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

Part 1: Generating and Managing GPG Keys

Screenshots

Verified GPG Installation

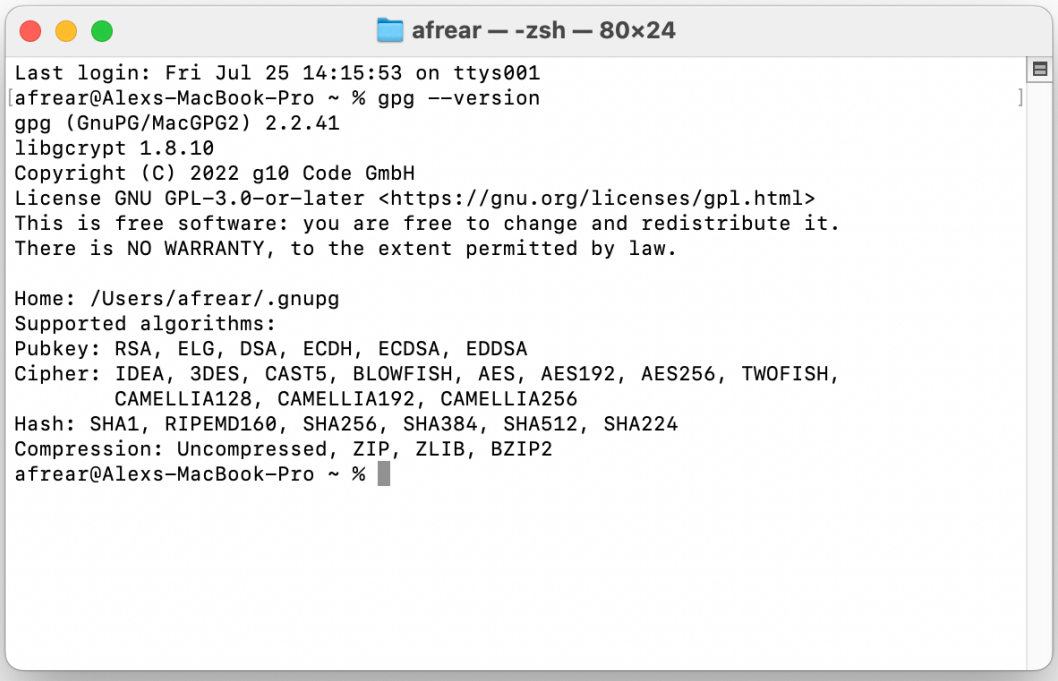

Figure 1 Verified GPG is installed and functioning on macOS.

GPG Keychain Generated Key

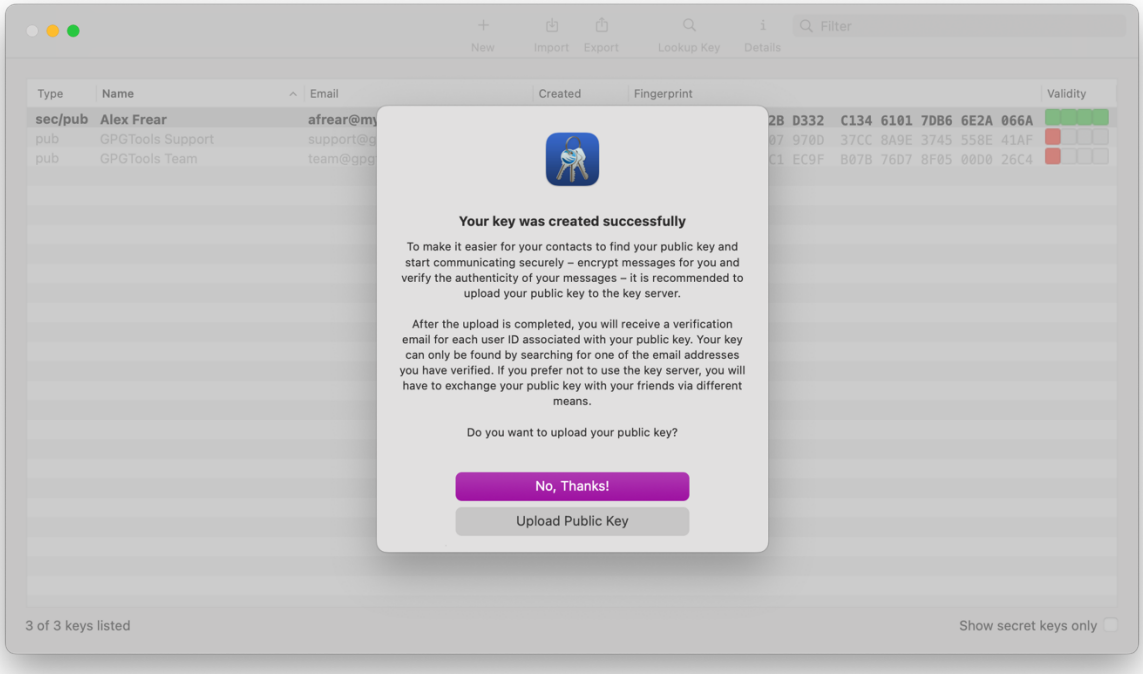

Figure 2 GPG Keychain showing newly created RSA key pair.

Symmetric Encryption via GPG GUI

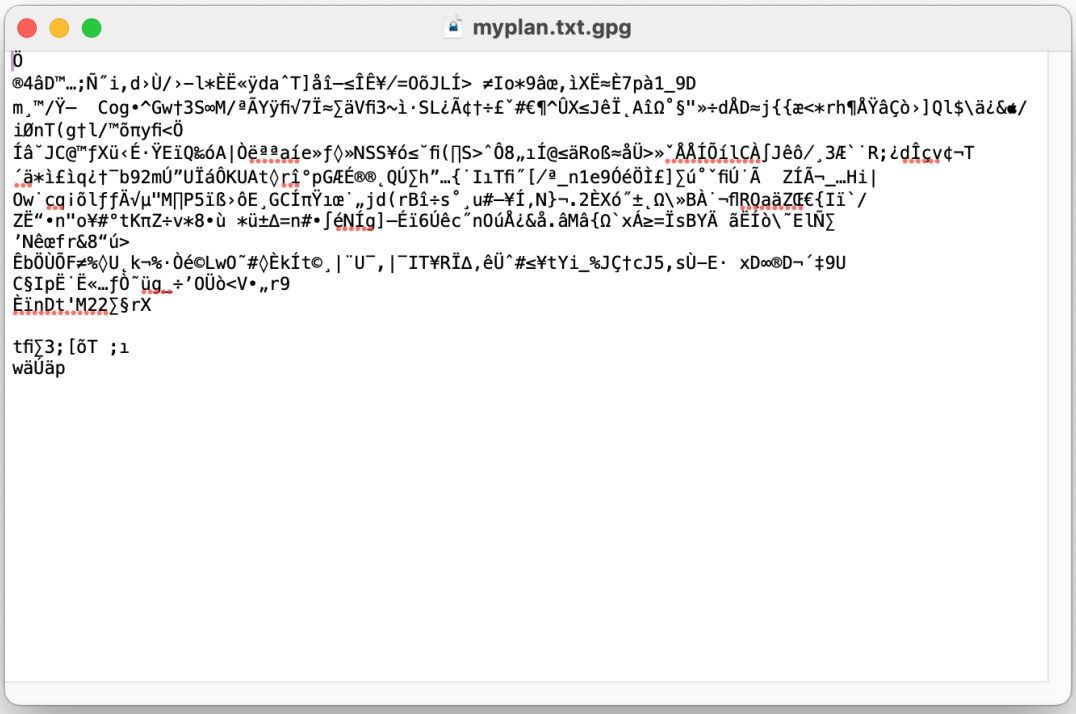

Figure 3 Encrypted file `myplan.txt.gpg` opened in TextEdit showing unreadable encrypted data.

Symmetric Encryption via CLI (ASCII Output)

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

ASCII Encrypted File in TextEdit

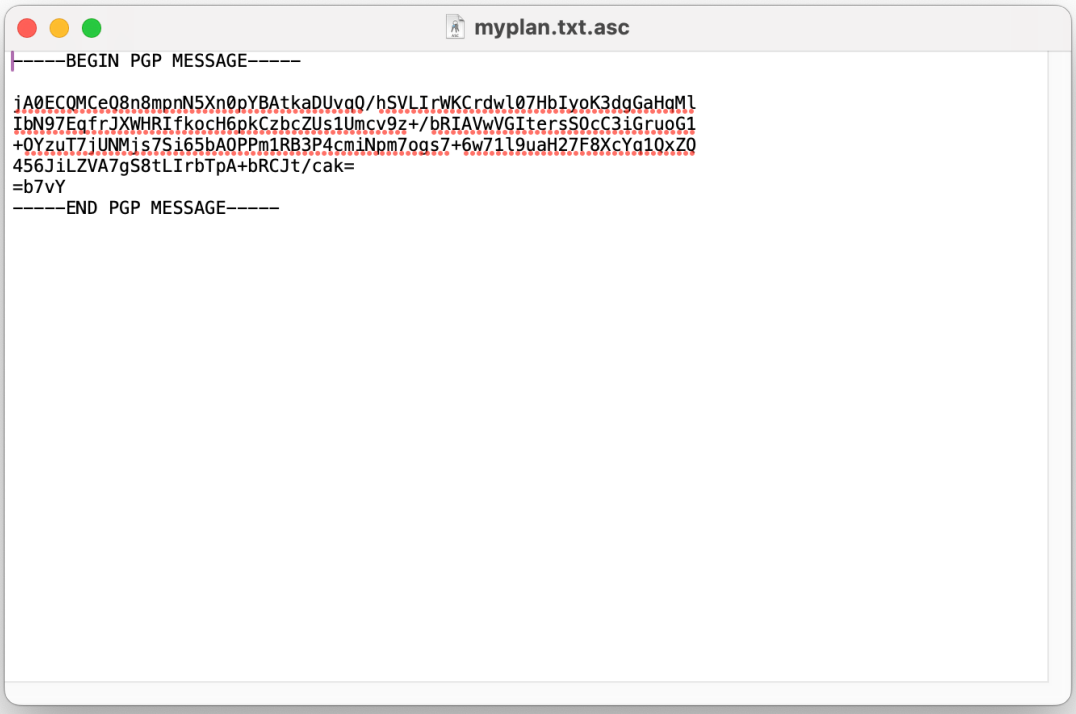

Figure 5 Encrypted `.asc` file opened in TextEdit showing ASCII armored content.

Decrypting the .asc File with GPG in Terminal

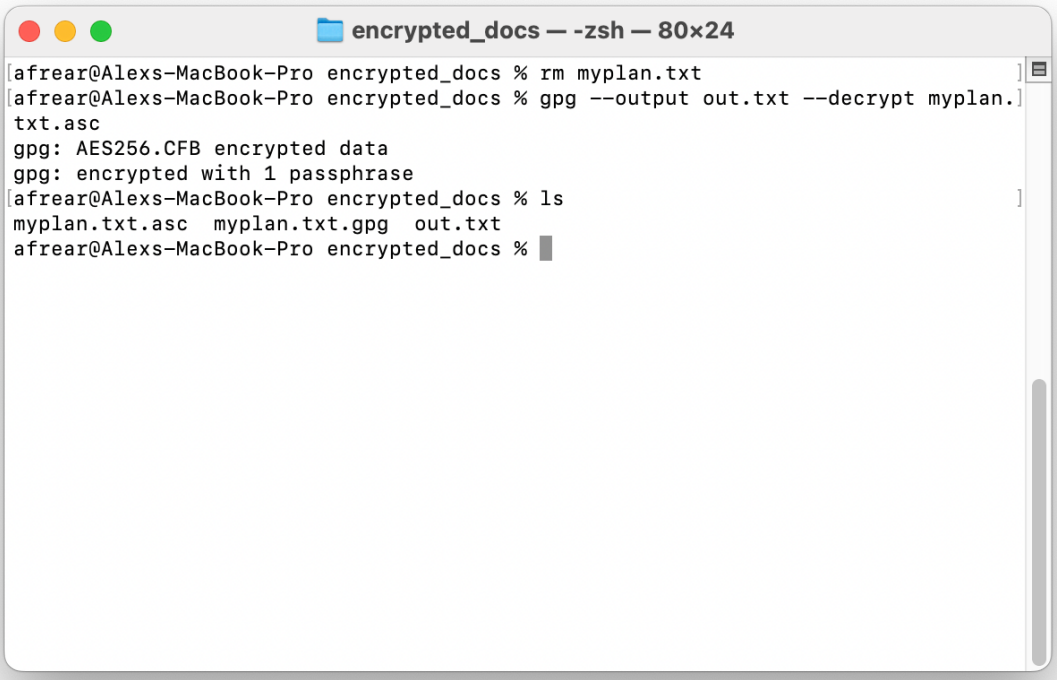

Figure 6 Terminal decrypting `myplan.txt.asc` into `out.txt` using GPG.

Decrypted File Confirmed in TextEdit

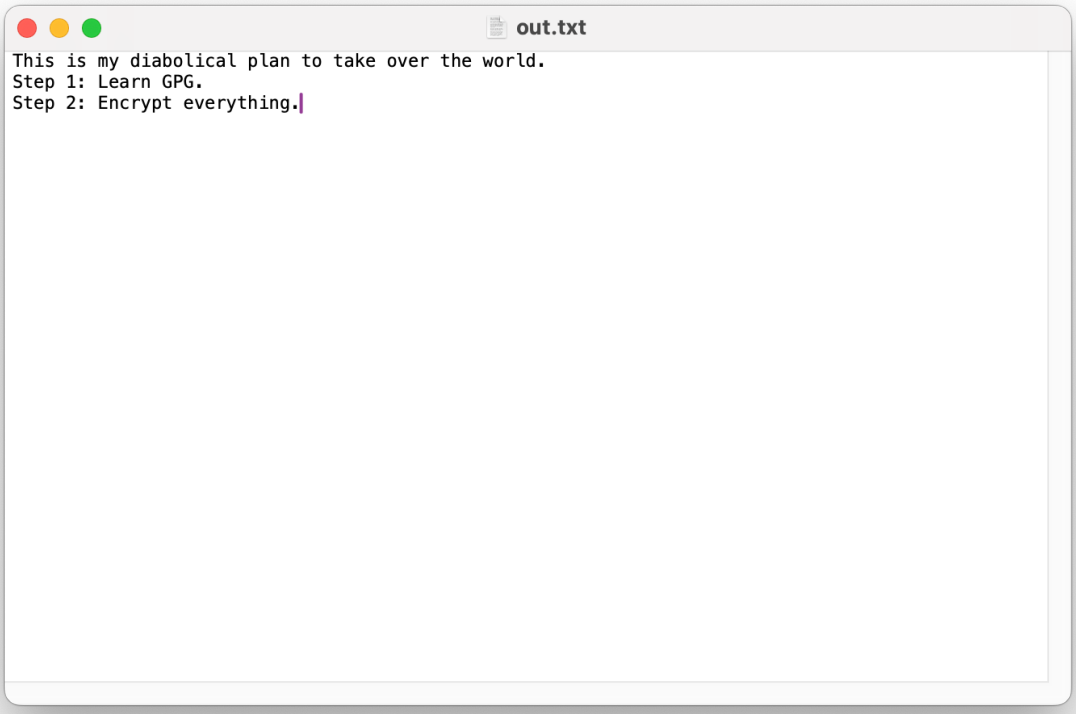

Figure 7 Decrypted `out.txt` opened in TextEdit showing restored original message.

Part 2: Signing and Decryption Verification

Screenshots

Exporting Public Key

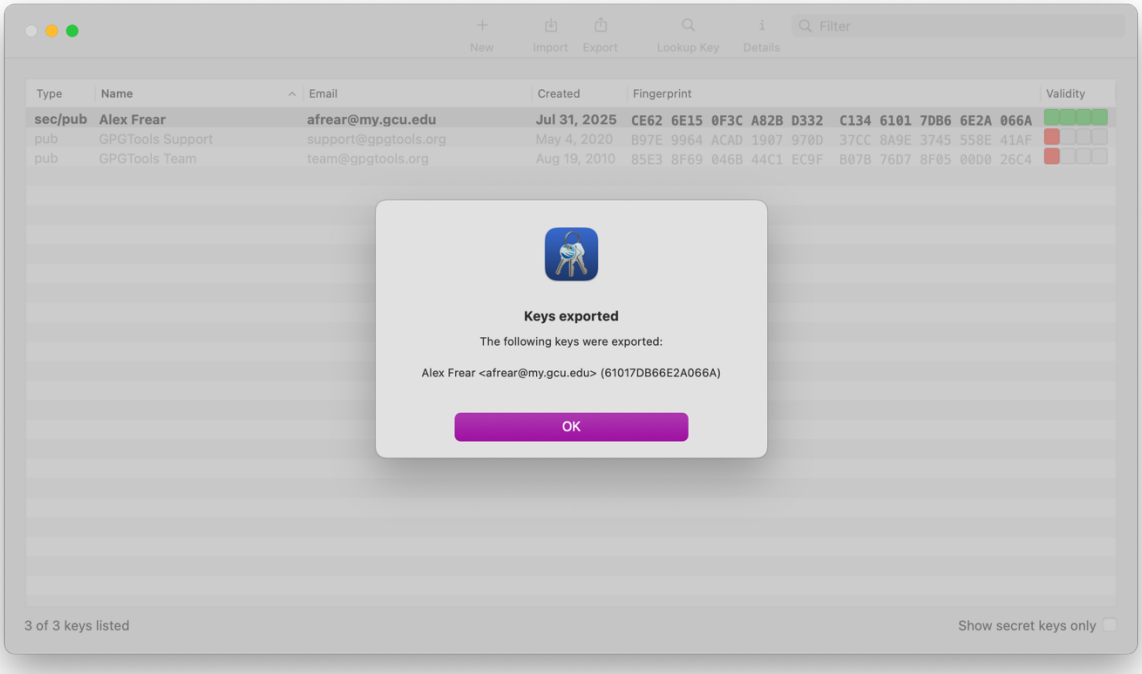

Figure 8 Export confirmation for the newly created GPG public key.

Importing Public Key

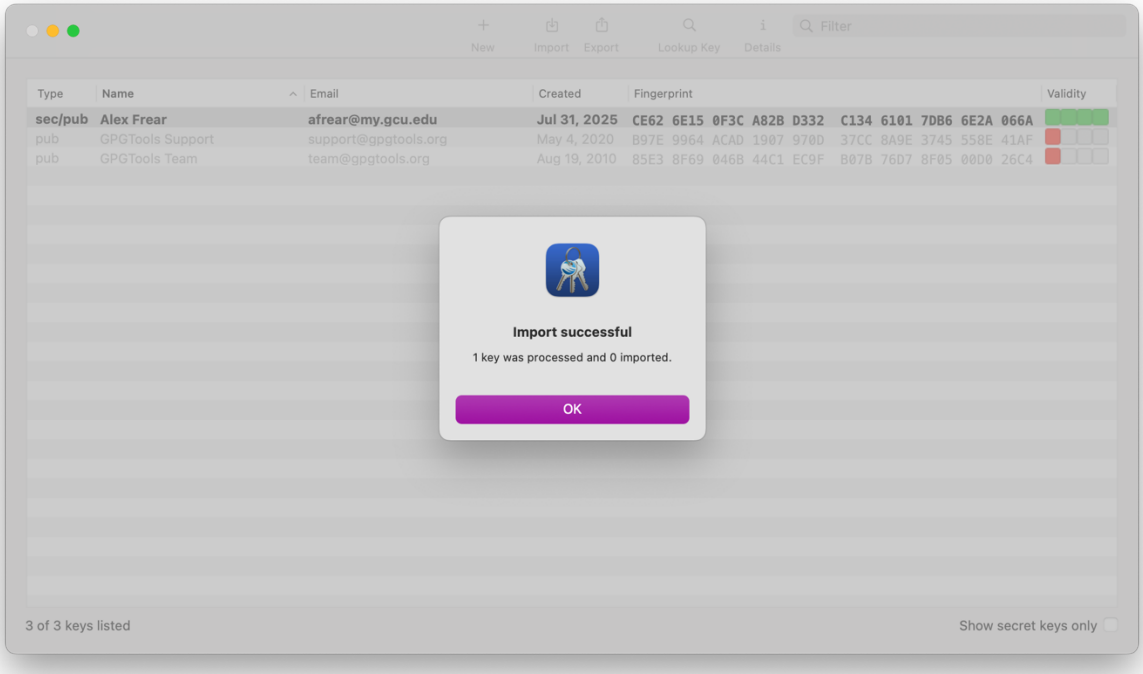

Figure 9 Successful import confirmation of a public key into GPG Keychain.

Encrypting File with Asymmetric Key

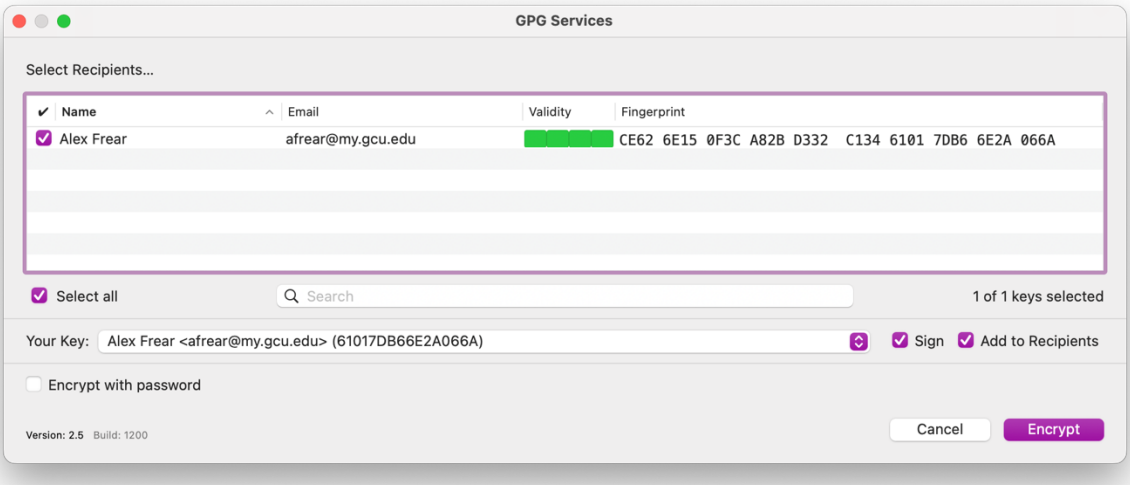

Figure 10 Encrypting a file using GPG with recipient selected and signing enabled.

Decryption and Signature Verification

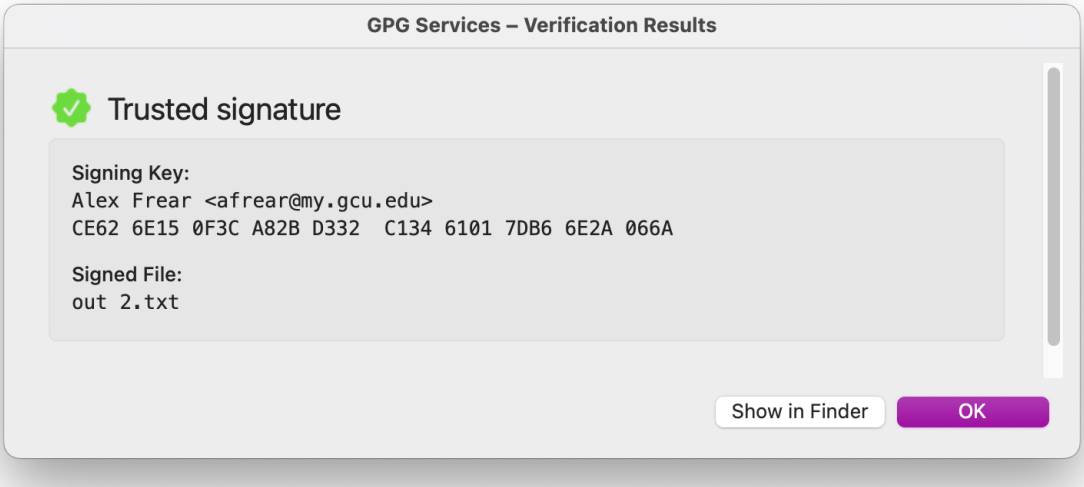

Figure 11 GPG confirms the file was decrypted and signed by a trusted key.

Part 3: Asymmetric Encryption and Digital Signatures via Command Line

Screenshots

Encrypting a Message with a Public Key

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

Decrypting the Encrypted Message

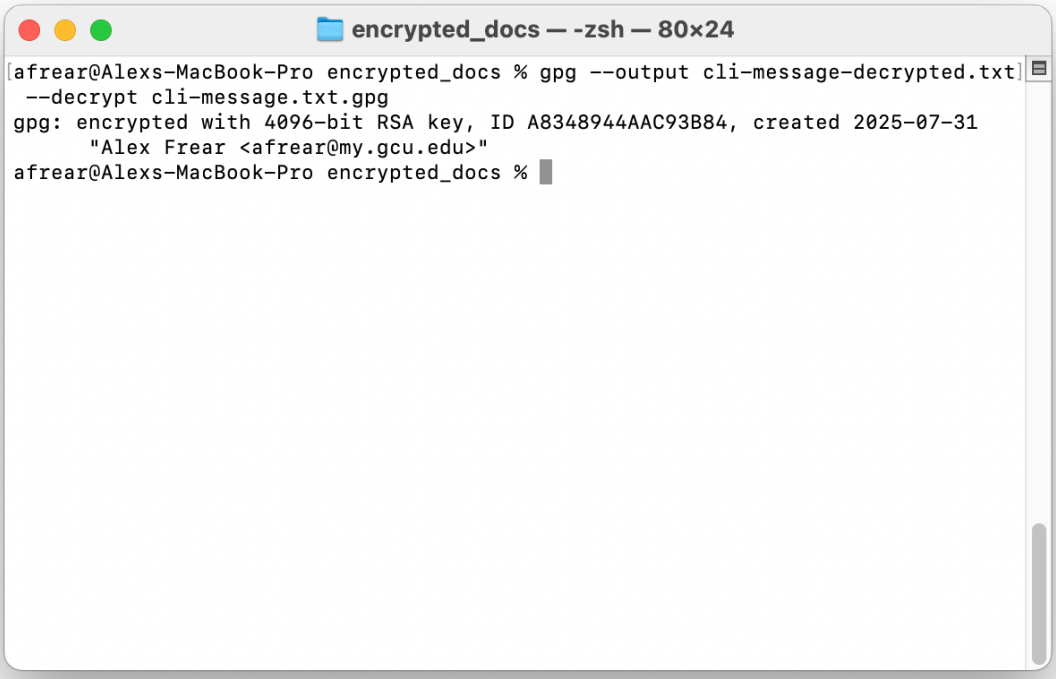

Figure 13 Decrypting the encrypted file cli-message.txt.gpg to produce cli-message-decrypted.txt.

Confirming Decryption Output

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

Verifying a Digitally Signed Message

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

Creating a Signed and Encrypted Message

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

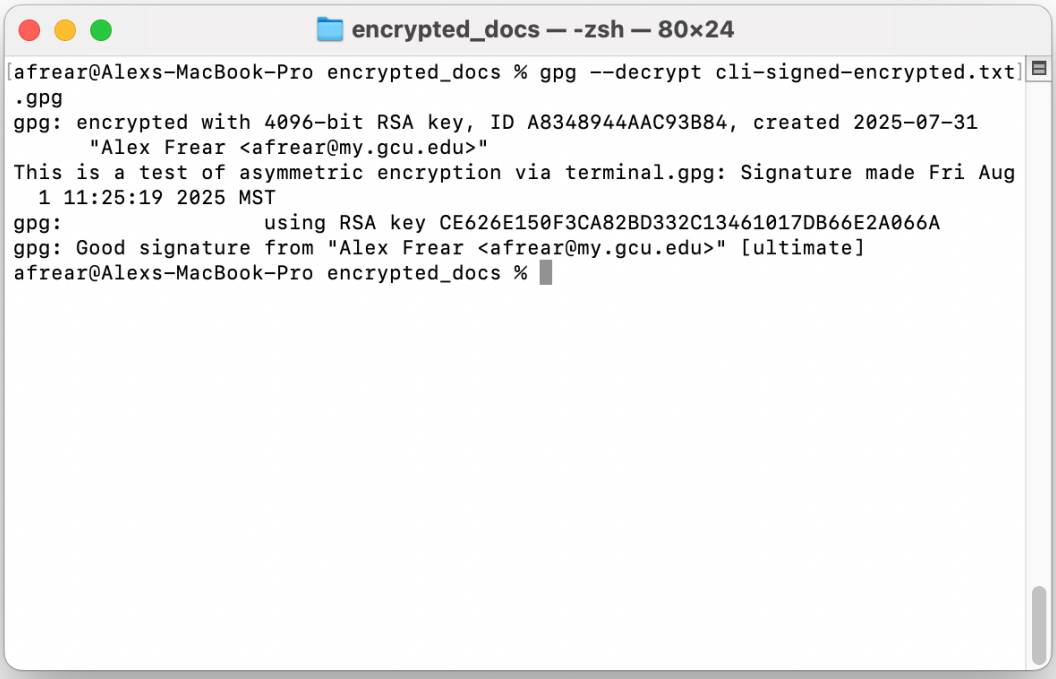

Figure 17 Decrypting and verifying the signature of cli-signed-encrypted.txt.gpg.

Part 4: Symmetric Encryption

Screenshots

Creating the Symmetric Message File

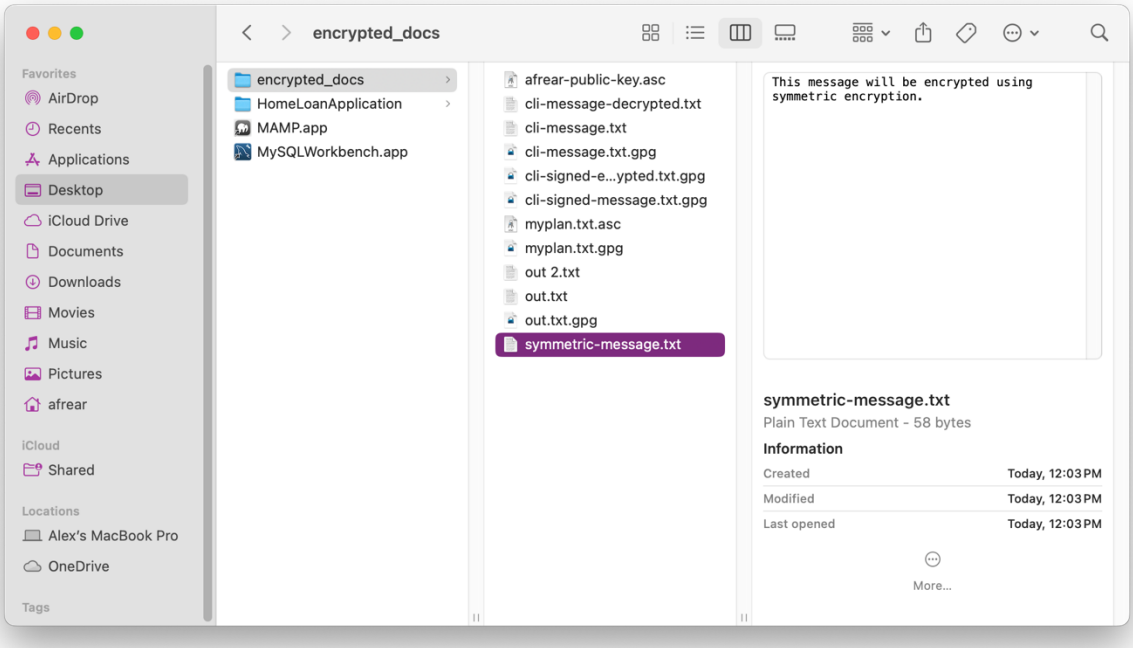

Figure 18 Plain text file symmetric-message.txt created using TextEdit. This file contains the message to be encrypted using symmetric encryption.

Encrypting the Message with a Passphrase

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

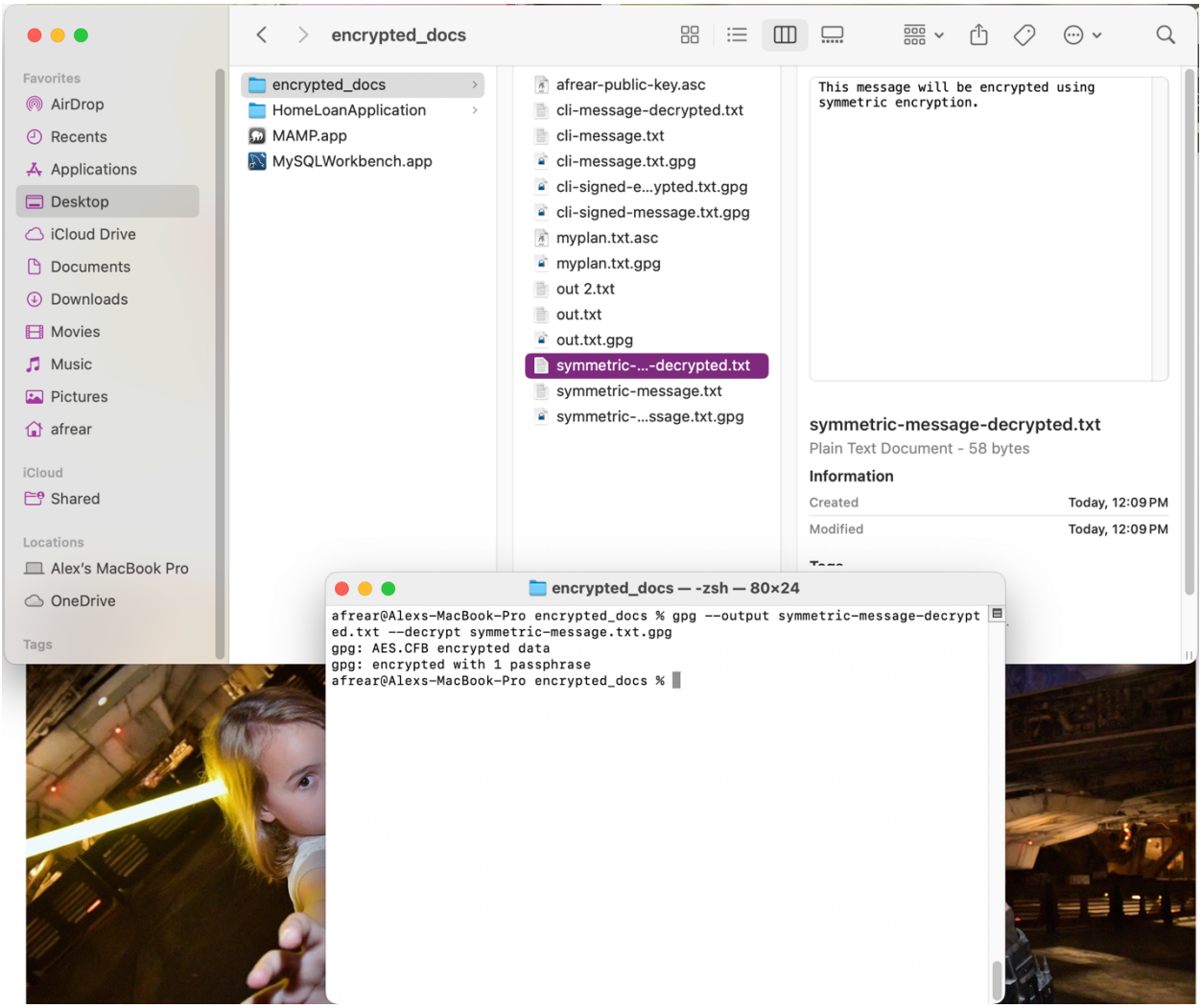

Figure 20 Terminal output confirming successful decryption of the symmetric file. The decrypted file symmetric-message-decrypted.txt appears alongside the original and encrypted versions in Finder.

Verifying the Decrypted Content Matches Original

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

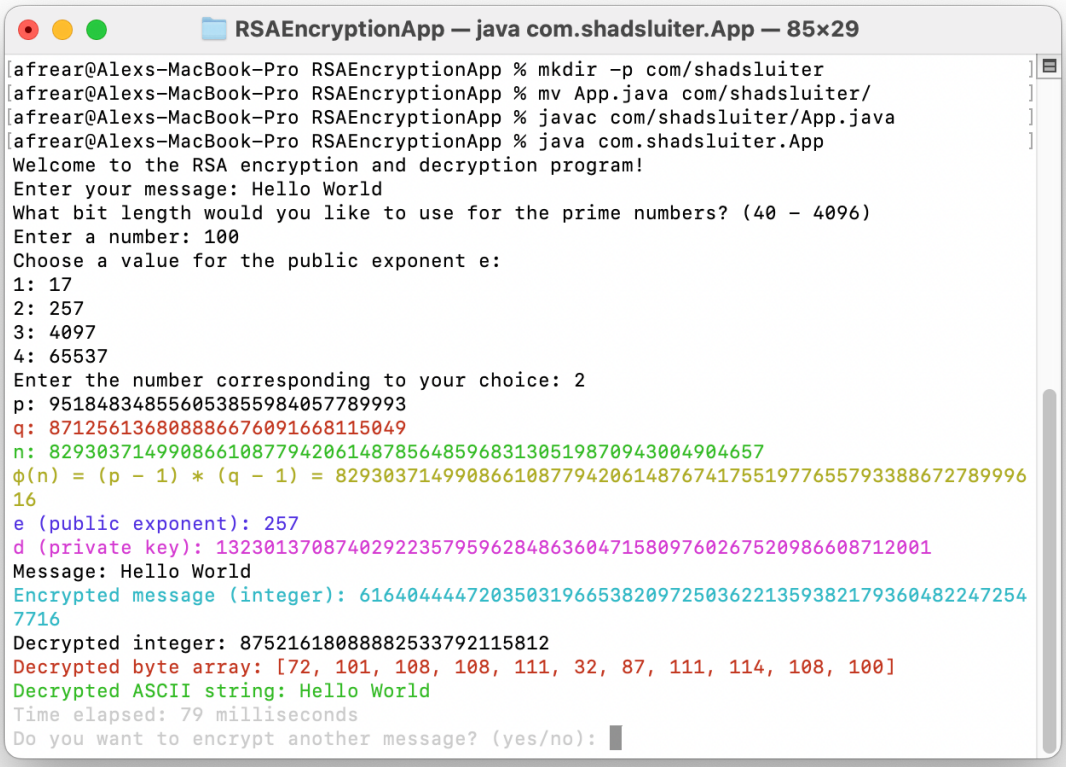

Figure 22 This screenshot captures the RSA encryption and decryption application running in Terminal on macOS. The program prompts the user to enter a message, generates a public/private key pair, encrypts the message to an integer format, and then decrypts it back into a readable ASCII string. In this 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.