TPM Workshop

TheCamp 2020

23/07-2020

1 Introduction

The purpose of this workshop is to get familiar with the IBM TSS and TPM Commands. We advice getting familiar (at least for reference) with the TPM Documentation (especially part 3 that includes the code), which can be found here: `https://trustedcomputinggroup.org/resource/tpm-library-specification/`.

The error messages from the TPM can be hard to decipher why it can be beneficial to look into the code itself. Furthermore, you can modify the Software TPM to provide debug information, if you desire.

2 Setup and installation

I expect that you got this document through the repository, otherwise pull the code located at `https://github.com/benlarsendk/TheCampTPM`. If you haven't installed IBMs TSS and Software TPM, you can use this script to do so `https://github.com/benlarsendk/SetupTPMEnv` **NOTE: It will chmod your /opt/ to 777 - you probably shouldn't do that.**.

If you are on Windows, you'll have to manually OpenSSL. Follow the guide in the packages you downloaded. A simple way to make it work on windows is using the Visual Studio Solutions provided. For the software TPM, you can simply open it, build and run; that's enough to run the TPM. For the TSS, you should make it, and then copy the generated DLL to your project folder. It is a bit more challenging on Windows (I've failed a couple of times, lost some hair too), so if you have Linux, it's probably easier.

3 Provided code

We have provided you with some code to get you started. The files you should be concerned about is:

1. TpmManager - this is where your code interacting with the TPM should be. It already has some code for booting the TPM, loading a key into the TPM, do a PCR Extend, and show error messages.
2. NetworkManager - This acts as the network manager we talked about, there shouldn't be any reason to extend this code (unless you want to)
3. Main - where your code is going to be

The rest of the files are for support functions. In the first exercise, don't think too much about it. For exercise two, you'll need to two .txt files: Antivirus.txt and database.txt. Do not edit these files!

4 Exercises

The scenario for both use cases is the one we talked about during the talk. An admin has been told he needs to ensure that *only* issued platforms can log onto his network. In the first exercise, we simply do the signature and verification, and in the second exercise, we protect the keys with the PCRs.

4.1 Exercise 0

Make a hello world! Make sure you have installed both IBM TSS and their Software TPM. Boot the software TPM and run the code provided ”as is.”You should not receive any errors, but a debug message saying, ”Booting TPM.”If you get errors - investigate!

4.2 Exercise 1

Now that you have your software TPM and TSS up and running let's start coding.

4.2.1 Create a primary key

The first thing you should do is create a primary key (The SK from the talk). To do this, you should execute `TPM2_CreatePrimary`.The key should be an ECC key, and *not* has neither password protection nor policy protection. Look in IBM TSS under ”tpmutils”and ”createprimary”to get help. If you'd instead just go on, look in the file ”CreatePrimaryHelp.txt.”In this file we've provided the public part that is used as a template for the TPM to generate the key, this can probably help you when you have to go to the next step.

When you have successfully created the key, you can access its keyhandle with the CreatePrimary_Out objects function: `out.ObjectHandle`. This handle is needed both when you need to flush it from the TPM and when creating the child key.

4.2.2 Create a function for flushing sessions and keys

To use the TPM at all, you'll need to be able to flush keys out of the TPM. Implement a function to flush a key. The TPM function is called `TPM2_FlushContext`. Look into the TSS Code to get inspiration. Test it by trying to flush the primary key you just created.

4.2.3 Create a function to create a child key

Now, you have to create a signing key as a child of your primary key. Create the function with inspiration from the TSS Examples. Remember, no password, no policy, but a simple, unrestricted signing key. Remember, not encryption but a signing key. This key should also be ECC, and it should have the primary key you just created as a parent, so remember to create the primary key before creating the signing key.

When you have created the key, try to load it using the load function in TpmManager. If you are successful, remember to flush both the primary key and the child key from the TPM.

4.2.4 Implement the signing function

Now we need to be able to sign. Implement a function to execute `TPM2_Sign` with inspiration from the examples in the TSS. Note that the nonce you will be receiving has a size of 32 bytes (`SHA256_DIGEST_SIZE`), so there is no need to hash it before signing it.

4.2.5 Run the usecase

When you have completed the above steps, try to run it with the use case in the following steps:

1. Create primary key
2. Create signing key

3. Register the signing key with network-manager
4. Request a nonce from the network manager
5. Load the signing key
6. Sign the nonce
7. Send it to verification with the network manager

If you did everything right, you shouldn't get any errors, but a confirmation that the signature was verified. Of course in a real-life scenario, the key would be registered at an earlier point in time.
You have completed the first exercise - an outstanding job. This stuff isn't easy, and a lot of errors can sneak up on you. If you want to be challenged further, continue to the next exercise.

4.3 Exercise 2

Now we're getting somewhere! Remember, we talked about trusting the platform had the right antivirus and database on the machine? That's what we're going to do now! For simplicity, both antivirus and database are represented as simple text files included in the project. Before starting, make sure you don't have exercise 1 running before or after the code you are about to create. Furthermore, make sure you run the commented code from the main file right after the boot - it simulates the trusted measurements we talked about, and hashes the two files and extends into the TPM. Take a look at the code if you are interested :)

4.3.1 Create a primary key

Do the same as before; there is no need to change anything. Wuhu! Ezpz exercise.

4.4 Create a PCR Selection

We need to create a structure that tells the TPM what PCR registers we want to work with. In our case, it's PCR 16. You can either use the PCRSelector provided in the code, or you can do it yourself by studying the code. Remember only to select PCR 16 (15 when counting from 0).

4.5 Create a policy that binds the signing key to PCR 16

Because we are nice people, we've created a function for you that calculates the *expected* session digest, if the PCRs are correct based on *expected* hashes of the two files. The function is called `calculatePolicyDigestForFiles` and simple takes a reference to the PCR Selection. If you want to do this using a trial session, you are very much welcome to do so. Simply create a trial session, run PolicyPCR for PCR 16 and execute a `TPM2_GetPolicyDigest` and use that. Of course, you'd need to implement these functions first. See us if you have questions about doing this.

4.5.1 Create the signing key

Now create the signing key as a child of the primary key as you did before, but ensure it's created with the policy digest we just made. Again, look in the examples for `CreateKey` in the IBM TSS for inspiration (it's not a lot of change from your earlier version). When you have created your key, try to run the use case as you did before. It should throw an error during sign, saying that there is a policy problem. If this happens, you did correct!

4.5.2 Create a function to start a session

As we talked about earlier, we need a session to satisfy a policy. Create a function to start an authorized session in the TPM (`TPM2_StartAuthSession`) of type Policy (`TPM_SE_POLICY`). This function should return a handle to the session.

4.5.3 Create a function to execute Policy command

To be able to modify the session digest, we need to implement a function to call the policy command in question. For this use case, it's `PolicyPCR`, and as the other commands well exemplified in the TSS Documentation. Remember, it needs to take a session as a parameter to be able to modify the session digest, and of course, a PCR(or more) to use.

4.5.4 Modify sign

To be able to sign in a session, the signing command needs to know what session. Modify your code to take a session handle. Hint: In the case, it's the first session handle (which we probably have had as NULL so far. Simply replace it with the started session).

4.5.5 Run the usecase

As earlier, let's run the use case.

1. (Run the trusted measurements)
2. Create the primary key
3. Create the signing key with the auth policy
4. Register the key with network-manager
5. Load the signing key
6. Request nonce
7. start an authorized session of type policy
8. Execute the PolicyPCR command
9. Execute sign over the nonce
10. Send signature for verification.

If you have done everything correctly, the signing operation will not fail. If something is wrong, you'll notice that the signing operation cannot continue. If it works: **great job**.

4.5.6 Edit the files

Try to change the files provided (Antivirus.txt or database.txt) and then rerun the use case. Notice that you will **not** be able to sign the nonce. Only if you have the correct files you will be able to sign the nonce, meaning that if a signature is returned and verified, the network-manager has *guarantees* that you indeed have the correct antivirus and database!