

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	--	--------------------

LAB7: Applying Encryption and Hashing Algorithms for Secure Communications

Section 2: Applied Learning

Table of Contents

Topology.....	1
Tools and Software.....	2
Part 1: Create a Text File on Linux.....	2
Part 2: Create a MD5sum and a SHA256sum hash string.....	3
Part 3: Modify a File and Verify Hash Values.....	3
Part 4: Generate GnuPG Keys.....	4
Part 5: Share a GnuPG Key.....	4
Part 6: Encrypt and Decrypt a ClearText Message.....	5

--	--	--

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	---	--------------------

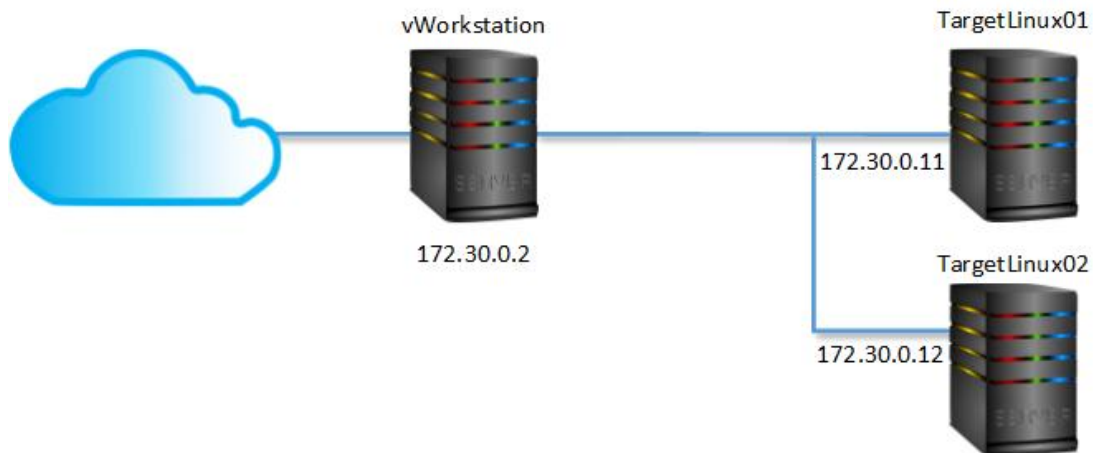
--	--	--

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	---	--------------------

Topology

This lab contains the following virtual devices. Please refer to the network topology diagram below.

- vWorkstation (Windows Server 2016)
- TargetLinux01 (Debian Linux)
- TargetLinux02 (Debian Linux)



Tools and Software

The following software and/or utilities are required to complete this lab. Students are encouraged to explore the Internet to learn more about the products and tools used in this lab.

- GNU Privacy Guard (GnuPG or GPG)
- KeyTransfer
- WinSCP
- vi Editor

--	--	--

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	--	--------------------

Part 1: Create a Text File on Linux



--	--	--

Sindy Morel

LAB7: Applying Encryption and Hashing Algorithms for Secure Communications

September 23, 2023

Part 2: Create a MD5sum and a SHA256sum hash string

```
student@TargetLinux01: ~/Documents
student@TargetLinux01:~/Documents$ md5sum Example2.txt > Example2.txt.md5
student@TargetLinux01:~/Documents$ ls
Example2.txt  Example2.txt.md5
student@TargetLinux01:~/Documents$ cat Example2.txt.md5
7033050e2c0735e8fcf7dc8c2e238669  Example2.txt
student@TargetLinux01:~/Documents$ md5sum -c Example2.txt.md5
Example2.txt: OK
student@TargetLinux01:~/Documents$ sha256sum --help
Usage: sha256sum [OPTION]... [FILE]...
Print or check SHA256 (256-bit) checksums.
With no FILE, or when FILE is -, read standard input.

  -b, --binary      read in binary mode
  -c, --check       read SHA256 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-180-2. 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 ('b' for binary, 't' 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
b383725409d94f42ce6664d8963c4eb882bdad16be884db8e24b1cb24019c852  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
b383725409d94f42ce6664d8963c4eb882bdad16be884db8e24b1cb24019c852  Example2.txt
student@TargetLinux01:~/Documents$
```

--	--	--

Sindy Morel

LAB7: Applying Encryption and Hashing Algorithms for Secure Communications

September 23, 2023

Part 3: Modify a File and Verify Hash Values

```
student@TargetLinux01: ~/Documents
-b, --binary      read in binary mode
-c, --check       read SHA256 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-180-2. 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 ('b' for binary, 't' 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
b383725409d94f42ce6664d8963c4eb882bdad16be884db8e24b1cb24019c852 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
b383725409d94f42ce6664d8963c4eb882bdad16be884db8e24b1cb24019c852 Example2.txt
student@TargetLinux01:~/Documents$ sha256sum -c Example2.txt.sha256
Example2.txt: OK
student@TargetLinux01:~/Documents$ echo "This example is testing hash values." >> Example2.txt
student@TargetLinux01:~/Documents$ cat Example2.txt
This file is from sindymorel.
This example is testing hash values.
student@TargetLinux01:~/Documents$ md5sum Example2.txt > Example2.txt.md5
student@TargetLinux01:~/Documents$ sha256sum Example2.txt > Example2.txt.sha256
student@TargetLinux01:~/Documents$ cat Example2.txt.md5
30548763e64161421eaa31e1aa81d Example2.txt
student@TargetLinux01:~/Documents$ cat Example2.txt.sha256
2f814bcb4ced0f10d42e10a4620e4aa41d5d4aa5306048674726b2b590cfa34 Example2.txt
student@TargetLinux01:~/Documents$
```

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	--	--------------------

Part 4: Generate GnuPG Keys

```
student@TargetLinux01: ~/Documents
generator a better chance to gain enough entropy.
Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 300 more bytes)
+++++
+++++
We need to generate a lot of random bytes. It is a good idea to perform
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.
Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 68 more bytes)
...+++++
Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 108 more bytes)
+++++
gpg: key D5E50E44 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: 2 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 2u
pub: 2048R/D5E50E44 2023-09-23
    Key fingerprint = 438F F458 40EF EE03 B40B BC3C F560 89C7 D5E5 0E44
uid: Student2 <student@securelabsondemand.com>
sub: 2048R/89307896 2023-09-23

student@TargetLinux01:~/Documents$ gpg --export -a > student.pub
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
-----
pub: 2048R/35695BC7 2023-09-23
uid: Student2 <student@securelabsondemand.com>
sub: 2048R/EC4E0568 2023-09-23

pub: 2048R/D5E50E44 2023-09-23
uid: Student2 <student@securelabsondemand.com>
sub: 2048R/89307896 2023-09-23

student@TargetLinux01:~/Documents$
```

--	--	--

Sindy Morel	LAB7: Applying Encryption and Hashing Algorithms for Secure Communications	September 23, 2023
-------------	--	--------------------

Part 5: Share a GnuPG Key

```
student@TargetLinux01: ~/Documents
uid      Student2 <student@securelabsondemand.com>
sub      2046R/89307896 2023-09-23

student@TargetLinux01:~/Documents$ gpg --export -a > student.pub
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
-----
pub      2046R/35695BC7 2023-09-23
uid      Student2 <student@securelabsondemand.com>
sub      2046R/EC4ED568 2023-09-23

pub      2046R/D5E50E44 2023-09-23
uid      Student2 <student@securelabsondemand.com>
sub      2046R/89307896 2023-09-23

student@TargetLinux01:~/Documents$ ls -l
total 20
-rw-r--r-- 1 student student 67 Sep 23 14:27 Example2.txt
-rw-r--r-- 1 student student 47 Sep 23 14:30 Example2.txt.md5
-rw-r--r-- 1 student student 79 Sep 23 14:35 Example2.txt.sha256
-rw-r--r-- 1 student student 1739 Sep 23 15:06 instructor2.pub
-rw-r--r-- 1 student student 3344 Sep 23 14:55 student.pub
student@TargetLinux01:~/Documents$ gpg --import instructor2.pub
gpg: key 2931104F: public key "Instructor2 <instructor@securelabsondemand.com>" imported
gpg: Total number processed: 1
gpg:   imported: 1 (RSA: 1)
student@TargetLinux01:~/Documents$ gpg --list-keys
/home/student/.gnupg/pubring.gpg
-----
pub      2046R/35695BC7 2023-09-23
uid      Student2 <student@securelabsondemand.com>
sub      2046R/EC4ED568 2023-09-23

pub      2046R/D5E50E44 2023-09-23
uid      Student2 <student@securelabsondemand.com>
sub      2046R/89307896 2023-09-23

pub      2046R/2931104F 2023-09-23
uid      Instructor2 <instructor@securelabsondemand.com>
sub      2046R/5AC96286 2023-09-23

student@TargetLinux01:~/Documents$
```

--	--	--

Part 6: Encrypt and Decrypt a ClearText Message

```
student@TargetLinux01: ~/Documents
--se -r Bob [file]          sign and encrypt for user Bob
--clearsign [file]         make a clear text signature
--detach-sign [file]       make a detached signature
--list-keys [names]        show keys
--fingerprint [names]     show fingerprints

Please report bugs to <gnupg-bugs@gnu.org>.
student@TargetLinux01:~/Documents$ echo "This clear-text message is from sindymorel" > cleartext2.txt
student@TargetLinux01:~/Documents$ cat cleartext2.txt
This clear-text message is from sindymorel
student@TargetLinux01:~/Documents$ gpg -e cleartext2.txt
You did not specify a user ID. (you may use "-r")

Current recipients:

Enter the user ID. End with an empty line: Instructor2
gpg: 5AC96286: There is no assurance this key belongs to the named user

pub 2048R/5AC96286 2023-09-23 Instructor2 <instructor@securelabsondemand.com>
Primary key fingerprint: 220D 182A 6D79 76CB 77A8 9D13 9A13 0668 2931 104F
Subkey fingerprint: 0D77 ECF5 CB10 FA13 FA33 E34F 604F 52B8 5AC9 6286

It is NOT certain that the key belongs to the person named
in the user ID. If you 'really' know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y

Current recipients:
2048R/5AC96286 2023-09-23 "Instructor2 <instructor@securelabsondemand.com>"

Enter the user ID. End with an empty line:
student@TargetLinux01:~/Documents$ ls
cleartext2.txt cleartext2.txt.gpg Example2.txt Example2.txt.md5 Example2.txt.sha256 instructor2.pub student.pub
student@TargetLinux01:~/Documents$ cat cleartext2.txt.gpg
-----BEGIN PGP MESSAGE-----
Version: 1.4.1
Comment: "vWorkstation 2023-09-23 18:34:11 Sindy Morel"
mQI=uxF3j5Gh...[truncated]...
Y-----END PGP MESSAGE-----
student@TargetLinux01:~/Documents$ PuTTYPuTTYPuTTY

Instructor@TargetLinux02: ~
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, use a random
disk) during the prime generation: this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 78 more bytes)
..+++++

Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 99 more bytes)
..+++++

gpg: /home/Instructor/.gnupg/trustdb.gpg: trustdb created
gpg: key 2931104F 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-, 0g, 0n, 0m, 0f, 1u
pub 2048R/2931104F 2023-09-23
uid Instructor2 <instructor@securelabsondemand.com>
sub 2048R/5AC96286 2023-09-23

Instructor@TargetLinux02:~$ gpg --export -a > instructor2.pub
Instructor@TargetLinux02:~$ gpg --list-keys
/home/Instructor/.gnupg/pubring.gpg
-----
pub 2048R/2931104F 2023-09-23
uid Instructor2 <instructor@securelabsondemand.com>
sub 2048R/5AC96286 2023-09-23

Instructor@TargetLinux02:~$ gpg -d cleartext2.txt.gpg

You need a passphrase to unlock the secret key for
user: "Instructor2 <instructor@securelabsondemand.com>"
2048-bit RSA key, ID 5AC96286, created 2023-09-23 (main key ID 2931104F)

gpg: encrypted with 2048-bit RSA key, ID 5AC96286, created 2023-09-23
"This clear-text message is from sindymorel"
Instructor@TargetLinux02:~$
```