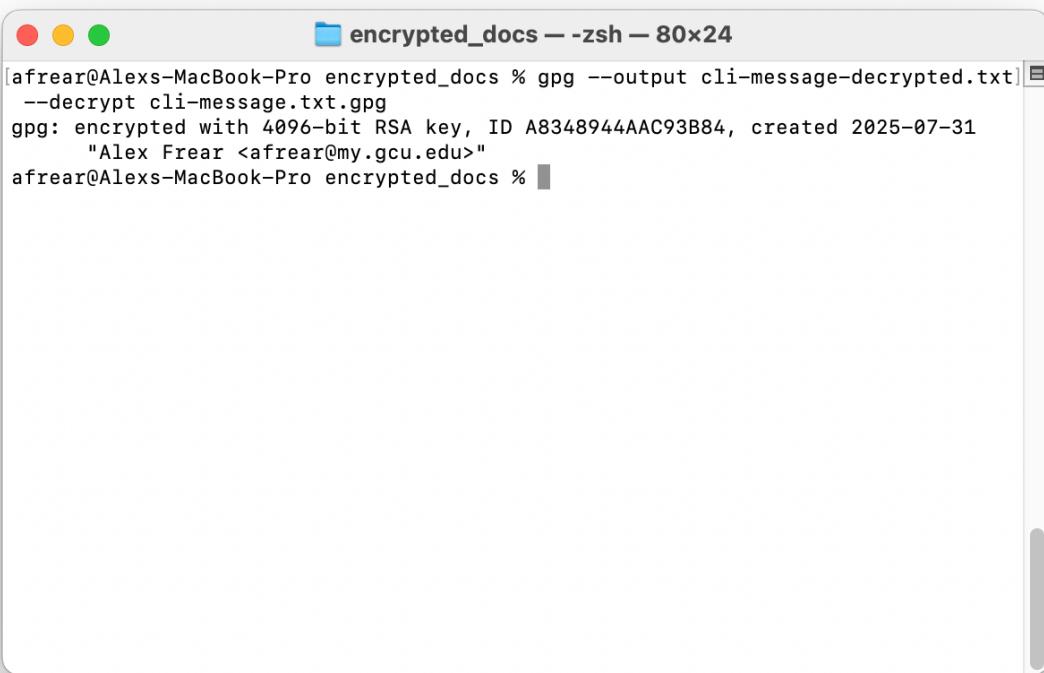
#### Encrypting a Message with a Public Key



```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --encrypt --recipient afrear@my.gc[u.edu cli-message.txt
[afrear@Alexs-MacBook-Pro encrypted_docs % ls -l
total 64
-rw-r--r--@ 1 afrear  staff  3143 Jul 31 16:56 afrear-public-key.asc
-rw-r--r--@ 1 afrear  staff    53 Jul 31 17:40 cli-message.txt
-rw-r--r--  1 afrear  staff   652 Jul 31 17:41 cli-message.txt.gpg
-rw-r--r--@ 1 afrear  staff   289 Jul 31 16:28 myplan.txt.asc
-rw-----@ 1 afrear  staff   733 Jul 31 16:23 myplan.txt.gpg
-rw-----@ 1 afrear  staff    97 Jul 31 17:05 out 2.txt
-rw-r--r--@ 1 afrear  staff    97 Jul 31 16:33 out.txt
-rw-----@ 1 afrear  staff  1278 Jul 31 17:04 out.txt.gpg
afrear@Alexs-MacBook-Pro encrypted_docs % ]
```

Figure 12 Encrypting `cli-message.txt` using the recipient's public key with GPG.

## Decrypting the Encrypted Message



The screenshot shows a terminal window titled "encrypted\_docs — -zsh — 80x24". The command entered is:

```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --output cli-message-decrypted.txt  
--decrypt cli-message.txt.gpg  
gpg: encrypted with 4096-bit RSA key, ID A8348944AAC93B84, created 2025-07-31  
    "Alex Frear <afrear@my.gcu.edu>"  
afrear@Alexs-MacBook-Pro encrypted_docs %
```

Figure 13 Decrypting the encrypted file `cli-message.txt.gpg` to produce `cli-message-decrypted.txt`.

## Confirming Decryption Output



The screenshot shows a terminal window on a Mac OS X desktop. The window title is "encrypted\_docs — -zsh — 80x24". The terminal session shows the following commands and output:

```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --output cli-message-decrypted.txt  
--decrypt cli-message.txt.gpg  
gpg: encrypted with 4096-bit RSA key, ID A8348944AAC93B84, created 2025-07-31  
      "Alex Frear <afrear@my.gcu.edu>"  
[afrear@Alexs-MacBook-Pro encrypted_docs % cat cli-message-decrypted.txt  
This is a test of asymmetric encryption via terminal.  
afrear@Alexs-MacBook-Pro encrypted_docs % ]
```

Figure 14 Viewing the contents of `cli-message-decrypted.txt` in the terminal to confirm successful decryption.

## Verifying a Digitally Signed Message



The screenshot shows a terminal window titled "encrypted\_docs -- -zsh -- 80x24". The window contains the following text:

```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --output cli-signed-message.txt.gpg --sign cli-message.txt  
afrear@Alexs-MacBook-Pro encrypted_docs % gpg --verify cli-signed-message.txt.gpg  
gpg: Signature made Thu Jul 31 17:50:02 2025 MST  
gpg:                 using RSA key CE626E150F3CA82BD332C13461017DB66E2A066A  
gpg: Good signature from "Alex Frear <afrear@my.gcu.edu>" [ultimate]  
afrear@Alexs-MacBook-Pro encrypted_docs %
```

Figure 15 Verifying the digital signature on *cli-signed-message.txt.gpg* using GPG.

## Creating a Signed and Encrypted Message

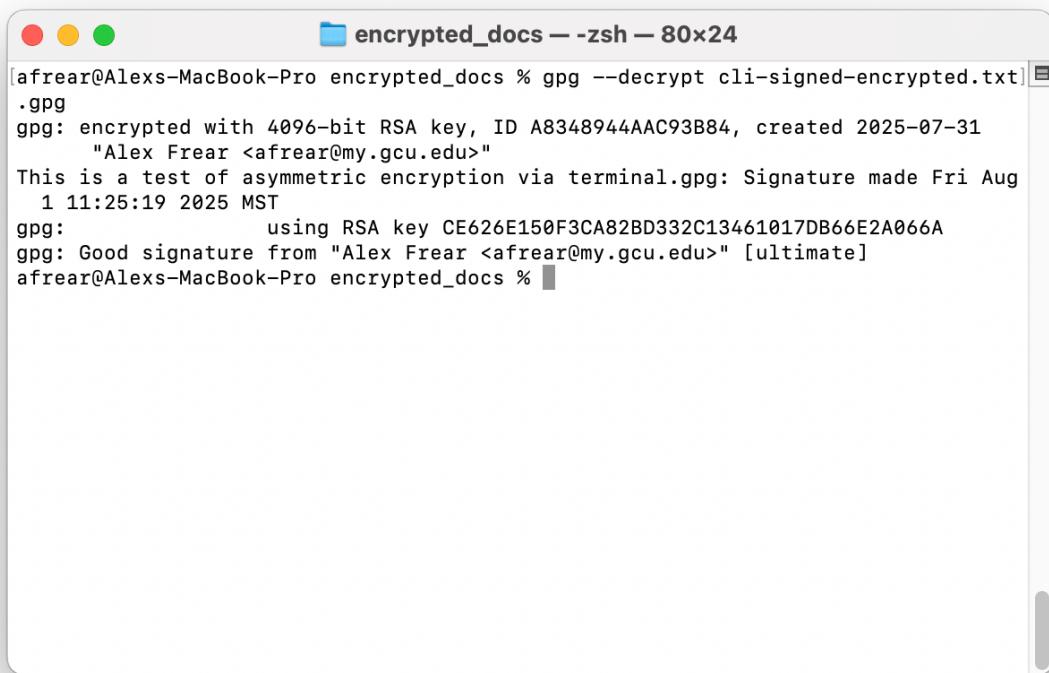


A screenshot of a terminal window titled "encrypted\_docs — -zsh — 80x24". The window shows a single command being run:

```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --encrypt --sign --recipient afrear@my.gcu.edu --output cli-signed-encrypted.txt.gpg cli-message.txt  
afrear@Alexs-MacBook-Pro encrypted_docs % ]
```

Figure 16 Encrypting and signing *cli-message.txt* in a single step, outputting *cli-signed-encrypted.txt.gpg*.

## Decrypting and Verifying a Signed-Encrypted Message



A screenshot of a terminal window titled "encrypted\_docs — -zsh — 80x24". The window shows the command "gpg --decrypt cli-signed-encrypted.txt.gpg" being run. The output indicates that the file was encrypted with a 4096-bit RSA key, ID A8348944AAC93B84, created on 2025-07-31 by "Alex Frear <afrear@my.gcu.edu>". It also shows a good signature from the same person using an RSA key with ID CE626E150F3CA82BD332C13461017DB66E2A066A. The terminal prompt "afrear@Alexs-MacBook-Pro encrypted\_docs %" is visible at the bottom.

```
[afrear@Alexs-MacBook-Pro encrypted_docs % gpg --decrypt cli-signed-encrypted.txt.gpg
gpg: encrypted with 4096-bit RSA key, ID A8348944AAC93B84, created 2025-07-31
      "Alex Frear <afrear@my.gcu.edu>"
This is a test of asymmetric encryption via terminal.gpg: Signature made Fri Aug
1 11:25:19 2025 MST
gpg:                 using RSA key CE626E150F3CA82BD332C13461017DB66E2A066A
gpg: Good signature from "Alex Frear <afrear@my.gcu.edu>" [ultimate]
afrear@Alexs-MacBook-Pro encrypted_docs %
```

Figure 17 Decrypting and verifying the signature of *cli-signed-encrypted.txt.gpg*.

# Part 4: Symmetric Encryption

## Screenshots

### Creating the Symmetric Message File

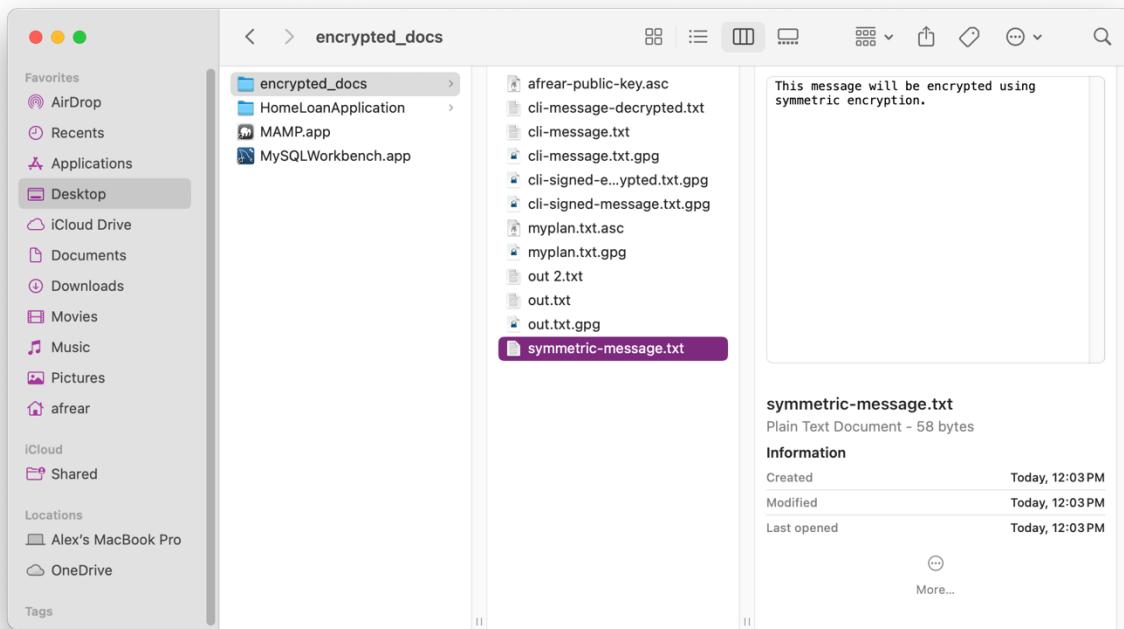


Figure 18 Plain text file `symmetric-message.txt` created using `TextEdit`. This file contains the message to be encrypted using symmetric encryption.

## Encrypting the Message with a Passphrase



A screenshot of a terminal window titled "encrypted\_docs — -zsh — 80x24". The window shows a command being run: "afrear@Alexs-MacBook-Pro encrypted\_docs % gpg --symmetric --output symmetric-message.txt.gpg symmetric-message.txt". The command is completed successfully, as indicated by the prompt "afrear@Alexs-MacBook-Pro encrypted\_docs %". The terminal has a dark theme with red, yellow, and green status indicators in the top-left corner.

Figure 19 GPG symmetric encryption command executed. The file `symmetric-message.txt.gpg` was successfully created using a passphrase.

## Decrypted File Output and File List View

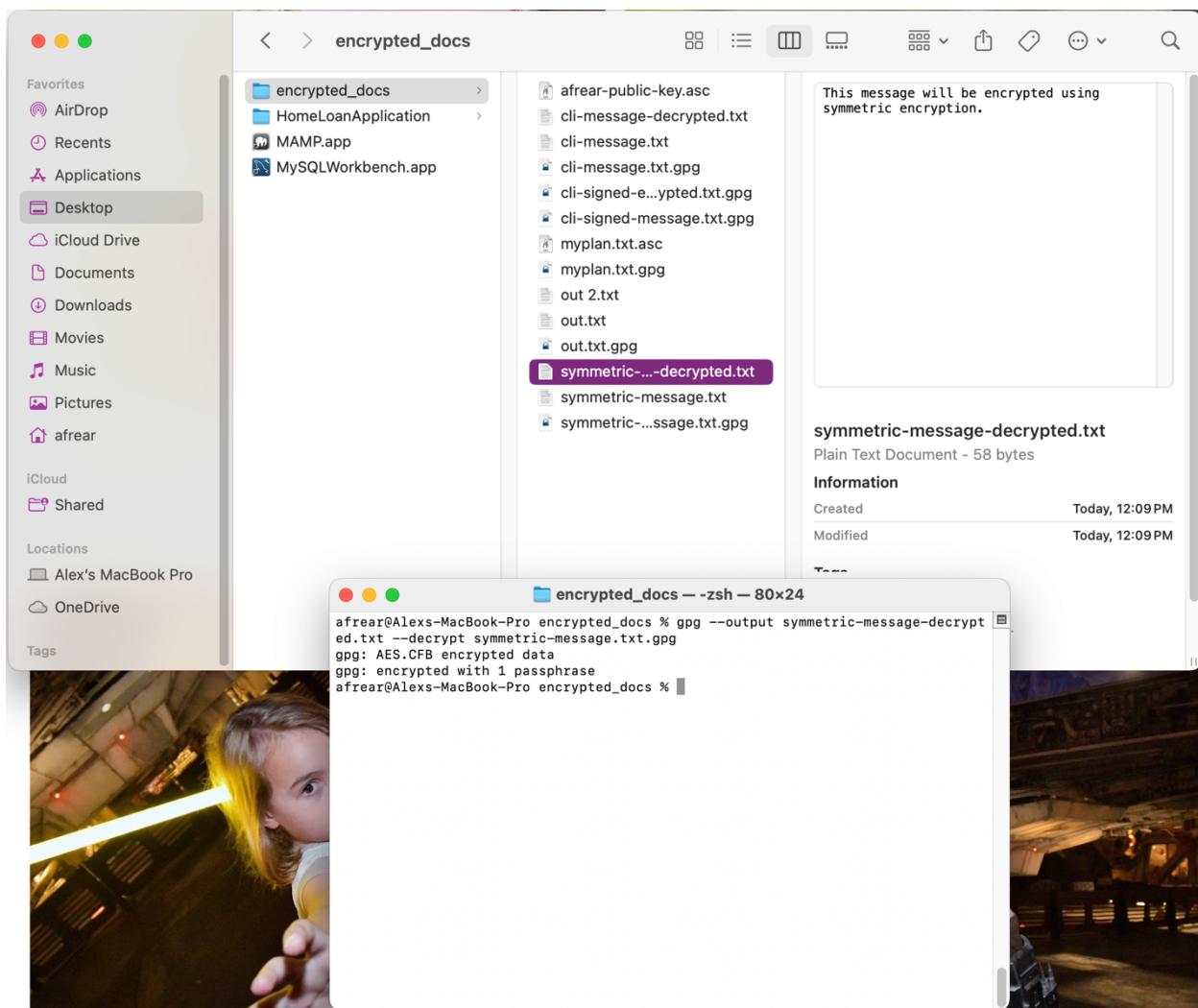


Figure 20 Terminal output confirming successful decryption of the symmetric file. The decrypted file `symmetric-message-decrypted.txt` appears alongside the original and encrypted versions in Finder.

## Verifying the Decrypted Content Matches Original



The screenshot shows a macOS terminal window titled "encrypted\_docs — -zsh — 80x24". The window contains the following text:

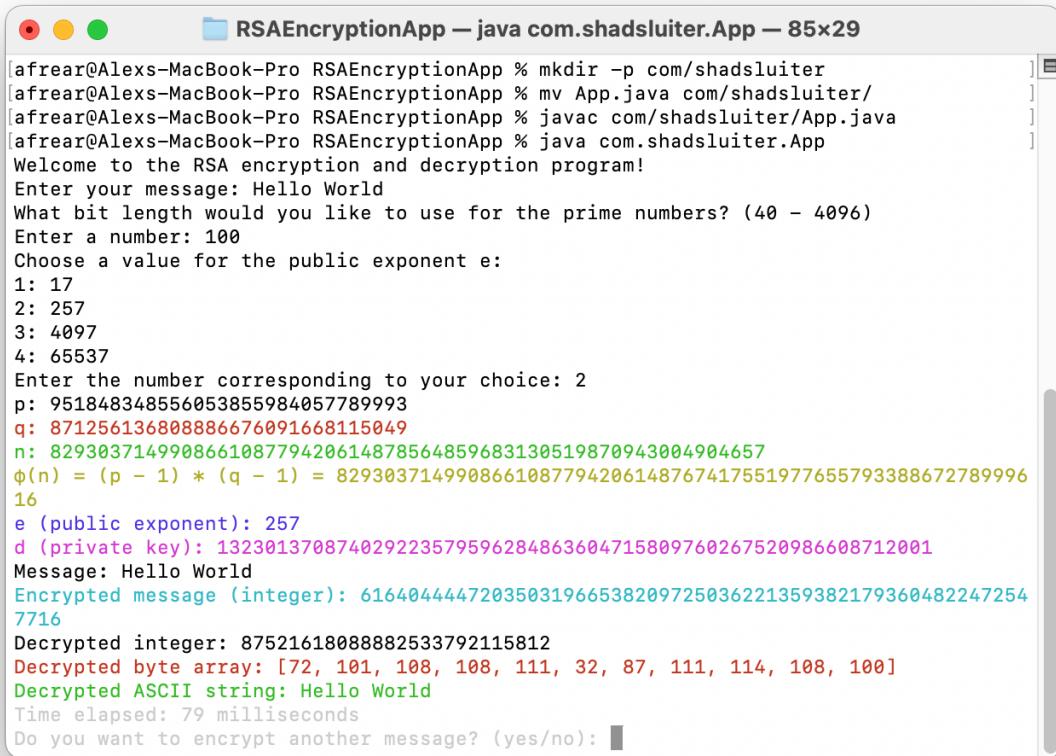
```
[afrear@Alexs-MacBook-Pro encrypted_docs % diff symmetric-message.txt symmetric-m  
essage-decrypted.txt  
afrear@Alexs-MacBook-Pro encrypted_docs % ]
```

Figure 21 *diff* command run to compare the original and decrypted files. No output confirms both files are identical, verifying successful symmetric encryption and decryption.

## Part 5: RSA Java Encryption Program

### Screenshots

#### Running the RSA Encryption Application



The screenshot shows a terminal window titled "RSAEncryptionApp — java com.shadsluiter.App — 85x29". The window displays the following interaction:

```
[afrear@Alexs-MacBook-Pro RSAEncryptionApp % mkdir -p com/shadsluiter
[afrear@Alexs-MacBook-Pro RSAEncryptionApp % mv App.java com/shadsluiter/
[afrear@Alexs-MacBook-Pro RSAEncryptionApp % javac com/shadsluiter/App.java
[afrear@Alexs-MacBook-Pro RSAEncryptionApp % java com.shadsluiter.App
Welcome to the RSA encryption and decryption program!
Enter your message: Hello World
What bit length would you like to use for the prime numbers? (40 - 4096)
Enter a number: 100
Choose a value for the public exponent e:
1: 17
2: 257
3: 4097
4: 65537
Enter the number corresponding to your choice: 2
p: 951848348556053855984057789993
q: 871256136808886676091668115049
n: 829303714990866108779420614878564859683130519870943004904657
φ(n) = (p - 1) * (q - 1) = 8293037149908661087794206148767417551977655793388672789996
16
e (public exponent): 257
d (private key): 132301370874029223579596284863604715809760267520986608712001
Message: Hello World
Encrypted message (integer): 61640444472035031966538209725036221359382179360482247254
7716
Decrypted integer: 87521618088882533792115812
Decrypted byte array: [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
Decrypted ASCII string: Hello World
Time elapsed: 79 milliseconds
Do you want to encrypt another message? (yes/no):
```

Figure 22 This screenshot captures the RSA encryption and decryption application running in Terminal on macOS. The program prompts the user to enter a message, generates a public/private key pair, encrypts the message to an integer format, and then decrypts it back into a readable ASCII string. In this example, the original message "Hello World" is successfully encrypted and decrypted, demonstrating how asymmetric cryptography functions using Java.

## Summary and Reflection

This activity introduced me to various methods of data encryption using GPG Suite and the command line. It was my first time doing anything like this, so I wasn't sure what to expect going in. From the very beginning, I realized this would be a challenge because I had no prior experience with cryptographic tools or working in the command line beyond basic navigation. That said, I was determined to follow the guide closely and take it one step at a time.

In **Part 1**, I started by generating a GPG key pair using the GPG Keychain app. I learned the difference between public and private keys, and how to use them for encrypting and decrypting files. It took a couple of tries to get it right, but once I understood where the files were saved and how to import keys, the process made a lot more sense. I also practiced symmetric encryption, which was easier in some ways but still required paying attention to the passphrase used.

**Part 2** introduced digital signatures and verifying encrypted messages. I learned how to sign a file with my private key and verify it with the public key. This part helped me understand how digital signatures help ensure the message wasn't tampered with and that it came from a trusted source. Seeing that verification message pop up was satisfying because it confirmed the encryption and signature had worked correctly.

In **Part 3**, I repeated similar encryption, decryption, and signing tasks using the command line instead of the graphical interface. This was the most challenging part of the activity for me. Even small typos in commands caused errors, so I had to stay patient and double-check everything. It took multiple attempts to get the syntax right, and I had to troubleshoot a few issues, but I ended up learning a lot through the process. I now feel more comfortable running GPG commands in Terminal, which I never thought I'd say at the beginning of this.

**Part 4** covered symmetric encryption with a passphrase via the CLI. This was a simpler approach to encryption since no key pair was involved, but it also made me realize the downside of sharing passphrases securely. Still, it was good to see both types of encryption in action and compare their pros and cons.

**Part 5** involved compiling and running a Java-based RSA encryption program. This was something completely new for me, and there was definitely a learning curve. I encountered errors related to class names and directories, but after reading the messages carefully and researching a bit, I was able to restructure the files and run the program successfully in Terminal. It took a couple of tries, but once the application launched, I got to see the entire RSA process in action — from selecting the public exponent to encrypting and decrypting an actual message. It was exciting to watch my "Hello World" message get encrypted into integers and then decoded back into readable text. This hands-on portion made everything I'd been learning in earlier parts click more clearly, especially how keys and encryption formulas work behind the scenes.

Throughout this process, I learned how public and private key encryption works, how to sign and verify messages, and the differences between symmetric and asymmetric encryption. I now have a much better understanding of how these techniques support confidentiality, integrity, and authenticity — key principles of application security.

While this assignment was definitely a learning curve, I'm proud of what I was able to accomplish. It took time, trial and error, and some frustration along the way, but going through each part helped me start to grasp the bigger picture. I now have real hands-on experience with tools that professionals use to secure data and communications, and that's a big step forward in my understanding of cybersecurity.