**<Linux TPM Quick Start>**

**Part A. Enable firmware TPM (fTPM) in UEFI BIOS**

1. Reboot your computer, go into the BIOS setting (by fiendishly pressing F1, F2, F4 or F10)

2. Go to Security Tab.

3. Go to TPM (or Trusted Platform Module) section.  
4. Enable (or activate) it.   
5. In the detailed setting (if there is), make sure the TPM’s SHA1 bank is turned on.   
5. Save and reboot.

6. From now on, Linux and GRUB bootloader can use the TPM.

**Part B. Downgrade to GCC 5.4 (only if your GCC version is > 6.0)**

- This is needed because GRUB2-TPM2 doesn’t compile from GCC 6.0 due to incompatibility. In general, GCC 6.0 has lots of compatibility issues.

1. Download GCC 5.4 at: <https://ftp.gnu.org/gnu/gcc/gcc-5.4.0/>

2. Extract it and go into the folder.

3. sudo apt-get install mpc libmpc-dev libmpfr-dev libmpfr-doc libmpfr4 libmpfr4-dbg gcc-multilib g++-multilib

4. ./configure

5. make -j 4 # This compilation will take about 0.5 ~ 1 hour. Do something else during this.

6. sudo make install

7. gcc --version # Check if your GCC version is now 5.4

**Part C. Install GRUB2-TPM2**

- This amazingly cool GRUB2 bootloader supports TPM2. This feature is needed to extend the SHA1 hash of GRUB2 bootloader and Linux into TPM’s PCR register 8 and 9.

1. Download GRUB2-TPM2 at: <https://github.com/mjg59/grub>   
2. Extract it and go into the folder  
3. sudo apt install flex bison

4. ./autogen.sh

5. mkdir build

6. ./configure --prefix=`pwd`/build --with-platform=efi # This configuration is for x86\_64.

7. make -j 4

8. make install

9. sudo fdisk -l # Check the name of your disk device. (e.g. /dev/sda, without the last digit)

10. sudo build/sbin/grub-install /dev/sda # Install new GRUB bootloader into your disk device

11. Reboot your computer and see if it boots up okay. If yes, GRUB2-TPM2 is successfully installed. It might say “error: no symbol table”, but just ignore it.

12. From now on, never run “update-grub” from your terminal by using your old GRUB2 tool. Otherwise, it may mess up all your OS images and you have to reinstall them. If you need to update GRUB for any reason, do it by going into this GRUB2-TPM2 folder and using the command: “sudo build/sbin/grub-mkconfig -o /boot/grub/grub.config”

**Part D. Compile a new Linux Kernel image**

- This is needed in order to hard-code the TPM device driver into the kernel image. If your TPM driver is installed as a kernel module, it won’t be available right after the kernel image has been loaded, which will break the trust chain cascaded from BIOS.

1. sudo apt-get install libncurses5 libncurses5-dev vim

2. sudo vim /etc/source.list

3. In source.list file, comment in all lines starting with “deb-src” keyword

4. :wq (save and close)

5. apt-get source linux # Download Linux source code

6. Go into newly downloaded Linux source code directory

8. make menuconfig  
9. Go to: Device Drivers --- Character Devices --- TPM Hardware Support --- TPM CRB Interface (\*) star-check   
10. Exit and Save  
11. make -j4 bindeb-pkg # Compile the kernel. This takes 0.5~1 hour. When you’re at the stage of finally creating a debug .deb package, containing letters \*dbg\*, you don’t have to wait anymore and just press Ctrl-x and finish.   
12. cd ..

13. sudo dpkg -i linux-image-\*.deb

14. rm \*.deb

19. Reboot your computer.  
20. In the GRUB menu, select the Linux kernel image you just newly installed. See if your new kernel image works okay. (You can also keep using your old kernel image.)

**Part E. Enable IMA\_TPM in Linux**

- This enables Linux to TPM-measure every binary, module, dynamic library, configuration file being loaded into the memory and extend their SHA1 hash to TPM’s PCR 10 register.

1. cd /boot/grub

2. sudo vim grub.config  
3. For every line starting with “linux” or “kernel” keyword for your newly installed kernel image entries (e.g. 4.8.17), add the additional command line argument “ima\_tcb” at the end. For example, the modified line will look like the following:

linux /boot/vmlinuz-4.8.17 root=UUID=3739bdbc-d46c-44f9-827f-87ff2270ea70 ro quiet splash $vt\_handoff **ima\_tcb**

4. Save and close.

5. sudo vim /init.d/tpm-tool

chmod 777 /dev/tpm0

chmod 444 /sys/kernel/security/ima/ascii\_runtime\_measurements

chmod 444 /sys/kernel/security/ima/binary\_runtime\_measurements

:wq (save and close)

5. The next booting will take much longer than before due to additional TPM measurement procedures (e.g. 1~2 minutes)

6. Once your Linux Desktop loads, open up the terminal.

7. sudo vim /sys/kernel/security/ima/ascii\_runtime\_measurements  
8. If you see thousands of lines of log, where each line is a SHA1 hash value of the loaded binary, module or dynamic library, you can be reassured that your Linux kernel is now TPM-enabled.

9. Note that Part E has to be done every time you re-compile and re-install your kernel image, because that’ll overwrite your current grub.cfg file.  
10. Otherwise, when you boot up, go to the kernel’s menu, press ‘e’, temporarily dd ‘ima\_tcb’ at the end of “linux” line, press F10 to boot that way.

**Part F. Compile TPM Userspace Toolkit**

- You’ve finally come to this almost final stage. This is the core TPM remote attestation Toolkit to be used within your Python implementation where the client and IFC server (or the proxy and IFC server) begins communication.

1. Extract my attached “tpm-ibm-stable” zip file.

2. cd tpm-ibm/utils  
3. sudo apt install openssl libssl-dev

4. make -j 4

5. ls -lah /dev/tpm0 # Make sure the TPM driver node’s access policy is rwx | rwx | rwx.

6. ./pcrread -ha 10 -halg sha1 # Check if it successfully reads TPM PCR 10 register’s SHA1

**Part G. Python TPM Attestation Server/Client**

1. Stay in the same tpm-imb/utils directory

2. ls -lah /sys/kernel/security/ima/binary\_runtime\_measurements # make sure it’s r-- | r-- | r--

2. ./create -hp 81000001 -si -opu pub.key -opr priv.key -nalg sha1 -halg sha1

3. python server.py # TPM remote attestor (IFC server)

4. Open up another terminal, and run “python client.py” # TPM challenger (proxy or client)

5. If the client terminal prints “TPM Remote Attestation Complete (PASS)”, congratulations!

6. Embed this server/client Python source code into your current implementation.

**Part H. OpenACC and CUDA**

Log into Odyssey

g++ -o acc -fopenacc parallelized\_openmp.c # OpenACC

nvcc -o cuda parallel\_cuda.cu # CUDA

./acc

./cuda

Let me know if anything doesn’t work.

[hrko@g.harvard.edu](mailto:hrko@g.harvard.edu)

Ronny Ko