**Embedded Security (Summer Term 2018)**
Dr. Stefan Nürnberger
CISPA Helmholtz Center i.G. / Saarland University

# Exercise #5 (21 Points)

**Date due: July 9., 9:59 am**

## Submission Instructions

The solutions must be submitted via CMS until **July 9., 9:59 am** in PDF format. Make sure that you clearly indicate the names and matriculation numbers of all students who authored the solutions **on the sheet itself**. Please do not share info that is not publicly known (via the lecture or the exercise sheet), especially not in the forum. If you have questions that might leak deduced information, write a mail to the tutors or visit the office hours instead of creating a post in the forum. If you wrote any code to solve the exercises, please add it to your solution. **Undocumented code is not accepted**. If it is more than a few lines, please put it into a **dedicated appendix** and refer to it from your solution. We only accept solutions written in *C*, *C++* and *Python* if not stated otherwise. Be kind and format your code nicely.
Label your solutions corresponding to the exercises on the task sheet. Handwritten solutions are accepted, but they have to be readable; otherwise points might be deducted.

### 1 Tea Pee Ehm? (12 points)

In this exercise, you will learn some TPM basics. Your task is to join a DRM network by first verifying the integrity of your operating system, then writing an app to download DRM-protected content.
WARNING: DO NOT UPGRADE YOUR KERNEL. This exercise relies on a specific kernel version and any update to it may break it. Furthermore connect the TPM to the raspberry exactly as shown in Figure 1.

(0 points) (a) Setup your Raspberry. Download the image from the CMS and write it to the provided SD-card. Then insert it into the Raspberry, attach the TPM and boot. Then, download the Trusted Software Stack [1]. Go into the *utils* folder, check that your TPM is found in /dev (as tpm0) and execute `./getcapability -cap 6`. This should provide some general information about the TPM. If you face any error, make sure the TPM type is `dev` (`export TPM_INTERFACE_TYPE=dev`) and to set the permissions properly (`sudo chmod 777 /dev/tpm0`).

(6 points) (b) Write a program that measures the operating system.
Take a look around in the *utils* folder and see if there is something that can help you. You need to create a handle and feed it with your file to create a hash of it (use sha256). Your file should accept the file to be measured as first parameter. Once this is completed, use this program to measure your operating

[1] `https://sourceforge.net/projects/ibmtpm20tss/files/ibmtss755.tar/download`

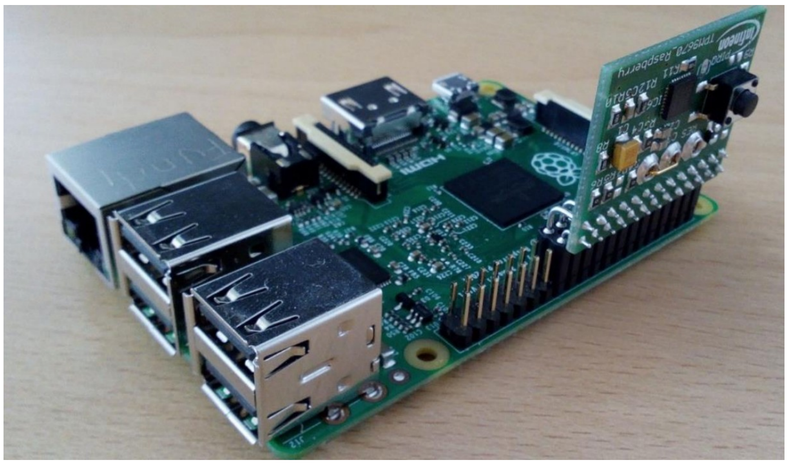

Figure 1: Setup of Raspberry and TPM module

system, i.e. `/boot/kernel.img`. You can verify your results by computing the sha256 hash on the commandline with, e.g. `sha256sum`.
Note that you could theoretically use this command line tool to compute the hash of the kernel image, however this will not yield you any points for this exercise. Additionally, you need to implement this program in C.

(0 points) (c) Visit `emsec.cispa.saarland/media/` and create an account there (you need the operating system integrity hash for it). Remember your credentials, you need them later on.

(4 points) (d) Now, you implement the client software. This software should be able to send a post request to `emsec.cispa.saarland/media/download/`. You need to supply your credentials as parameters, the username is set in the `username` field, password in the `pwd` field. Additionally, your program should call the program from the first part to measure itself. This measurement needs to be provided as `app` parameter. Issueing the post request should now yield you an error that your application is not a trusted application. To access the protected content, you need to login again at `emsec.cispa.saarland/media/login/` and upload your application. Afterwards, posting to `emsec.cispa.saarland/media/download/` will reward you with the protected content. Provide it in your solution.
Again, the integrity hash can be computed using standard command line tools. This will not yield any points. Unlike the measurement program, the programming language may be C, C++ or Python.

(2 points) (e) The current setup has a flaw which invalidates the idea of using a TPM to establish a chain of trust. Briefly explain why and explain how to fix this problem!

## 2 More TPM stuff (9 points)

(2 points) (a) Briefly explain the difference between *sealing* and *encrypting* data. Further-

more, give an example where sealing is wanted in contrast to encrypting.

(1 point) (b) Briefly explain why the restricted attribute is always set for storage keys.

(2 points) (c) Briefly explain why there is a certificate for the attestation key in the attestation concept.

(d) Show how a policy session can be established for the following scenarios[2]. You should do that in the same fashion as done in the lecture.

(1 point) (a) Any app that is in possesion of a specific password *pwd* and was build by developer $D$ (with keypair $sk/pk$) should be given access to key $K1$ for the usecase of decryption.

(1 point) (b) Any app running on a specific operating system ($PCR1 = mOS$) should be able to key $K2$ for decryption, as long as a user shows *physical presence*.

(2 points) (c) Any app that was build by a Developer $D$, or $D'$ that was authorized by $D$, is allowed to use key $K1$ for signing.

[2]This source as well as the cheat sheet in the CMS might help you: `https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf`