Exercise 4: Data Sealing

1 Introduction

This exercise will introduce the Tspi functions for binding and sealing. In the practical assignments you will extend a given program with sealing functions.

In addition to the TPM functions we already know from the lectures and the exercises (integrity measurement, authenticated boot, or creating and storing various keys), the TPM also provides encryption and decryption mechanisms to protect your data from unauthorized reading. The two functions provided by the TPM to encrypt data are **binding** and **sealing**. Both of them encrypt data using asymmetric techniques. Since asymmetric algorithms are very time-consuming, we run into perfomance problems if we have to encrypt large amounts of data. The TPM by itself does not support mechanisms to encrypt data with symmetric algorithms, but it is able to create symmetric keys that will be securely stored by the TPM and can later be used by an external encryption engine to guarantee data confidentiality.

If the TPM binds data, then data is simply encrypted using asymmetric cryptography. The Tspi functions for binding are `Tspi_Data_Bind` and `Tspi_Data_Unbind`. The asymmetric keys used for binding can be migratable or non-migratable storage keys. If non-migratable storage keys are used, the encrypted data is bound to a specific platform. Otherwise, we have no platform binding and the ciphertext can be decrypted on different platforms using the appropiate private key.

Sealing is an extension to binding. Contrary to binding, only non-migratable storage keys can be used to seal data. Consequently, the encrypted data is always bound to a specific platform. Additionally, the Tspi function `Tspi_Data_Seal` allows to specify one or more PCR registers for the sealing operation. Thus, the ciphertext includes the platform's state at the time of encryption. Therefore, the ciphertext can only be decrypted with `Tspi_Data_Unseal` if the platform is in the same state as it was at the time of encryption.

In the remainder of this section we introduce some new TSP objects that are necessary for sealing and unsealing data. These TSP objects are all connected to a context object and can be created with the Tspi function `Tspi_Context_CreateObject`:

```
TSS_RESULT Tspi_Context_CreateObject (TSS_HCONTEXT hContext,
        TSS_FLAG objectType, TSS_FLAG initFlags, TSS_HOBJECT* phObject);
```

The second argument indicates the object type to create. The object types that can be defined in this function are listed in table 1. The third argument holds the attributes of the object type. A full list of `initflags` can be found in the appendix in table 4.

1.1 Policy Objects

Some commands need authorization data to execute successfully. Among these are commands for loading and creating keys, encrypting and decrypting data, or taking

Object Type	Description
TSS_OBJECT_TYPE_POLICY	Policy object
TSS_OBJECT_TYPE_RSAKEY	RSAKey object
TSS_OBJECT_TYPE_ENCDATA	Encrypted data object; sealed data or bound data
TSS_OBJECT_TYPE_PCRS	PCR composite object
TSS_OBJECT_TYPE_HASH	Hash object
TSS_OBJECT_TYPE_NV	Non Volatile RAM object
TSS_OBJECT_TYPE_MIGDATA	CMK-Migration data object
TSS_OBJECT_TYPE_DAA	DAA object

Table 1: Object Types for `Tspi_Context_CreateObject`

ownership of a TPM. The authorization data is held by policy objects. There are three policy types, listed in table 2. The most common policy type is the usage policy. Migration policies are only used for creating migratable keys and operator policies only hold authorization data when the TPM is deactivated temporarily. The method to retrieve

InitFlag	Description
TSS_POLICY_USAGE	Policy object used for authorization
TSS_POLICY_MIGRATION	Policy object used for migration
TSS_POLICY_OPERATOR	Policy object used for operator authorization

Table 2: Policy Types

authorization data is called **secret mode**. By default, a pop-up dialog box appears where the user enters the proper password. The TSS also provides additional secret modes that are listed in table 3.

For the context object and the TPM object a policy is created implicitly by the TSP. All subsequently created TSP objects get a reference to the policy of the context object. In case that a specific secret/policy is needed for a particular TSP object, the following sequence of commands should be issued.

1. `Tspi_Context_CreateObject`: Create a policy object
2. `Tspi_Policy_SetSecret`: Set the secret for the policy
3. `Tspi_Policy_AssignToObject`: Assign the policy object to a TSP object

1.2 Key Objects

TPM Keys consist of RSA public/private key pairs and are represented by TSP key objects. To create a new TPM key, it is necessary to create a TSP key object that holds

Secret Mode	Description
TSS_SECRET_MODE_NONE	No authorization will be processed
TSS_SECRET_MODE_SHA1	Secret in the form of 20 bytes of SHA-1 data The secret will not be touched by the TSP
TSS_SECRET_MODE_PLAIN	The secret string passed in will be hashed using SHA-1
TSS_SECRET_MODE_POPUP	TSP will ask for a secret by displaying a GUI pop-up window
TSS_SECRET_MODE_CALLBACK	The application will provide a callback function for authorization data

Table 3: Secret Modes

all the properties of the key (e.g., length and type) first. Afterwards, the key object has to be forwarded to the TPM that generates the key pair. Finally, the generated key pair will be returned to the TSP. Since the generated key pair can only be a child of another key pair (root key pair is always the Storage Root Key (SRK)), the parent key pair has to be loaded into the TPM before creating the new key pair. Thus, the following three commands should be issued to create a new key pair:

1. `Tspi_Context_LoadKeyByUUID`: Load the parent key pair
2. `Tspi_Context_CreateObject`: Create a key object
3. `Tspi_Key_CreateKey`: Generate the key pair

The TSS provides additional storage for keys. Keys can either be stored (or registered) on disk at the TSS Core Services (TCS) level or at the TSS Service Provider (TSP) level. If keys are registered on disk at the TCS level, then they are stored in the so-called system persistent storage (`TSS_PS_TYPE_SYSTEM`). These keys are accessible by any application and any application is able to unregister keys at the TCS level without restriction. Keys that are registered on disk at the TSP level instead are only accessible by applications of a particular user. These keys are stored in the user persistent storage (`TSS_PS_TYPE_USER`).

To address keys in the persistent storage, we use the structure TSS_UUID:

```
typedef struct tdTSS_UUID
{
        UINT32 ulTimeLow;
        UINT16 usTimeMid;
        UINT16 usTimeHigh;
        BYTE bClockSeqHigh;
        BYTE bClockSeqLow;
        BYTE rgbNode[6];
} TSS_UUID;
```

The Tspi functions that are necessary to register and unregister keys are the following ones:

1. `Tspi_Context_RegisterKey`
2. `Tspi_Context_UnregisterKey`

1.3 Encrypted Data Objects

If data has to be sealed (or bound), an encrypted data object is necessary that holds the sealed data blobs. If we want to unseal encrypted data, it is necessary to encapsulate the encrypted data into an encrypted data object first. The Tspi functions `Tspi_SetAttribData` and `Tspi_GetAttribData` can be used to insert or extract data to or from an encrypted data object. However, if the Tspi function for sealing data is issued, the encrypted data will automatically be inserted into a specified encrypted data object.

1.4 PCR Composite Objects

Since sealing should yield platform binding, a TSP object is needed that can hold hash values that are stored in the PCR registers. For this reason, the appropiate TSP object is the PCR composite object. The Tspi function `Tspi_TPM_PcrRead` is used to retrieve a particular hash value from one specified PCR register. This function will return a hash value that can be loaded into the PCR composite object using the Tspi function `Tspi_PcrComposite_SetPcrValue`.

2 Theoretical Assignments (5 Points)

1. What is the main difference between binding and sealing?

2. Sealing can only be enforced with non-migratable storage keys. Explain the reasons why we cannot use migratable storage keys for sealing and why we can use them for binding?

3. If you use a non-migratable storage key for binding, data is bound to a specific platform. Is binding with a non-migratable storage key equal to sealing? Keep in mind that there are different modes how to seal data. Justify your answer!

4. Consider a user that wants to create a key that should be accessible by any application. Further, the user desires that the key can be used for signing on different platforms. Moreover, the key size should be 2048 bit. Which initflags have to be set? Explain in which persistent storage part the key has to be registered!

5. Explain why keys that should be accessible by any application have to be registered at the TCS level and not at the TSP level? Justify your answer!

3 Practical Assignments

Your task is to extend a given program with Tspi functions in order to provide sealing of some data. The program simulates accounting transactions. The effective transaction is delegated to a shared library that returns the result of a transaction back to the accounting program. Moreover, all transactions are logged in a file named *transaction.log*.

The accounting program performs its task as follows:

1. If the log file *transaction.log* is found in the file system, the log file will be opened and the content will be displayed on `stdout`. Otherwise, the accounting program will create a new log file.
2. Afterwards, the accounting program invokes the function `executeTransaction` from the shared library `libtransact.so`. This function returns a fixed value that is stored in the variable `amount`.
3. After writing the value of `amount` to `stdout`, the new amount will be appended to *transaction.log*.
4. Finally, the modified file will be saved on disk and the accounting program terminates by closing the *transaction.log*.

The directory *Exercise Data* contains an archive with all necessary files to run the accounting programm. For loader to find the shared library `libtransact.so`, you have to set the environment variable `LD_LIBRARY_PATH` properly.[1] Further, you will find a make file in the */account/src* directory that you can use to compile the accounting program.

Your task is to seal the data retrieved from the shared library to PCR register 13 and append the sealed data to the *transaction.log*. Since the log file only holds sealed data, it is also necessary to unseal the data before writing it to `stdout`. Follow the steps described in the next sections!

3.1 Creating a Storage Key (5 Points)

In your exercise folder you will find a file named *generateKey.c*. Extend the program in order to generate a Storage Key that can be used for sealing.

1. Modify the initflags in a way that you can create a key object with the following properties:

   Key length: 2048 Bit

   Key type: Storage Key

   Not migratable and authorization disabled

2. Load the Storage Root Key (SRK) from system persistent storage!

[1] `$ export LD_LIBRARY_PATH="..."`

3. Create a key object with all initflags necessary for the new storage key!
4. Generate the new storage key without binding it to PCR values! The parent key is the SRK.
5. Register the key in the system persistent storage with the UUID already specified in the source code!

3.2 Setup for sealing and unsealing (10 Points)

In the archive you will find a file named *accounting.c*. Further, in the source code you will find a function called `TSS_initialize`. At the moment, this function only creates a context and a TPM object. Extend the function in the following way by using the already declared global TPM variables:

1. Create an encrypted data object!
2. Create a usage policy object that you will need to set the authorization data for the encrypted data object!
3. For the encrypted data object it is necessary to set a secret in the newly created policy. Set a secret and assign the policy to the encrypted data object! Hint: The secret and the secret mode are defined at the beginning of the source code.
4. Since you have to seal the data to PCR register 13, create a PCR composite object!
5. Read the value of PCR register 13 and set the value in the PCR composite object!
6. Finally, load the storage key created in 3.1 by its UUID!

3.3 Sealing and Unsealing (5 Points)

1. Complete the function `addAmount()`! Seal the value of the variable `amount` to PCR register 13! The sealing key should be the key created in Section 3.1.
2. To append the encrypted data to a file, you have to extract the encrypted data from the encrypted data object. Implement the appropriate Tspi function with the attribflag `TSS_TSPATTRIB_ENCDATA_BLOB` and subflag `TSS_TSPATTRIB_ENCDATABLOB_BLOB`! Use `*blob` to point to the extracted data!
3. Add the Tspi function for unsealing to the function `printAccount()`!

3.4 Testing and Modifications (5 Points)

1. Reboot the system and start the accounting program with the process starter created in exercise 2! Repeat this procedure several times! Write down your observations and explain why you have to reboot the system before running the accounting program with the process starter!

2. Now that you have executed the accounting program according to 3.4.1, modify the accounting program by adding a constant value to the value you are receiving from the library function `executeTransaction`! Please store the old program binary and the source code since we still need them in this exercise! Compile the modified program and follow the procedure from 3.4.1! Write down your observations!

3. Now once again use the old program binary to append a new amount to the *transaction.log*! Write down your observations and explain the different behavior!

4. In 3.4.2 you modified the source file to add a constant value to the variable `amount`. An alternative to change the value of `amount` would be to modify the library function `executeTransaction` directly. Therefore, change the value directly in the library function, compile the modified library[2] and execute the accounting program according to 3.4.1! Write down your observations! Do you see any security problems regarding the modification you made?

[2] You will find a make file in *.../ex1-account/src/libtransact*

Appendix/Bibliography

InitFlag	Description
TSS_KEY_SIZE_DEFAULT	Default size
TSS_KEY_SIZE_512	Key size 512
TSS_KEY_SIZE_1024	Key size 1024
TSS_KEY_SIZE_2048	Key size 2048
TSS_KEY_SIZE_4096	Key size 4096
TSS_KEY_SIZE_8192	Key size 8192
TSS_KEY_SIZE_16384	Key size 16384
TSS_KEY_TYPE_STORAGE	Key for wrapping keys
TSS_KEY_TYPE_SIGNING	Key for signing operations
TSS_KEY_TYPE_BIND	Binding Key
TSS_KEY_TYPE_AUTHCHANGE	Ephemeral key used during the ChangeAuthAsym process only
TSS_KEY_TYPE_IDENTITY	Key for an identity
TSS_KEY_TYPE_LEGACY	Key that can perform signing and binding
TSS_KEY_TYPE_AUTHCHANGE	An ephemeral key used to change authorization value
TSS_KEY_NON_VOLATILE	Key is non-volatile. MAY be unloaded at startup
TSS_KEY_VOLATILE	Key is volatile. MUST be unloaded at startup
TSS_KEY_NOT_MIGRATABLE	Key is not migratable (DEFAULT)
TSS_KEY_MIGRATABLE	Key is migratable
TSS_KEY_CERTIFIED_MIGRATABLE	Key is certified migratable
TSS_KEY_NOT_CERTIFIED _MIGRATABLE	Key is not certified migratable
TSS_KEY_NO_AUTHORIZATION	Key needs no authorization (DEFAULT)
TSS_KEY_AUTHORIZATION	Key needs authorization
TSS_KEY_AUTHORIZATION_PRIV_US _E_ONLY	Key needs authorization for use of private portion of key
TSS_KEY_STRUCT_DEFAULT	Key object uses a 1.1 TCPA_KEY or 1.2 TCPA_KEY12 structure based on the Context's TSS_TSPATTRIB_CONTEXT _VERSION_MODE attribute (DEFAULT)
TSS_KEY_STRUCT_KEY	Key object uses a 1.1 TCPA_KEY structure
TSS_KEY_STRUCT_KEY12	Key object uses a 1.2 TCPA_KEY12 structure
TSS_KEY_EMPTY_KEY	no TCG key template (empty TSP key object)

InitFlag	Description
TSS_KEY_TSP_SRK	use a TCG SRK template (TSP key object for SRK)
TSS_ENCDATA_SEAL	Data object is used for seal operation
TSS_ENCDATA_BIND	Data object is used for bind operation
TSS_ENCDATA_LEGACY	Data for legacy bind operation
TSS_HASH_DEFAULT	Default hash algorithm
TSS_HASH_SHA1	Hash object with algorithm SHA1
TSS_HASH_OTHER	Hash object with other algorithm
TSS_POLICY_USAGE	Policy object used for authorization
TSS_POLICY_MIGRATION	Policy object used for migration
TSS_POLICY_OPERATOR	Policy object used for operator authorization
TSS_PCRS_STRUCT_DEFAULT	PcrComposite object uses a 1.1 TCPA_PCR_INFO structure or a 1.2 TCPA_PCR_INFO_LONG structure based on the Context's TSS_TSPATTRIB_CONTEXT _VERSION_MODE attribute (DEFAULT)
TSS_PCRS_STRUCT_INFO	PcrComposite object uses a 1.1 TCPA_PCR_INFO structure (DEFAULT)
TSS_PCRS_STRUCT_INFO_LONG	PcrComposite object uses a 1.2 TCPA_PCR_INFO_LONG structure
TSS_PCRS_STRUCT_INFO_SHORT	PcrComposite object uses a 1.2 TCPA_PCR_INFO_SHORT structure

Table 4: Initflags for `Tspi_Context_CreateObject`

References

[1] TCG Software Stack (TSS) Specification Version 1.2, `https://www.trustedcomputinggroup.org/specs/TSS/TSS_Version_1.2_Level_1_FINAL.pdf`

[2] A Practical Guide to Trusted Computing, D. Challener, K. Yoder, R. Catherman, D. Safford and L. Van Doorn, 2008, ISBN-10: 0132398427

[3] manpages, `http://linux.die.net/man/`