

Nicholas Leoncito wdf712 lab 02

Nicholas Leoncito: wdf712

IS 3423-001 – Fall 22024

09/29/2024

Lab 02 | Applying Encryption and Hashing Algorithms for Secure Communications

INTRODUCTION

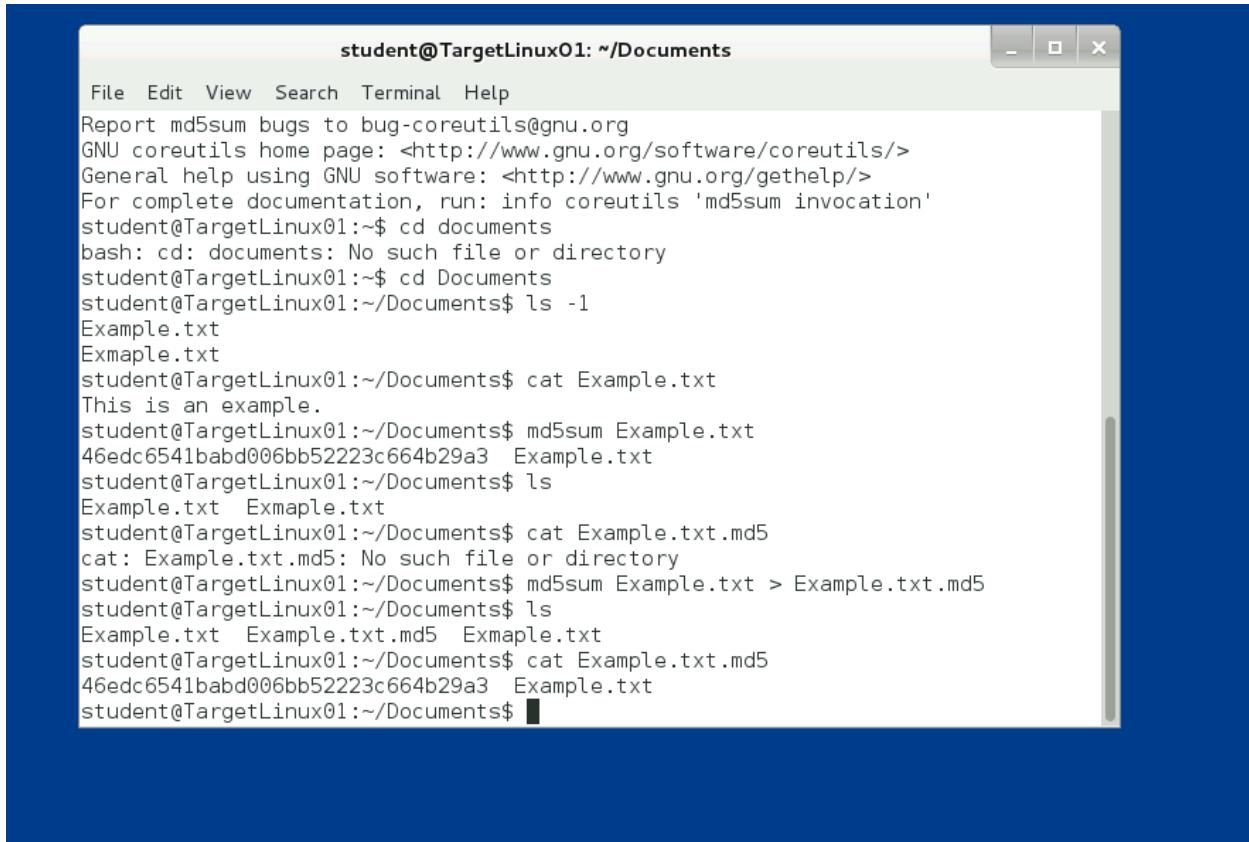
The purpose of this lab was to gain hands-on experience with encryption and decryption processes using GnuPG (GNU Privacy Guard) and RSA encryption, a widely-used public-key cryptographic system. Throughout the lab, we generated GnuPG keys for two fictitious users—Student and Instructor—exchanged public keys, and securely communicated via encrypted messages. This exercise provided valuable insight into how public and private keys function to ensure data confidentiality, integrity, and authenticity. Additionally, the lab explored the differences between cleartext and encrypted messages, allowing for a deeper understanding of the processes that protect sensitive information from unauthorized access. By simulating encryption and decryption tasks, this lab reinforced the fundamental principles of secure communication in a controlled environment.

Part 2: Create a MD5sum and a SHA1sum hash string

Step 6 – Create an MD5sum hash string for the Example.txt file:

Describe what you are doing in this step and provide your screenshot.

After running the command `md5sum Example.txt`, the system generated a unique MD5 hash string for the `Example.txt` file. At this point, I took a screenshot of the terminal displaying the MD5sum hash string to document the process for the lab report.



The screenshot shows a terminal window titled "student@TargetLinux01: ~/Documents". The terminal displays the following command-line session:

```
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
Example.txt
Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
student@TargetLinux01:~/Documents$ md5sum Example.txt
46edc6541babd006bb52223c664b29a3 Example.txt
student@TargetLinux01:~/Documents$ ls
Example.txt Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt.md5
cat: Example.txt.md5: No such file or directory
student@TargetLinux01:~/Documents$ md5sum Example.txt > Example.txt.md5
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt.md5
46edc6541babd006bb52223c664b29a3 Example.txt
student@TargetLinux01:~/Documents$
```

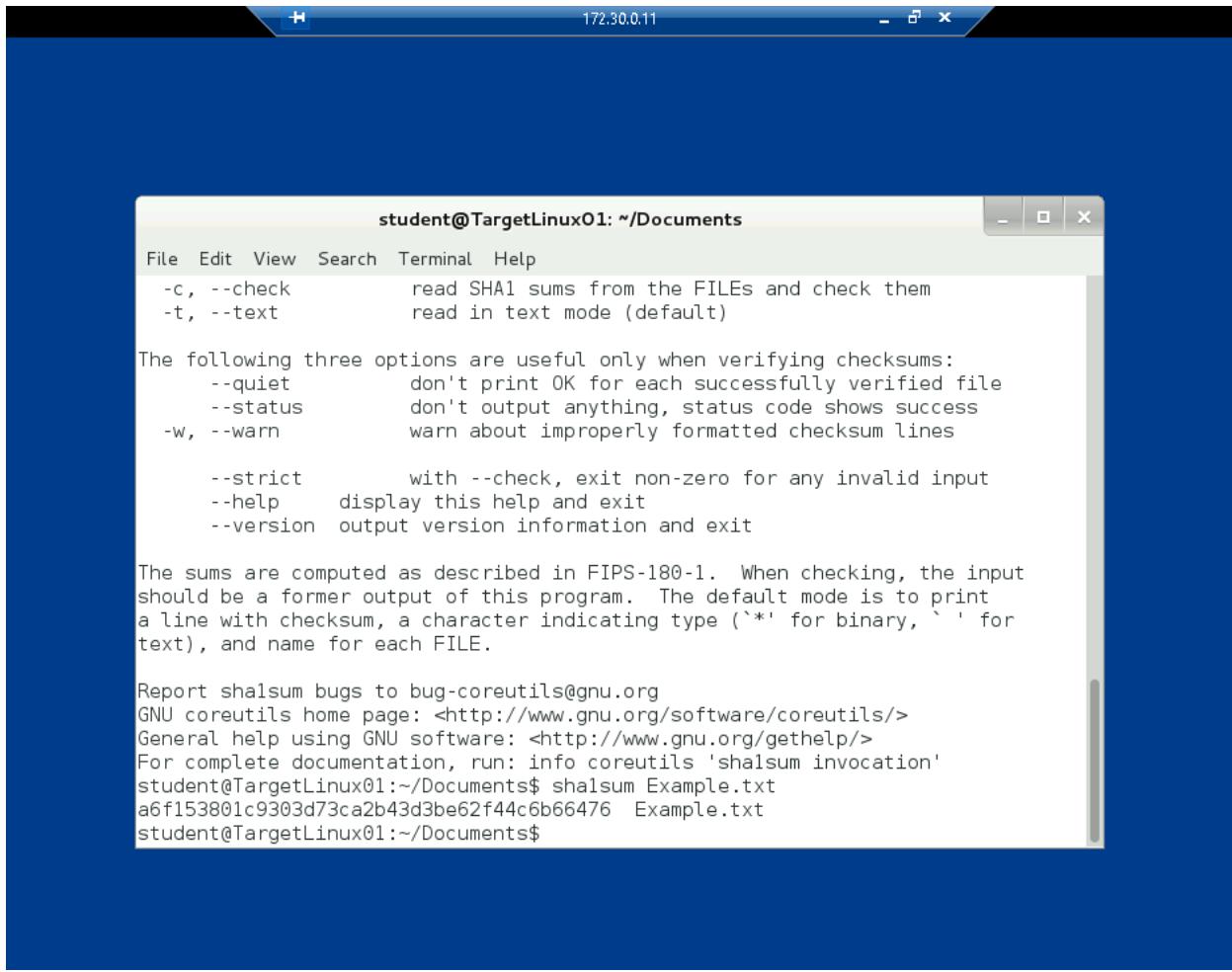
Step 10 – View the contents of Example.txt.md5:

After using the command `md5sum Example.txt > Example.txt.md5` to save the hash, I ran the `ls` command to verify that the new file `Example.txt.md5` had been added to the `Documents` folder. I took a screenshot showing the contents of the folder, including the newly created `.md5` file.

When I typed `cat Example.txt.md5`, the terminal displayed the contents of the `Example.txt.md5` file, which contained the same MD5 hash generated earlier. I took a screenshot showing the hash string in the `.md5` file for comparison and verification.

Step 14 – Create SHA1sum hash string for the Example.txt file:

Describe what you are doing in this step and provide your screenshot. After switching to the SHA1 hashing algorithm, I used the command `sha1sum Example.txt` to create a SHA1 hash string for the same file. The terminal returned a unique string, which I captured in a screenshot to document the SHA1 hash generation.



The terminal window shows the following text:

```
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$
```

Step 18 – View the contents of Example.txt.sha1:

After saving the SHA1 hash using sha1sum Example.txt > Example.txt.sha1 and listing the folder contents with ls, I took a screenshot showing the addition of the Example.txt.sha1 file in the Documents folder.

When I used cat Example.txt.sha1, the terminal displayed the SHA1 hash string stored in the Example.txt.sha1 file. I took a screenshot to confirm that the hash string matched the one generated earlier.

The screenshot shows a terminal window titled "student@TargetLinux01: ~/Documents". The window contains the following text:

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

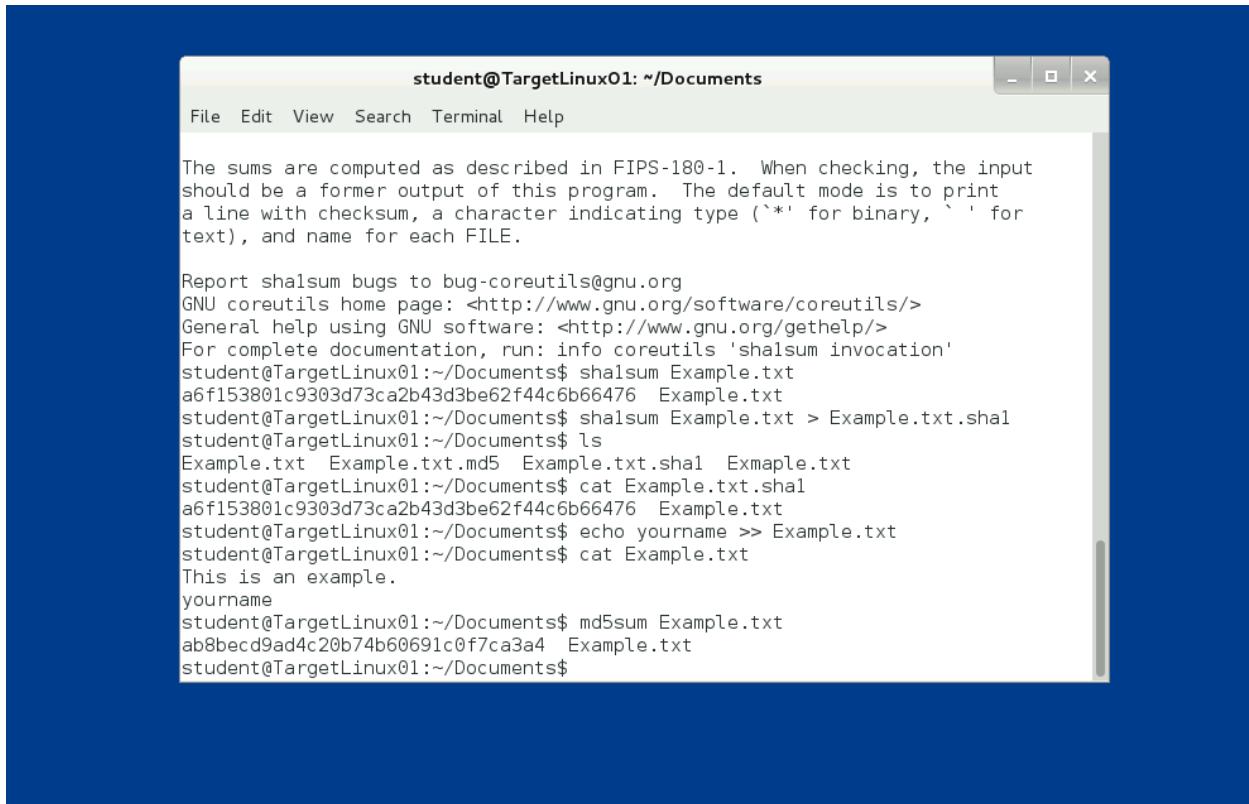
--strict          with --check, exit non-zero for any invalid input
--help           display this help and exit
--version         output version information and exit

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 shasum 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 'shasum invocation'
student@TargetLinux01:~/Documents$ shasum Example.txt
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$ shasum Example.txt > Example.txt.shal
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.shal Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt.shal
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$
```

I used the command echo yourname >> Example.txt (replacing "yourname" with my actual name) to add my name to the end of the Example.txt file. After running the command, I typed cat Example.txt to view the modified file contents, which now included my name at the end. I captured a screenshot of the terminal displaying the updated contents of the Example.txt file.

After modifying the file, I typed md5sum Example.txt to generate a new MD5 hash for the modified Example.txt. The tool returned a new hexadecimal hash string, which was different from the original hash due to the changes in the file's contents. I took a screenshot showing the new MD5sum hash string to document the change for the lab report.

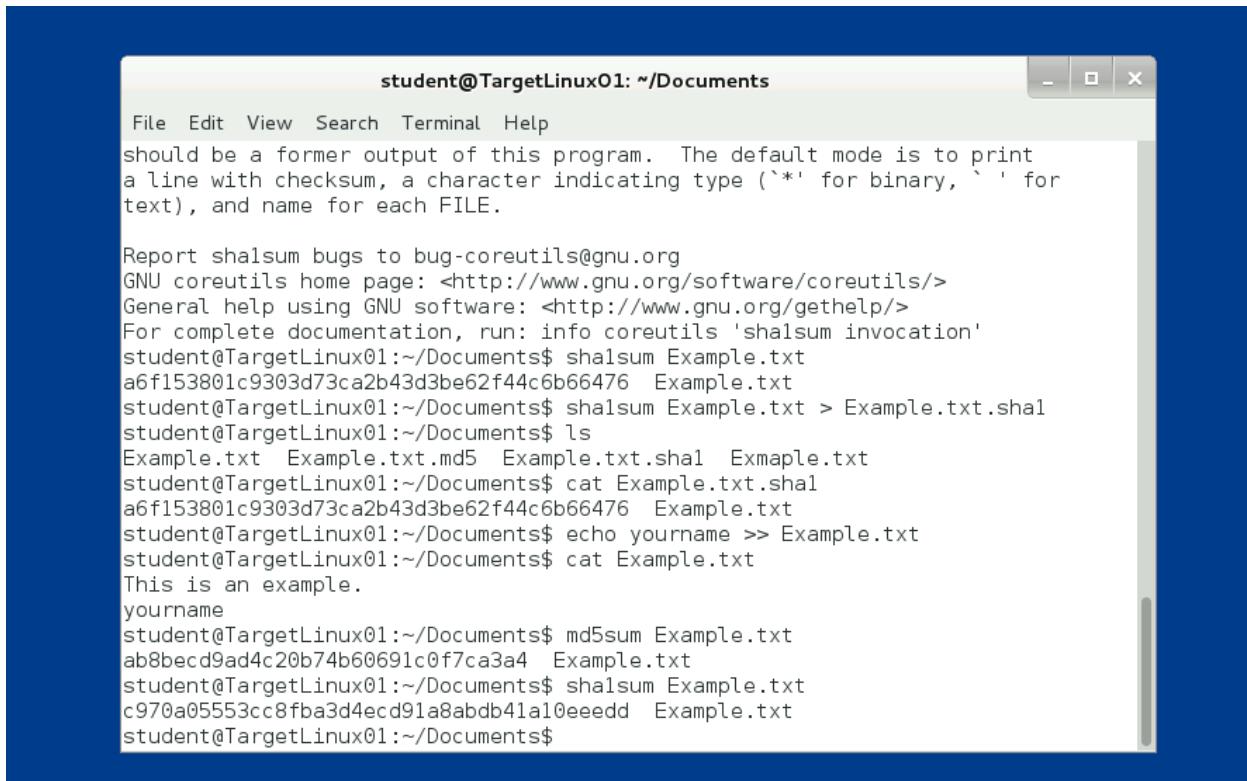


The screenshot shows a terminal window titled "student@TargetLinux01: ~/Documents". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area of the terminal displays the following text:

```
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 shasum 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 'shasum invocation'
student@TargetLinux01:~/Documents$ shasum Example.txt
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$ shasum Example.txt > Example.txt.sha1
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.sha1 Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt.sha1
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$ echo yourname >> Example.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
yourname
student@TargetLinux01:~/Documents$ md5sum Example.txt
ab8becd9ad4c20b74b60691c0f7ca3a4 Example.txt
student@TargetLinux01:~/Documents$
```

Finally, I typed `sha1sum Example.txt` to create a new SHA1 hash for the modified file. The SHA1 hash was also different from the original due to the added content. I captured a screenshot showing the new SHA1 hash string and included it in the lab report to demonstrate the effects of modifying the file on the integrity hashes.



The screenshot shows a terminal window titled "student@TargetLinux01: ~/Documents". The window has standard close, minimize, and maximize buttons at the top right. The terminal displays the following command-line session:

```
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 shasum 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 'shasum invocation'
student@TargetLinux01:~/Documents$ shasum Example.txt
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$ shasum Example.txt > Example.txt.shal
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.shal Exmaple.txt
student@TargetLinux01:~/Documents$ cat Example.txt.shal
a6f153801c9303d73ca2b43d3be62f44c6b66476 Example.txt
student@TargetLinux01:~/Documents$ echo yourname >> Example.txt
student@TargetLinux01:~/Documents$ cat Example.txt
This is an example.
yourname
student@TargetLinux01:~/Documents$ md5sum Example.txt
ab8becd9ad4c20b74b60691c0f7ca3a4 Example.txt
student@TargetLinux01:~/Documents$ shasum Example.txt
c970a05553cc8fba3d4ecd91a8abdb41a10eeeeed
student@TargetLinux01:~/Documents$
```

I started by verifying that I was logged in as the student by checking the command prompt (student@TargetLinux01:~/Documents\$). Then, I initiated the GPG key generation process by typing gpg --gen-key. I followed the prompts by selecting the default RSA key type, setting the key size to 1024 bits, and choosing to make the key valid indefinitely. After entering the necessary information, including "Student" as the real name and email address, I input the passphrase ("today is a nice day") as prompted. When the system generated the error message about insufficient random bytes, I captured the screenshot showing the "Not enough random bytes available" error.

```
student@TargetLinux01: ~/Documents
File Edit View Search Terminal Help
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
.....+++++
+++++
gpg: /home/student/.gnupg/trustdb.gpg: trustdb created
gpg: key A475782E marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
pub 1024R/A475782E 2024-09-30
      Key fingerprint = 8EAB E945 1629 E113 2E71 D0D2 EA93 5E52 A475 782E
uid             Student <student@securelabsondemand.com>
sub 1024R/CF63B4AB 2024-09-30

student@TargetLinux01:~/Documents$ gpg -- export -a > student.pub
usage: gpg [options] [filename]
student@TargetLinux01:~/Documents$ pwd
/home/student/Documents
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.sha1 Exmaple.txt student.pub
student@TargetLinux01:~/Documents$
```

To generate enough entropy for the key pair, I opened a second terminal window and ran the ./entropy_loop.sh script. I also played a game of Solitaire (via Applications > Games > AisleRiot Solitaire) to increase entropy. Once enough random bytes were generated and the command prompt returned in the first terminal window, I captured a screenshot showing that the key pair had been successfully created.

After generating the key, I exported the GPG public key for the student account using the command gpg --export -a > student.pub. I then confirmed the current directory by typing pwd to ensure I was in /home/student/Documents. Using the ls command, I verified that the student.pub file had been saved correctly. I took a screenshot showing the contents of the /home/student/Documents folder, including the student.pub file, for the lab report.

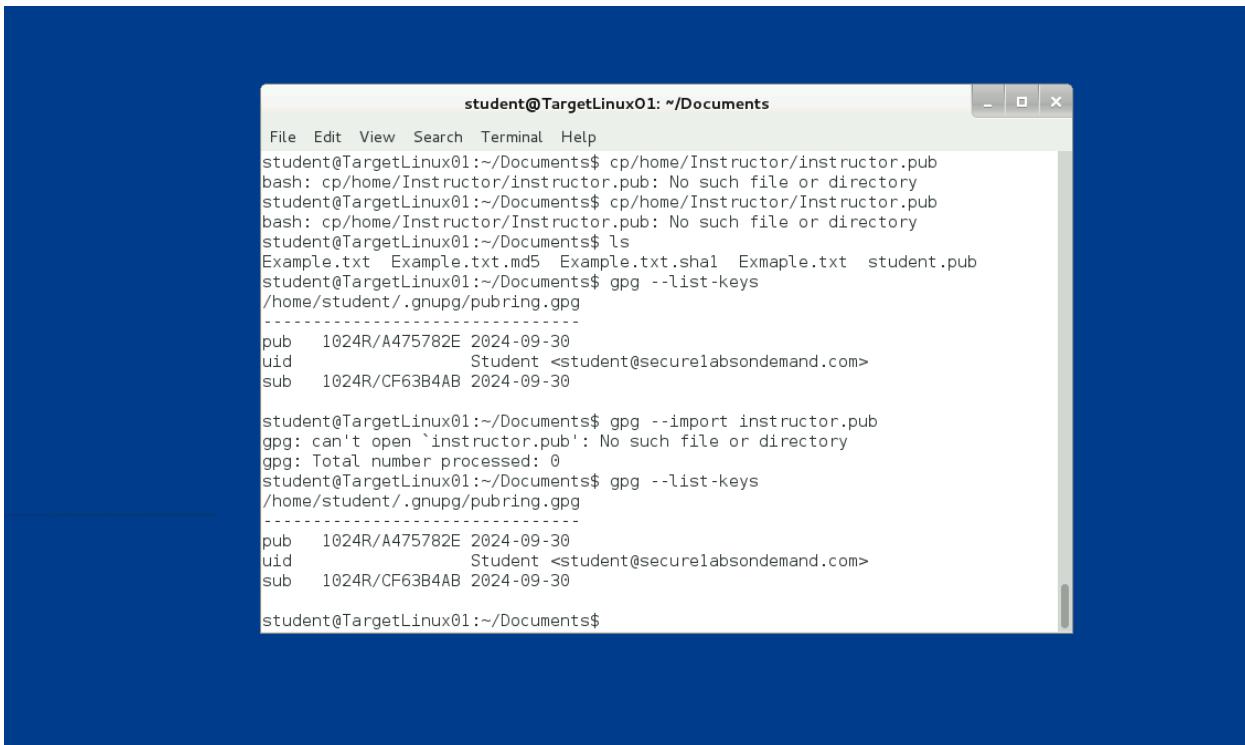
```
Instructor@TargetLinux01: ~
File Edit View Search Terminal Help
usage: gpg [options] [filename]
student@TargetLinux01:~/Documents$ pwd
/home/student/Documents
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.shal Exmaple.txt student.pub
student@TargetLinux01:~/Documents$ su instructor
No passwd entry for user 'instructor'
student@TargetLinux01:~/Documents$ su Instructor
Password:
Instructor@TargetLinux01:/home/student/Documents$ cd /home/Instructor
Instructor@TargetLinux01:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
Instructor@TargetLinux01:~$ gpg --export -a > instructor.pub
gpg: directory `/home/Instructor/.gnupg' created
gpg: new configuration file `/home/Instructor/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/Instructor/.gnupg/gpg.conf' are not yet active
uring this run
gpg: keyring `/home/Instructor/.gnupg/secring.gpg' created
gpg: keyring `/home/Instructor/.gnupg/pubring.gpg' created
gpg: WARNING: nothing exported
Instructor@TargetLinux01:~$ ls
Desktop Downloads Music Public Videos
Documents instructor.pub Pictures Templates
Instructor@TargetLinux01:~$
```

Next, I switched to the Instructor account by typing su Instructor and entering the password. After navigating to the /home/Instructor directory with cd /home/Instructor, I listed the folder contents using the ls command and captured a screenshot showing the files in the Instructor folder.

I repeated the steps to generate a GnuPG key for the Instructor account. This involved entering "Instructor" as the real name, providing the appropriate email address, and following the key generation prompts. Once the key pair was generated, I exported the Instructor's GPG public key using gpg --export -a > instructor.pub and verified that the instructor.pub file had been saved by listing the folder contents. I captured a screenshot showing the files in the /home/Instructor folder, including the instructor.pub file.

Finally, I typed exit to switch back to the student account, confirming that the command prompt had returned to student@TargetLinux01:~/Documents\$.

Keys list



The screenshot shows a terminal window titled "student@TargetLinux01: ~/Documents". The user runs several commands related to GPG key management:

```
student@TargetLinux01:~/Documents$ cp/home/Instructor/instructor.pub
bash: cp/home/Instructor/instructor.pub: No such file or directory
student@TargetLinux01:~/Documents$ cp/home/Instructor/Instructor.pub
bash: cp/home/Instructor/Instructor.pub: No such file or directory
student@TargetLinux01:~/Documents$ ls
Example.txt Example.txt.md5 Example.txt.shal Exmaple.txt student.pub
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
-----
pub 1024R/A475782E 2024-09-30
uid          Student <student@securelabsondemand.com>
sub 1024R/CF63B4AB 2024-09-30
-----
student@TargetLinux01:~/Documents$ gpg --import instructor.pub
gpg: can't open `instructor.pub': No such file or directory
gpg: Total number processed: 0
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
-----
pub 1024R/A475782E 2024-09-30
uid          Student <student@securelabsondemand.com>
sub 1024R/CF63B4AB 2024-09-30
student@TargetLinux01:~/Documents$
```

SECTION 3: LAB CHALLENGE AND ANALYSIS

RSA (Rivest-Shamir-Adleman) and ECDSA (Elliptic Curve Digital Signature Algorithm) are both public-key encryption algorithms, but they differ in their efficiency and underlying mathematics. RSA, one of the oldest encryption methods, relies on the difficulty of factoring large integers and typically uses key sizes of 2048 or 4096 bits. While widely supported and easy to implement, RSA becomes inefficient with larger key sizes, requiring more computational power. On the other hand, ECDSA is based on elliptic curve cryptography (ECC), providing the same security level as RSA with much smaller key sizes, making it faster and more efficient, especially for mobile and resource-constrained devices. ECDSA's complexity and lower support are its main weaknesses compared to RSA, but its adoption is growing. RSA is commonly used in SSL/TLS certificates for securing websites, while ECDSA is used in Bitcoin for signing transactions.

References:

- Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM.
- NIST. (2013). Recommendation for Key Management: Part 1. National Institute of Standards and Technology.
- Schneier, B. (2015). Applied Cryptography: Protocols, Algorithms, and Source Code in C.