CSci 4271W (011 and 012 Sections) Lab Instructions

Lab 11 **April 14th, 2025**

Ground Rules. You may choose to complete this lab in a group of up to three students. Before you leave the lab, **make sure you have submitted to Gradescope, you included all group members on the submission, and the autograder found all required files!**

1 gpg

In today's lab, we'll see a little bit about how to use GnuPG, the GNU project's PGP-compatible command-line software for encrypting and signing (and decrypting and verifying) files, and handling public keys. GPG is used by many open-source projects to authenticate source and binary files, and also sometimes still used to encrypt emails so that email providers cannot read their contents.

2 Installing gpg

gpg is already installed on your VM, so nothing to do here. Yay!

(You can also complete this lab on a regular (non-virtual) CSELabs machine, but if you actually use gpg in your regular account you may not want to mess up your keyring.)

You can download the necessary files for this lab using:

```
$ git clone https://github.umn.edu/badly-coded-alpha/lab11.git
$ cd lab11
```

3 Verifying signatures

To verify any digital signature, you need to know the (public) verification key it was signed with, the signature, and the message (file) that was signed. After cloning the lab 11 repo, you'll find several files in your directory; the pertinent files for this part of the lab are `bci.asc` (a public verification key for BCI), `badlycoded.tgz` (a file to be signed), and four *candidate* signature files, `badlycoded.tgz.N.asc`. (Note: one of these signatures is correct, one is a correct signature but with a different key, one is a correct signature for a version of `badlycoded.tgz` that has a single bit flipped, and one is the correct signature, with a single bit flipped.) Let's figure out which of the four is a valid signature.

1. First, we need to import BCI's public key into our GPG keyring:

```
$ gpg --import bci.asc
```

   This adds the key to the ring of (untrusted) public keys gpg knows.

2. Next, we can try verifying one of the signatures with `gpgv` (take your pick!)

```
$ gpgv --keyring pubring.kbx badlycoded.tgz.N.asc badlycoded.tgz
```

Here, the `--keyring pubring.kbx` option is telling gpgv not to bother looking for a trusted key, and just try all the keys it knows about (which at this point, is just BCI's.)

Did you get lucky? Repeat the verification command with the other three signature files, to see which is which; at the end of the lab, you'll upload the correct signature file to Gradescope.

Note: many open-source software projects use gpg to sign both binary and source code releases, and many developers use gpg internally to sign git commits and/or tags. (The flag to do this with git is `-S`.)

4 Encrypting messages

Some open-source projects have vulnerability-reporting inboxes that only accept encrypted messages. Let's see how to encrypt a message to a given public (encryption) key. For this part of the lab, we'll need to import another public key file that was in the lab archive, `mccamant.asc`:

```
$ gpg --import mccamant.asc
```

Now we need to make a plaintext file to encrypt. In your favorite text editor, create the file 'plain.txt', write a short note (e.g. "baby shark doo doo doo doo doo doo" or "Sator arepo tenet opera rotas") and save the file. Now we're ready to encrypt the file:

```
$ gpg -a -e -r mccamant@cs.umn.edu plain.txt
```

This will create the ciphertext file `plain.txt.asc`. Here, the `-a` option tells gpg to "ASCII armor" the file (so that it can be sent as a text file), `-e` tells it to encrypt, and `-r mccamant@cs.umn.edu` specifies the (r)ecipient of the message.

5 Generating and exporting keys

OK, let's see one last task: generating public keys. To generate your own public key do:

```
$ gpg --expert --full-gen-key
```

You'll be prompted to choose a type of key to generate; the cool kids are using elliptic curve keys these days, so choose 9. Then you'll be asked to choose an elliptic curve; any of the 256-bit curves is fine, so choose one of 1,3,6, or 9. Answer the rest of the prompts as you see fit. Finally you'll see some text like:

```
pub   secp256k1 2024-11-13 [SC]
      0F1E3AAA23C5BEE82E2B5A71857BF2BE48957C97
uid                      some text you entered
sub   secp256k1 2024-11-13 [E]
```

The second line of this block (which will be different in your output) is the "fingerprint" of your public key. If you now ask gpg to list its keys:

```
$ gpg --list-keys
```

It should print out a list that include the BCI key, the mccamant key, and the key you've just generated. (More if you've done this before). If you want to share this key with others, you need to **export** the key, and you do this by the key fingerprint:

```
$ gpg -a --export $yourfingerprint > student.asc
```

(Where you've replaced `$yourfingerprint` with the appropriate fingerprint printed earlier.) If you now examine the file `student.asc` you'll see that it is an "ASCII-armored" public key, which you'll upload to Gradescope momentarily.

5.1 All done!

Once you've exported your public key, you're done with Lab 11! Use `scp` to copy the `badlycoded.tgz.N.asc` (**only the Good signature**), `plain.txt.asc` and `student.asc` files off of your VM, so you can submit them to the Lab 11 assignment on Gradescope. Make sure you include all of the members of your group!

Once you've submitted the files, the autograder will test to make sure the proper files were submitted, check that they include the right information, and notify you if anything went wrong, within a few minutes.

Congratulations, you've finished Lab 11!