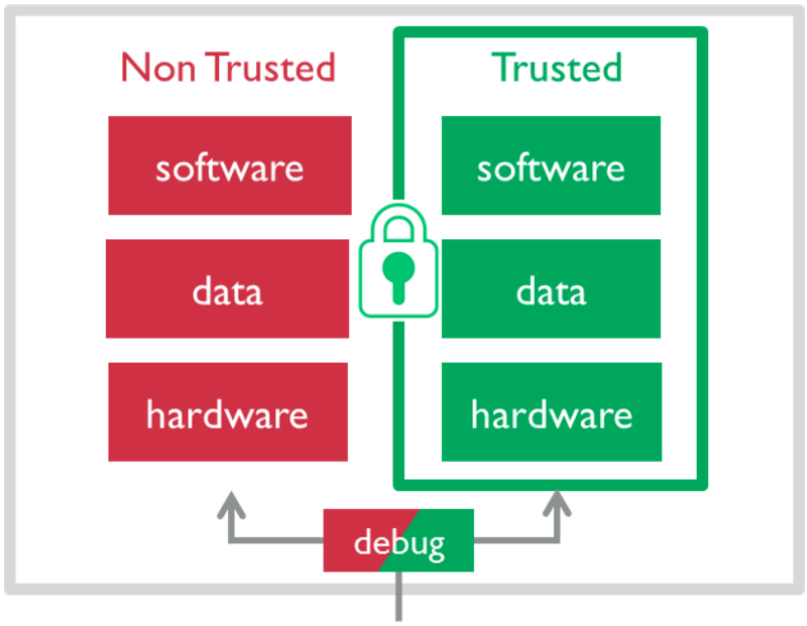
Security Tutorial

ARM® TrustZone®

ARM® TrustZone® technology is a System on Chip (SoC) and CPU system-wide approach to security. TrustZone is hardware-based security built into SoCs. The family of TrustZone technologies can be integrated into any ARM Cortex-A and the latest Cortex-M23 and Cortex-M33 based systems.

At the heart of the TrustZone approach is the concept of secure and non-secure worlds that are hardware separated. Non-secure software is blocked from accessing secure resources directly. By creating a security subsystem, assets can be protected from software and common hardware attacks. The diagram below illustrates the high-level architecture of TrustZone technology.



Trusted Execution Environment (TEE)

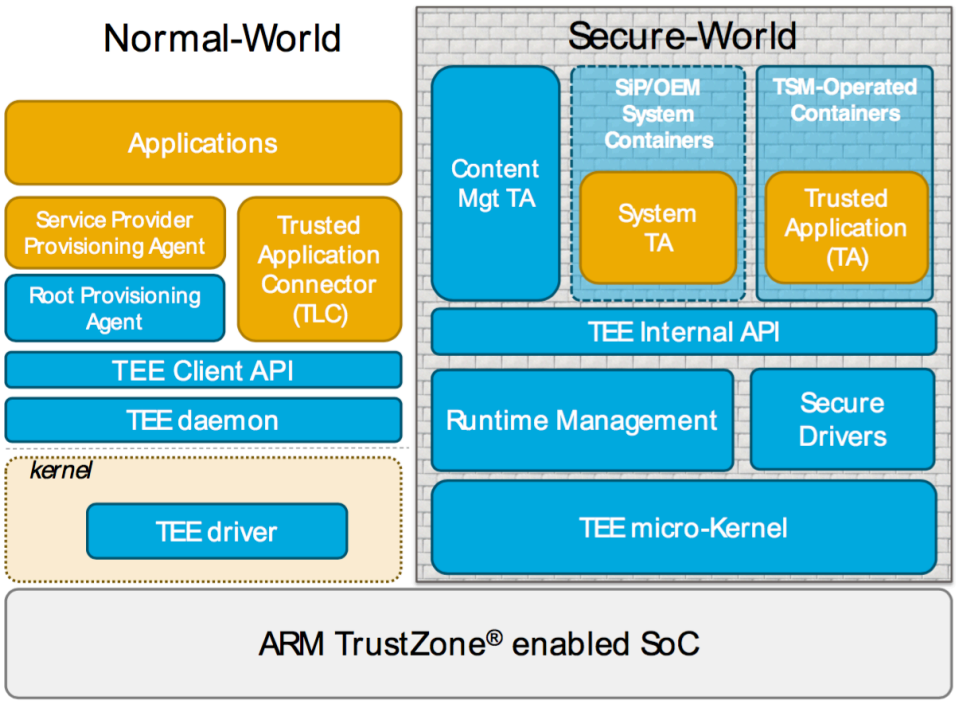
TrustZone technology within Cortex-A based application processors is commonly used to run trusted boot and a trusted OS to create a Trusted Execution Environment (TEE). Applications that run in the secure world are called **Trusted Applications (TA)**.

The TEE offers the security properties of confidentiality and integrity to multiple Trusted Applications. Many TEE providers follow GlobalPlatform's API standard to enable their TEE to deliver a common security capability across platforms and markets.

Trustonic TEE

For ARTIK 5/7/10, Samsung works with industry leading security TEE provider Trustonic and introduces Trustonic Trusted Execution Environment platform to ARTIK modules.

Trustonic TEE is a portable and open TEE aiming at executing Trusted Application on a device. It includes built-in cryptographic algorithms and secure objects for secure data persistence.



Trustonic TEE uses ARM TrustZone to separate the platform into two distinct areas, Normal World with a conventional rich OS and rich applications and the Secure World.

The Secure World contains Trustonic TEE core operating system and trusted applications. It uses on-device client/server architecture to provide security functionalities to Normal World. The Normal World contains mainly software which is not security sensitive and it calls the Secure-World to get security functionality via a communication mechanism and APIs provided by Trustonic TEE. The caller in the Normal

World is usually an application, also called a Client Application.

Client Application (CA)

A Client Application uses Trusted Application Connector (TLC) to communicate with Trusted Application. TLC establishes a connection with the Trusted Application and exchanges data with it, to make Trusted Application's security features accessible to the Normal World.

The communication between Normal world and the Secure World is implemented by writing data to a world shared memory buffer. This memory is configured that both worlds have access to the shared section, each with their own execution context

Trustonic SDK

Trustonic SDK includes 2 sets of APIs. One set is the Trustonic Legacy API, The other set conforms to the standard defined by GlobalPlatform.
More information about GlobalPlatform APIs and how to use these APIs, can be found at: http://www.globalplatform.org/specificationsdevice.asp

TEE Client API

TEE Client Application is responsible for:

- Initiating a context to the TEE
- Opening and closing sessions to one or more Trusted Applications
- Registering and unregistering shared memory
- Invoking commands on sessions, passing command data and memory buffers to a Trusted Application
- Handling Trusted Application return codes and data returned either directly or in memory.

When a Client Application sends a command to a Trusted Application, the command contains a command identifier and an operation payload. An operation includes parameters that can be used to exchange data from CA to TA and from TA and CA.

TEE Trusted Application API

A trusted application is command-orientated. Client application access a service by opening a session with trusted application and sending messages with the session. When a Trusted Application receives a command, it parses the messages associated with the command, processes them and sends a response back to the Client Application.

When a Client Application opens a session with a Trusted Application, Trustonic TEE creates a new process and loads the TA binary into it and starts execution of this process. This process is then called an instance of that Trusted Application. Trusted Application processes are single-threaded and have their own address space separating them from all other processes.

Each Trusted Application must implement a few callback functions, collectively called the “TA interface”. These functions are the entry points called by the Trustonic TEE to create the instance, notify the instance that a new client is connecting, notify the instance when the client invokes a command and to notify when a client is disconnecting and also before destroying the instance. These functions must be implemented in order to link the TA binary.

Here is an example of trusted application interface:

TA Interface Operation	Description
TA_CreateEntryPoint	Trusted Application constructor. It is called once and or once in the life-time of the Trusted Application instance
TA_OpenSessionEntryPoint	The function is called whenever a client attempts to cor to the Trusted Application instance to open a new sessi
TA_InvokeCommandEntryPoint	The function is called whenever a client invokes a Trust Application command.
TA_CloseSessionEntryPoint	The function is called when the client closes a session a disconnects from the Trusted Application instance.
TA_DestroyEntryPoint	Trusted Application destructor.

```
TEE_Result TA_EXPORT TA_CreateEntryPoint(void)
{
return TEE_SUCCESS;
}


void TA_EXPORT TA_DestroyEntryPoint(void)
{
}


TEE_Result TA_EXPORT TA_OpenSessionEntryPoint(
uint32_t nParamTypes, INOUT TEE_Param pParams[4],
OUT void** ppSessionContext)
{
return TEE_SUCCESS;
}


void TA_EXPORT TA_CloseSessionEntryPoint(INOUT void*
pSessionContext)
{
}
```

```
/* ------------------------------------------------
 *   Main entry point to our TA
 * ------------------------------------------------
*/
TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint(
        INOUT void* pSessionContext,
        uint32_t nCommandID,
        uint32_t nParamTypes,
        TEE_Param pParams[4])
{
  …
   switch(nCommandID)
   {
    case CMD_SAMPLE_ROT13:
        …
        pInput      = pParams[0].memref.buffer;
        nInputSize  = pParams[0].memref.size;
        pOutput     = pParams[1].memref.buffer;
        nOutputSize = pParams[1].memref.size;

        …
        rot13((char*)pInput, (char*)pOutput,
nInputSize);
          …

        return TEE_SUCCESS;
    default:
         return TEE_ERROR_BAD_PARAMETERS;
   }
}
```

Correspondingly, these are the Client operations interface:

Client Operation	Trusted Application Effect
TEEC_OpenSession	Create a new process and call TA_CreateEntryPoint to initialize, then invoke TA_OpenSessionEntryPoint
TEEC_InvokeCommand	Unblock the process and call TA_InvokeCommandEntryPoint
TEEC_CloseSession	Call TA_CloseSessionEntryPoint.

Here is an example of the corresponding Client Application that interacts with the Trusted Application above:

```
TEE_Operation sOperation;
TEEC_Result   nError;
*session = (TEEC_Session *)malloc(sizeof(TEEC_Session));
  …
 nError = TEEC_OpenSession(context,
              *session,             /* OUT session */
              &uuid,                /* destination UUID */
              TEEC_LOGIN_PUBLIC,    /* connectionMethod */
              NULL,                 /* connectionData */
              &sOperation,    /* INOUT operation */
              NULL            /* OUT returnOrigin, optional */
          );

 memset(&sOperation, 0, sizeof(TEEC_Operation));
 sOperation.paramTypes = TEEC_PARAM_TYPES(
                         TEEC_MEMREF_TEMP_INPUT,
                         TEEC_MEMREF_TEMP_OUTPUT,
                         TEEC_NONE,
                         TEEC_NONE);
 sOperation.params[0].tmpref.buffer = (void*)plainText;
 sOperation.params[0].tmpref.size   = plainTextLength;
 sOperation.params[1].tmpref.buffer = pCipherText;
 sOperation.params[1].tmpref.size   = *nCipherTextLength;
```

```
 nError = TEEC_InvokeCommand(session,
              CMD_SAMPLE_ROT13,
              &sOperation, /* IN OUT operation */
              NULL         /* OUT returnOrigin, optional */
          );
*nCipherTextLength = sOperation.params[1].tmpref.size;


if (nError != TEEC_SUCCESS) {
…
}


TEEC_CloseSession(session);
free(session);
…
```

TEE Internal API

GlobalPlatform TEE Internal API specification defines data encryption/decryption APIs, arithmetic APIs, memory management APIs etc. Developers can use these APIs to design security algorithms to protect their data.

Here is an example that uses GlobalPlatform TEE Internal APIs to sign the data we received from normal world.

```
  TEE_Result nResult = TEE_SUCCESS;
  TEE_ObjectHandle priKey = TEE_HANDLE_NULL;
  TEE_ObjectHandle pubKey = TEE_HANDLE_NULL;
  TEE_OperationHandle op_sig = NULL;


  /* Load public and private keys */
  …
  /* Allocate a handle for a new cryptographic operation and
mode and algorithm type */
  nResult = TEE_AllocateOperation(&op_sig,
                  TEE_ALG_ECDSA_P256,
                  TEE_MODE_SIGN, bitlength);
  if (nResult != TEE_SUCCESS) {
      goto cleanup;
  }


 /* Program the key of an operation */
  nResult = TEE_SetOperationKey(op_sig, priKey);
  if (nResult != TEE_SUCCESS) {
      goto cleanup;
  }
```

```
 /*Sign a message digest within an asymmetric operation. Only
hashed message can be signed */

  nResult = TEE_AsymmetricSignDigest(op_sig, NULL, 0, toSign,
bytelength, signature, &signatureLen);

  if (nResult != TEE_SUCCESS) {

    goto cleanup;

  }

…

cleanup:

  TEE_FreeTransientObject(pubKey);

  TEE_FreeTransientObject(priKey);

  TEE_FreeOperation(op_sig);

  return nResult;
```