FEBRUARY 4, 2024

LAB01

APPLYING ENCRYPTION AND HASHING ALGORITHMS FOR SECURE COMMUNICATIONS

ANTHONY CAMPBELL: YVW316
Professor Darniet Kendrick Jennings
University of Texas at San Antonio IS 3423-ON1

Contents

Introduction

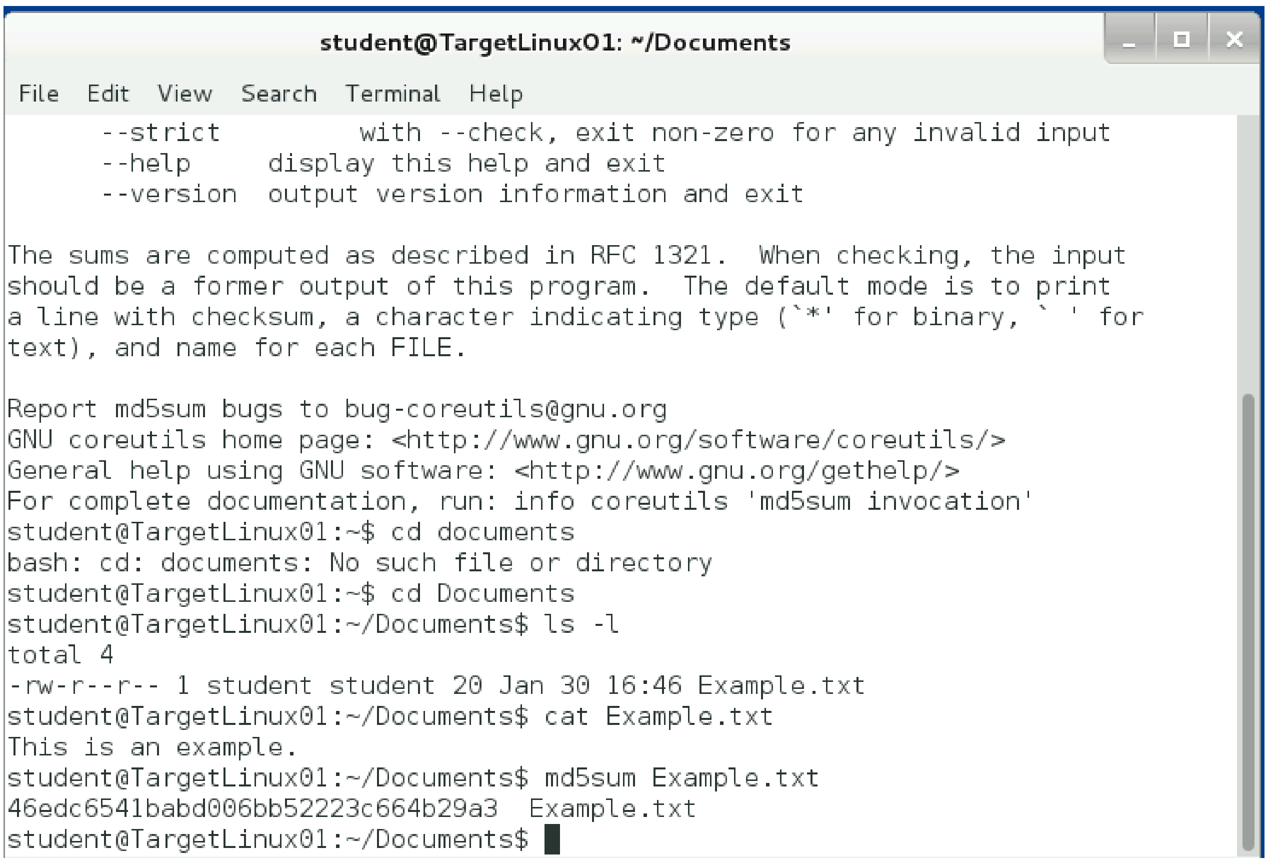
This lab provides a practical exploration of secure communication and cryptography, focusing on essential concepts and hands-on applications. There will be insights into cryptographic hashing using tools like md5sum and sha256sum for generating hash values. The emphasis then shifts to GnuPG, where we generate public encryption keys for both Instructor and Student accounts, enabling secure communication.

Key exchange mechanisms will be explored, including the export and import of GnuPG keys between accounts through WinSCP. Throughout the lab, Linux terminal commands are employed to enhance participants' proficiency in executing cryptographic operations. The theoretical knowledge and hands-on application provide a comprehensive understanding of secure communication and cryptography, participants will gain practical skills applicable in various cybersecurity contexts.

Section 1

MD5sum Hash

I used the Linux terminal to perform various tasks related to MD5 hashing. This includes opening the MD5 help document, navigating to the Documents folder, listing files, and creating an MD5 hash string for a file named Example.txt, with the content "This is an example."

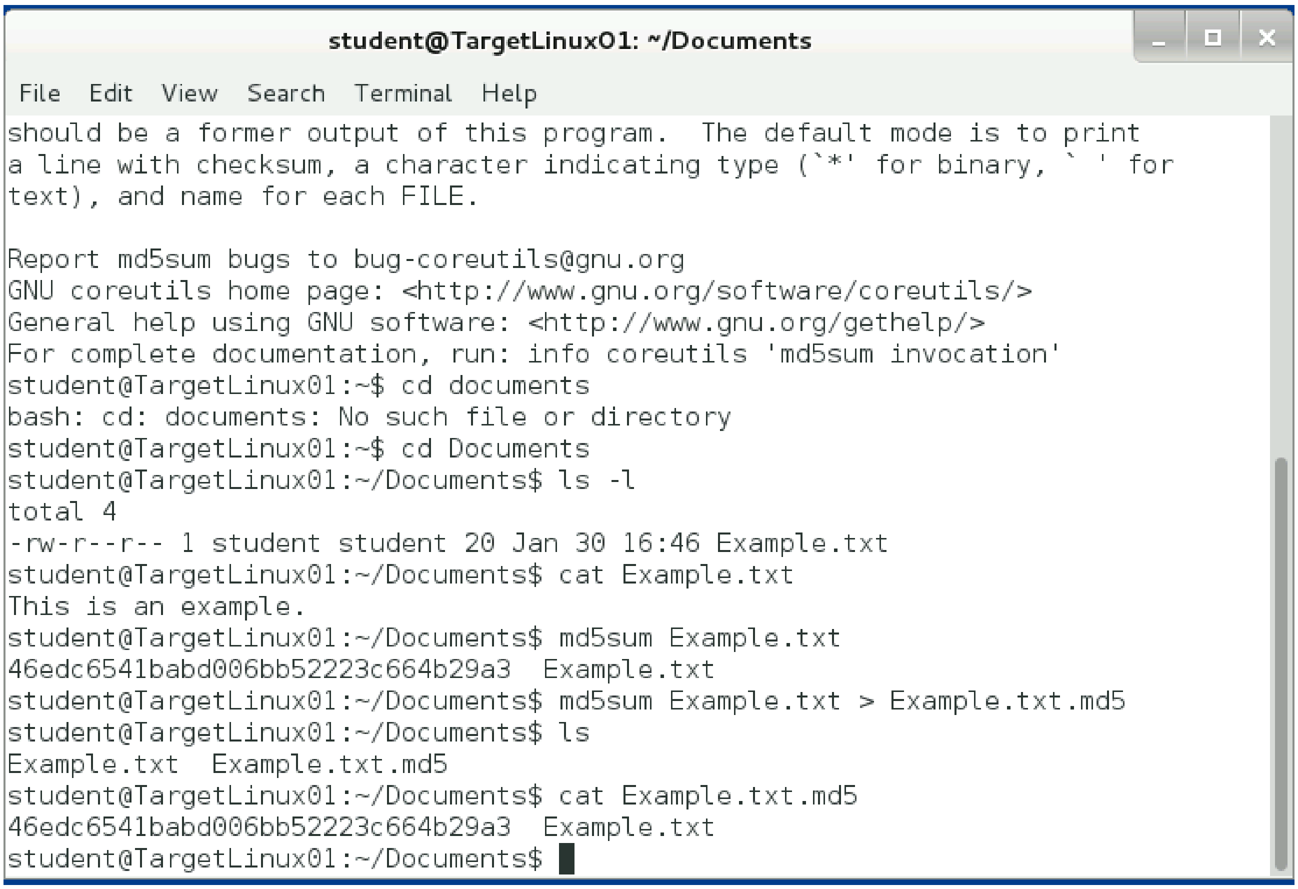


```
student@TargetLinux01: ~/Documents
File  Edit  View  Search  Terminal  Help
      --strict         with --check, exit non-zero for any invalid input
      --help     display this help and exit
      --version  output version information and exit

The sums are computed as described in RFC 1321.  When checking, the input
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating type (`*' for binary, ` ' for
text), and name for each FILE.

Report md5sum bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'md5sum invocation'
student@TargetLinux01:~$ cd documents
bash: cd: documents: No such file or directory
student@TargetLinux01:~$ cd Documents
student@TargetLinux01:~/Documents$ ls -l
total 4
-rw-r--r-- 1 student student 20 Jan 30 16:46 Example.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
student@TargetLinux01:~/Documents$ md5sum Example.txt
46edc6541babd006bb52223c664b29a3  Example.txt
student@TargetLinux01:~/Documents$
```

Figure 1: MD5sum hash for the Example.txt file

MD5 File Contents

I have created a new file, Example.txt.md5, to store the MD5sum hash string for the Example.txt file. I then verify the addition of this new file in the Documents folder by listing the files with command *ls*. Finally, I view the contents of Example.txt.mdS to confirm that it contains the MD5sum hash string recorded in a previous step with the command *cat Example.txt.md5*.

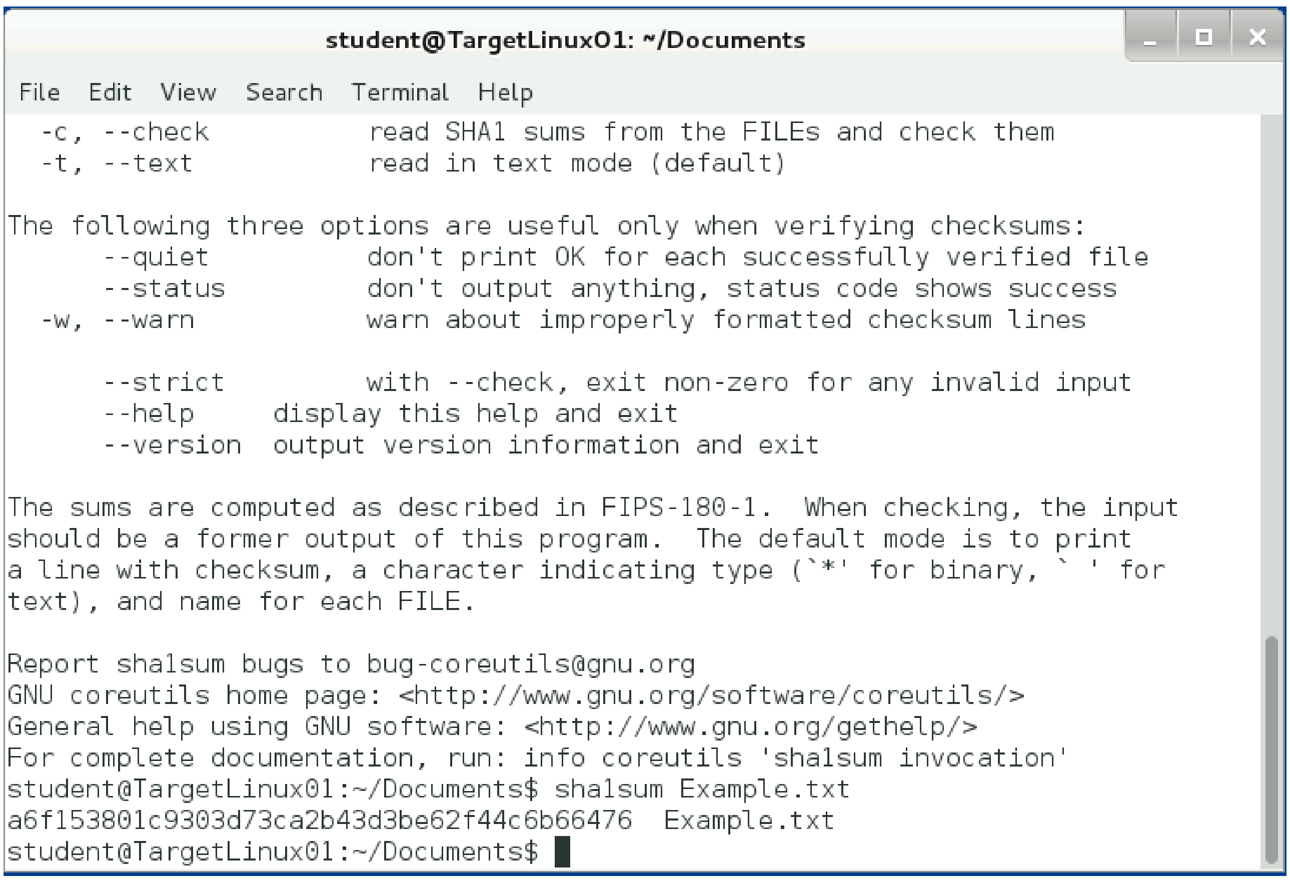


```
student@TargetLinux01: ~/Documents
File  Edit  View  Search  Terminal  Help
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating type (`*' for binary, ` ' for
text), and name for each FILE.

Report md5sum bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'md5sum invocation'
student@TargetLinux01:~$ cd documents
bash: cd: documents: No such file or directory
student@TargetLinux01:~$ cd Documents
student@TargetLinux01:~/Documents$ ls -l
total 4
-rw-r--r-- 1 student student 20 Jan 30 16:46 Example.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
student@TargetLinux01:~/Documents$ md5sum Example.txt
46edc6541babd006bb52223c664b29a3  Example.txt
student@TargetLinux01:~/Documents$ md5sum Example.txt > Example.txt.md5
student@TargetLinux01:~/Documents$ ls
Example.txt  Example.txt.md5
student@TargetLinux01:~/Documents$ cat Example.txt.md5
46edc6541babd006bb52223c664b29a3  Example.txt
student@TargetLinux01:~/Documents$
```

Figure 2: Contents of Example.txt.md5 file

SHA1sum Hash

Firstly, open the sha1sum help document to review its features and switches. Following that, apply the sha1sum command to create a SHA1sum hash string for the Example.txt file. The returned string of hexadecimal numbers represents a unique hash for my file in the virtual session.



```
student@TargetLinux01: ~/Documents
File  Edit  View  Search  Terminal  Help
  -c, --check          read SHA1 sums from the FILEs and check them
  -t, --text           read in text mode (default)

The following three options are useful only when verifying checksums:
      --quiet          don't print OK for each successfully verified file
      --status         don't output anything, status code shows success
  -w, --warn           warn about improperly formatted checksum lines

      --strict         with --check, exit non-zero for any invalid input
      --help     display this help and exit
      --version  output version information and exit

The sums are computed as described in FIPS-180-1.  When checking, the input
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating type (`*' for binary, ` ' for
text), and name for each FILE.

Report sha1sum bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'sha1sum invocation'
student@TargetLinux01:~/Documents$ sha1sum Example.txt
a6f153801c9303d73ca2b43d3be62f44c6b66476  Example.txt
student@TargetLinux01:~/Documents$ 
```

Figure 3: SHA1sum hash for the Example.txt file

SHA1sum Example and contents

We initiate the process by entering the command *sha1sum Example.txt* at the terminal prompt, generating a SHA1sum hash string unique to the contents of this Example.txt file. This string is then written into new file Example.txt.sha1 using the command *sha1sum Example.txt > Example.txt.sha1*. To confirm the successful creation of this new file, execute the *ls* command, listing the contents of the student folder and ensuring the presence of Example.txt.sha1 in the Documents folder.

To review the contents of the newly generated hash file, use the *cat* command in conjunction with *Example.txt.sha1*. This action displays the SHA1sum hash string, verifying the integrity of the Example.txt file.

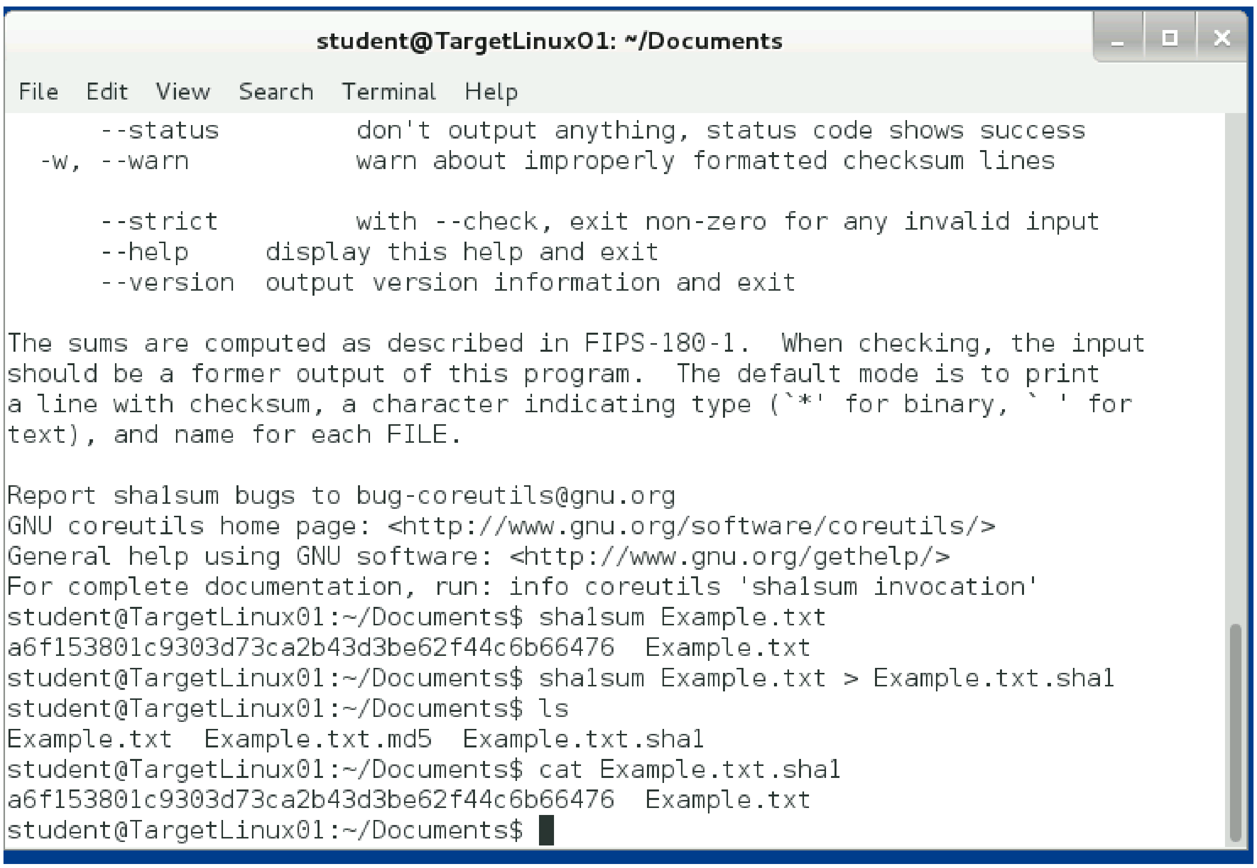
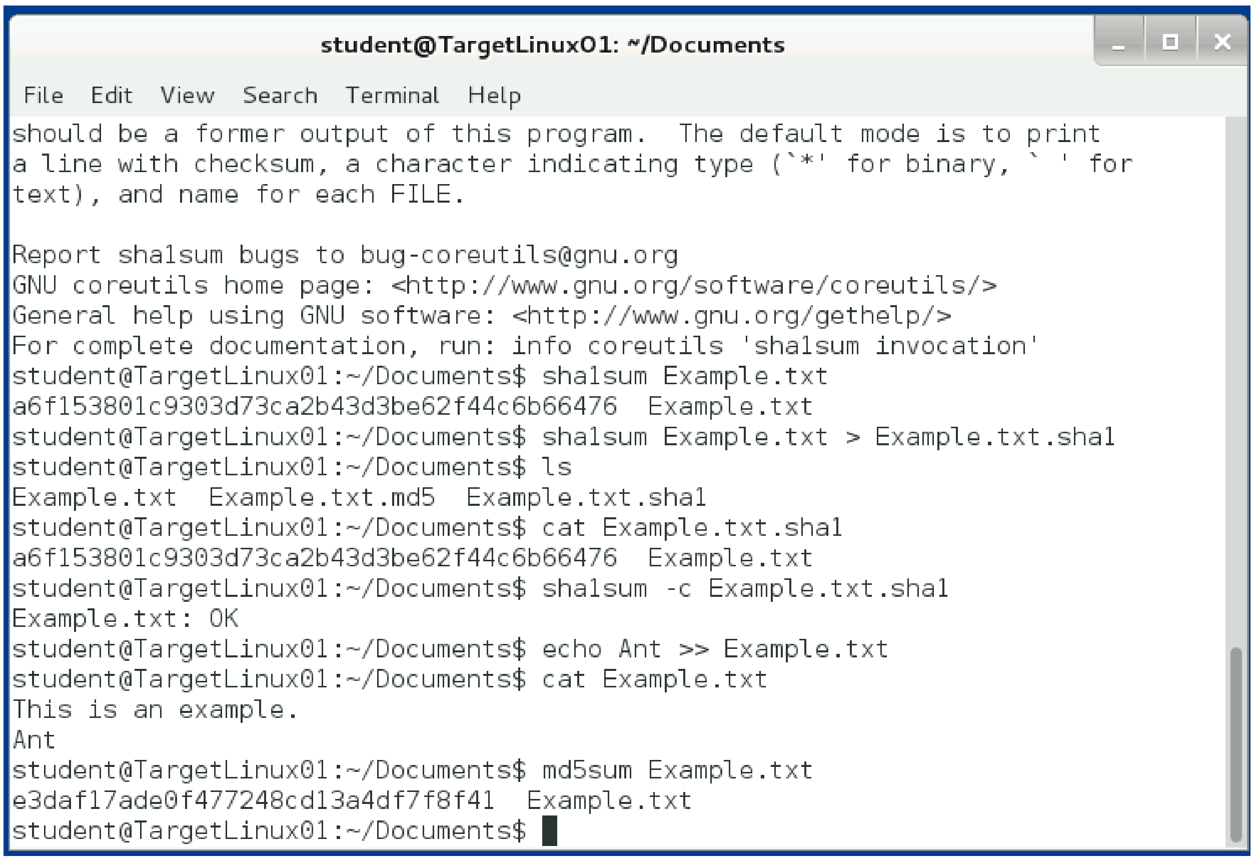


Figure 4: Contents of Example.txt.sha1 file

Modify a File and Verify Hash Values

First, the *echo* command is used, appending a name to the Example.txt file. The file's updated contents are then displayed using *cat*. Following this, the *md5sum* command is executed on the modified Example.txt file, generating a new MD5sum hash string distinct from the original. This hash string serves as a means to verify alterations in the file's content.




```
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating type (`*' for binary, ` ' for
text), and name for each FILE.

Report sha1sum bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'sha1sum invocation'
student@TargetLinux01:~/Documents$ sha1sum Example.txt
a6f153801c9303d73ca2b43d3be62f44c6b66476  Example.txt
student@TargetLinux01:~/Documents$ sha1sum Example.txt > Example.txt.sha1
student@TargetLinux01:~/Documents$ ls
Example.txt  Example.txt.md5  Example.txt.sha1
student@TargetLinux01:~/Documents$ cat Example.txt.sha1
a6f153801c9303d73ca2b43d3be62f44c6b66476  Example.txt
student@TargetLinux01:~/Documents$ sha1sum -c Example.txt.sha1
Example.txt: OK
student@TargetLinux01:~/Documents$ echo Ant >> Example.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
Ant
student@TargetLinux01:~/Documents$ md5sum Example.txt
e3daf17ade0f477248cd13a4df7f8f41  Example.txt
student@TargetLinux01:~/Documents$
```

Figure 5: New md5sum hash string

The next step involves using the *sha1sum* command on the modified Example.txt file. This action generates an SHA1sum hash string. The use of different hash algorithms, such as MD5 and SHA1, enhances the security and integrity verification capabilities, offering multiple perspectives on file alterations.

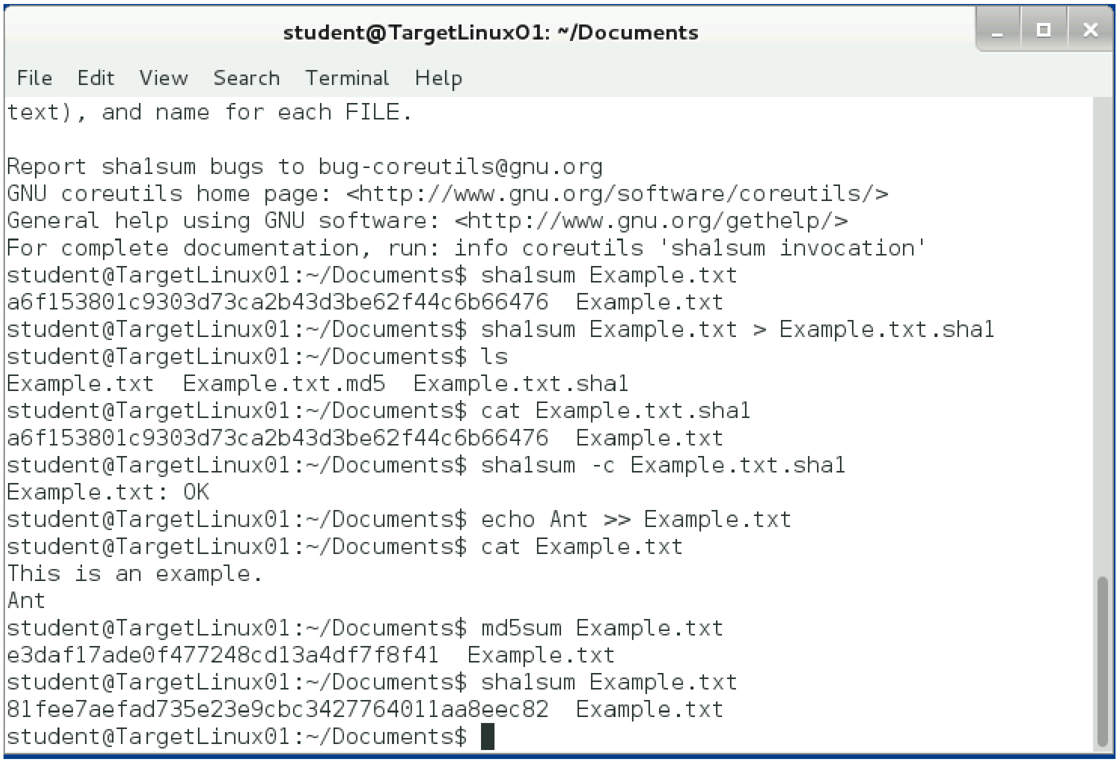


Figure 6: New sha1 hash string

Generate GnuPG Keys

In the Linux terminal, ensure the logged-on user is identified as *student@TargetLinux01:~/Documents$*. Initiate the GPG key generation process by executing the command *gpg --gen-key* and respond to on-screen prompts. An error indicates an insufficient random bytes issue, open a second terminal window and run the script *./entropy_loop.sh* to generate entropy. After sufficient bytes are available, return to the initial terminal window.

Next, close the second terminal window and execute *gpg --export -a > student.pub* to save the GnuPG key to a new file named student.pub. Confirm the working directory with *pwd,* ensuring it displays *home/student/Documents*, the student's Documents folder. List files using ls and verify that student.pub is correctly saved in the Documents folder.

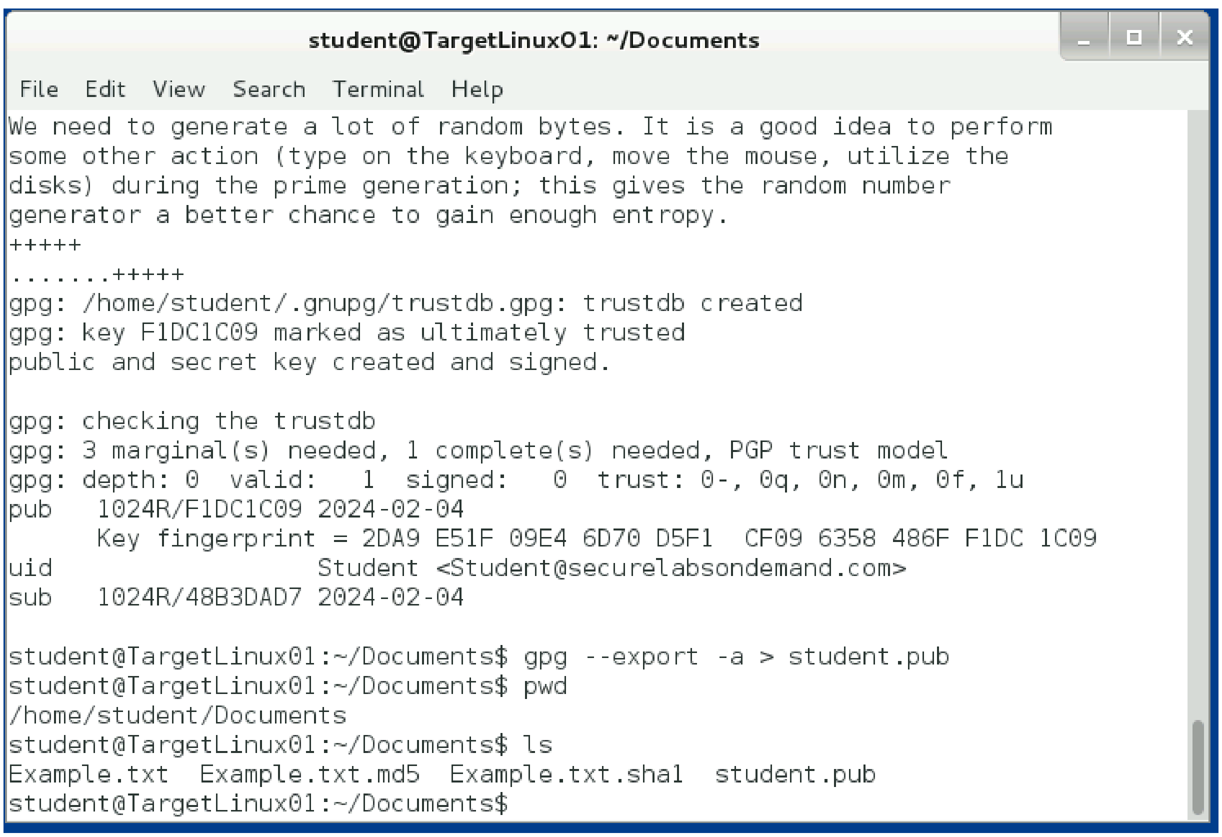
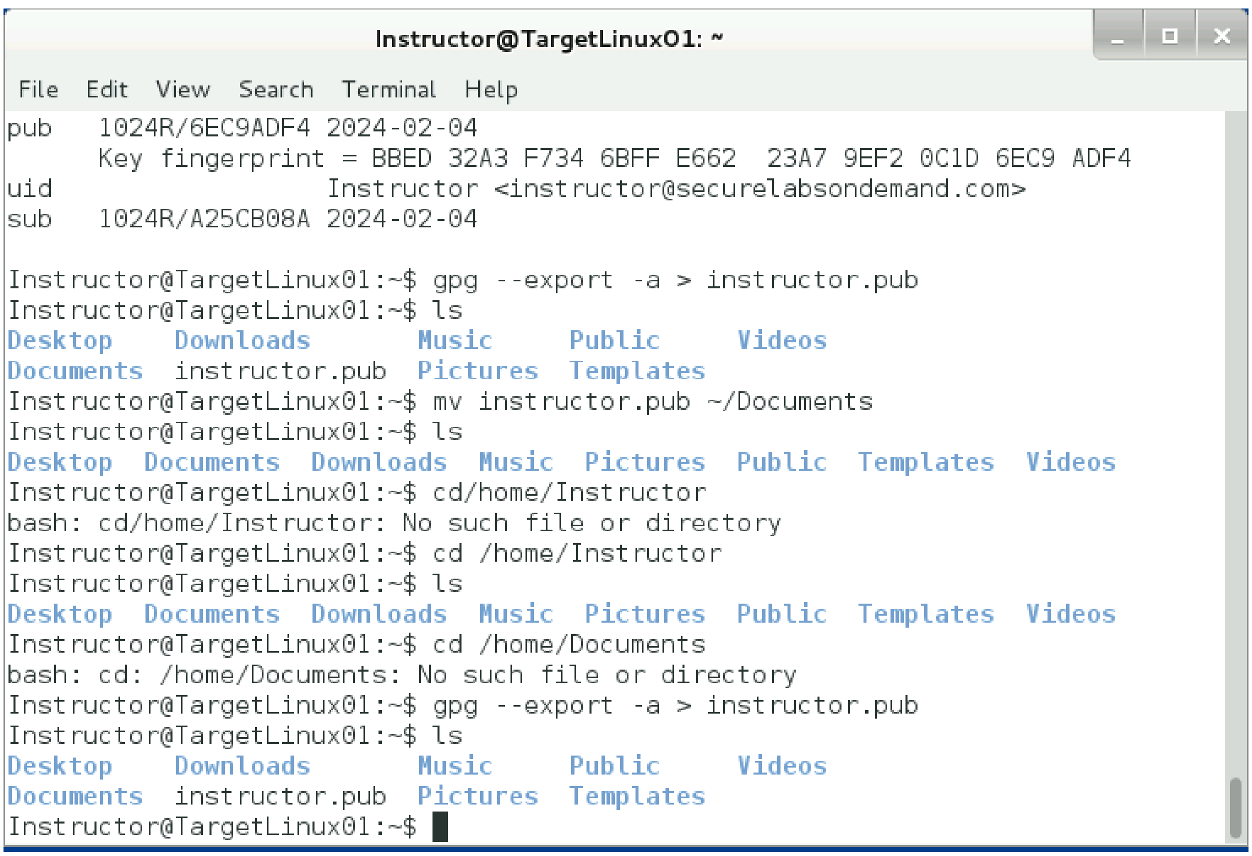


Figure 7: Documents folder contents post student.pub creation

Instructor Key and Sharing a GnuPG Key

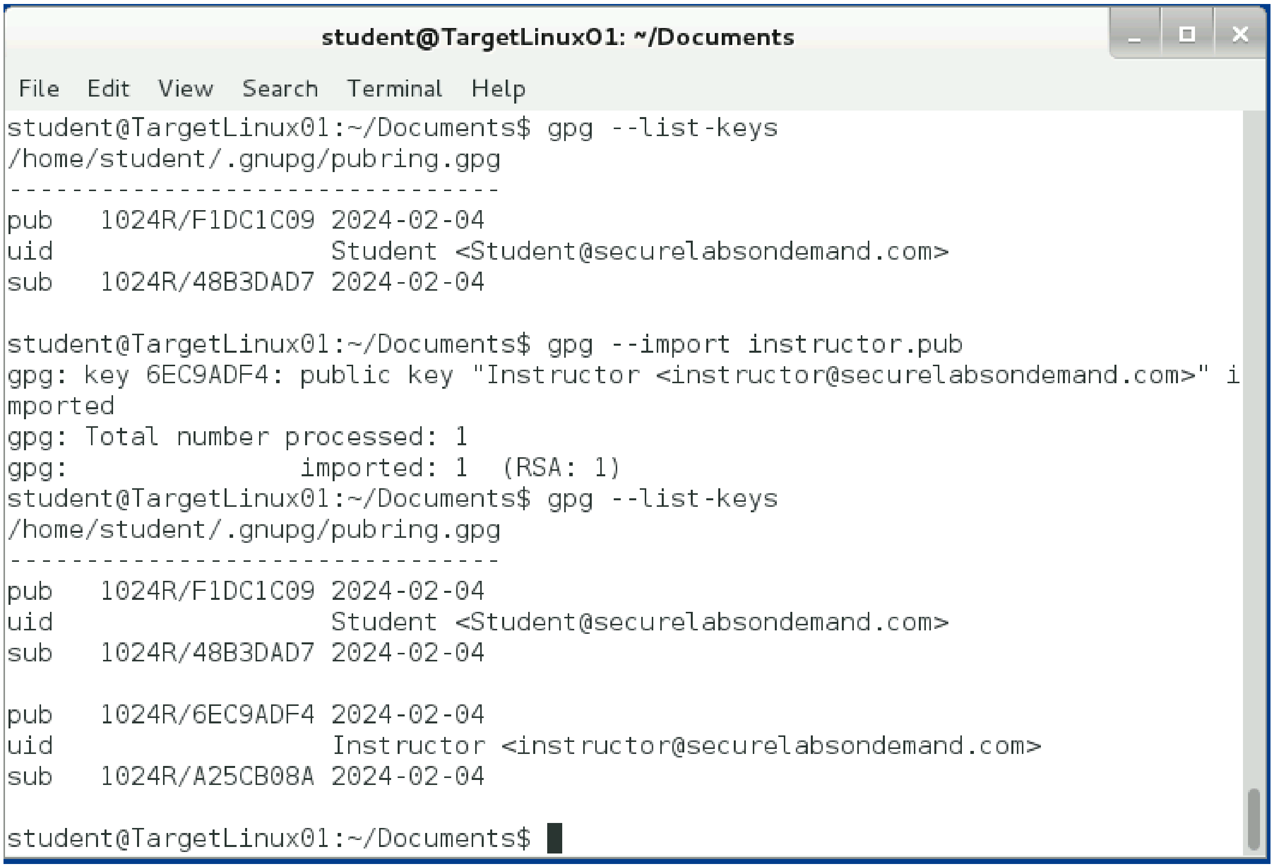
On the instructor's machine, the GnuPG key is created with by following the last steps, and then exported using the command *gpg --export -a > instructor.pub*, saving it as instructor.pub. Verification of the correct file save is done by listing the files in the folder using *ls*. To facilitate secure communication, the instructor's GnuPG keys (instructor.pub) are then copied to the student's Documents folder via the keytranfer11 & keytransfer12 connections. The student can confirm the successful transfer by using the *ls* command to list folder contents.

Next, the student updates their public key ring by importing the instructor's GnuPG keys. Initially, the student's public key ring is listed using *gpg --list-keys*, which should show only the student-pub key. The import command, *gpg --import instructor.pub*, adds the instructor's public key to the student's key ring. Finally, the student verifies the update by listing the public key ring again using *gpg --list-keys*. This process ensures that the instructor's GnuPG key is successfully shared and integrated into the student's key ring for secure communication.



```
Instructor@TargetLinux01: ~
File  Edit  View  Search  Terminal  Help
pub   1024R/6EC9ADF4 2024-02-04
      Key fingerprint = BBED 32A3 F734 6BFF E662  23A7 9EF2 0C1D 6EC9 ADF4
uid                  Instructor <instructor@securelabsondemand.com>
sub   1024R/A25CB08A 2024-02-04

Instructor@TargetLinux01:~$ gpg --export -a > instructor.pub
Instructor@TargetLinux01:~$ ls
Desktop    Downloads      Music     Public     Videos
Documents  instructor.pub  Pictures  Templates
Instructor@TargetLinux01:~$ mv instructor.pub ~/Documents
Instructor@TargetLinux01:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
Instructor@TargetLinux01:~$ cd/home/Instructor
bash: cd/home/Instructor: No such file or directory
Instructor@TargetLinux01:~$ cd /home/Instructor
Instructor@TargetLinux01:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
Instructor@TargetLinux01:~$ cd /home/Documents
bash: cd: /home/Documents: No such file or directory
Instructor@TargetLinux01:~$ gpg --export -a > instructor.pub
Instructor@TargetLinux01:~$ ls
Desktop    Downloads      Music     Public     Videos
Documents  instructor.pub  Pictures  Templates
Instructor@TargetLinux01:~$
```

Figure 8: Contents of /home/instructor folder, after a snafu with rereading instructions




```
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
--------------------------------
pub   1024R/F1DC1C09 2024-02-04
uid                  Student <Student@securelabsondemand.com>
sub   1024R/48B3DAD7 2024-02-04

student@TargetLinux01:~/Documents$ gpg --import instructor.pub
gpg: key 6EC9ADF4: public key "Instructor <instructor@securelabsondemand.com>" i
mported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
--------------------------------
pub   1024R/F1DC1C09 2024-02-04
uid                  Student <Student@securelabsondemand.com>
sub   1024R/48B3DAD7 2024-02-04

pub   1024R/6EC9ADF4 2024-02-04
uid                  Instructor <instructor@securelabsondemand.com>
sub   1024R/A25CB08A 2024-02-04

student@TargetLinux01:~/Documents$
```

Figure 9: Student's public key ring

Encrypt and Decrypt a ClearText Message

In this section, the GnuPG (Gnu Privacy Guard) is employed to encrypt a clear-text message exchanged between the users, Instructor and Student. The GnuPG help document is accessed using the command *gpg --help* to understand the tool's options and functionalities. A clear-text message is created and saved to a file named cleartext.txt through the command *echo "this is a clear-text message from yourname" > cleartext.txt*, with the user replacing "yourname" with their own name. The file's contents are then displayed using *cat cleartext.txt*.

The encryption process is initiated with the command *gpg -e cleartext.txt*. During encryption, the user is prompted to enter the recipient's user ID (Instructor), confirm the key usage, and press Enter to complete the process. The encrypted file, cleartext.txt.gpg, is verified to exist through the *ls* command. Finally, the contents of the encrypted file are displayed using *cat cleartext.txt.gpg* to confirm successful encryption.

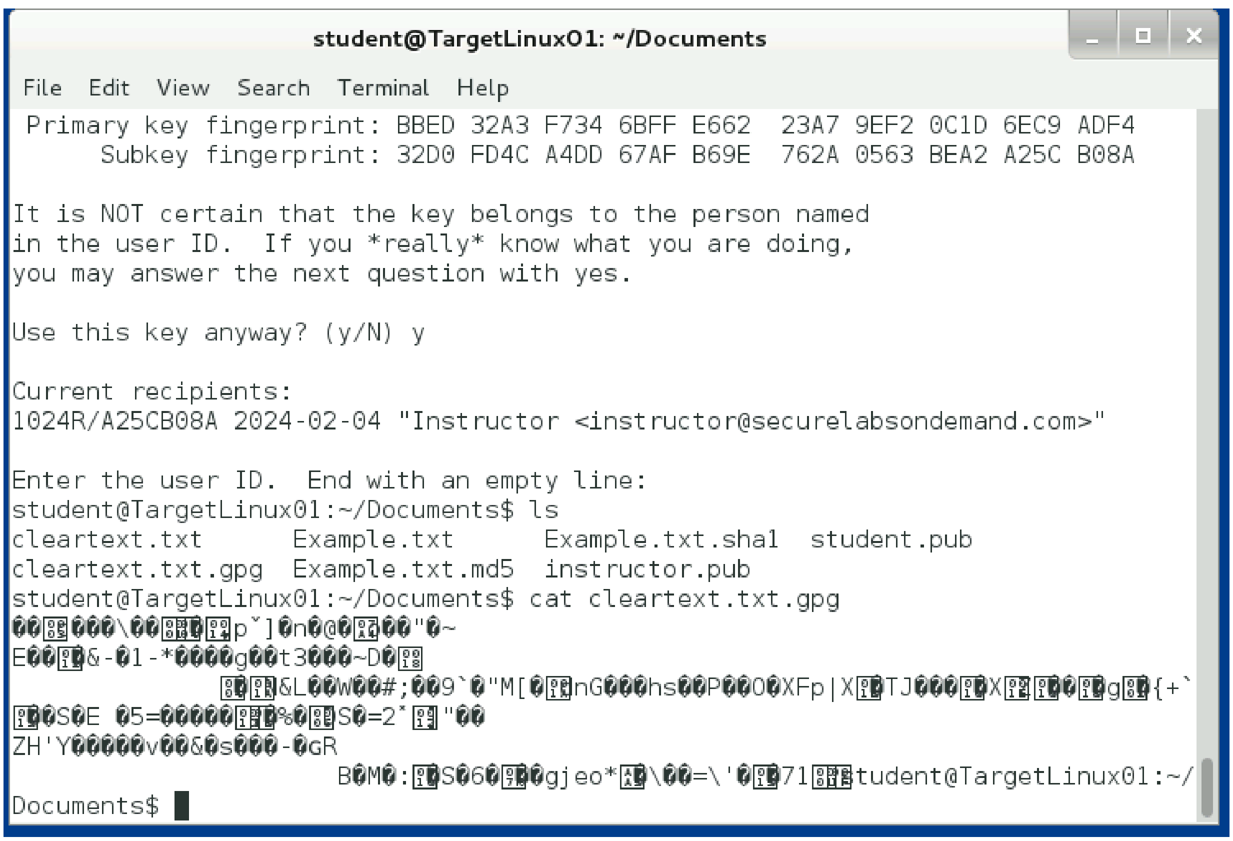


Figure 10: Encrypted File

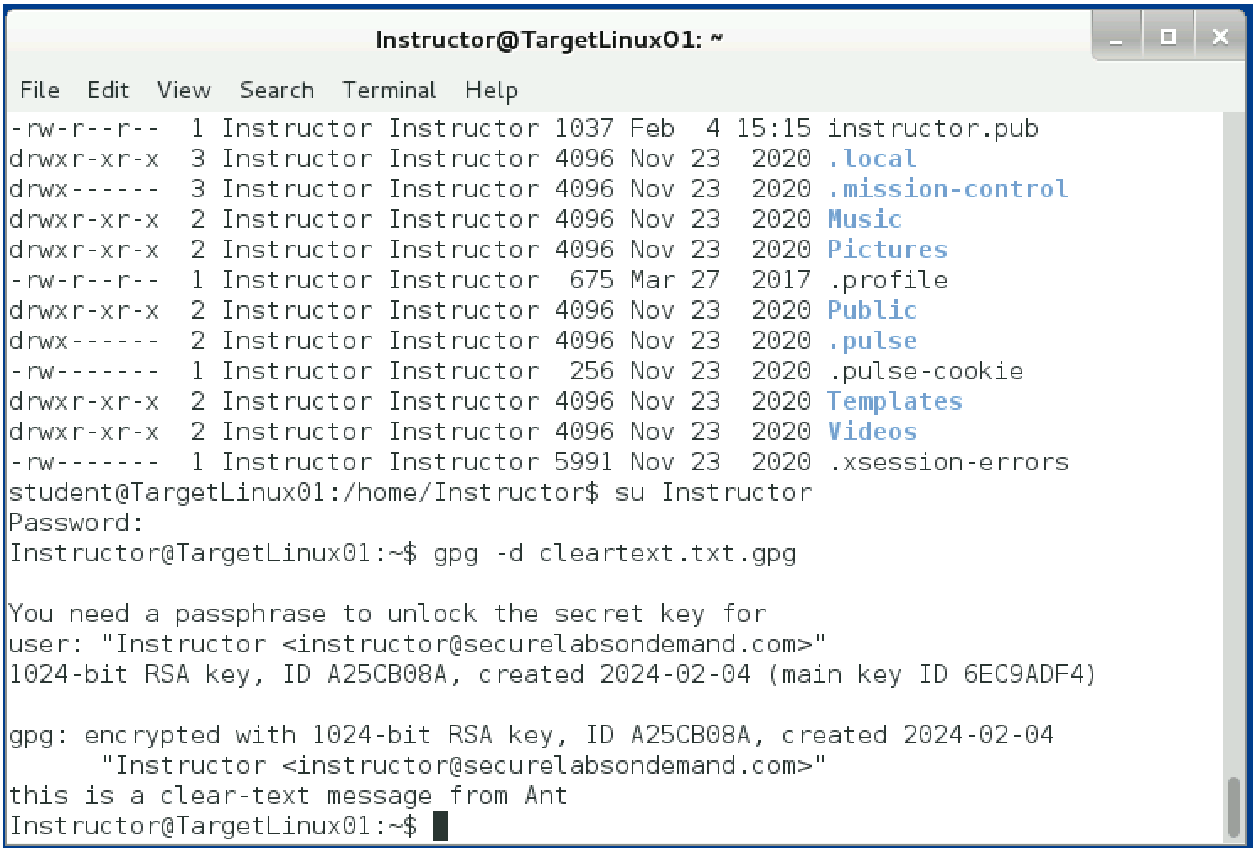


Figure 11: Contents of the decrypted cleartext.txt.gpg file

Section 2

A new text file, Example2.txt, is created in the student's Documents folder on the TargetLinux01 virtual machine. The process begins by connecting to the TargetLinux01 machine using PuTTY, ensuring the user is logged in as "student." The working directory is then changed to the student's Documents folder using the *cd* command (cd Documents), and a new file, Example2.txt, is created using the vi editor (vi Example2.txt).

Within the vi editor, the user enters edit mode by typing *i* and adds the text "This file is from yourname." (with "yourname" replaced by the user's own name). Exiting edit mode is done by pressing *Esc*, followed by *:wq!* to save changes and return to the command prompt. Afterward, the contents of Example2.txt are displayed using the command *cat Example2.txt*.

The md5sum tool is employed to create an MD5sum hash string for Example2.txt by executing the command *md5sum Example2.txt*. The generated hash string is stored in a new file, Example2.txt.md5, through the command *md5sum Example2.txt > Example2.txt.md5*. The files in the Documents folder are listed with *ls*, and the content of the newly created Example2.txt.md5 file is displayed using *cat Example2.txt.md5*.

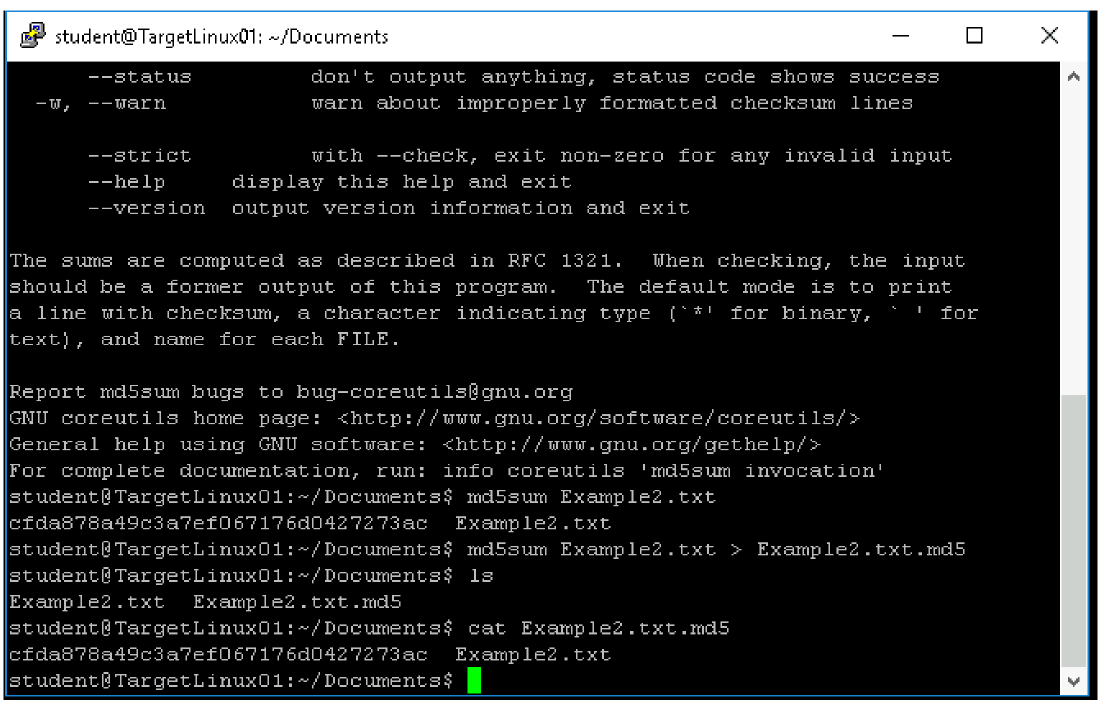
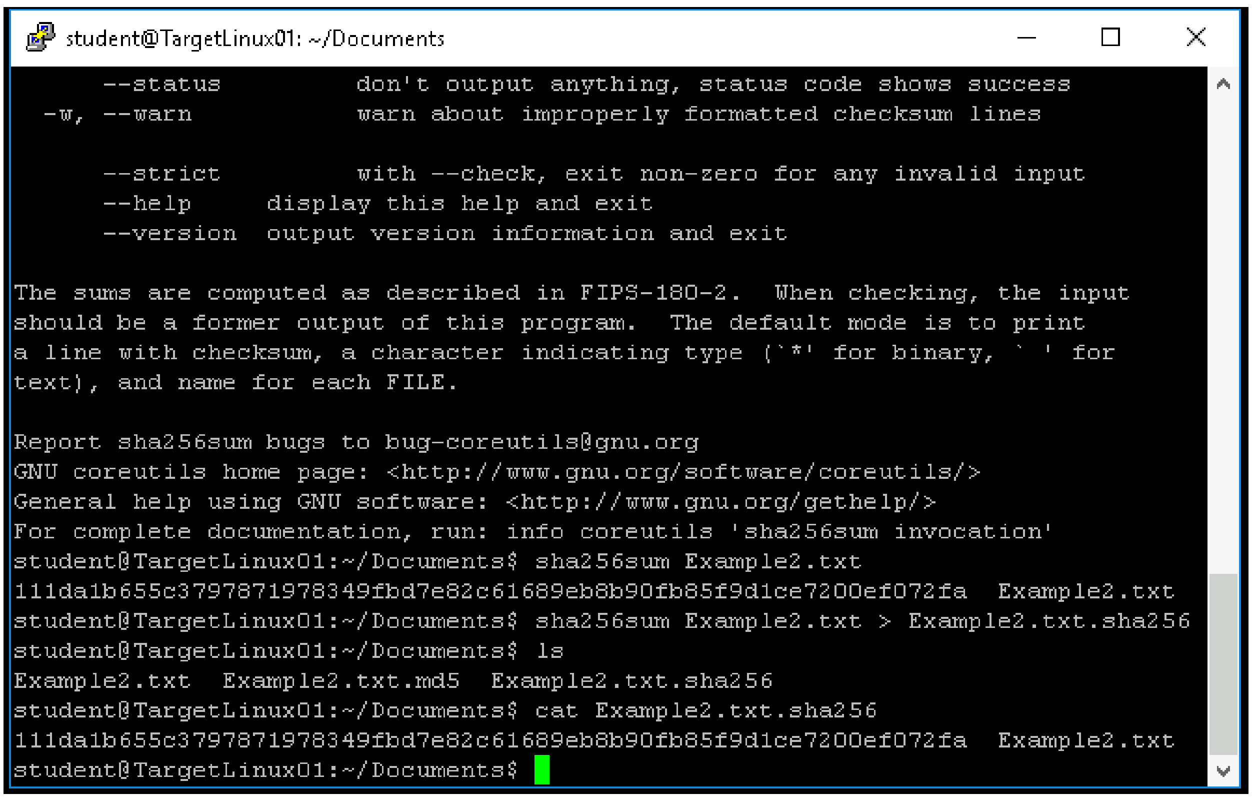


Figure 12: Contents of Example2.txt.md5 file

SHA256sum

A SHA—256sum hash string will be generated for the Example2.txt file. The sha256sum help document was accessed by executing the command *sha256sum --help* at the command prompt. Following this, the hash string was produced using the command *sha256sum Example2.txt*.

To preserve this hash string in a newly created file, the command *sha256sum Example2.txt > Example2.txt.sha256* is used. To validate the successful creation of the new file, a list of files in the Documents folder was obtained with the command *ls*. Lastly, the contents of the Example2.txt.sha256 file were examined using the command *cat Example2.txt.sha256* to confirm the hash string's integrity.



```
student@TargetLinux01: ~/Documents
      --status         don't output anything, status code shows success
  -w, --warn           warn about improperly formatted checksum lines

      --strict         with --check, exit non-zero for any invalid input
      --help     display this help and exit
      --version  output version information and exit

The sums are computed as described in FIPS-180-2.  When checking, the input
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating type (`*' for binary, ` ' for
text), and name for each FILE.

Report sha256sum bugs to bug-coreutils@gnu.org
GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
General help using GNU software: <http://www.gnu.org/gethelp/>
For complete documentation, run: info coreutils 'sha256sum invocation'
student@TargetLinux01:~/Documents$ sha256sum Example2.txt
111da1b655c3797871978349fbd7e82c61689eb8b90fb85f9d1ce7200ef072fa  Example2.txt
student@TargetLinux01:~/Documents$ sha256sum Example2.txt > Example2.txt.sha256
student@TargetLinux01:~/Documents$ ls
Example2.txt  Example2.txt.md5  Example2.txt.sha256
student@TargetLinux01:~/Documents$ cat Example2.txt.sha256
111da1b655c3797871978349fbd7e82c61689eb8b90fb85f9d1ce7200ef072fa  Example2.txt
student@TargetLinux01:~/Documents$
```

Figure 13: Contents of Example2.txt.sha256 file

Modify a File and Verify Hash Values

Modifications are made to the Example2.txt file created earlier in this section of the lab. Additional text, "This example is testing hash values," was appended to the file using the command *echo "This example is testing hash values." >> Example2.txt*, altering its contents. The modified content of Example2.txt is then displayed using the command *cat Example2.txt* to confirm.

To generate and overwrite the MD5sum hash for the modified Example2.txt file, the command *md5sum Example2.txt > Example2.txt.md5* is executed. The SHA—256sum hash for the modified file is also created and the existing Example2.txt-sha256 file is replaced with the command *sha256sum Example2.txt > Example2.txt.sha256*.

To ensure the successful update of the modified files, the contents of Example2.txt.md5 and Example2.txt.sha256 are displayed using the *commands cat Example2.txt.md5* and *cat Example2.txt.sha256*, respectively.

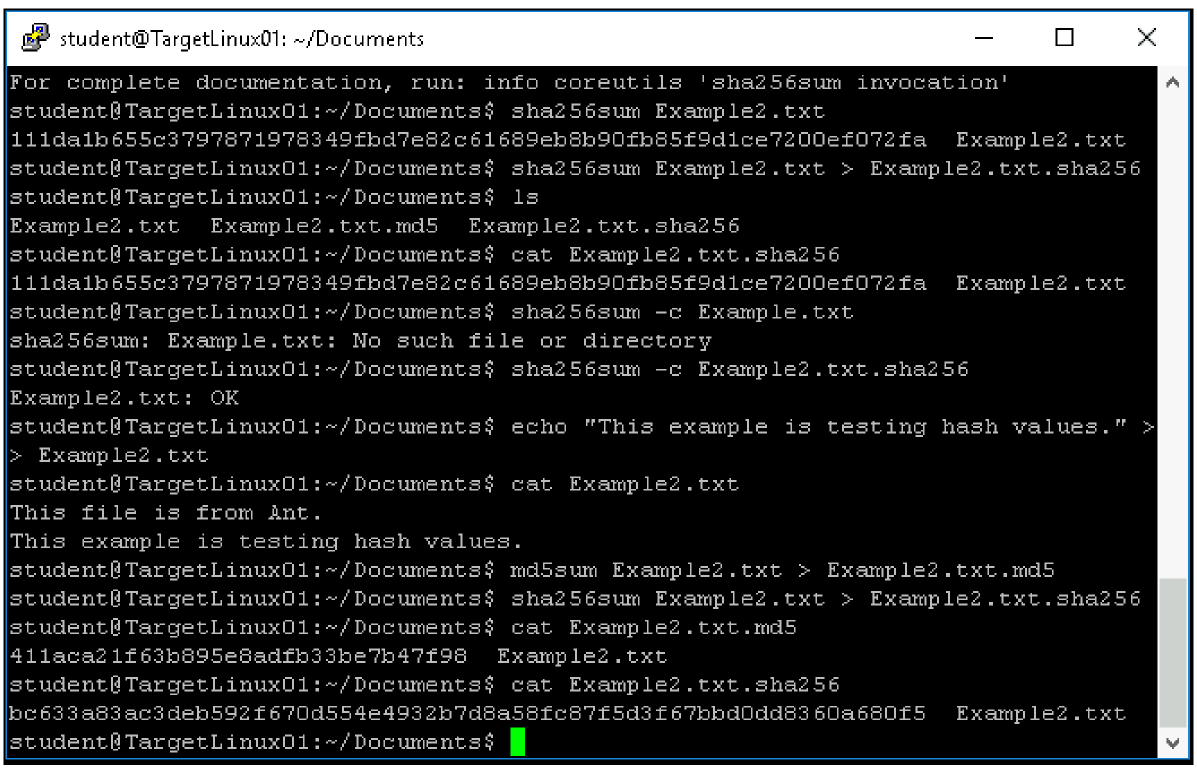


Figure 14: Modified md5sum and sha256 hash strings

Generate GnuPG Keys

In the ensuing steps, GnuPG (GNU Privacy Guard) was employed to encrypt a message using the RSA (Rivest-Shamir-Adleman) algorithm, securing communication between users, Instructor and Student. The process initiated with the generation of a GnuPG key from the Student account on the TargetLinux01 machine. The command *gpg --gen-key* was executed to generate a public encryption key, and subsequent on-screen prompts were addressed for user identification.

To enhance the generation process, a new PuTTY connection was established, and the script *./entropy_loop.sh* was executed in the new PuTTY window to generate entropy. Afterward, the windows were resized for visibility, and upon completion, the second PuTTY connection was closed, leaving the original PuTTY window open. The student's new GnuPG key was saved to a file, student2.pub, using the command *gpg --export -a > student2.pub*. The student's public key ring was then listed with *gpg --list-keys*.

The procedure continued with the creation of the GnuPG key for the Instructor account on the TargetLinux02 machine. Steps were repeated with specific alterations, including changing the real name to "Instructor2" and the email address. The key for the Instructor account was saved to a new file, instructor2.pub, with the command *gpg --export -a > instructor2.pub*. The keys in the instructor's public key ring were listed with *gpg --list-keys*. The keys are then traded using the KeyTransfers and the permissions set so that the users can access the files.

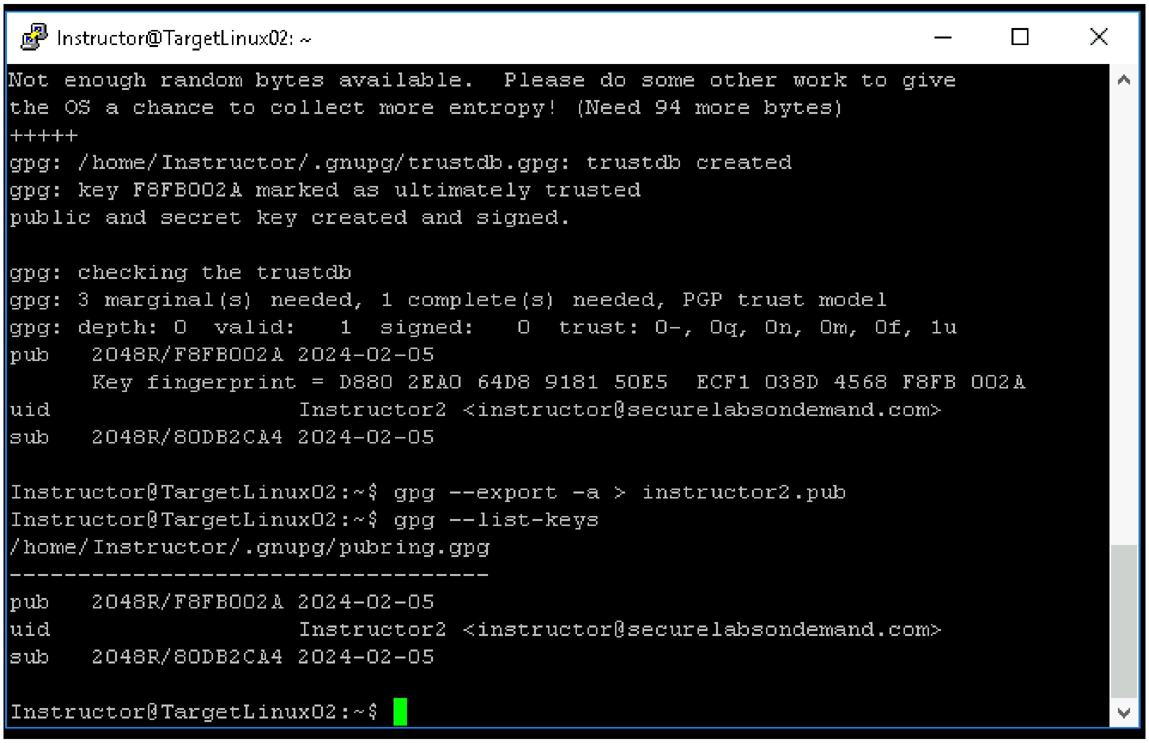


Figure 15: Instructor's public key ring

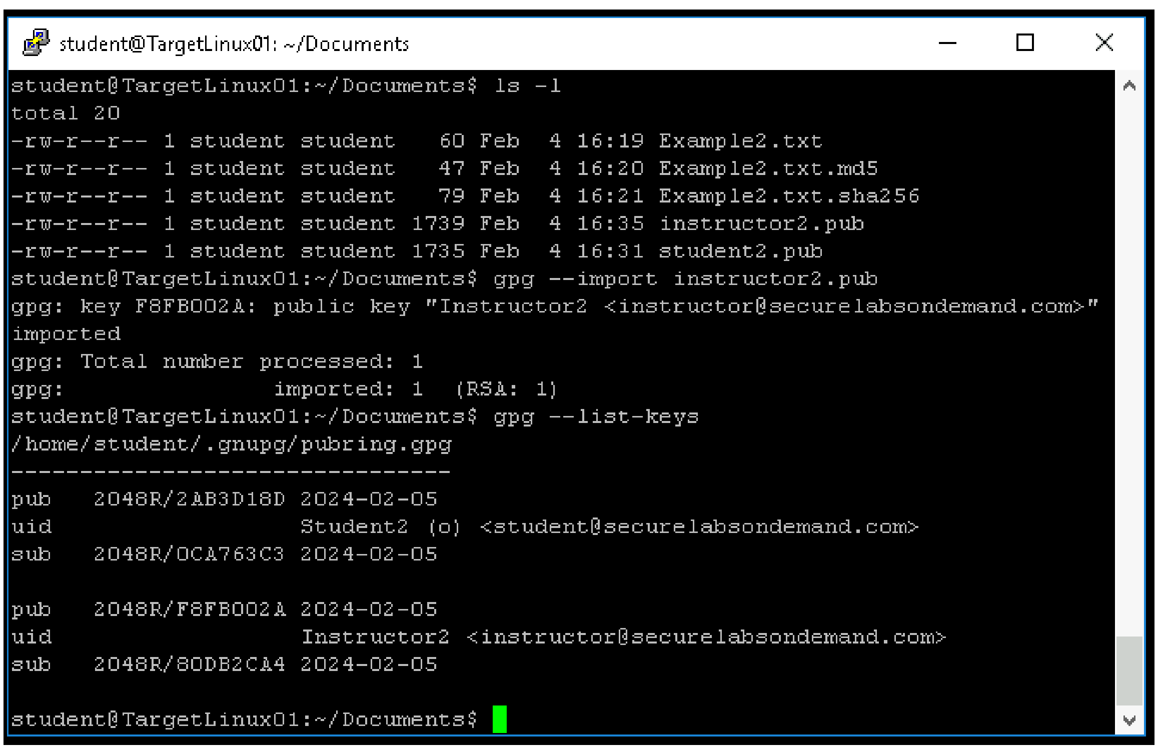


Figure 16: Student's pubic key ring updated

Encrypt and Decrypt a ClearText Message

The exchanged keys were utilized to encrypt and decrypt a clear-text message between the Instructor and Student users. With the command *gpg --help* to review the GnuPG tool's options.

The command *echo "This clear-text message is from yourname" > cleartext2.txt* was executed to create a clear-text message, and its contents were displayed using *cat cleartext2.txt*. Encryption was then applied with the command *gpg -e cleartext2.txt*. Throughout the encryption process, responses were provided, including the recipient's user ID (Instructor2) and confirmation to use the key. The encrypted file, cleartext2.txt.gpg, was verified with *ls* and its contents inspected with *cat cleartext2.txt.gpg*.

To facilitate the transfer of the encrypted message from the student's TargetLinux01 machine to the instructor's TargetLinux02 machine, the KeyTransfer shortcuts in the Connections folder were used. Using WinSCP, the encrypted file was copied first to the vWorkstation and subsequently to the TargetLinux02 machine. Permissions for the file were adjusted to make the instructor the owner.

Returning to the TargetLinux02 PuTTY connection, the decryption process commenced with *gpg -d cleartext2.txt.gpg*. The passphrase associated with the encryption certificate was provided when prompted.

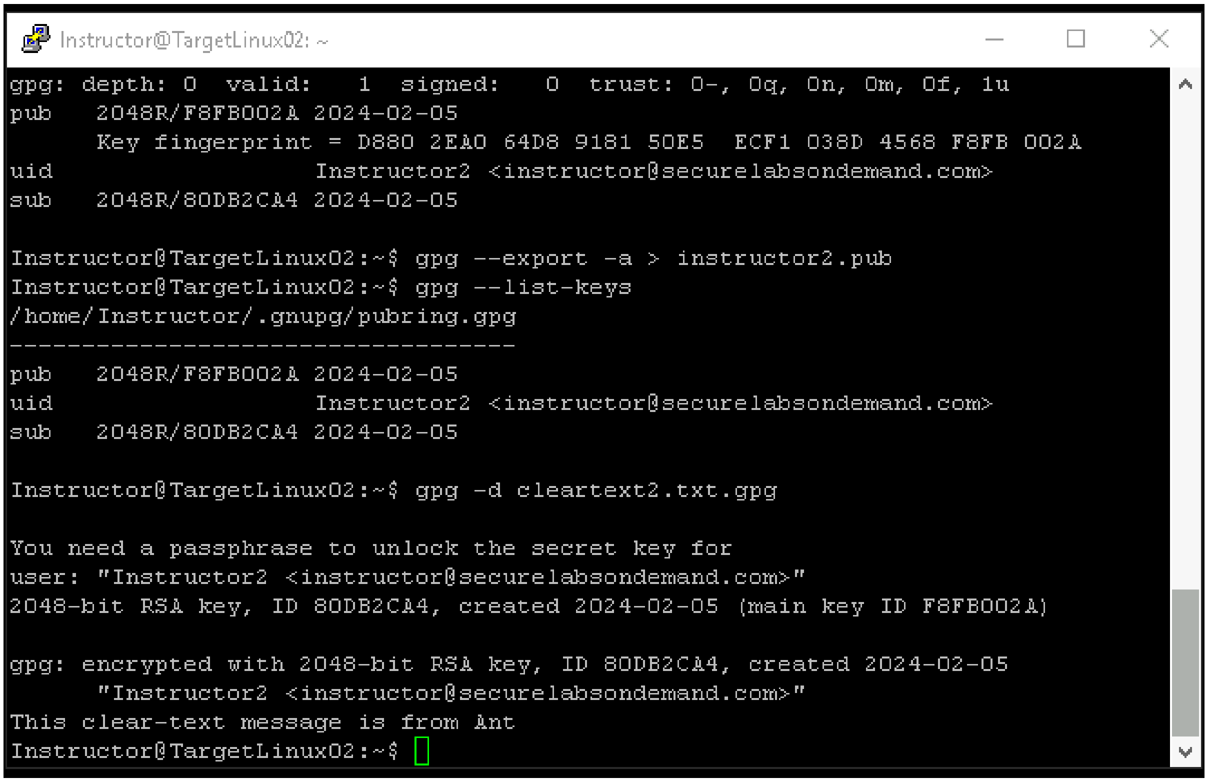


Figure 17: Contents of the decrypted cleartext2.txt.gpg file

Section 3

Part 1: Analysis of RSA and ECDSA Encryption

Aspect	RSA Encryption	ECDSA Encryption
Key Generation	Relies on mathematical difficulty of factoring large composite numbers	Relies on difficulty of solving the elliptic curve discrete logarithm problem
Key Length	Generally uses longer key lengths (2048 bits or higher) for equivalent security	Short key lengths are sufficient for equivalent security (256 bits)
Computational Complexity	Intensive, especially for key generation and decryption	Generally faster, more efficient for certain applications
Signature Size	Signature is larger compared to ECDSA	Compact, more suitable for bandwidth-constrained environments
Performance	Slower performance in key generation and decryption, suitable for less frequent operations	Generally faster performance, more suitable for real-time operations
Application	Commonly used in digital signatures, key exchange, and secure communication	Frequently used in blockchain technology and scenarios with resource constraints
Example products	PGP (Pretty Good Privacy), SSL/TLS for secure communications, and SecurID	Bitcoin and Ethereum blockchains, digital wallets, and IoT devices (like Google Home, Echo, etc.)

(Cobb, 2021), (Rivest, Shamir, & Adleman, 1978), (Johnson, Menezes, & Vanstone, 2014)

Part 2: Tools and Commands

a & b

I expedited the process a little by adding the md5sum hash and recording the string at the same time. I also used *whoami* to verify the machine and user account I was on.

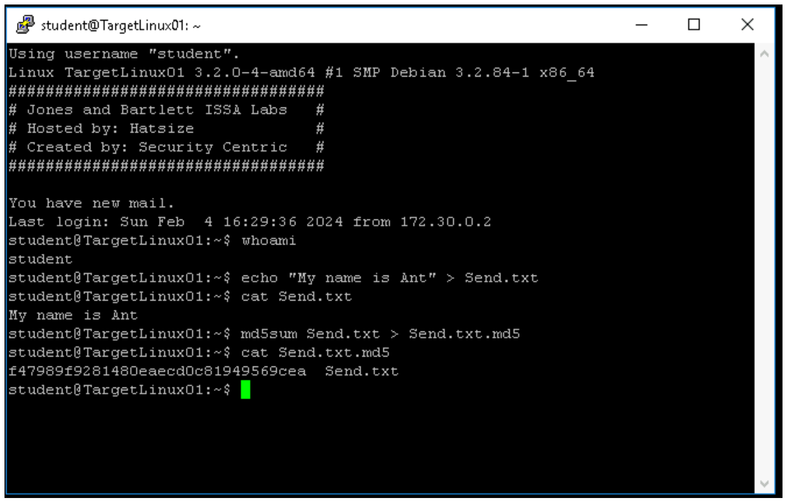


Figure 18: Hashed and saved to Send.txt.md5

c

I proceeded to use keytranfer11 & keytransfer12 to move the new Send.txt, Send.txt.md5 and student.pub to the Instructor machine, as well as changed the permissions of the files to the Instructor.

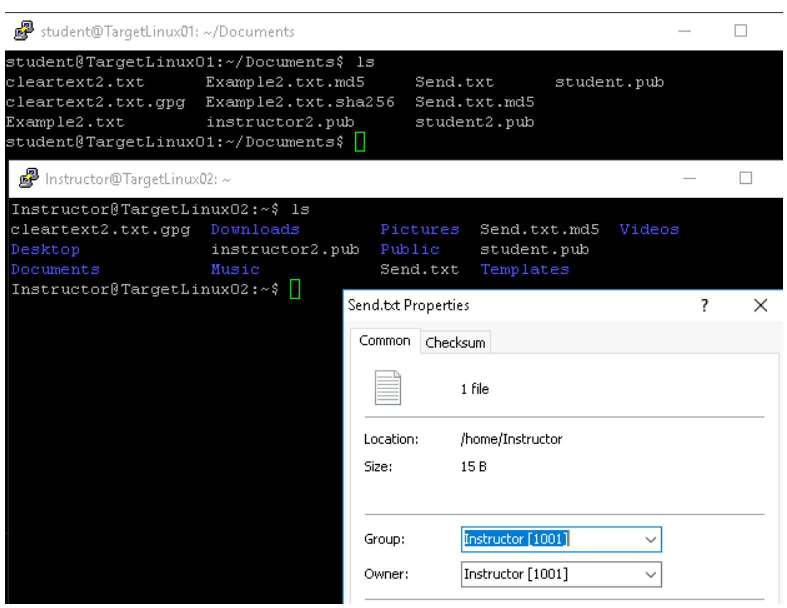


Figure 19: Documents transferred successfully.

d

Because I still had the other keys from earlier, I actually had to delete the Student2 key because the Instructor was trying to use the 'old' Student2 key instead of my new Student key generated in step c. I deleted it from the student VM as well, using *gpg –delete-key Student2* to do so. Then I used *gpg -e Send.txt* to encrypt the file.

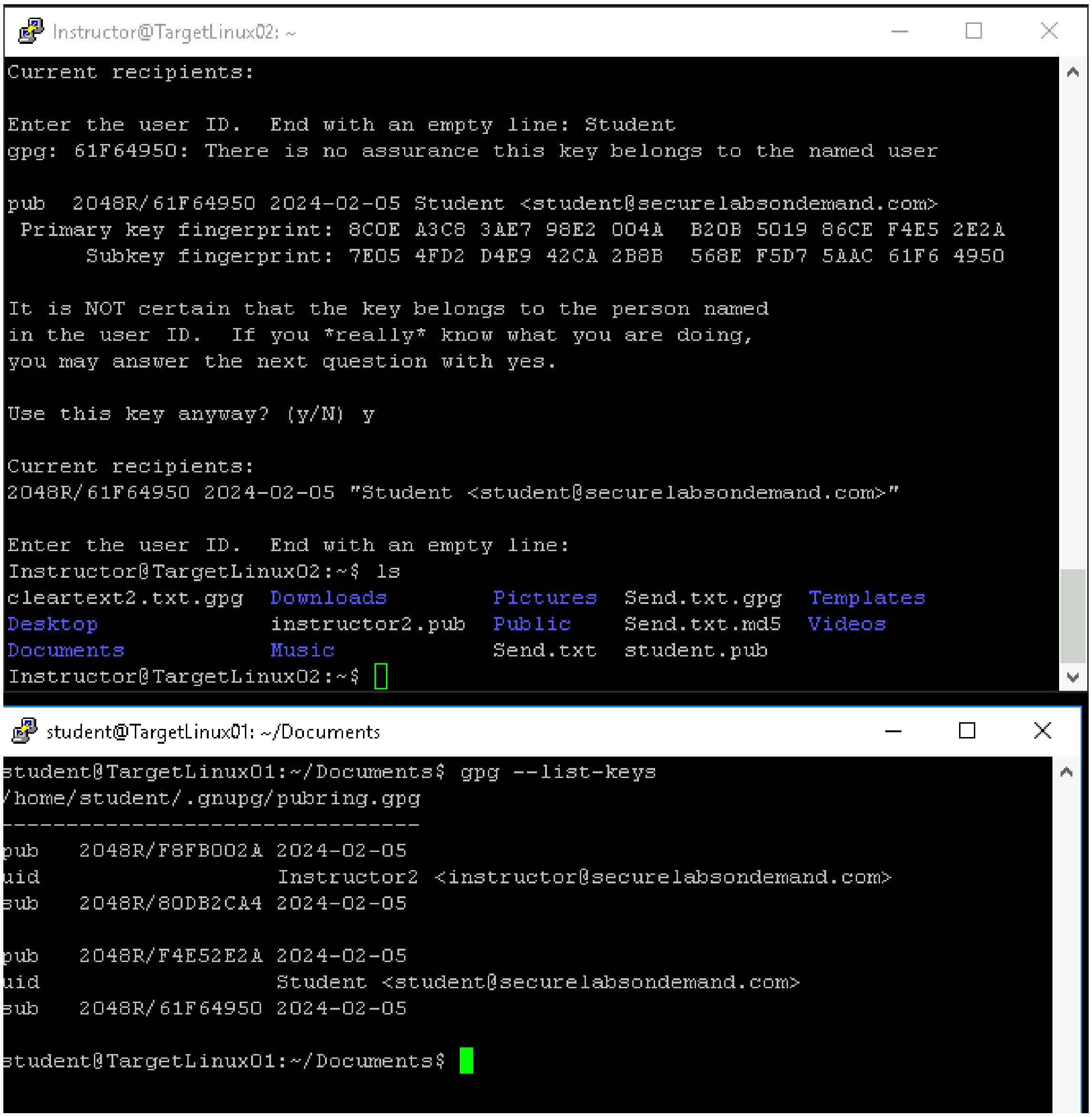


Figure 20: Student key and Instructor2 key is all that remains. The Send.txt file has been encrypted.

e

I copied the encrypted file to the student machine and changed the permissions, and then decrypted the Send.txt.gpg file after a small typo error entering in the passphrase.

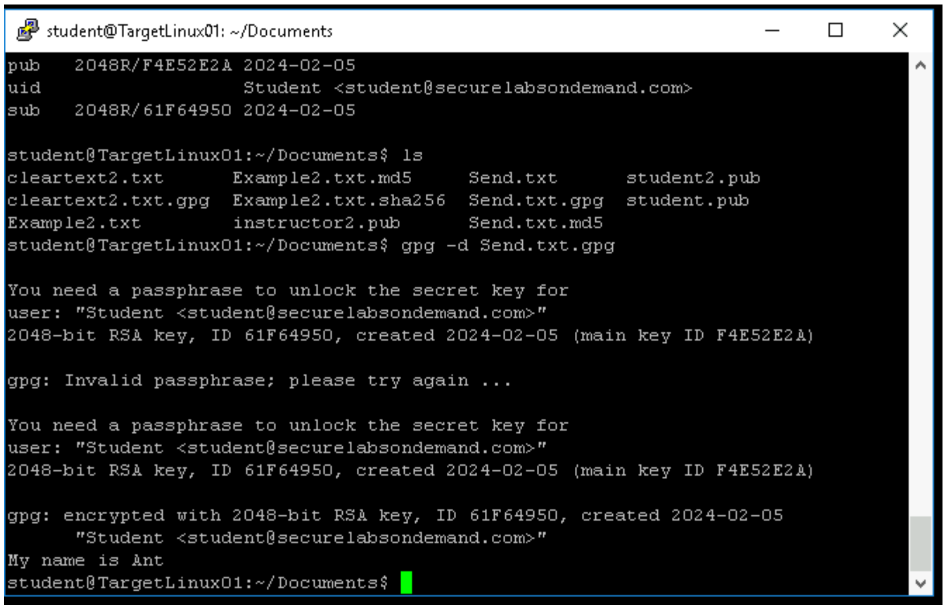


Figure 21: A freshly decrypted Send.txt.gpg

f

While the decryption command, *gpg -d Send.txt.gpg* can be view in Figure 21 above, the *gpg -e Send.txt.gpg* command used to encrypted the file was much higher up in the terminal lines in Figure 20. Below, in Figure 22, is a screenshot of the command line used to encrypt the file.

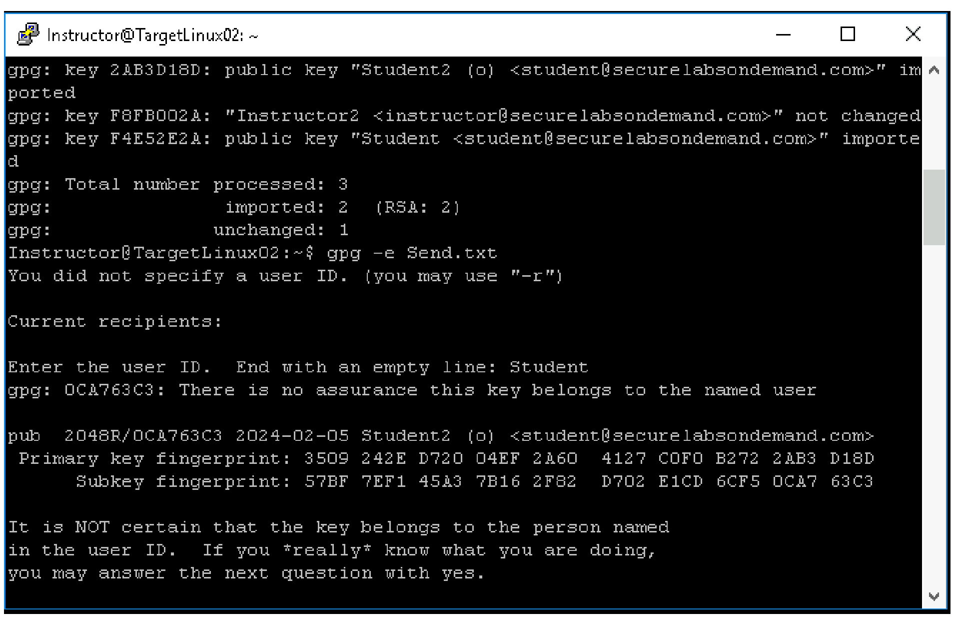


Figure 22: The command that encrypted the file

Part 3

a&b

On the student device I used *echo "This is a test of AE256 encryption." > ant.txt* to create the file with the text provided. After browsing a couple forums I was able to encrypt the file with the following command: *gpg –cipher-algo AES256 –symmetric ant.txt* and then used *rm ant.txt* to remove the original file. I used *ls* command to confirm.

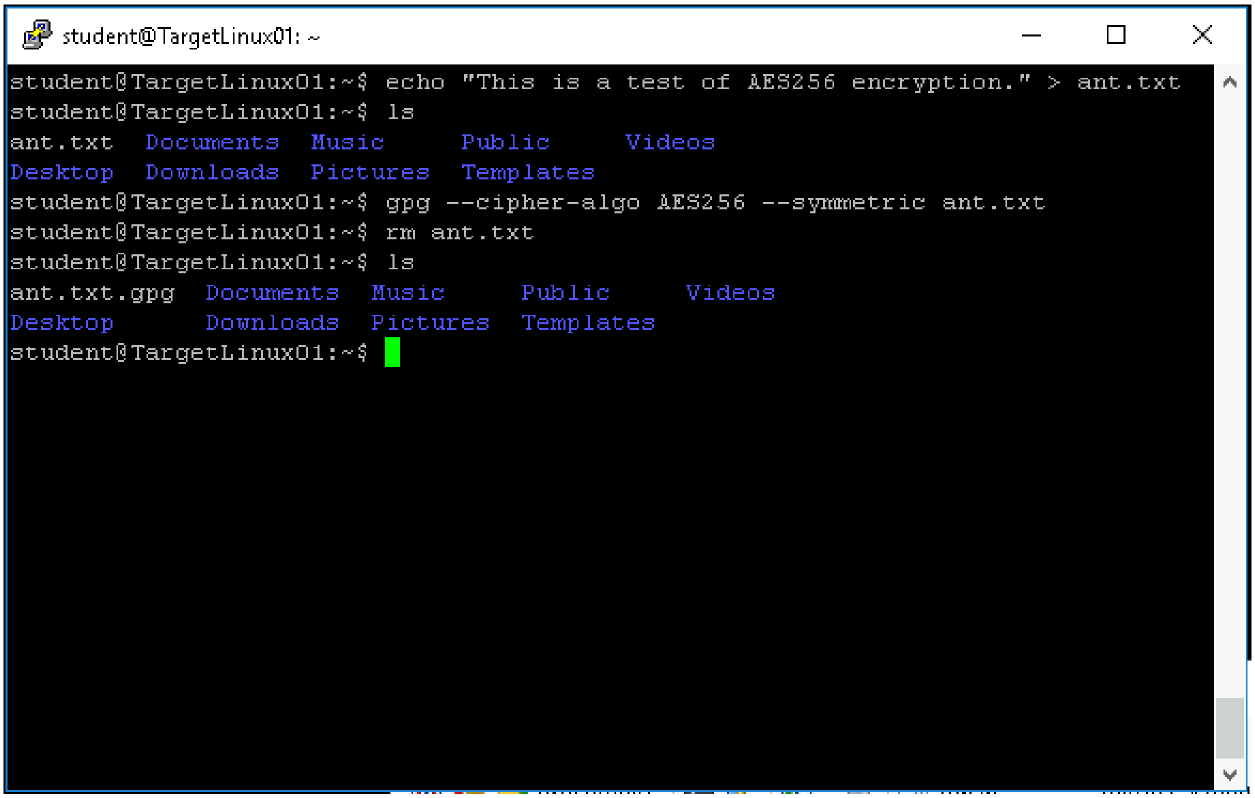


Figure 23: ant.txt.gpg all alone in the home folder

C

Using the KeyTransfer11 and KeyTransfer12 connections, I moved the ant.txt.gpg file over to the Instructor machine. After changing the permissions, and verifying that the file was moved successfully with *ls* I decrypt the file with the following command: *gpg -d ant.txt.gpg* and I am met with success!

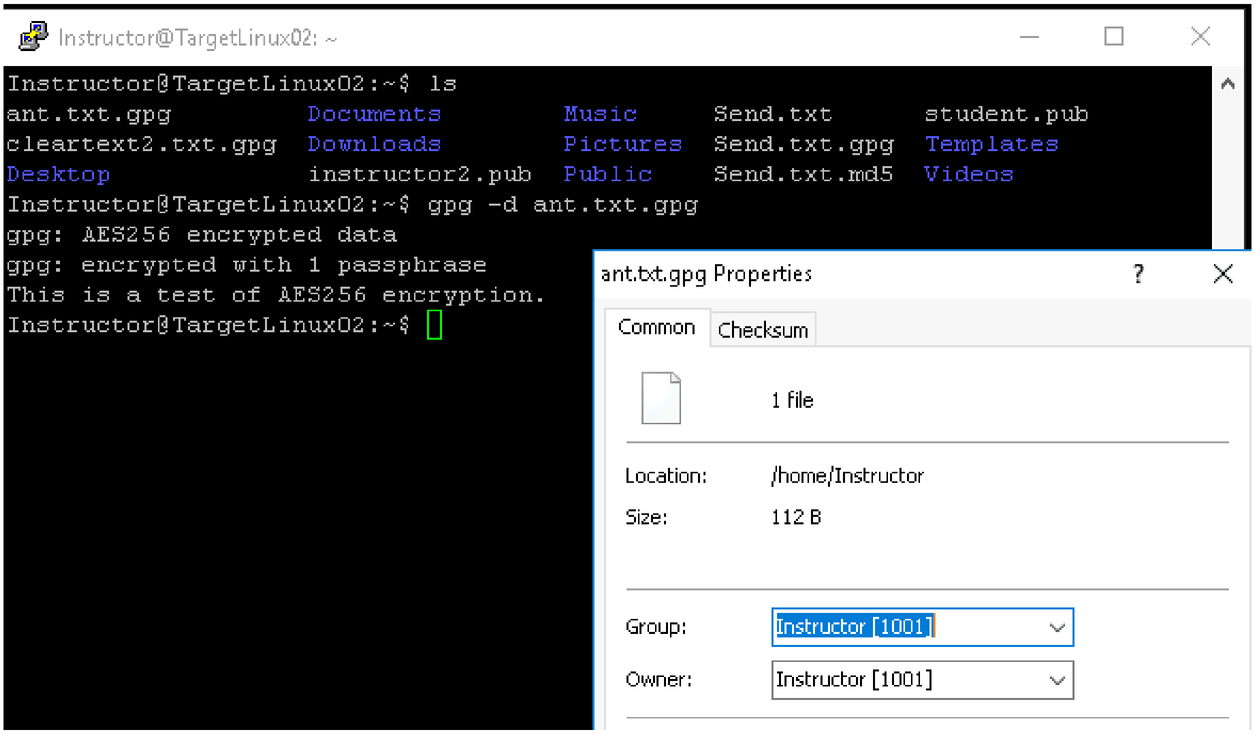


Figure 24: Moved, permitted and decrypted

Bibliography

Cobb, M. (2021, November n.d.). *RSA algorithm (Rivest-Shamir-Adleman)*. Retrieved from techtarget: https://www.techtarget.com/searchsecurity/definition/RSA

Johnson, D., Menezes, A., & Vanstone, S. (2014, January 31). *The Elliptic Curve Digital Signature Algorithm (ECDSA)*. Retrieved from University of Miami: https://www.cs.miami.edu/home/burt/learning/Csc609.142/ecdsa-cert.pdf

Rivest, R. L., Shamir, A., & Adleman, L. (1978, February 01). *A Method for Obtaining Digital*. Retrieved from MIT: https://people.csail.mit.edu/rivest/Rsapaper.pdf