Design Document: Java API for Trusted Application

RUI YANG

rui.yang@aalto.fi

Abstract

Based on GlobalPlatform specification, Open-TEE paved a way for normal application developers to develop and deploy GP-compiant Trusted Applications (TA) in Trust Execution Environment (TEE). However, in order to develop a Client Application (CA) (especially Android application), there still lacks an efficient way of exposing the underlining C-based APIs to the application layer which is written in Java. In this design document, this problem will be adderssed in more details and the possibility to wrap the C APIs into Java APIs is discussed.

KEYWORDS: Open-TEE, Java API, JNI

1 Problem Description

Open-TEE is a virtual TEE which is based on GlobalPlatform (GP) TEE specifications. It can run on a mobile device on which without real GP-Compliant TEE hardwares. If the mobile device is equipped with GP-Compliant TEE hardware, via one module of Open-TEE called "libtee" `https://github.com/Open-TEE/libtee` with corrsponding lower layer linux driver `https://github.com/Open-TEE/tee-engine-Driver`, Open-TEE can allow the communications between CAs in Rich Execution Environment (REE) and TAs in the real TEE.

If the mobile device does not have a real GP-Compliant TEE hardware, Open-TEE can run as an application in REE itself, which does not provide the security features of a TEE. TAs can be deployed in Open-TEE and the communications between CAs and TAs are provided.

The corresponding GP specifications are written in C as such Open-TEE is implemented in C programming language. The problems are list as follows which will occur once Android application developers start using Open-TEE.

1.1 Problem With C Level API

The first problem is how to use the C APIs in the application layer. Java Native Interface (JNI) provides the ways to allow the communications between C and Java. So a middle layer between the application layer and C API layer should be provided to reduce the redundant works of application developers.

1.2 Problem With Shared Memory

As stated above, the GP TEE specifications only focus on C. There is one notion called "Shared Memory" which is utilized in the specification to avoid big string of memory copies between the CA and TA if they wants to communicate with each other. In C programming language, sharing memory between two processes can be achieved by declaring that these two processes want to share part of their memory. However, since the CA in our scenario is written in Java which does not have the notions of referencing to the memory address in run time environment, which makes the notion of "Shared Memory" hard to implement.

1.3 Problem With Transferring Big String of Data

Moreover, since the architecture of Open-TEE for android has been changed in a way that the Open-TEE will run in a seperate android application which running as a service, there is a problem with transferring big string of data from one user application to our Open-TEE CA. For instance, the user application wants to transfer a big file to the TA via CA in Open-TEE to encrypt or decrpt.

2 Possible Solutions

Android Shared Memory: Ashmem How to use `http://stackoverflow.com/questions/16099904/how-to-use-shared-memory-ipc-in-android` and `http://notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory`

Currently there are two ways to solve the problem No.3. The first solution is to create a shared memory between the user application and CA. So I am trying to share the MemoryFile FD between two different applications. The second solution is to create a parcable object.

3 Data Type

All the constants specified in GlobalPlatform TEE Client API specification are the same in this report. The following list defines the extra data types.

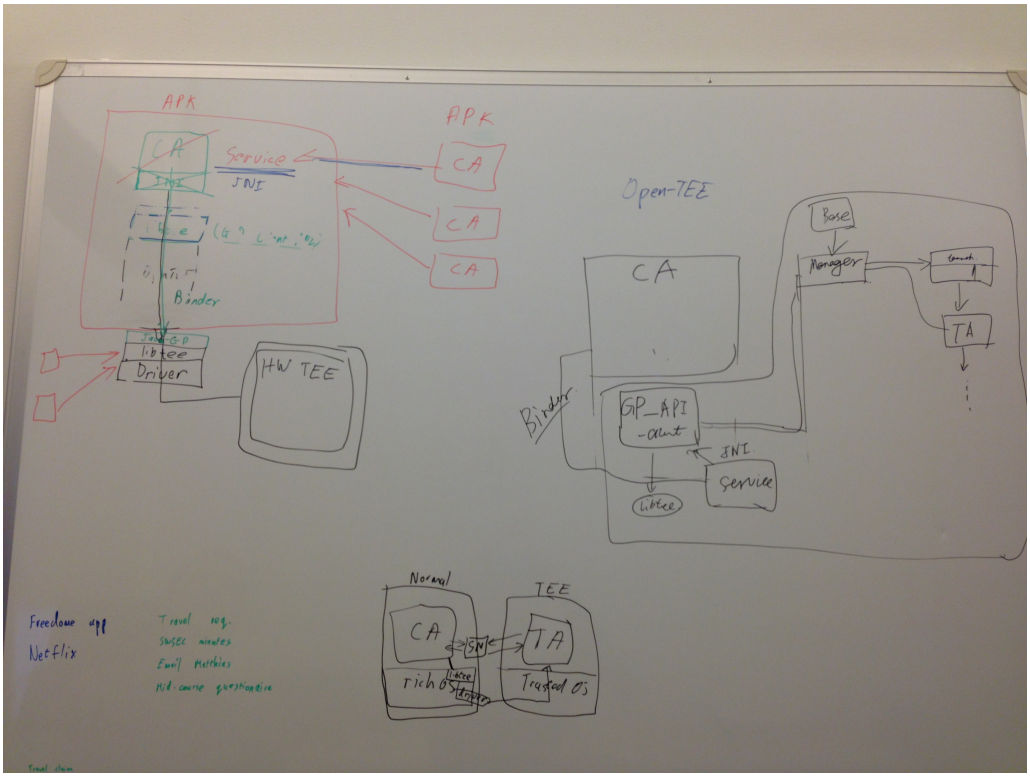

Figure 1: Draft for reminder

1. class TEEC_SharedMemory{

   int reference = -1; // reference number to shared memory

   int size = 0; // size of shared memory

   int flags = 0; // flags for the I/O direction

   set{}, get{} // set and get functions for each variable

   }

2. class TEEC_RESULT_UNION{

   TEEC_RESULT result;

   int referenceNum;

   set{}, get{} // set and get functions for each variable

   }

3. class TEEC_OPERATION{

   int started;

   int paramsTypes;

   TEEC_PARAMETER[] params = new TEEC_PARAMETER[4];

   set{}, get{} // set and get functions for each variable

   }

4. class TEEC_PARAMETER{

```
TEEC_TempMemoryReference tmpref;
TEEC_RegisteredMemoryReference memref;
TEEC_Value value;
set{}, get{} // set and get functions for each variable
}
```

5.
```
class TEEC_TempMemoryReference{
int size;
int[] buffer;
set{}, get{} // set and get functions for each variable
}
```

6.
```
class TEEC_RegisteredMemoryReference{
int memRef;
int size;
int offset;
set{}, get{} // set and get functions for each variable
}
```

7.
```
class TEEC_Value{ int a; int b;
set{}, get{} // set and get functions for each variable
}
```

4 Java APIs

1.
```
int TEEU_GetAPPID(
static String name);
```

2.
```
TEEC_RESULT TEEC_InitializeContext(
int APP_ID);
```

3.
```
void TEEC_FinalizeContext(
int APP_ID);
```

4.
```
TEEC_RESULT_UNION TEEC_RegisterSharedMemory(
int APP_ID);
```

5.
```
TEEC_RESULT_UNION TEEC_AllocateSharedMemory(
int APP_ID);
```

6. `void TEE_ReleaseSharedMemory(`
   `int APP_ID,`
   `TEEC_RESULT_UNION.referenceNum);`

7. `TEEC_RESULT_UNION TEEC_OpenSession(`
   `int APP_ID,`
   `int destination,`
   `int connectionMethod,`
   `int connectionData,`
   `TEEC_Operation operation);`

8. `void TEEC_CloseSession(`
   `int APP_ID,`
   `TEEC_RESULT_UNION.referenceNum);`

9. `TEEC_RESULT TEEC_InvokeCommand(`
   `int APP_ID,`
   `int mSessionReferenceNum,`
   `int command_ID,`
   `TEEC_OPERATION operation);`

10. `void TEEC_RequestCancellation(`
    `int APP_ID,`
    `TEEC_OPERATION operation);`