Trusted Execution Environments on Mobile Devices

ACM CCS 2013 tutorial

Jan-Erik Ekberg, Trustonic
Kari Kostiainen, ETH Zurich
N. Asokan, University of Helsinki and Aalto University
What is a TEE?

Trusted Execution Environment

Isolated and integrity-protected

Processor, memory, storage, peripherals

From the “normal” execution environment (Rich Execution Environment)

Chances are that:
You have devices with hardware-based TEEs in them!
But you don’t have (m)any apps using them
Outline

• A look back (10 min)
  – Why mobile devices have TEEs?
• Mobile hardware security (30 min)
  – What constitutes a TEE?
• Application development (30 min)
  – Mobile hardware security APIs + DEMO

Break (10 min)

• Current standardization (60 min)
  – NIST, Global Platform, TPM 2.0
• A look ahead (10 min)
  – Challenges and summary

Trusted Execution Environments on Mobile Devices

**Lecturers:** Jan-Erik Ekberg, Kari Kostiainen, N. Asokan

**Time:** Wednesday, Nov 6th, 2013, 9:30 am – 12:30 pm in Room B07-B08

**Abstract:** A trusted execution environment (TEE) is a secure processing environment that is isolated from the “normal” processing environment where the device operating system and applications run. The first mobile phones with hardware-based TEEs appeared almost a decade ago, and today almost every smartphone and tablet contains a TEE like ARM TrustZone. Despite such a large-scale deployment, the use of TEE functionality has been limited for developers. With emerging standardization this situation is about to change. In this tutorial, we explain the security features provided by mobile TEEs and describe On-board Credentials (ObC) system that enables third-party TEE development. We discuss ongoing TEE standardization activities, including the recent Global Platform standards and the Trusted Platform Module (TPM) 2.0 specification, and identify open problems for the near future of mobile hardware security. Slides to be presented at the tutorial can be found [here](#tee).
Why do most mobile devices today have TEEs?

A LOOK BACK
Platform security for mobile devices

Mobile network operators
1. Subsidy locks $\rightarrow$ immutable ID
2. Copy protection $\rightarrow$ device authentication, app separation
3. ...

Regulators
1. RF type approval $\rightarrow$ secure storage
2. Theft deterrence $\rightarrow$ immutable ID
3. ...

End users
1. Reliability $\rightarrow$ app separation
2. Theft deterrence $\rightarrow$ immutable ID
3. Privacy $\rightarrow$ app separation
4. ...

Closed $\rightarrow$ open
Different expectation compared to PCs
Early adoption of platform security

Both IMSI and IMEI require physical protection.

Physical protection means that manufacturers shall take necessary and sufficient measures to ensure the programming and mechanical security of the IMEI. The manufacturer shall also ensure that the IMSI (where applicable) remains secure.

GSM 02.09, 1993

The IMSI is stored securely within the SIM.
The IMEI shall not be changed after the ME's final production process. It shall resist tampering, i.e. manipulation and change, by any means (e.g. physical, electrical and software).

NOTE: This requirement is valid for new GSM Phase 2 and Release 96, 97, 98 and 99 MEs type approved after 1st June 2002.

3GPP TS 42.009, 2001

Different starting points compared to PCs:
Widespread use of hardware and software platform security

~2001

~2002

~2005

~2008

J2ME

TrustZone

Security Foundation by ARM

Symbian OS Platform Security

Android
Historical perspective

1970
Cambridge CAP
Reference monitor
Simple smart cards
Protection rings

1980
VAX/VMS

1990
Java security architecture
Late launch

1990
Trust Platform Module (TPM)

2000
Mobile hardware security architectures
Mobile OS security architectures
Mobile Trusted Module (MTM)

2010
GP TEE standards
TPM 2.0

First part
Second part

Hardware-assisted secure boot
Java Card platform

TPM Mobile

On-board Credentials

Mobile hardware security architectures

Computer security
Mobile security
Smart card security
What constitutes a TEE?

MOBILE HARDWARE SECURITY
1. Platform integrity
2. Secure storage
3. Isolated execution
4. Device identification
5. Device authentication
Secure boot vs. authenticated boot

Secure boot

Authenticated boot
Platform integrity

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Boot sequence

Boot code certificate

Boot code hash

Device key

Base identity

Non-volatile memory

TEE management

Trusted Application (TA)

Secure storage and isolated execution

Device identification

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Platform integrity

Launch boot code
Secure storage

Mobile device hardware TCB

- Verification root
- Cryptographic mechanisms
  - Volatile memory
  - Boot sequence
  - Trusted Application (TA)
  - TEE management
  - Device key
  - Non-volatile memory

Legend
- Trust anchor (Hardware)
- Trust anchor (Code)
- TEE code
- External certificate

Platform integrity

Secure storage

Device identification

Base identity

External certificate

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity

Legend

Trust anchor (Code)

TEE code

External certificate

Device key

Non-volatile memory

Legend

Trust anchor (Hardware)

Base identity
Isolated execution

Legend

- Trust anchor (Hardware)
- Trust anchor (Code)
- TEE code
- External certificate

Mobile device hardware TCB

- TA code certificate
- TA code hash
- Verification root
- Cryptographic mechanisms
- Volatile memory
- Device key
- Non-volatile memory
- Trusted Application (TA)
- TEE management
- Boot sequence

Platform integrity

Secure storage and isolated execution

Device identification

TEE Entry from Rich Execution Environment

Trust anchor
(Code)
Device authentication (and remote attestation)

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Device key

Boot sequence

Trusted Application (TA)

Non-volatile memory

TEE management

Secure storage and isolated execution

Platform integrity

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

External trust root

Device certificate

Identity

Device public key

Device authentication
Hardware security mechanisms (recap)

1. Platform integrity
   - Secure boot
   - Authenticated boot

2. Secure storage
3. Isolated execution
   - Trusted Execution Environment (TEE)

4. Device identification
5. Device authentication
   - Remote attestation

Legend

- Trust anchor (Hardware)
- Trust anchor (Code)
- TEE code
- External certificate

Mobile device hardware TCB

- Verification root
- Cryptographic mechanisms

- Volatile memory
- Device key

- Non-volatile memory
- Base identity

Platform integrity

- TEE mgmt layer
- Secure storage and isolated execution

- Device identification

Launch boot code

TEE Entry from Rich Execution Environment
Device hardware and firmware with TEE support

Architectures with single TEE
- ARM TrustZone
- TI M-Shield
- Smart card
- Crypto co-processor
- TPM

Architectures with multiple TEEs
- Intel SGX
- TPM (and “Late Launch”)
- Hypervisor

Figure adapted from: Global Platform. TEE system architecture. 2011.
TEE hardware realization alternatives

**TEE component**

![Diagram of TEE hardware realization alternatives]

- External Security Co-processor
- Off-chip memory
- External Peripherals
- RAM
- ROM
- Processor core(s)
- Internal peripherals
- OTP Fields
- External Secure Element (TPM, smart card)
- Embedded Secure Element (smart card)
- Processor Secure Environment (TrustZone, M-Shield)

*Figure adapted from: Global Platform. TEE system architecture. 2011.*
ARM TrustZone architecture

System on chip (SoC)

- On-chip memory
- Boot ROM
- Main CPU
- Access control hardware
- Memory controller
- Off-chip/main memory (DDR)
- Peripherals (touchscreen, USB, NFC…)
- Modem

SoC internal bus (carries status flag)

Secure World and Normal World

TrustZone hardware architecture

TrustZone system architecture

- Normal world
  - App
  - App
  - Mobile OS
- Secure world
  - Trusted app
  - Trusted app
  - Trusted OS
  - TEE entry

Device hardware
TrustZone overview

Normal World (NW)  Secure World (SW)

User mode

SCR.NS=1

User

SCR.NS=0

User

Privileged mode

SCR.NS := 1

Supervisor

Supervisor

Secure Monitor call (SMC)

Boot sequence

Address space controllers

TZ-aware MMU

On-chip ROM

On-chip RAM

Main memory (DDR)

physical address range

On-chip ROM

On-chip RAM

Main memory (DDR)
1. Boot begins in Secure World Supervisor mode (set access control)

2. Copy code and keys from on-chip ROM to on-chip RAM

3. Configure address controller (protect on-chip memory)

4. Prepare for Normal World boot
TrustZone example (2/2)

5. Jump to Normal World Supervisor for traditional boot

Secure World Supervisor → Normal World Supervisor

An ordinary boot follows: Set up MMU, load OS, drivers...

6. Set up trusted application execution

Normal World User → Normal World Supervisor

7. Execute trusted application

Normal World Supervisor → Secure World Monitor

On-chip ROM

On-chip RAM

Main memory (DDR)

SW NA
NW NA

SW RW
NW NA

SW RW
NW RW

SMC, NS→0

coupled trusted app and parameters
Mobile TEE deployment

• TrustZone support available in majority of current smartphones

• Mainly used for manufacturer internal purposes
  – DRM, Subsidy lock...

• *Third-party APIs emerging*...
Mobile hardware security APIs

APPLICATION DEVELOPMENT
Mobile hardware security APIs

1. Standardized key stores:
   - JSR 177
   - PKCS #11

2. Proprietary hardware key stores:
   - iOS Key Store
   - Android Key Store

3. Programmable TEE “credential platforms”:
   - On-board Credentials
   - Trustonic TEE API
Android Key Store API

Android Key Store example

// create RSA key pair
Context ctx;
KeyPairGeneratorSpec spec = new KeyPairGeneratorSpec.Builder(ctx);
spec.setAlias("key1")
…
spec.build();

KeyPairGenerator gen = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
gen.initialize(spec);
KeyPair kp = gen.generateKeyPair();

// use private key for signing
AndroidRsaEngine rsa = new AndroidRsaEngine("key1", true);
PSSSigner signer = new PSSSigner(rsa, …);
signer.init(true, …);
signer.update(signedData, 0, signedData.length);
byte[] signature = signer.generateSignature();

Android Key Store implementation

Selected devices
- Android 4.3
- Nexus 4, Nexus 7

Keystore operations
- GENERATE_KEYPAIR
- IMPORT_KEYPAIR
- SIGN_DATA
- VERIFY_DATA

Persistent storage on Normal World

Android Key Store

• Available operations
  – Signatures
  – Encryption/decryption

• Developers cannot utilize programmability of mobile TEEs
  – Not possible to run arbitrary trusted applications

• Different API abstraction and architecture needed...
On-board Credentials goal

An open credential platform that enables existing mobile TEEs

Secure yet inexpensive

Design constraints:
- Open provisioning model
- Limited secure (on-chip) secure memory
- No access control architecture within TEE
On-board Credentials (ObC) architecture

- Mobile device
  - Rich execution environment (REE)
    - ObC API: Provisioning, execution, sealing
    - ObC scheduler
      - Trusted app persistent store
      - Trusted app dynamic state
  - Mobile OS
- Trusted execution environment (TEE)
  - ObC Interpreter
    - I/O data
    - Interpreted code
    - Interpreter state
    - Loaded trusted app
- Mobile device hardware with TEE support
  - Driver

Centralized provisioning vs. open provisioning

Centralized provisioning
(smart card, Trustonic)

Open provisioning
(On-board Credentials)
Open provisioning model

1. Certified device key + user authentication
   PK

2. Provision new family
   Enc(PK, FK)

3. Provision new secrets
   AuthEnc(FK, secret)

4. Provision trusted applications
   AuthEnc(FK, hash(app)) + app

Principle of same-origin policy

On-board Credentials development

- Trusted application development
  - BASIC like scripting language
  - Common crypto primitives available (RSA, AES, SHA)

- REE application counterpart
  - Standard smartphone app (Windows Phone)
  - ObC API: provisioning, trusted application execution

ObC counterpart application pseudo code

```c
// install provisioned credential
secret = obc.InstallSecret(provSecret)
app = obc.InstallCode(provApplication)
credential = obc.CreateCredential(secret, app, authData)

// run installed credential
output = obc.RunCredential(credential, input)
```

ObC trusted application extract

```c
rem --- Quote operation
if mode == MODE_QUOTE
read_array(IO_SEALED_RW, 2, pcr_10)
read_array(IO_PLAIN_RW, 3, ext_nonce)

rem --- Create TPM_PCR_COMPOSITE
pcr_composite[0] = 0x0002 rem --- sizeOfSelect=2
pcr_composite[1] = 0x0004 rem --- PCR 10 selected (00 04)
pcr_composite[2] = 0x0000 rem --- PCR selection size 20
pcr_composite[3] = 0x0014
append_array(pcr_composite, pcr_10)
sha1(composite_hash, pcr_composite)

rem --- Create TPM_QUOTE_INFO
quote_info[0] = 0x0101 rem --- version (major/minor)
quote_info[1] = 0x0000 rem --- (revMajor/Minor)
quote_info[2] = 0x5155 rem --- fixed (`Q' and `U')
quote_info[3] = 0x4F54 rem --- fixed (`O' and `T')
append_array(quote_info, composite_hash)
append_array(quote_info, ext_nonce)
write_array(IO_PLAIN_RW, 1, pcr_composite)

rem --- Hash QUOTE_INFO for MirrorLink PA signing
sha1(quote_hash, quote_info)
write_array(IO_PLAIN_RW, 2, quote_hash)
```
Example application: MirrorLink attestation

- MirrorLink system enables smartphone services in automotive context
- Car head-unit needs to enforce driver distraction regulations
- Attestation protocol
  - Defined using TPM structures (part of MirrorLink standard)
  - Implemented as On-board Credentials trusted application (deployed to Nokia devices)

[Diagram showing attestation process]


[Image of MirrorLink system]

http://www.mirrorlink.com
Example application: Public transport ticketing

- Mobile ticketing with NFC and TEE
- 110 traveler trial in New York (summer 2012)
  - Implemented as On-board Credentials trusted application
  - Deployed to Nokia devices

Transaction evidence
(authenticated counter as ObC app)
Application development summary

• Previously mainly internal purposes
  – DRM, subsidy lock

• Third-party APIs have started to emerge
  – Android KeyStore (TrustZone)
  – Trustonic security API

• Research for open TEEs
  – On-board Credentials with open provisioning

• Standardization would help developers...
See you in 10 minutes...

BREAK
**Trustonic <t-base TEE**

- **L4**: minimized kernel: IPC, scheduling, MMU
- **Run-Time Manager**: Installation, I/O.
- **Crypto driver**: key access, crypto, RNG, secure storage
- Smart-card like provisioning and life-cycle model for TAs
- Global Platform compatibility
<t-base TA invocation

Rich world application

- mcOpenSession (void *, int len, ..)
- opt. mapping
- TCI buffer

Secure World

- MMU (Sec world)
- TCI buffer
- stack, code, bss
- Run-Time Manager
- Kernel VMM mgr
- Phys. memory
- MMU (Rich world)

User space

Privileged mode

VM address

void *secVirt = mcMap (void *, int len)

void tlMain(
  addr tciBuffer,
  int tciBufferLen)

1MB

opt. mapping

1MB

1MB

1MB
1. Open connection to TEE

2. Open session
   - provide TA
   - Opt: provide shared mem.

3. Communicate

4. Terminate session and connection

```c
static TEEC_Result Run (TEEC_Session *session, unsigned char *pData)
{
    TEEC_Result nError;
    TEEC_Operation sOperation;

    memset(&sOperation, 0, sizeof(TEEC_Operation));
    sOperation.paramTypes = TEEC_PARAM_TYPES(
        TEEC_MEMREF_TEMP_INOUT, TEEC_NONE,
        TEEC_NONE, TEEC_NONE);
    sOperation.params[0].tmpref.buffer = pData;
    sOperation.params[0].tmpref.size = 512;

    nError = TEEC_InvokeCommand(session, CMD_GENKEY, &sOperation, NULL);

    return nError;
}
```

#define CMD_GENKEY 1
1. Provide handlers for
   - instantiation / unload
   - session open / close

2. Provide code for
   - function that is called

```c
TA_InvokeCommandEntryPoint(void* pSessionContext,
                           uint32_t nCommandID,
                           uint32_t nParamTypes, TEE_Param pParams[4])
{
 ...
 switch(nCommandID)
 {
   case CMD_GENKEY:
     if (nParamTypes != CMD_GENKEY_PTYPES) {...}
     pInput = pParams[0].memref.buffer;
     size = (uint32_t)pParams[0].memref.size;
     if (TEE_CheckMemoryAccessRights( ...) { ... } )
     TEE_AllocateTransientObject(TEE_TYPE_RSA_KEYPAIR,
                                   maxObjectSize, &keyObj))
     TEE_GenerateKey(keyObj, 2048, NULL, 0);
     TEE_GetObjectBufferAttribute(keyObj,
                                  TEE_ATTR_RSA_MODULUS, ...);
     TEE_FreeTransientObject(keyObj);
     return TEE_SUCCESS;
 ...
```

#define CMD_GETKEY 1
• Run a dev-board so that we can see the activity
Application development summary

• Previously mainly internal purposes
  – DRM, subsidy lock

• Third-party APIs have started to emerge
  – Android KeyStore (TrustZone)
  – Trustonic <tbase

• Research for open TEEs
  – On-board Credentials with open provisioning

• Standardization would help developers...
Break
Outline

• A look back (10 min)
  – Why mobile devices have TEEs?
• Mobile hardware security (30 min)
  – What constitutes a TEE?
• Application development (30 min)
  – Mobile hardware security APIs + DEMO

Break (10 min)

• Current standardization (60 min)
  – NIST, Global Platform, TPM 2.0
• A look ahead (10 min)
  – Challenges and summary

NIST guidelines, Global Platform, Trusted Computing Group, Jedece

STANDARDIZATION
TEE-related standards and specifications

- First versions of standards already out
- Needed for compliance/interoperability
- Enables app developers to leverage TEEs
EFI SECURE BOOT
**UEFI – boot principle**

- **Firmware init**
  - EFI applications
  - EFI drivers
  - EFI OS loaders

- **OS**

**Things that e.g. sets up the device (like TZ)**

**Driver firmware setup**

**Boot loaders**

**Replacement for BIOS**

*Secure Boot is an optional feature*

---

Unified Extensible Firmware Interface Specification

Nyström et al: UEFI Networking and Pre-OS security (2011)
Key management for updates

Platform Firmware
Key Storage
→ tamper-resistant
→ updates governed by platform key

Key Exchange Keys

Platform Key (Pub/Priv)
UEFI – secure boot

Signature Database (s)
- tamper-resistant
- (rollback prevention)
- updates governed by keys

Key management for updates

Platform Firmware Key Storage
- tamper-resistant
- updates governed by platform key

Key Exchange Keys

Platform Key (Pub/Priv)

Keys allowed to update

(ref: UEFI spec)
UEFI – secure boot

Key management for update

Platform Firmware
Key Storage
→ tamper-resistant
→ updates governed by platform key

Key Exchange Keys

Platform Key (Pub/Priv)

Signature Database (s)
→ tamper-resistant
→ (rollback prevention)
→ updates governed by keys

(ref: UEFI spec)

Image Information Table
→ hash
→ name, path
→ Initialized / rejected

White list + Black list for database images
ROOTS OF TRUST (HARDWARE ANCHORS)
Guidelines on Hardware-Rooted Security in Mobile Devices (SP800-164, draft)

Required security components are

a) **Roots of Trust (RoT)**

b) an **application programming interface (API)** to expose the RoT to the platform

c) a **Policy Enforcement Engine (PEnE)**”

“RoTs are preferably implemented in hardware”
Secure Capabilities built from Roots-of-Trust

Security Capabilities

Roots of Trust

RoT for Reporting
RoT for Integrity
RoT for Storage
RoT for Verification
RoT for Measurement

Picture: Andrew Regenshield: NIST/Computer Security Division
ARM TrustZone + Secure Boot + Secrets = RoT?

1. Secure boot → Root of Trust for Verification
2. Measuring in secure boot → Root of Trust for Measurement
3. Device key + code in TZ TEE → Root of Trust for Reporting
4. TEE secure memory → Root of Trust for Integrity
5. Device key + TEE → Most of Root of Trust for Storage. No easy rollback protection.

Trusted Execution Environment (TEE)

- Storage
- Isolation
- Integrity
Specifications: www.globalplatform.org

GLOBALPLATFORM™
Most of the smart-card based ecosystems around authentication, payment and ticketing make use of Global Platform standards:
- For card interaction and provisioning protocols
- For reader terminal architecture and certification

The Global Platform Device Committee specifies architecture and interfaces for a trusted operating system in a TEE

References:
- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal API Specification v1.0
- Trusted User Interface API v 1.0
Global Platform Device Architecture

- API to communicate with the TEE
- System interface library (libc ..) for Trusted Applications with RPC, crypto and necessary I/O functions

Eventually, these APIs may become the reference model for writing code for and interacting with a TEE. Missing pieces still include provisioning and compliance aspects.
Interaction with a TEE (GP) -- caller

(adapted from example in TEE Client API specification)

result = TEEC_InitializeContext( NULL, &context);
result = TEEC_OpenSession(&context, &session, &cryptoTEEApp, TEEC_LOGIN_USER,
                          NULL, NULL, NULL);
commsSM.size = 20; commsSM.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
result = TEEC_AllocateSharedMemory(&context, &commsSM);

// omitted: registration of additional shared memory for in-place encryption of data
operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_MEMREF_PARTIAL_INPUT,
                                         TEEC_NONE, TEEC_NONE);
ivPtr = (uint8_t*)commsSM.buffer; memset(ivPtr, 0, 16); // Set input (IV)
operation.params[0].value.a = 1;                          // Set input (key handle=1)
operation.params[1].memref.parent = &commsSM;
operation.params[1].memref.offset = 0;
operation.params[1].memref.size = 20;

result = TEEC_InvokeCommand(&session, CMD_ENCRYPT_INIT, &operation, NULL);

Parameters:

- **CMD**: Val:1
- **Ref**: N/A
- **N/A**: N/A
Interaction with a TEE (GP) -- callee

*Mandatory handler functions:*

```
TA_CreateEntryPoint(void); / TA_DestroyEntryPoint(void);

TA_OpenSessionEntryPoint(uint32_t param_types, TEE_Param params[4], void **session)

TA_CloseSessionEntryPoint(..)

TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
                        uint32_t param_types, TEE_Param params[4])

{
    switch(cmd)
    {
    case CMD_ENCRYPT_INIT:
        ....
    }
}
```

*Parameters:*

- **CMD**
- **Val:** 1
- **Ref**: N/A
- **N/A**: N/A

Constructor / Destructor

May point to any memory chosen by TA
Interaction with a TEE (GP)

TA pointer to shared memory in the callers’ context.

Efficient mechanism for in-place encryption / decryption etc.

The TA **programmer must be aware of differences in memory references.**

Secure storage: Memory / objects in a TA can be persistently stored

TEE_CreatePersistentObject(TEE_STORAGE_PRIVATE, objID, objIDLen, flags, attributes, .., handle)

bytes read
handle

TEE_ReadObjectData(handle, buffer, size, count);
TEE_WriteObjectData(handle, buffer, size);
TEE_SeekObjectData(handle, offset, ref);
TEE_TruncateObjectData(handle, size);

RPC: Communication with other TAs

TEE_OpenTASession(TEE_UUID* destination, ..., paramTypes, params[4], &session);
TEE_InvokeTACCommand(session, ..., commandId, paramTypes, params[4]);

(The invocation calls the same interface as the one used for external calls)
Trusted path to user (GP)

• Trustworthy user interaction needed
  – Provisioning
  – User authentication
  – Transaction confirmation

• Trusted User Interface API 1.0:
  – Set up widget structures
  – Call TEE_TUIDisplayScreen
  – Collect results

• Only for I/O directly wired to to the trusted OS
GP User-Centric provisioning model

**User-centric provisioning white paper**

**Trad:**
issuer / service provider → manufacturer → user

**New:**
token provider → user → service manager → service provider

GP device committee is working on a TEE provisioning specification
Specifications: www.jedec.org

JEDEC ™
JEDEC RPMB in e·MMC v4.41 and v4.5

Jedec is primarily known for standards like DDR, MMC, UFS, but is important esp. in microelectronics.

RPMB: Replay-Protected Memory Block
- Separate partition in the MMC
- Authenticated channel

- Memory write/reads protected with HMAC-SHA256
  - Random values for freshness
  - Counter binding for replay protection (write)
Specifications: www.trustedcomputinggroup.org

TRUSTED COMPUTING GROUP
TPM / TPM2 / TPM MOBILE
TCG Trusted Platform Module (TPM)

- an application interface to secure services

- deployed to hundreds of millions of PCs and laptop (v1.2. chip + drivers)

- potential way applications and OS services interact with platform security
TPM

- Component that collects state and is separate from system on which it reports

- **Relies on Roots of Trust**

- For remote parties
  - Remote attestation in well-defined manner
  - Authorization for functionality provided by the TPM

- Locally
  - **Key generation** and **key use** with TPM-resident keys
  - Secure **binding** with encryption, as well as **non-volatile storage**
  - An **engine** for encryption / decryption and signing, also for hash algorithms and symmetric ciphers
A TPM is NOT

• An enforcing component or mechanism for services outside the TPM

• An eavesdropping channel for remote monitoring

HOWEVER

Secure Boot + (GP TEE OR TPM)

can potentially be used to violate privacy
alternatively, it can be used to protect user privacy
Platform Configuration Register (PCR)

... Measurement aggregation for eventual binding or attestation

... A given expected PCR value can ONLY be reached by a correct extension sequence

... In an aggregate with a trustworthy root, any divergence in reported events causes an irrevocable change in the eventual PCR value.

Authenticated boot

measure \texttt{m1} 
send \texttt{m1} to TPM
launch code 1

measure \texttt{m2} 
send \texttt{m2} to TPM
launch code 2

measure \texttt{m3} 
send \texttt{m3} to TPM
launch code 3

Remote Attestation:
\texttt{SIG}(\texttt{chall}, \texttt{PCR value})

\[ H = H(new \mid H-old) \rightarrow \]
\[ H = H(m3 \mid H(m2 \mid H(m1))) \]
\[ H(0) = 0 \]
TPM Mobile (Mobile Trusted Module)

A TPM profile for Mobile devices (v 1.2. & v.2) that adds mechanisms for

**Adaptation to TEEs:**
New RoT definitions and requirements for TEE adaptation

**Multi-Stakeholder Model (MSM):**
Rich Application – Trusted Application – TPM relation
Measurements, lifecycle models
Relations between different ”types” of TPM mobiles

”Certified boot”:
Secure boot with TCG authorizations
(RIM Certificates → TPM2 authorization)
TPM Mobile on GP TEE

- Do GP TEEs provide needed functionality?
- Do GP TEEs provide needed security assurance?
TPM Mobile Multi-Stakeholder Model (MSM)

A TEE can host a number of “simultaneous” TPMs
One TPM (platform) is needed for OS services – say secure boot

Most applications do not need dedicated code (a TA) in the TEE. But they may need secure storage, state-aware keys, and attestation for those

“Platform” TPM
Application specific TPMs
TAs
Secure Storage
Crypto
I/O
RPC

“Rich Execution Environment” OS
TPM TSS
Normal application

Isolation boundary

TEE driver

Trusted Operating System
TPM authorization

- Many users of varying security levels
- System state awareness is a fundamental to TPMs – sets TPMs apart from e.g. removable smartcards.
- To implement any TPM service that enforces control, authorization is essential
Authorization (policy) TPM 1

MTM added key authorization, but only for PCRs
Authorization (policy) TPM2

System

TPM2

Commands to include some part of TPM2 (system) state in policy validation

Object invocation

Object authorization

- System state info
- Other TPM objs
- Object (e.g. key)
- Reference value: authVal
- External auth
- Session

Commands to include some part of TPM2 (system) state in policy validation
TPM2 Policy Session

- different types of preconditions can be part of an authorization policy (session)

- In addition, logical relations should be applicable on the set of atomic preconditions that constitutes the policy (AND, OR)

- A policy session accumulates all policy information needed to make the authorization decision.
TPM2 Policy Session Contents

- An accumulated session policy value called `policyDigest`

\[ \text{newDigestValue} := H(\text{oldDigestValue} \ || \ \text{policyCommand} \ || \ \text{stateinfo}) \]

- Some policy commands reset the value

IF condition THEN
\[ \text{newDigestValue} := H(0 \ || \ \text{policyCommand} \ || \ \text{stateinfo}) \]

- Session also contains optional assertions to be made at object access.

Deferred checks:
- PCRs changed
- Applied command
- Command locality
TPM2 Policy Command Examples

\[\text{Tpm2} \text{PolicyPCR}: \text{Include a set of PCR values in the authorization}\]

\[\text{sessionUpdate.state_info := \{pcr value, pcr index\}}\]

\[\text{Tpm2} \text{PolicyNV}: \text{Include a reference value and operation index in case a comparison (<, >, eq) of a non-volatile memory area with the reference succeeds.}\]

\[\text{e.g., if counter5 > 2 then}\]
\[\text{sessionUpdate.state_info := [ref, op, mem.area]}\]
TPM2 Deferred Policy Examples

- **TPM2_PolicyCommandCode**: Include the command code specification in session:
  
  ```
  sessionUpdate.state_info := command code
  deferred : policySession->commandCode := command code
  ```

- **TPM2_PolicyLocality**: Restrict the operation to a given locality:
  
  ```
  sessionUpdate.state_info := locality
  deferred : policySession->commandLocality := locality
  ```
TPM2 PolicyOR:

**TPM2_PolicyOR**: Authorize one of several options:

**Input**: List of digest values <D1, D2, D3, .. >

**IF** policySession->policyDigest **in** List **THEN**

newDigestValue := H(0 || policyCommand || List)

**Reasoning**: H(List) is known (fixed) policy. For a wrong digest Dx (not in set <D1 D2 D3>) it is difficult to find another List2 = <Dx Dy, Dz, .. > where H(List) == H(List2)
“TPM2 PolicyAND”

There is no explicit AND command

AND is achieved by two consecutive policy commands → order dependence

Theoretical example: Use an OR to hide the order dependence of an AND
External Authorization

**TPM2_PolicyAuthorize:** Validate a signature on a policyDigest:

**IF** signature validates

**AND** `policySession->policyDigest` in signed content

**THEN**

`newDigestValue := H(0 || policyCommand || pub || ..)`
Simple secure boot is not always enough

Secure boot can have the following properties

A) Extend all the way into OS / application booting

B) Can include platform-dependent policy

C) Can include optional / complementary boot branches

D) Order in which components are booted may matter

TPM2 authorizations can be used for secure boot: Example follows
Secure boot “constructed example”

1. UEFI started the boot process
2. A UEFI program loads the TEE, TPM etc (PCR 1)
3. A UEFI OS loader loads the OS (PCR 2)
4. The OS boots
5. We want to (dynamically) **load the driver** that communicates with some aspect of the TEE --- the TPM must of course be accessible
Example policy

OS driver for TEE will be measured and launched

We need something to authorize..

Assumptions

Driver supplier can change policy later

Policy applies only to PCR update

IF

AND

Platform A kernel

Ext.sign.

Platform B kernel

OR

measurement → PCR 5

measurement → PCR 2

measurement → PCR 1

Secure side loaded

Secure side loaded

UEFI drivers M completed successfully

UEFI program N completed successfully

Assumptions

Measurement → PCR 3

Rollback protection ..

CTR5 > 2

Driver supplier can change policy later

Policy applies only to PCR update
Example policy

We 'own' PCR 5 authorization. Let's add authValue X (non-modifiable)

What is a good value for X?
Example policy

If X is H(pubA) we can authorize any value Y as policy for PCR 5

\[
Y \overset{\text{PolicyAuthorize}}{\rightarrow} Y \overset{\text{H(pubA)}=X}{\rightarrow} X
\]

[ actually \( H(0 \ || \ \text{PolicyAuthorize} \ || \ \text{pubA} \ || \ ..) \) ]
Example policy

Assumptions
- Driver supplier can change policy later
- Policy applies only to PCR updates

IF

AND

Platform A kernel
- Ext. sign.
- Measurement → PCR 2
- Secure side loaded
- UEFI program N completed successfully
- Measurement → PCR 1

Platform B kernel
- Measurement → PCR 2
- Secure side loaded
- UEFI drivers M completed successfully
- Measurement → PCR 3

OR

AND

OS driver for TEE will be measured and launched
- Measurement → PCR 5

Assumptions

Y → PolicyAuthorize(Sig_A(Y)) → X
Example policy

If we want to make sure PCREx tend is used and not e.g. PCRReset:

- **TPM2_PolicyCommandCode**
- **TPM2_PolicyCPhash**

Assumptions

- Driver supplier can change policy later
- Policy applies only to PCR updates

Assumptions

- **CTR5 > 2**

Assumptions

- Secure side loaded
- UEFI drivers M completed successfully
- Measurement → PCR 3

Assumptions

- Secure side loaded
- UEFI program N completed successfully
- Measurement → PCR 1

Assumptions

- **IF**
- **AND**

Platform B kernel

- Measurement → PCR 2

Example policy

- OS driver for TEE will be measured and launched
- Measurement → PCR 2

Example policy

- IF
- AND

Y → PolicyAuthorize(Sigₐ (Y)) → X

Example policy
Example policy

OS driver for TEE will be measured and launched

**Assumptions**
- Driver supplier can change policy later
- Policy applies only to PCR updates

**Platform A kernel**
- Measurement $\rightarrow$ PCR 2

**Platform B kernel**
- Measurement $\rightarrow$ PCR 2

**OR**

Secure side loaded
- UEFI program N completed successfully

Secure side loaded
- UEFI drivers M completed successfully

**Assumption**
- CTR5 $>$ 2

**Assumptions**
- Rollback protection

**Z** $\rightarrow$ PolicyCommandCode(TPM_PCRExtend) $\rightarrow$ Y $\rightarrow$ PolicyAuthorize(Sig$_A$(Y)) $\rightarrow$ X

{**Check:** Eventual command == TPM_PCRExtend}
Example policy

**Assumptions**

- Driver supplier can change policy later
- Policy applies only to PCR updates

**To bind a PCR value:**

\[
\text{TPM2\_PolicyPCR (index(1), value(expected meas.))}
\]

(actually an aggregate PCR hash)

- Secure side loaded
- UEFI program N completed successfully
- Measurement \(\rightarrow\) PCR 1

- Measurement \(\rightarrow\) PCR 2

- Measurement \(\rightarrow\) PCR 5

- Ext.sign.
- Platform Advanced
- UEFI drivers M completed successfully

**Z \(\rightarrow\) PolicyCommandCode(TPM\_PCRExtend) \(\rightarrow\) Y \(\rightarrow\) PolicyAuthorize(Sig_A(Y)) \(\rightarrow\) X**

{Check: Eventual command == TPM\_PCRExtend}
Example policy

OS driver for TEE will be measured and launched

**Example policy**

- **Assumptions**
  - Driver supplier can change policy later
  - Policy applies only to PCR updates

- **Platforms**
  - **Platform A kernel**
    - Measurement → PCR 2
  - **Platform B kernel**
    - Measurement → PCR 2

- **Secure side loaded**
  - UEFI program N completed successfully
    - Measurement → PCR 1

- **OR**

- **Rollback protection**
  - CTR5 > 2

- **W → PolicyPCR(1, meas.) → Z**
  - Z → PolicyCommandCode(TPM_PCRExtend) → Y → PolicyAuthorize(SigA(Y)) → X
    - **Check:** Eventual command == TPM_PCRExtend
Example policy

We want to support two OS variants based on a PCR2 value:

\[\text{TPM2\_PolicyOR \{V1, V2\}}\]

**OS driver for TEE will be measured and launched**

**Assumptions**
- Driver supplier can change policy later

**Platform A kernel**
- Measurement → PCR 2
- Rollback protection
- Secure side loaded
- UEFI program N completed successfully

**Platform B kernel**
- Measurement → PCR 2
- Secure side loaded
- UEFI drivers M completed successfully

\[\text{CTR5} > 2\]

\[\text{Y} \rightarrow \text{PolicyAuthorize(Sig}_A(Y)) \rightarrow X\]

\{Check: Eventual command == TPM\_PCRExtend\}

**Measurement → PCR 1**

\[\text{W} \rightarrow \text{PolicyPCR(1, meas.)} \rightarrow Z\]

\[\text{Z} \rightarrow \text{PolicyCommandCode(TPM\_PCRExtend)} \rightarrow Y\]
Example policy

OS driver for TEE will be measured and launched

Platform A kernel

Platform B kernel

Ext. sign.

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

AND

Rollback protection...

Secure side loaded

UEFI program N completed successfully

Measurement → PCR 1

Secure side loaded

UEFI drivers M completed successfully

Measurement → PCR 3

V1 → PolicyOr({V1,V2} → W → PolicyPCR(1, meas.) → Z

V2 → Z → PolicyCommandCode(TPM_PCRExtend) → Y → PolicyAuthorize(Sig_A(Y)) → X

{Check: Eventual command == TPM_PCRExtend}
Example policy

Provider of OSB may do certified or authenticated boot. Thus:

Possibly there are many more authorizations needed (like a **PolicyNV**) or

The OS provider updates PCR2 with result of some **PolicyAuthorize**(Sig$_{OSB}(...)) and guarantees its own freshness

---

OS driver for TEE will be measured and launched

**OR**

**AND**

Platform A kernel

**AND**

Platform B kernel

**AND**

UEFI drivers M completed successfully

Secure side loaded

UEFI program N completed successfully

Secure side loaded

**Rollback protection ..**

**V1** → PolicyOr({V1,V2} → **W** → PolicyPCR(1, meas.) → **Z**

**Z** → PolicyCommandCode(TPM_PCRExtend) → **Y** → PolicyAuthorize(Sig$_A$(Y)) → **X**

**Check:** Eventual command == TPM_PCRExtend
Example policy

**Assumptions**
- Driver supplier can change policy later
- Policy applies only to PCR updates

**Platform A kernel**
- Measurement → PCR 2
  - AND
    - Ext. sign.
    - Policy PCR(2, H(...)) → V1 → PolicyOr({V1, V2} → W → Policy PCR(1, meas.) → Z
      - Z → Policy Command Code (TPM_PCRExtend) → Y → Policy Authorize (Sig_A(Y)) → X
      - {Check: Eventual command == TPM_PCRExtend}

**Platform B kernel**
- Measurement → PCR 2
  - AND
    - Ext. sign.
    - Policy PCR(2, H(...)) → V2 → PolicyOr({V1, V2} → W → Policy PCR(1, meas.) → Z
      - Z → Policy Command Code (TPM_PCRExtend) → Y → Policy Authorize (Sig_A(Y)) → X
      - {Check: Eventual command == TPM_PCRExtend}

**OS driver for TEE**
- Will be measured and launched

**IF**
- AND
- CTR5 > 2
- Secure side loaded
  - UEFI program N completed successfully
  - Measurement → PCR 1

**OR**
- Secure side loaded
  - UEFI drivers M completed successfully
  - Measurement → PCR 3
Recap: Example “boot sequence”

.setDefault(UEFI starts TEE and lauches OS → PCR1 updated)

setDefault(Operating System boots up)

setDefault(TPM PolicyAuthorize → OS manufacturer PCR2 updated)

 nues(THM_PolicyPCR (PCR 2 ”Sign of OS provider”), → OS OK)

 nues(THM_PolicyOR → One of two OSs values accepted)

 nues(THM_PolicyPCR (PCR 1, ”H(TEE meas.)”) → TEE version correct)

 nues(THM_PolicyCommand (PCRExtract) → Only authorize a PCRExtract command)

 nues(THM PolicyAuthorize →”I” authorize the collected state)

 nues(THM_PCRExtract(PCR 5, measurement value)

 {Check: Eventual command == TPM_PCRExtract}
Deployed standards: Nokia Lumia Secure Boot Flow

1. Transitive trust chain begins (Platform RoT is in eFuse)
2. Trusted OS integrity is verified
3. Trusted OS verifies UEFI
4. UEFI loads TPM app into Trusted OS
5. UEFI verifies OS boot manager using certificates from RPMB (Replay Protected Memory Block)
6. UEFI Boot Manager verifies OS kernel
7. OS kernel verifies OS binaries

Source: Nokia, Presented at RSA conf. 2013
A LOOK AHEAD

Challenges ahead and summary
Challenges ahead

• **What is the right TEE architecture?**
  – Processor secure environments vs. Separate secure elements vs ...?

• **Hardware security and privacy**
  – Secure boot and control points, TEE rootkits

• **Provisioning**
  – Does ‘open provisioning’ emerge as viable alternative for centralized model?

• **Trusted user interaction**
  – How to establish a secure channel between TEE and the user?

• **Certification / verification**
  – How to gain confidence in TEE designs?
What is the right TEE architecture?

- Processor security architecture vs. embedded secure element vs. some combination?
- New designs like Intel SGX
- Multiple cores multiple TEEs
- Dealing with peripherals (UI, sensors, NFC, ...)
Hardware security and user privacy?

- Secure boot can be used to limit user choice
- Vulnerabilities in TEE implementation $\rightarrow$ rootkits
What is the right provisioning model?

- **Open provisioning**
  - Easy service deployment
  - But challenging lifecycle management

- **Hybrid model?**

---

How to provide trusted path to the user?

- Trustworthy user interaction needed
  - Provisioning
  - User authentication
  - Transaction confirmation

- Technical implementation possible

- But how does the user know?
  - Secure attention key (ctrl-alt-del)
  - Security indicator
Verification and certification?

• Common Criteria model may not be suitable for TEEs
  – too slow
  – too inflexible (cannot efficiently deal with software upgrades)

• Alternatives may/will emerge
  – UK: CPA
    http://www.cesg.gov.uk/servicecatalogue/CPA/Pages/CPA.aspx
Summary

• Hardware-based TEEs are widely deployed on mobile devices
  – But access to application developers has been limited
  – This is about to change

• TEE functionality and interfaces are being standardized
  – Promise of better third-party developer access
  – GlobalPlatform TEE architecture
  – Trusted Computing Group: TPM 2.0 specification

• Many open issues lie ahead...

• Thank you for any feedback (contact info in author copy)

Forthcoming e-book

“Mobile Platform Security”
(to be published by Morgan-Claypool)

Draft version at publisher stand (lobby)

Publisher offers to give you a free copy!