CS-E4760: Platform Security (2023)
Exercise 5:
Measurement & Attestation

1. System boot [10 points]

a. Why does a PC use a BIOS/UEFI, rather than executing the operating system directly? **[2 points]**

b. How does authenticated boot protect sensitive data, despite allowing attackers to run whatever code they like? **[2 points]**

c. Can a TPM protect against an attacker with physical access to the machine? State your assumptions. **[4 points]**

d. Late launch eliminates the need to measure BIOS code, by including a Root of Trust for Measurement in the CPU itself, which will measure and jump to a specified piece of code. Is a BIOS still needed in a system that always uses Late Launch? Why [not] [2 points]

2. Remote attestation [8 points]

a. Property-based attestation allows attestation of a program's name, version, and developer instead of the hash of its binary.

   What is an advantage and disadvantage of this approach? **[2 points]**

b. Can any code run before a static root of trust for measurement in a TPM-based system? Why [not]? **[2 points]**

c. In a system using a TPM, normally some PCRs will be allocated to the BIOS, others to the OS, and others to applications. This means that, the BIOS code can change without affecting the PCR containing the measurement of operating system kernel.

   Can this system attest itself to a verifier that doesn't know what is and not valid BIOS code, without using Late Launch? Why [not]? **[2 points]**

d. Suppose we make a query to some machine, asking it to perform some computation. The machine returns a response, along with a valid attestation stating that a machine is running a known piece of software. Does this mean that the response to the query is valid? Why [not]? **[2 points]**

4. TPM extended authorization [12 points]

Recall the example from the lecture where developer D wanted to ensure that a TPM-protected RSA key (k_1) was available for use by application A_1. D could set a suitable TPM authPolicy and define the sequence of commands that the app needed to issue in order to decrypt a ciphertext c using the private key k_1.

More precisely, developer D defines a particular authPolicy for the private key of k_1. In order to use k_1, the TPM's policyDigest (i.e. the running policy session hash) must match the defined authPolicy. For simplicity, assume that the decryption command takes the form:

```
TPM2_RSA_Decrypt(k_1, c)
```

where "k_1" is an identifier of the private key and "c" the encrypted data to be decrypted using "k_1".

In that example, the authPolicy for k_1 is:

```
H( H( H(0 || PolicyPCR || 1 || mOS)
          || PolicyPCR || 2 || mA_1)
          || PolicyCommandCode || RSA_Decrypt)
```

The sequence of TPM commands for A_1 is:

```
v11 ← TPM2_PolicyPCR(1, mOS)
v12 ← TPM2_PolicyPCR(2, mA_1)
v13 ← TPM2_PolicyCommandCode(RSA_Decrypt)
      TPM2_RSA_Decrypt(k_1, c)
```

The intermediate values of the TPM's policyDigest (i.e. v11-v13) are:

```
v11   = H( 0 || PolicyPCR || 1 || mOS)

v12   = H( v11 || PolicyPCR || 2 || mA_1)
      = H( H(0 || PolicyPCR || 1 || mOS) || PolicyPCR || 2 || mA_1)

v13   = H( v12 || PolicyCommandCode || RSA_Decrypt)
      = H( H( H( 0 || PolicyPCR         || 1 || mOS)
                   || PolicyPCR         || 2 || mA_1)
                   || PolicyCommandCode || RSA_Decrypt)
```

Note that the final value of the TPM's policyDigest (i.e. v13) is precisely the same as the authPolicy given above, so the TPM allows the decryption.

Now suppose that developer D has two applications: A_1 and A_2. D wants to ensure that a TPM-protected RSA key k_1 is available for use by A_1 *or* A_2.

To achieve this, D can use the TPM2_Policy_OR() command. TPM2_PolicyOR() is passed a list of hash values <X1, X2, ..., Xn>, each corresponding to a possible value of the TPM's policyDigest. If the TPM's current policyDigest is in the supplied list, the TPM2_Policy_OR() command succeeds. The TPM's policyDigest is updated as follows:

```
IF policySession->policyDigest in List
THEN newDigest := h(0 || TPM2_CC_Policy_OR || <X1, X2, ..., Xn>)
```

The reasoning behind this scheme is that for a wrong digest Xw (not in <X1, X2, ..., Xn>), it is computationally infeasible, due to the properties of the hash function h, to find another list of digests <X1', X2', ..., Xn'> such that

```
H(X1' || X2' || ... || Xn') = H(X1 || X2 || ... || Xn).
```

a. What should authPolicy of the RSA private key (identified by k_1) be set to in order to allow A_1 or A_2 to decrypt a ciphertext c using k_1? **[2 points]**

b. Based on the authPolicy in (a), what sequence of TPM commands does the app (A_1) need to issue in order to decrypt a ciphertext c using k_1? **[2 points]**

Now suppose developer D now wants all his applications (A_1, ... A_i, ... A_n) to be able to use the private key k_1 to decrypt ciphertext c.

c. One possible approach would be to use multiple PolicyOR statements, in order to include the measurements of all the applications. Briefly explain the main disadvantage of this approach using a concrete example. **[2 points]**

A better solution would be to use the TPM2_PolicyAuthorize() command. The TPM2_PolicyAuthorize() command asserts that the current policyDigest is authorized by an external entity (i.e. the external entity has signed the policyDigest value). If the policyDigest is authorized in such a way, the TPM resets its policyDigest and replaces it with the hash of the external entity's public key.

```
IF Sig_X(signedDigest) validates using PK_D AND
   signedDigest == policySession->policyDigest

THEN newDigest := H(0 || TPM2_CC_PolicyAuthorize || PK_D)
```

d. Using TPM2_PolicyAuthorize(), what should the authPolicy of k_1 now be set to in order to avoid the disadvantage identified in (c)? **[2 points]**

e. Based on the authPolicy in (d), what sequence of TPM commands should app A_2 issue in order to decrypt a ciphertext c using k_1? **[2 points]**

f. If the developer has already allowed a particular application (e.g. A_3) to use key k_1, is it possible for the developer to later revoke this application's access to the key? Assume that the developer is using the authPolicy you specified in (d) and explain your answer. **[2 points]**

Please give your answers to (a), (b), (d), and (e) in the same format as the example above.

For simplicity, in your answers, you can leave out the "TPM2_" or "TPM_" prefixes (e.g. use "RSA_Decrypt()" rather than "TPM2_RSA_Decrypt()")

Use the following notation: public signature verification key of an entity X is PK_X (e.g. PK_D is D's public key) and signature made by X on a message m is denoted Sig_X(m)

Recall the assumptions we made during the lecture:

(1) the OS is measured and extended into PCR1; if the correct OS is running, PCR1 should have value "mOS".

(2) only one app (process) has access to the UI at any given time and the OS ensures that the app with access to the UI is measured and extended into PCR2. If application "A_1" is extended this way into PCR2, PCR2 will have value "mA_1". Use the same assumptions and notation for all apps A_1 - A_n.