Exercises - Week 8
Trusted Computing

1. In the following scenarios explain whether you need isolation, attestation, or both. Explain what would happen if these properties were not met.

(a) A bank uses an HSM to sign very large transactions.

(b) In a phone, the SIM card provided by the phone company signs a message to authenticate itself to the network.

(c) In a Cloud application called remotely from a client, Intel SGX is used to perform statistical computations (e.g., average of a column) from values stored in the clear in a database.

(a) Needs isolation. If an adversary can extract the secret key from the HSM, this adversary can sign very large transactions.

Attestation is not absolutely needed. The bank owns the trusted hardware in its premises and therefore it does not need to prove that the trusted hardware is there (first attestation property) Regarding software, the signature on the transaction can be verified externally. If the HSM performs any other operation, the signature would not pass the verification.

One could also argue that attestation is needed to ensure that the HSM signs once, and only once, the transaction, and does not leak the secret key.

(b) Needs isolation. An adversary that can extract the secret key from the SIM card can impersonate this card towards the network, i.e., can generate/receive calls as if she was the legitimate user.

Same considerations regarding attestation as in 4(a).

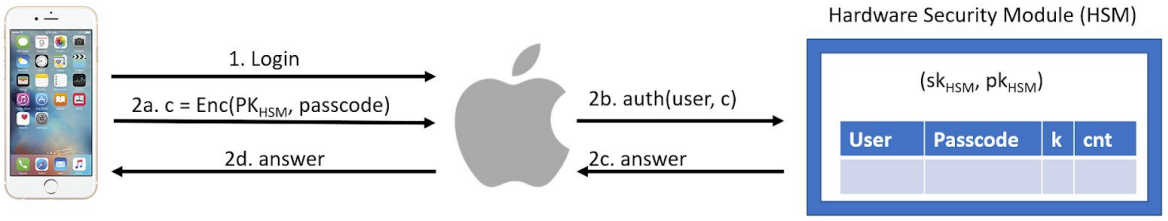
(c) One needs attestation. Since the application is run remotely on an untrusted environment (the Cloud), it is important that the SGX ensures the client that it has performed the desired operation.

One could argue that, since no one has physical access to the cloud where SGX is installed, there is no need for isolation. On the other hand, it could be argued that the Cloud is adversarial, and SGX needs to be protected to avoid tampering attacks. Since values are stored in the clear, we are not concerned with the cloud trying to breach the confidentiality of data (It already has read access to data).

2. In question 4(a) what do you think is a better security practice to avoid fraudulent transactions: giving the bank director the right to execute the signature in the HSM, or have the HSM requiring the credentials of one subdirectory and one teller to permit the operation (use the design security principle(s) to justify your answer).

The second option: subdirectory + teller is a better option. This configuration would follow the separation of privilege principle. To avoid that one single action (the director becoming malicious) can break the system, we require more than one actor to execute the operation to be secured, namely the large transaction.

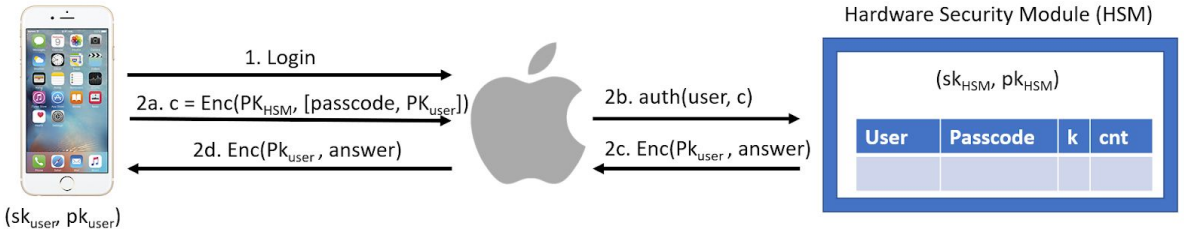
3. Consider Apple's data backup system. Is there a possibility for a replay attack if the passcode encrypted in step 2a is correct? And if the passcode is not correct? What security property (not principle!) is the attack threatening?



The attacker needs to login to the apple server as a user to send an HSM authenticate for user u. Hence, with just observing the message 2a the attacker cannot perform a replay. On the other hand, if the attacker can observe the message 2b (or gain access to apple login) it can replay it to the HSM, and the result depends on the passcode in the message 2a:

Correct passcode: Yes, replay attacks threaten confidentiality and integrity. There is no nonce(IV) in the request and the client and server don't perform a challenge-response protocol. Hence, replaying the 2a request with a correct passcode would lead to a successful key retrieval. The apple server can detect that two request message are equal (simplified: equality only applies to symmetric encryption and we cannot check equality for asymmetric encryption in practice), but it cannot determine whether the second request is a real request from the user or a replay. This attack gains access to the key, but the attacker needs to access the data in the cloud to be able to use this key. If the attacker has read access to the data this breaches the confidentiality, and if she has write access, this breaches integrity. Having key and data in separate systems is a form of separation of privilege.

Wrong passcode: Yes, replay attacks threaten availability. The attacker cannot gain access to the keys with a wrong passcode, but with sending multiple wrong passcode requests for a user, he can block the user from the system. The HSM prevents access after MAX wrong password attempt, so replaying wrong passcodes would lead to denial of service and attack availability.



Would encrypting the HSM's answer with the user's key affect the replay attack. Justify.

Correct passcode: Yes, replay attack no longer works. Only the user can decrypt the answer, so replaying the user's request leads to receiving a valid but indecipherable secret from the HSM.

Wrong passcode: No, encryption is not helpful here. The attacker cannot read the answer, but he doesn't need to know what is inside the answer. The passcode is wrong, and the attacker is only trying to get failed attempts to block the user.

4. The command TPM_Seal is used to encrypt data and bind it to some Platform Configuration Registers (PCRs) values. Suppose TPM_Seal is called with arguments including the usual arguments concerning keys and authorization data, and also the PCR p and associated value v. The command runs successfully and returns a blob.
(a) Explain the conditions under which the blob can later be decrypted.
(b) Explain a scenario in which TPM_Seal might be useful.

(a) The blob can be decrypted only if the PCR p has value v, which means that the hardware repeated the same configuration and instruction sequence, and the relevant authorization data is given as part of the command.
(b) Scenarios:
    (i) For example, in Microsoft BitLocker, the volume encryption key is sealed by the TPM, and can be decrypted only if the relevant PCRs have the correct values, thus ensuring that the system is in the correct configuration state
    (ii) A music store application wants to prevent any direct access to the music files. If a user can find a way to go beyond the interface of the program and access the files then he can listen to them on a different device or with another media player, or illegally upload the files to a sharing site. Music store seals the music files, and this ensures that only a valid music store code, not cracked, can unseal and decrypt the music files.

5. What is the difference between a covert channel and a side channel?

A covert channel consists of using mechanisms that are **not** intended for communications in order to transmit information. For instance, creating or deleting a file to transmit a "1" or a "0". Covert channels are used by insiders (e.g., a trojan horse, a rogue user with access to confidential information) with access to privileged information to leak information to the "outside".

A side-channel attack, on the contrary, is deployed by an external adversary that exploits execution differences (in time, power, etc.) of secret-dependent operations that happen inside a protected environment. Usually, these secret-dependent operations are key-dependent operations executed inside of a cryptographic device.

6. Is there any timing side-channel in the following code?

If a timing side-channel exists, (1) give an explanation on how to exploit it. (2) change the code to prevent the attack (Hint: try to make time measurements useless for the adversary).

Example A:

```
Bool CheckPin(string check, string pin){
        for ( int i = 0; i < 4; i++)
                If (check[i] != passcode[i])
                        return false;
        return true;
}
```

In the CheckPin the function spends time linear to the size of the largest check prefix which matches the passcode. The attacker can use this to infer the size of the correct prefix. The attacker has a partially correct passcode (initially empty). In each step, it adds each of the ten digits to the end of the current pass, and the one which spends more time in the function has guessed the correct number. Fix: Have a constant time function

```
Bool CheckPin(string check, string pin){
        Bool ok = true;
        for ( int i = 0; i < 4; i++)
                If (check[i] != passcode[i])
                        Ok = false;
        return ok;
}
```

Example B:

```
// exp(sec, x) computes x^sec mod n
const BigNumber n = [A big number];
BigNumber exp(BigNumber sec, BigNumber  x){
        BigNumber out = 1;
        for ( int i = 0; i < sec.size(); i++){
                If (sec[i] == 1)
                        out = (out*x) %n;
                out = (out * out)%n;
        }
        Return out;
}
```

The exp(sec, x) computes $x^{sec} mod\ n$. Multiplying $out * x$ is time-consuming. The system only performs this action in the i^{th} step if the secret key's i^{th} bit is one. Based on this timing difference the adversary can guess secret key's i^{th} bit. Afterward, the attacker can repeat

this process for other bits. Fix: make the function constant time. The computation time should be independent of the secret key.

```
BigNumber exp(BigNumber sec, BigNumber x){
        BigNumber out = 1;
        for ( int i = 0; i < sec.size(); i++){
                tmp = (out*x) %n;
                If (sec[i] == 1)
                        memcpy(out,tmp);
                else
                        memcpy(out,out);
                out = (out * out)%n;
        }
        Return out;
}
```

Besides having a constant time, there are other possible countermeasures:

Hiding: The system adds noise to the sensitive operations to hide the signal (valuable data).

```
Bool RandomizedCheckPin(string check, string pin){
        Bool ok = true;
        for ( int i = 0; i < rand(); i++)
                check[i] == passcode[i];     // waste time
        for ( int i = 0; i < 4; i++)
                If (check[i] != passcode[i])
                        Ok = false;
        return ok;
}
```

In this example, the perform a random number of comparison prior to the real check. Hence, the final timing won't have a correlation with the length of the correct prefix.

Masking: The system doesn't directly operate on the sensitive data.

```
Bool HidingCheckPin(string check, string pin){
        // There is no correlation between the input of the hash
        //  and the necessary time to compute the hash.
        check_hash = Hash(check);
        pin_hash = Hash(pin);
        return check_hash == pin_hash;
}
```

In this example, the check is performed on the hashes of the inputs (check and pin) rather than the inputs directly.