Part-1: Cryptography Basics
Part-2 Policy
Part-3 Quote
Part-4 Putting It All Together

Part-1: Cryptography Basics

1. Download dummy.txt, hash it using sha256 and extend the hash to PCR-10. What are the values of the PCRs before and after? Explain the result of PCR-10 following extending the hash value.

   Values before extend:

   ```
   tpm2_pcrread sha256:10
    sha256:
       10: 0x0000000000000000000000000000000000000000000000000000000000000000
   ```

   Values after extend:

   ```
   tpm2_pcrread sha256:10
     sha256:
       10: 0x9AAA405426BA342B7E1742BE635D8CBE4BE66921E33507A052B4EA76D6C21837
   ```

   The process of extend is an operation of sha256(original value | extend value), "|" indicates concatenation. Mimic the extend process:

   ```
   EXTEND_DATA=188f4d5762cb433104f16162136c1234d4c20dab156910e651c6ff7cf46b5b71

   ORIGINAL_DATA=0000000000000000000000000000000000000000000000000000000000000000
   00

   CONCATENATED=`echo -ne $ORIGINAL_DATA; echo $EXTEND_DATA`

   echo $CONCATENATED
   0000000000000000000000000000000000000000000000000000000000000188f4d5762cb
   433104f16162136c1234d4c20dab156910e651c6ff7cf46b5b71

   echo $CONCATENATED | xxd -r -p | openssl dgst -sha256
   (stdin)= 9aaa405426ba342b7e1742be635d8cbe4be66921e33507a052b4ea76d6c21837
   ```

2. Create a new EK and perform signing followed by verifying operations. How do you guarantee that only an admin can use the EK?

   Endorsement key is a key which never leaves TPM and is used only to certify AIKs in TPM 1.2.

   The primary key is similar with EK in TPM2.0 and the **signing** is not a the default attributes of it. Hence we can use primary key create a signing key to sign data.

   The processing of creating a Primary key and using primary key to generate a signing key for signing:

```
tpm2_createprimary -C e -c primary.ctx -p password

tpm2_create -G rsa -u sign.pub -r sign.priv -C primary.ctx -c sign.ctx -P
password

echo "data" > message.dat

tpm2_sign -c sign.ctx -g sha256 -o sig.rssa message.dat

tpm2_verifysignature -c sign.ctx -g sha256 -s sig.rssa -m message.dat
```

When creating the primary key, we can use -p to set a password for admin to guarantee only an admin can use this primary key.

3. Create a new SRK, specify the authorization "value = 9854" to prevent another user from using the new generated SRK. Attempt to perform a brute force attack starting from 0000 to 9999. Are you able to launch a successful attack? Explain the result you get from the TPM.

   In TPM2.0, we can use primary to create a storage key with "restricted|decrypt" attributes as a SRK.

```
tpm2_createprimary -c primary.ctx -C e

tpm2_create -C primary.ctx -G rsa -u srk.pub -r srk.priv -a
'fixedtpm|fixedparent|sensitivedataorigin|userwithauth|restricted|decrypt' -
p 9854 -Q

tpm2_flushcontext -t
tpm2_load -C primary.ctx -u srk.pub -r srk.priv -c srk.ctx -Q

tpm2_evictcontrol -C o -c srk.ctx -o srk.handle
```

   We cannot launch a brute force, after 5 times of wrong authorization values, TPM will enter locked mode:

```
echo "data" > data
for((i=0;i<=9999;i++));
do
tpm2_sign -c srk.handle -g sha256 -o sig.rssa data -p $i
done
```

   locked mode

```
tpm:warn(2.0): authorizations for objects subject to DA protection are not
allowed at this time because the TPM is in DA lockout mode
```

   This is the protection mechanism for brute force.

4. Create a new symmetric key (AES-128) with a policy of your choice. What is the advantage of using a TPM to generate the key vs generating a symmetric key with OpenSSL?

   The process of creating a AES-128

```
tpm2_createprimary -c primary.ctx -C e

tpm2_create -C primary.ctx -G aes128 -u key.pub -r key.priv

tpm2_load -C primary.ctx -u key.pub -r key.priv -c key.ctx

echo "my secret" > secret.dat

tpm2_encryptdecrypt -c key.ctx -o secret.enc secret.dat

tpm2_encryptdecrypt -d -c key.ctx -o secret.dec secret.enc

cat secret.dec
```

The key generated by TPM will be protected by its parent key, when using this key, we need to use its parent key to decrypt it. Meanwhile, we can specify a policy or password for this key to protect the using process. Hence the key generated by TPM are in a more security situation.

Part-2 Policy

Tom would like to store a secret file on his hardrive. He plans to use his Laptop TPM to securely store his file encrypted. Design and implement a solutions using TPM and policy that would allow Tom to protect the secrets and seal it to his laptop only. Note: There are multiple ways to choose the authorisation method to access the key/data

Implement:

1. Encrypt data
    - Randomly generate key and vi
    - Use Openssl to encrypt the file with key and vi
    - Seal the key and vi with a policy.
2. Decrypt data
    - Unseal the object by the policy to get key and vi
    - Use key and vi to decrypt the encrypted file

1. Randomly generate key and vi

```
AESKEY=`tpm2_getrandom 16 --hex`
AESIV=`tpm2_getrandom 16 --hex`
echo "key: $AESKEY" > aes.key
echo "iv: $AESIV" >> aes.key
echo "secret" > secret.dat
```

2. Use Openssl to encrypt the file with key and vi

```
openssl enc -e -aes-128-cbc -in secret.dat -out encrypt.data -K $AESKEY -iv
$AESIV
```

3. Seal the key and vi with a policy.

```
tpm2_startauthsession -S session.ctx
tpm2_policysecret -S session.ctx -c o -L secret.policy -Q
tpm2_flushcontext session.ctx
rm session.ctx
tpm2_flushcontext -t
tpm2_createprimary -Q -C o -g sha256 -G rsa -c prim.ctx

tpm2_create -Q -g sha256 -u sealing_key.pub -r sealing_key.priv -i aes.key -
C prim.ctx -L secret.policy
rm aes.key
tpm2_flushcontext -t
tpm2_load -C prim.ctx -u sealing_key.pub -r sealing_key.priv -c
sealing_key.ctx -Q
```

4. For decrypting the file, user needs to unseal the object by the policy to get key and vi

```
tpm2_startauthsession --policy-session -S session.ctx
tpm2_policysecret -S session.ctx -c o -L secret.policy -Q

tpm2_unseal -p "session:session.ctx" -c sealing_key.ctx > unseal.dat
tpm2_flushcontext session.ctx
rm session.ctx
tpm2_flushcontext -t

UNSEALKEY=`grep key unseal.dat | awk '{print $2}'`
UNSEALIV=`grep iv unseal.dat | awk '{print $2}'`

#Use key and vi to decrypt the encrypted file
openssl aes-128-cbc -d -in encrypt.data -out secret.dec -K $UNSEALKEY -iv
$UNSEALIV
cat secret.dat
```

Part-3 Quote

1. Can we use the SRK to produce a quote? Why is the endorsement hierarchy used for quote?

   SRK is used to encrypt and store other keys, we cannot use SRK to produce a quote. Instead of SRK, we can use AIK to prodece a quote, because AIK is used to sign data created by the TPM, such as quotes and certifications of other TPM keys.

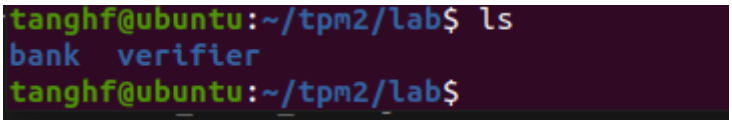
   Endorsement hierarchy is intended for privacy-sensitive operations. And quote contains what are possibly privacy-sensitive fields: resetCount , restartCount and the firmware version. To avoid this information exposured, the endorsement hierarchy is used for quote.

2. You want your bank to provide a quote of the PCRs on a server before you start sharing private information. Assume that the public key to verify a quote can be shared with you in a secure manner.Your task is to demonstrate how your bank would generate a quote to prove the state of their server/software. ; either a detailed description or an example implementation. You are free to use placeholder measurements for the PCR values used in this remote attestation process. If you use any placeholder values, ensure you explain what would take their place in a real environment. The purpose of providing PCR is for you to

make a decision if you can trust your bank server. What information do you want to request from your bank to capture in the PCRs? Think about why this information is important for you to make a trust decision.

Implementation:

I used two folders to demonstrate the remote attestation, one folder as the bank and the other as the verifier.



- Data preparation,Key exchange

  1. Use a random hash extend the specified PCRs of TMP in Bank which will be placeholder measurements in attestation.

```
pcr0=`echo -n "measurement0" | openssl dgst -sha256 -binary | xxd -p
-c 32` #PCR0: Code of BIOS
pcr1=`echo -n "measurement1" | openssl dgst -sha256 -binary | xxd -p
-c 32` #PCR1: Host Platform Configuration
pcr2=`echo -n "measurement2" | openssl dgst -sha256 -binary | xxd -p
-c 32` #PCR2: Option ROM Code

tpm2_pcrextend 0:sha256=$pcr0
tpm2_pcrextend 1:sha256=$pcr1
tpm2_pcrextend 2:sha256=$pcr2
```

  2. Create EK and AK in bank, then sending the AK to verifier

```
tpm2_createek -c ek.ctx -G rsa -u ekpub.pem -f pem
tpm2_createak -C ek.ctx -c ak.ctx -G rsa -s rsassa -g sha256 -u
akpub.pem -f pem -n ak.name
cp ./akpub.pem ../verifier/
```

- Simulate the communication process

  1. Verifier sends the request and nonce to bank for attestation

```
echo "pcr-selection: $GOLDEN_PCR_SELECTION" > pcrlist.txt
NONCE=`dd if=/dev/urandom bs=1 count=32 status=none | xxd -p -c32`
echo "nonce: $NONCE" >> pcrlist.txt
cp pcrlist.txt $bank_location/.
```

  2. When bank receives the request, it generates quote and send this quote to verifier

```
tpm2_quote --key-context ak.ctx --message attestation_quote.dat \
    --signature attestation_quote.signature \
    --qualification "$service_provider_nonce" \
    --pcr-list "$pcr_selection" \
    --pcr pcr.bin -Q
```

  3. After verifier gets the quote, it verifies the signature and nonce firstly.

```
tpm2_checkquote --public akpub.pem --qualification "$NONCE" \
--message attestation_quote.dat --signature
attestation_quote.signature \
--pcr pcr.bin -Q
```

4. The verifier then extracts a PCRdigest from the TPMS_ATTEST to match the digest information it holds

```
testpcr=`tpm2_print -t TPMS_ATTEST attestation_quote.dat | \
grep pcrDigest | awk '{print $2}'`
rm -f attestation_quote.dat
if [ "$testpcr" == "$GOLDEN_PCR" ];then
    LOG_INFO "$software_status_string"
else
    LOG_ERROR "$software_status_string"
    echo -e "      \e[97mDevice-PCR: $testpcr\e[0m"
    echo -e "      \e[97mGolden-PCR: $GOLDEN_PCR\e[0m"
    return 1
fi
```

What would take placeholder values' place in a real environment.

In real world the measurements in PCRs may be:

- hash of BIOS boot block
- hash of Boot loader
- hash of Important OS kernel file
- hash of application binaries and configuration
- hash a set of malware signatures applications have been assessed against

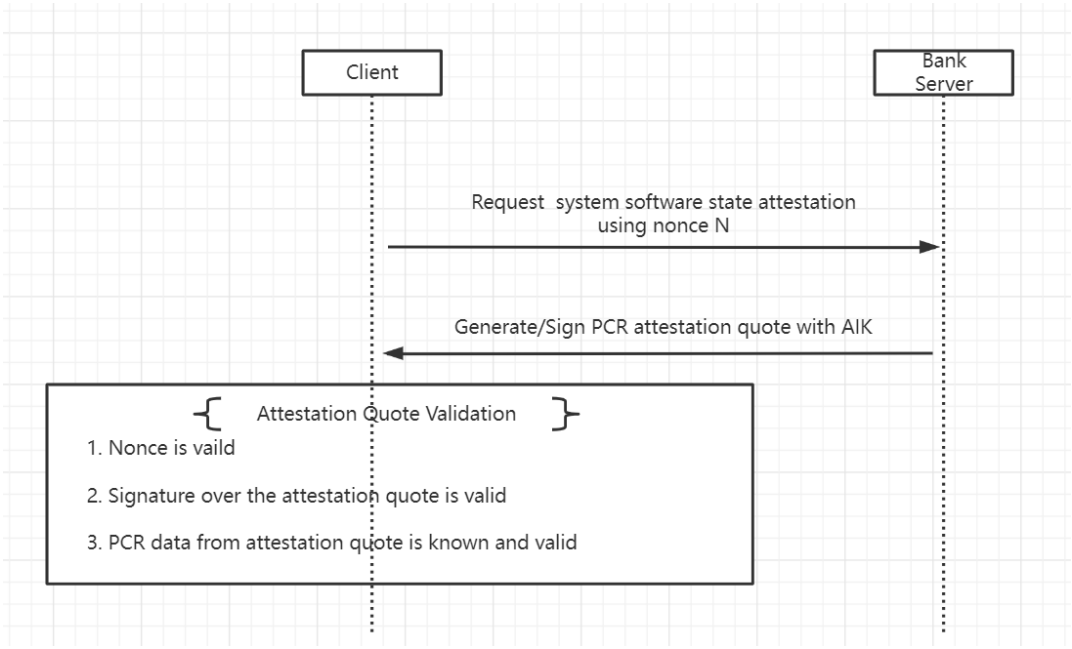
What information do you want to request from your bank to capture in the PCRs?

- The quote message that makes up the data that is signed by the TPM.
- A Signature file of the quote
- A file includes the list of PCR values that were included in the quote.

3. Describe the protocol between you and your bank to request, exchange and verify the quote. How can you be sure of the identity/code/configuration of the bank server and if the quote is legitimate?

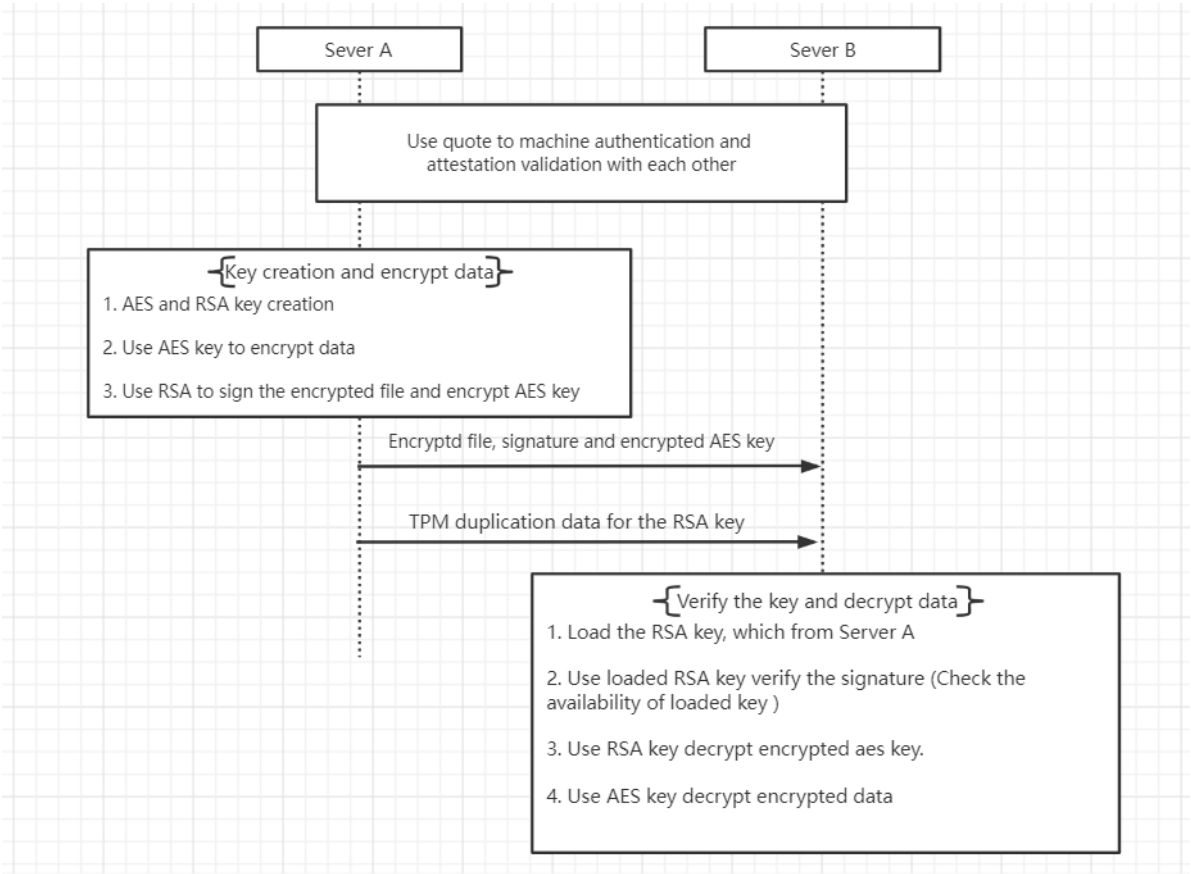
   Note: If we think the bank server is compromised and some kind of an attacker is sending the same (old) quote all over again to you, in order to make you believe nothing is wrong with the server. Can you provide an algorithm/method to prevent this kind of attack from happening again

   1. Extend the PCRs using hash of the software version, code and configurations
   2. Verifying process:



The nonce N is used to against replay attacks for preventing attacker sending the same (old) quote.

Part-4 Putting It All Together



I use two TPMs and two folders to mimic this process. - Folder serverA with TPM-A on port 2321, folder serverB with TPM-B on port 2323.

1. **Machine authentication and attestation validation**

   Before moving the data from Server A and B, it is necessary to machine authentication and attestation validation with each other. (Demonstration in Part-3)

2. **Key creation and encrypt data**

- Use TPM generate a symmetric key and a duplicable asymmetric key in folder serverA

```
#Use TPM generate a duplicable asymmetric key
tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx -T
mssim:host=localhost,port=2321

tpm2_startauthsession -S session.dat -T mssim:host=localhost,port=2321

tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate -
T mssim:host=localhost,port=2321

tpm2_flushcontext session.dat -T mssim:host=localhost,port=2321

rm session.dat

tpm2_create -C primary.ctx -g sha256 -G rsa -r key.prv -u key.pub  -L
dpolicy.dat -a "sensitivedataorigin|userwithauth|decrypt|sign" -T
mssim:host=localhost,port=2321

tpm2_flushcontext -t  -T mssim:host=localhost,port=2321
tpm2_load -C primary.ctx -r key.prv -u key.pub -c key.ctx -T
mssim:host=localhost,port=2321
tpm2_readpublic -c key.ctx -o dup.pub -T mssim:host=localhost,port=2321

#Use TPM generate a symmetric key for openssl
AESKEY=`tpm2_getrandom 16 --hex -T mssim:host=localhost,port=2321`
AESIV=`tpm2_getrandom 16 --hex -T mssim:host=localhost,port=2321`
echo "key: $AESKEY" > aes.key
echo "iv: $AESIV" >> aes.key
```

- Encrypt data and symmetric key for transmitting
  - Use Openssl with aes key to encrypt protected data and then sign this encrypted file by asymmetric key.

```
#Encrypt data and symmetric key
echo 'important data....' > protect.dat

openssl enc -e -aes-128-cbc -in protect.dat -out encrypt.data -K
$AESKEY -iv $AESIV

tpm2_sign -c key.ctx -g sha256 -o sign.raw encrypt.data -T
mssim:host=localhost,port=2321
```

  - Use asymmetric public key to encrypt the aes key.

```
tpm2_rsaencrypt -c key.ctx  -o aes.encrypted aes.key -T
mssim:host=localhost,port=2321
```

3. **Send the encrypted data, signature and encrypted aes key to Server B**

```
cp aes.encrypted encrypt.data sign.raw ../serverB/
```

4. **Asymmetric key Duplication**

   Use tpm2_duplicate to migrate the RSA key from Server A to Server B

On Server B，create a Storage key as parent key.

```
#!/bin/sh
tpm2_createprimary -C o -g sha256 -G rsa -c primary.ctx  -T
mssim:host=localhost,port=2323

tpm2_create  -C primary.ctx -g sha256 -G rsa -r new_parent.prv  -u
new_parent.pub -a "restricted|sensitivedataorigin|decrypt|userwithauth" -T
mssim:host=localhost,port=2323

cp new_parent.pub ../serverA
```

On Server A， start the duplication

```
#!/bin/sh

tpm2_startauthsession --policy-session -S session.dat -T
mssim:host=localhost,port=2321

tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate -T
mssim:host=localhost,port=2321

tpm2_loadexternal -C o -u new_parent.pub -c new_parent.ctx -T
mssim:host=localhost,port=2321

tpm2_flushcontext --transient-object -T mssim:host=localhost,port=2321

tpm2_duplicate -C new_parent.ctx -c key.ctx -G null -p "session:session.dat"
-r dup.dpriv -s dup.seed -T mssim:host=localhost,port=2321

cp dup.* ../serverB
```

On Server B， load the asymmetric key under new parent

```
#!/bin/sh
tpm2_startauthsession --policy-session -S session.dat -T
mssim:host=localhost,port=2323

tpm2_policycommandcode -S session.dat -L dpolicy.dat TPM2_CC_Duplicate  -T
mssim:host=localhost,port=2323

tpm2_flushcontext --transient-object -T mssim:host=localhost,port=2323

tpm2_load -C primary.ctx -u new_parent.pub -r new_parent.prv -c
new_parent.ctx -T mssim:host=localhost,port=2323

tpm2_import -C new_parent.ctx -u dup.pub -i dup.dpriv -r dup.prv -s dup.seed
-L dpolicy.dat -T mssim:host=localhost,port=2323

tpm2_flushcontext --transient-object  -T mssim:host=localhost,port=2323

tpm2_load -C new_parent.ctx -u dup.pub -r dup.prv -c dup.ctx -T
mssim:host=localhost,port=2323
```

5. **Verify the infomation**

- Use asymmetric private key to verify the signature, on Server B

```
tpm2_verifysignature -c dup.ctx -g sha256 -s sign.raw -m encrypt.data -T
mssim:host=localhost,port=2323
```

- Decrypt the aes key, and then use aes key to decrypt the encrypted data.

```
tpm2_flushcontext --transient-object  -T mssim:host=localhost,port=2323
tpm2_rsadecrypt -c dup.ctx -o aes.key aes.encrypted -T
mssim:host=localhost,port=2323

KEY=`grep key aes.key | awk '{print $2}'`
IV=`grep iv aes.key | awk '{print $2}'`

openssl aes-128-cbc -d -in encrypt.data -out data -K $KEY -iv $IV
cat data
```