Mobile Platform Security
Trust Execution Environments

Summer School, 2014
N. Asokan
Aalto University and University of Helsinki

Jointly prepared with:
Jan-Erik Ekberg, Trustonic
Kari Kostiainen, ETH Zurich
What is a TEE?

Execution Environment
What is a TEE?

Trusted Execution Environment

Isolated and integrity-protected

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

Processor, memory, storage, peripherals

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
  – Why do mobile devices have TEEs?

• Mobile hardware security
  – What constitutes a TEE?

• Application development
  – Mobile hardware security APIs

• Current standardization
  – UEFI, NIST, Global Platform

• Current standardization: TCG
  – TPM1.2 and TPM 2.0 extended authorization model

• A look ahead
  – Challenges and summary
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 (where applicable) remains

**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 J2ME

~2002 TrustZone® Security Foundation by ARM®

~2005 Symbian OS Platform Security

~2008 Android
Historical perspective

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology/Innovation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>Cambridge CAP</td>
</tr>
<tr>
<td></td>
<td>VAX/VMS</td>
</tr>
<tr>
<td></td>
<td>Reference monitor</td>
</tr>
<tr>
<td></td>
<td>Simple smart cards</td>
</tr>
<tr>
<td></td>
<td>Protection rings</td>
</tr>
<tr>
<td>1980</td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>Trusted Platform Module (TPM)</td>
</tr>
<tr>
<td></td>
<td>Java security architecture</td>
</tr>
<tr>
<td></td>
<td>ARM TrustZone</td>
</tr>
<tr>
<td></td>
<td>TI M-Shield</td>
</tr>
<tr>
<td></td>
<td>On-board Credentials</td>
</tr>
<tr>
<td></td>
<td>Mobile hardware security architectures</td>
</tr>
<tr>
<td></td>
<td>Hardware-assisted secure boot</td>
</tr>
<tr>
<td></td>
<td>Mobile OS security architectures</td>
</tr>
<tr>
<td></td>
<td>Mobile Card platform</td>
</tr>
<tr>
<td>2000</td>
<td>Late launch</td>
</tr>
<tr>
<td></td>
<td>GP TEE standards</td>
</tr>
<tr>
<td></td>
<td>NIST</td>
</tr>
<tr>
<td></td>
<td>TPM 2.0</td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
</tbody>
</table>

Computer security
Smart card security
Mobile 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**

- OS Kernel checker
- Boot block checker
- Firmware checker

**Authenticated boot**

- OS Kernel measurer
- Boot block measurer
- Firmware measurer

**Why?**

Start only known trusted configurations

**Why?**

Start any configuration, but remember the state
State can be:
- bound to stored secrets (sealing)
- reported to external verifier (remote attestation)
Certified by device manufacturer: \( \text{Sig}_{\text{SKM}}(H(\text{boot code})) \)

Platform integrity

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

**Mobile device hardware TCB**

- Verification root

**Cryptographic mechanisms**

- Volatile memory

**Boot sequence**

- Stores measurements for authenticated boot

**Device key**

**Signature verification algorithm**

**Device manufacturer public key: \( \text{PK}_M \)**

**Platform integrity**

**Launch boot code**

- Boot code certificate
  - Boot code hash

**Device manufacturer public key: \( \text{PK}_M \)**

**Mobile device hardware TCB**

- Verification root

**Cryptographic mechanisms**

- Volatile memory

**Boots sequence**

- Stores measurements for authenticated boot

**Device key**

**Signature verification algorithm**

**Device manufacturer public key: \( \text{PK}_M \)**
Secure storage

Legend

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

External certificate

Mobile device hardware TCB

Platform integrity

Secure storage

Volatile memory

Boot sequence

Trusted Application (TA)

TEE management

Device key $K_D$

Non-volatile memory

Device identification

Encryption algorithm

Base

Protected memory

Rollback protection

Sealed-data = AuthEnc_{K_D}(data | ...)

Encryption algorithm
Isolated execution

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Device key $K_D$

Non-volatile memory

TEE management

Trusted Application (TA)

Secure storage and isolated execution

Device identification

Platform integrity

Boot sequence

Controls TA execution

Certified by device manufacturer

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Legend

TA code certificate

TA code hash
Device identification

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Device key $K_D$

Non-volatile memory

Trusted Application (TA)

TEE management

Secure storage and isolated execution

Platform integrity

Boot sequence

Device identification

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Identity certificate

Base identity

Assigned identity

Multiple assigned identities

One fixed device identity
Device authentication (and 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 $K_D$

- Boot sequence
- Trusted Application (TA)
- Secure storage and isolated execution

Platform integrity

Used to protect/derive signature key

Issued by device manufacturer

External trust root

Device certificate

- Identity
- Device public key $PK_D$
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 $K_D$
  - Non-volatile memory
  - Base identity

Platform integrity
- TEE mgmt layer
- Secure storage and isolated execution

Device identification
- TEE Entry from Rich Execution Environment

Launch boot code

Device authentication
- Device certificate
  - Identity
  - Device pub. key $PK_D$

External trust root
TEE system architecture

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

Legend:
SoC: system-on-chip
OTP: one-time programmable

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
- Modem
- Interrupt controller
- Memory controller
- Off-chip/main memory (DDR)
- Peripherals (touchscreen, USB, NFC...)

SoC internal bus (carries status flag)

Secure World and Normal World

TrustZone system architecture

- Normal world (REE)
  - App
  - Memory controller
  - Mobile OS
  - Boot ROM

- Secure world (TEE)
  - Trusted OS
  - Trusted app
  - Device hardware
  - TEE entry

TrustZone hardware architecture
TrustZone overview

Normal World (NW)

- User
- Supervisor

Secure World (SW)

- User
- Supervisor

Boot sequence

Secure Monitor call (SMC)

SCR.NS := 1

SCR.NS=1

SCR.NS=0

Address space controllers

TZ-aware MMU

On-chip ROM

On-chip RAM

Main memory (DDR)

physical address range

Privileged mode

User mode
TrustZone example (1/2)

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

   Boot vector → Secure World Supervisor

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

   Secure World Supervisor

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

   Secure World Supervisor

4. Prepare for Normal World boot

   Secure World Supervisor

**Diagram:**
- On-chip ROM: SW NA, NW NA
- On-chip RAM: SW RW, NW NA
- Main memory (DDR): SW RW, NW RW

**Legend:**
- SW: Read + Write
- NW: No Write
- NA: Not Accessible

**Notes:**
- Code (trusted OS) device key
- Code (boot loader)
5. Jump to Normal World Supervisor for traditional boot

6. Set up trusted application execution

7. Execute trusted application
Mobile TEE deployment

- TrustZone support available in **majority** of current smartphones

- **Are there any APIs for developers?**
Mobile hardware security APIs

APPLICATION DEVELOPMENT
Mobile hardware security APIs

1. Secure element APIs: (smart cards)
   - JSR 177
   - PKCS #11

2. Mobile 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

Keymaster operations
• GENERATE_KEYPAIR
• IMPORT_KEYPAIR
• SIGN_DATA
• VERIFY_DATA

Persistent storage on Normal World

Android Key Store

• Only predefined operations
  – Signatures
  – Encryption/decryption

• Global Platform is standardizing TEE APIs

• Developers cannot utilize programmability of mobile TEEs
  – Not possible to run arbitrary trusted applications
  – (Same limitations hold for hardware protected iOS key store)

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

Skip ObC
On-board Credentials goal

An open credential platform that enables existing mobile TEEs

Secure yet inexpensive
On-board Credentials (ObC) architecture

Mobile device hardware with TEE support

Mobile device

Rich execution environment (REE)

App

App

ObC API
Provisioning, execution, sealing

ObC scheduler
Trusted app persistent store
Trusted app dynamic state

Mobile OS


Centralized provisioning vs. open provisioning

Centralized provisioning (smart card)

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 trusted application extract

```plaintext
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)
```

ObC counterpart application pseudo code

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

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

Attestation protocol

Example application: Public transport ticketing

- Mobile ticketing with NFC phones and TEE
  - Offline terminals at public transport stations
  - Mobile devices with periodic connectivity
    - Such use case requires ticketing protocol with state keeping (authenticated counters)

- 110 traveler trial in New York (summer 2012)
  - Implemented as On-board Credentials trusted application

Transport ticketing protocol

Authenticated counters implemented as an ObC program

**Command 1**: Read card state and counter commitment
- **REE**: “Read”: CHALL, d
- **REE**: ctr, ack, Sigk(id, ctr)
- **REE**: SigX(“READ”, CHALL, d, ctr-ack, Sigk(id, ctr-d))

**Command 2**: Sign and increment
- **REE**: “Increment”: CHALL
- **REE**: ctr, SigX(“INCR”, CHALL, ctr)

**Command 3**: Release commitment
- **REE**: “Release”: ctrN, Sigk2(idN, ctrN)
- **REE**: ack := ctrN

**Command 4**: Sign challenge
- **REE**: “OK/Fail”
- **REE**: “Sign”: CHALL
- **REE**: SigX(“SIGN”, CHALL)

---

*Ekberg and Tamrakar. Tapping and Tripping with NFC. TRUST 2013*
Application development summary

- Mobile TEEs previously used mainly for internal purposes
  - DRM, subsidy lock

- Currently available third-party APIs enable only limited functionality
  - Signatures, decryption
  - Android key store
  - iOS key store

- Programmable TEE platforms
  - On-board Credentials
  - Demonstrates that mobile TEEs can be safely opened for developers
UEFI, NIST, Global Platform, Trusted Computing Group

STANDARDIZATION
TEE standards and specifications

- First versions of standards already out
- Goal: easier development and better interoperability
Secure Boot

UEFI
UEFI – boot principle

- UEFI standard intended as replacement for old BIOS
- Secure boot an optional feature

---

Unified Extensible Firmware Interface Specification
Nyström et al: UEFI Networking and Pre-OS security (2011)
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

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

White list + Black list for database images

(ref: UEFI spec)
UEFI secure boot

• Thus far primarily used in PC platforms
  – Also applicable to mobile devices

• Can be used to limit user choice?
  – The specification defines user disabling
  – Policy vs. mechanism
Hardware-based Trust Roots for Mobile Devices

NIST
Required security components are

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

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

“RoTs are preferably implemented in hardware”

“the APIs should be standardized”
Roots of Trust (RoTs)

**Root of Trust for Storage (RTS):** repository and a protected interface to store and manage keying material.

**Root of Trust for Measurement (RTM):** reliable measurements and assertions.

**Root of Trust for Verification (RTV):** engine to verify digital signatures associated with software/firmware.

**Root of Trust for Integrity (RTI):** run-time protected storage for measurements and assertions.

**Root of Trust for Reporting (RTR):** environment to manage identities and sign assertions.
Root of Trust mapping

RoT Verification

RoT Storage

Verification root

Cryptographic mechanisms

Volatile memory

Device key

Base identity

Platform integrity

Boot sequence

Trusted application

TEE mgmt layer

Secure storage and isolated execution

Device identification

RoT Measurement

RoT Integrity

RoT Reporting

Device certificate

Identity

Public device key

Device authentication
Trusted Execution Environment (TEE) specifications

GLOBAL PLATFORM
Global Platform (GP)

GP standards for smart card systems used many years
  • Examples: payment, ticketing
  • Card interaction and provisioning protocols
  • Reader terminal architecture and certification

Recently GP has released standards for mobile TEEs
  • Architecture and interfaces

http://www.globalplatform.org/specificationsdevice.asp
- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal API Specification v1.0
- Trusted User Interface API v 1.0
GP TEE System Architecture

REE

- Application
- TEE Client API v.1.0
- Rich Execution Environment OS

TEE

- Trusted Application
- TEE Internal API v.1.0
- Trusted Operating System
- Secure Storage
- Crypto
- I/O
- RPC
- Trusted User Interface API v.1.0

Isolation boundary

TEE Driver
// 1. initialize context
TEEC_InitializeContext(&context, …);

// 2. establish shared memory
sm.size = 20;
sm.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
TEEC_AllocateSharedMemory(&context, &sm);

// 3. open communication session
TEEC_OpenSession(&context, &session, …);

// 4. setup parameters
operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, …);
operation.params[0].value.a = 1;  // First parameter by value
operation.params[1].memref.parent = &sm;  // Second parameter by reference
operation.params[1].memref.offset = 0;
operation.params[1].memref.size = 20;

// 5. invoke command
result = TEEC_InvokeCommand(&session, CMD_ENCRYPT_INIT, &operation, NULL);
Interaction with Trusted Application

REE App provides a pointer to its memory for the Trusted App
- Example: Efficient in place encryption

1. TEE Driver
2. Isolation boundary
// each Trusted App must implement the following functions…

// constructor and destructor
TA_CreateEntryPoint();
TA_DestroyEntryPoint();

// new session handling
TA_OpenSessionEntryPoint(uint32_t param_types, TEE_Param params[4], void **session)
TA_CloseSessionEntryPoint(…)

// incoming command handling
TA_InvokeCommandEntryPoint(void *session, uint32_t cmd,
uint32_t param_types, TEE_Param params[4])
{
    switch(cmd)
    {
    case CMD_ENCRYPT_INIT:
        ....
    }
}
Secure storage: Trusted App can persistently store memory and objects

```
TEE_CreatePersistentObject(TEE_STORAGET_PRIVATE, flags, ..., 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_InvokeTACommand(session, ..., commandId, paramTypes, params[4]);
```

Also APIs for crypto, time, and arithmetic operations...
Trusted User Interface API

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

- Trusted User Interface API 1.0:
  - TEE_TUIDisplayScreen
GP device committee is working on a TEE provisioning specification

User-centric provisioning white paper
GP standards summary

• Specifications provide sufficient basis for TA development

• Issues
  – Application installation (provisioning) model not yet defined
  – Access to TEE typically controlled by the manufacturer
  – User interaction

• Open TEE
  – Virtual TEE platform for prototyping and testing
  – Implements GP TEE interfaces
  – [https://github.com/Open-TEE](https://github.com/Open-TEE)
TPM 1.2 and TPM 2.0 EA

TRUSTED COMPUTING GROUP
Trusted Platform Module (TPM)

• Collects state information about a system
  • separate from system on which it reports

• 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
  • **Sealing**: Secure **binding** with **non-volatile storage**
  • **Engine** for cryptographic operations
Platform Configuration Registers (PCRs)

- Integrity-protected registers
  - in volatile memory
  - represent current system configuration

- Store aggregated platform "state" measurement
  - a given state reached ONLY via the correct extension sequence
  - Requires a root of trust for measurement (RTM)

\[ H_{\text{new}} = H(\text{new} \mid H_{\text{old}}) \]
\[ H_0 = 0 \]
\[ H_3 = H(m3 \mid H(m2 \mid H(0 \mid m1))) \]
Use of platform measurements (1/2)

Remote attestation

– verifier sends a challenge
– attestation is $\text{SIG}_{\text{AIK}}(\text{challenge, PCRvalue})$

– AIK is a unique key specific to that TPM ("Attestation Identity Key")

– attests to current system configuration
Use of platform measurements (2/2)

Sealing

– bind secret data to a specific configuration

– Create RSA key pair PK/SK when PCR$_x$ value is Y

– Bind private key: Enc$_{SRK}(SK, PCR_x=Y)$
  – SRK is known only to the TPM
  – “Storage Root Key”

– TPM will “unseal” key only if PCR$_x$ value is Y
  – Y is the “reference value”
TPM Mobile (Mobile Trusted Module)

A TPM profile for Mobile devices that adds mechanisms for

- **Adaptation to TEEs:**
  - New roots of trust definitions and requirements

- **Multi-Stakeholder Model (MSM):**

- "**Certified boot**": Secure boot with TCG authorizations
  - Reference Integrity Metric (RIM) certificates:
    - "if $\text{PCR}_x$ matches \textit{reference}, set $\text{PCR}_x$ to target"
“If $\text{PCR}_x$ has value $H_{\text{old}}$, extend $\text{PCR}_y$ (from 0) by $H_{\text{new}}$. 

1. Verify

2. Update

Verification key

$\text{PCR}_x$

$H_{\text{old}}$

$\text{PCR}_y$

$H_{\text{new}}$
TPM 2.0

• Recent specification, in public review
  – Algorithm agility
  – New extended authorization model
  – “Library specification”
    → Defines interface, not physical security chip
    → Intended for various devices (not only PCs)
TPM 2.0 Mobile Reference Architecture

“Protected Environment”
- “the device SHALL implement Secure Boot”
- “the Protected Environment SHALL provide isolated execution”

![Diagram of TPM 2.0 Mobile Reference Architecture]

- **REE**
  - Application
  - Rich Execution Environment OS
  - **TPM 2.0 Interface**

- **TEE**
  - Trusted Operating System
  - Trusted Application
  - **TEE Driver**
TPM 2.0 on Mobile Devices

• Trusted application on TrustZone TEE is likely deployment

• Other alternatives
  – Embedded secure element (smart card)
  – Removable secure element (microSD card)
  – Virtualization
Authorization (policy) in TPM 1.2

- **System**
  - System state info
- **TPM 1.2**
  - Object (e.g. key)
    - Reference values: "PCR selection" authData
- **External auth (e.g. password)**
- **Object invocation**
- **Object authorization**
TPM 2.0

- More expressive policy definition model
- Various policy preconditions
- Logical operations (AND, OR)
- A policy session accumulates all authorization information
Authorization (policy) in TPM 2.0

System

Commands to include some part of TPM 2.0 (system) state in policy validation

Policy assertions

External authorization: passwords

signatures

Object invocation ("policy command")

Object authorization

TPM 2.0

System state info

policySession: policyDigest

Other TPM objs

Object (e.g. key)

Reference values:
authPolicy
authValue

Authorized objects

76
TPM2 Policy Session Contents

Contains accumulated session policy value: `policyDigest`

\[
\text{newDigestValue} := \text{H}(\text{oldDigestValue} || \text{commandCode} || \text{state\_info})
\]

Some policy commands `reset` the value

IF condition THEN
\[
\text{newDigestValue} := \text{H}(0 || \text{commandCode} || \text{state\_info})
\]

Can contain optional assertions for `deferred policy checks` to be made at object access time.
TPM2 Policy Command Examples

TPM2_PolicyPCR: Include PCR values in the authorization

update policyDigest with [pcr index, pcr value]

newDigest := H(oldDigest || TPM_CC_PolicyPCR || pcrs || digestTPM)

TPM2_PolicyNV: Include a reference value and operation (<, >, eq) for non-volatile memory area

e.g., if counter5 > 2 then
update policyDigest with [ref, op, mem.area]

newDigest := H(oldDigest || TPM_CC_PolicyNV || args || nvIndex->Name)
TPM2 Deferred Policy Example

TPM2_PolicyCommandCode: Include command code for later checking during ”object invocation” operation:

update policyDigest with [command code]

newDigest := H(oldDigest || TPM_CC_PolicyCommandCode || code)

additionally save policySession->commandCode := command code

policySession->commandCode checked at the time of object invocation!
Policy disjunction

TPM2_PolicyOR: Authorize one of several options:
   Input: List of digest values <D1, D2, D3, .. >
   
   IF policySession->policyDigest in List THEN
   newDigest := H(0 || TPM2_CC_PolicyOR || List)

   Reasoning: For a wrong digest Dx (not in <D1 D2 D3> ) difficult
to find List2 = <Dx Dy, Dz, .. > where H(List) == H(List2)
Policy conjunction

- No explicit AND command
- AND achieved by consecutive authorization commands $\rightarrow$ 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**

newDigest := H(0 || TPM2_CC_PolicyAuthorize || H(pub) || ..)
Let’s try this out: example 1

• Developer D
  – Has TPM2-protected keypair \( k1 \) and Application A
  – Wants **only** A can use \( k1 \) via
    • TPM2_RSA_Decrypt (key, ciphertext)

• Assume that
  – OS measured into PCR1 (if correct OS: PCR1 = mOS)
  – Foreground app into PCR2 (if A: PCR2 = mA)

• What should authPolicy of \( k1 \) be?
Exercise 1

System

Command sequence

v11 <- PolicyPCR(1, mOS)
  // v11 = h (0  || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
  // v12 = h (v11 || PolicyPCR || 2 || mA)
v13 <- PolicyCommandCode(RSA_Decrypt)
  // v13 = h (v12 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)

NOTE: We drop “TPM2_” and “TPM_” prefixes for simplicity...

Two checks:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == RSA_Decrypt?
  - PCR 1 == mOS?
  - PCR 2 == mA?
Exercise 2

• What if D wants to authorize app A (PCR2=mA) or app A’ (PCR2=mA’)

Exercise 2

Command sequence

v11 <- PolicyPCR(1, mOS)
// v11 = h (0 || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
// v12 = h (v11 || PolicyPCR || 2 || mA)
v13 <- PolicyCommandCode(RSA_Decrypt)
// v13 = h (v12 || PolicyCommandCode || RSA_Decrypt)
RSA_Decrypt(k1, c)
Exercise 2

Command sequence

v11 <- PolicyPCR(1, mOS)
   // v11 = h (0 || PolicyPCR || 1 || mOS)
v12 <- PolicyPCR(2, mA)
   // v12 = h (v11 || PolicyPCR || 2 || mA)
v22 <- PolicyOR({v12, v12'})
   // v22 = h (0 || PolicyOR || (v12 || v12'))
v23 <- PolicyCommandCode(RSA_Decrypt)
   // v23 = h (v22 || PolicyCommandCode || RSA_Decrypt)

RSA_Decrypt(k1, c)
Exercise 2

Command sequence

\[
\begin{align*}
v_{11} & \leftarrow \text{PolicyPCR}(1, \text{mOS}) \\
& \quad \text{// } v_{11} = h(0 || \text{PolicyPCR} || 1 || \text{mOS})
\end{align*}
\]

\[
\begin{align*}
v_{12}' & \leftarrow \text{PolicyPCR}(2, \text{mA'}) \\
& \quad \text{// } v_{12}' = h(v_{11} || \text{PolicyPCR} || 2 || \text{mA'})
\end{align*}
\]

\[
\begin{align*}
v_{22} & \leftarrow \text{PolicyOR}('v_{12}, v_{12}') \\
& \quad \text{// } v_{22} = h(0 || \text{PolicyOR} || (v_{12} || v_{12}'))
\end{align*}
\]

\[
\begin{align*}
v_{23} & \leftarrow \text{PolicyCommandCode(RSA\_Decrypt)} \\
& \quad \text{// } v_{23} = h(v_{22} || \text{PolicyCommandCode} || \text{RSA\_Decrypt})
\end{align*}
\]

\[
\begin{align*}
\text{RSA\_Decrypt}(k_1, c)
\end{align*}
\]
More Exercises

• Exercise 2’
  – D wants to allow *all his future apps*?
  – D has app signing keypair PK_D/SK_D

• Exercise 3
  – D wants to license the use of k1 to any app of another developer D1
  – D1’s app signing keypair PK_D1/SK_D1

• Exercise 4
  – D wants to license use of k1 to any app of any developer that he later authorizes!

Skip to Secure Boot example
Exercise 2’

Command sequence

Allow any app by D

Object invocation

RSA_Decrypt(k1, c)

Object authorization

PCR 1: mOS

PCR 2: mA

PCR 2: mA'

PCR 2: ?

policySession

authPolicy

k1 : private decryption key

Other TPM objs

System

TPM2

Object (e.g. key)

? (must be independent of PCR 2 value)
Exercise 2’

Command sequence

\[
\begin{align*}
\text{v11} &\leftarrow \text{PolicyPCR}(1, \text{mOS}) \\
&= h(0 \ || \ \text{PolicyPCR} \ || \ 1 \ || \ \text{mOS}) \\
\text{v12} &\leftarrow \text{PolicyPCR}(2, \text{mA}) \\
&= h(\text{v11} \ || \ \text{PolicyPCR} \ || \ 2 \ || \ \text{mA}) \\
\text{v13} &\leftarrow \text{PolicyCommandCode}(\text{RSA_Decrypt}) \\
&= h(\text{v12} \ || \ \text{PolicyCommandCode} \ || \ \text{RSA_Decrypt}) \\
\text{v24} &\leftarrow \text{PolicyAuthorize} (\text{signature}) \\
&= h(0 \ || \ \text{PolicyAuthorize} \ || \ PK_D) \\
\text{RSA_Decrypt}(k1, c)
\end{align*}
\]

Allow any app by D
Exercise 2’

Allow any app by D

PK_D/SK_D

Command sequence

\[ v_{11} \leftarrow \text{PolicyPCR}(1, \text{mOS}) \]
\[ v_{12}'' \leftarrow \text{PolicyPCR}(2, \text{mA''}) \]
\[ v_{13}''' \leftarrow \text{PolicyCommandCode}(	ext{RSA_Decrypt}) \]
\[ v_{24} \leftarrow \text{PolicyAuthorize} (\text{signature''}) \]

RSA_Decrypt(k1, c)
**Exercise 2**

But we don’t want to allow any OS or any policyCommand!

**Command sequence**

\[
\begin{align*}
v_{11} & \leftarrow \text{PolicyPCR}(2, mA'') \\
& \quad // v_{11} = h(0 || \text{PolicyPCR} || 2 || mA)
\end{align*}
\]

\[
\begin{align*}
v_{12} & \leftarrow \text{PolicyAuthorize (signature'')} \\
& \quad // \text{where signature’’} \leftarrow \text{Sig}_D(v_{11’’}) \\
& \quad // v_{12} = h(0 || \text{PolicyAuthorize} || \text{PK}_D)
\end{align*}
\]

\[
\begin{align*}
v_{13} & \leftarrow \text{PolicyPCR}(1, mOS) \\
& \quad // v_{13} = h(v_{12} || \text{PolicyPCR} || 1 || mOS)
\end{align*}
\]

\[
\begin{align*}
v_{14} & \leftarrow \text{PolicyCommandCode} (\text{RSA} \_\text{Decrypt}) \\
& \quad // v_{14} = h(v_{14} || \text{PolicyCommandCode} || \text{RSA} \_\text{Decrypt})
\end{align*}
\]

\[
\begin{align*}
\text{RSA} \_\text{Decrypt}(k_1, c)
\end{align*}
\]
Exercise 3

• D wants to license the use of k1 to any app of another developer D1
  – D1’s app signing keypair PK_D1/SK_D1
Exercise 3

Allow any app by D or D1
PK_D/SK_D
PK_D1/SK_D1

System

Object (e.g. key)

Object authorization

Command sequence

RSA_Decrypt(k1, c)

PK_D/SK_D
PK_D1/SK_D1

Other TPM objs

policySession

Object (e.g. key)

authPolicy

k1 : private decryption key

PCR 1: mOS
PCR 2: mA1

? (must be independent of PCR 2 value; must allow either public key to policyAuthorize PCR 2 value)
Exercise 3

Allow *any* app by D or D1

**PK_D/SK_D**

**PK_D1/SK_D1**

Command sequence

\[
\text{v31} \leftarrow \text{PolicyPCR}(2, mA) \\
\text{v32} \leftarrow \text{PolicyAuthorize (signature)} \\
\text{v33} \leftarrow \text{PolicyOR} \{(v32, v32')\} \\
\text{v34} \leftarrow \text{PolicyPCR}(1, mOS) \\
\text{v35} \leftarrow \text{PolicyCommandCode}(	ext{RSA Decrypt})
\]

\[
\text{RSA Decrypt}(k1, c)
\]

**Object** (e.g. key)

**authPolicy**

**PCR 1: mOS**

**PCR 2: mA**

**Other TPM objs**

**TPM2**

**System**

**System state info**

**Object invocation** ("policy command")

**Object authorization**

Allow *any* app by D or D1

PK_D/SK_D

PK_D1/SK_D1
Exercise 4

• D wants to license use of k1 to any app of any developer that he later authorizes!
Exercise 4

Allow any app certified by any developer authorized by D

PK_D/SK_D

System

Command sequence

PK_D’/SK_D’

policyAuthorize

TPM2

PCR 1: mOS

PCR 2: mA’

Other TPM objs

policySession

Object (e.g. key)

authPolicy

k1 : private decryption key

Object authorization

RSA_Decrypt (k1, c)

? (must be independent of PCR 2 value; independent of public key used to authorize PCR 2 value)
Example policy: Simple Secure Boot

- Suppose PCR 2 has value mA when Platform A kernel loaded

- Sequence of commands to ensure secure boot
  - [PCRExtend(2, measurement value); Start new authorization session]
  - V1 <- PolicyPCR (2, mA)
  - V2 <- PolicyCommandCode (PCRExtend)
    \[ PCRExtend(5, mA) \]

- authPolicy for PCR 5 is V2
  - V1 = h (0 || PolicyPCR || 2 || mA)
  - V2 = h (V1 || PolicyCommandCode || PCR_Extend)

Continue boot only if Platform A kernel has been loaded

Platform A kernel

measurement mA → PCR 2

measurement mA → PCR 5

Object invocation for authorization

Skip to Standards Summary
Simple secure boot not always enough

Secure boot can have the following properties

A) Extend to start up of applications

B) Include platform-dependent policy

C) Include optional or complementary boot branches

D) Order in which components are booted may matter
Advanced Secure Boot example

1. Root-of-Trust-for-Measurement starts Boot Loader and boot process
2. It loads the TEE and TPM (PCR 1)
3. It loads the REE OS (PCR 2)
4. We want to verify **loading of the OS TEE driver** (PCR 3)

*Authorization policy conditional to correct execution of previous steps*
Advanced Boot: example policy

- Policy applies to extending of PCR5 (authPolicy = X)
- Create policy session with policyDigest = X
Advanced Boot Policy

OS TEE driver will be measured and launched

IF

AND

Assumptions

Driver supplier can change policy later

Policy applies only to PCR update

External signature

OR

Platform A kernel

Platform B kernel

IF

AND

TEE OS driver loaded

measurement → PCR 3

measurement → PCR 2

measurement → PCR 2

Rollback protection...

CTR5 > 2

AND

TEE successfully loaded

measurement → PCR 1

Skip to Standards Summary
Advanced Boot Policy

- authPolicy $X = (PK_A)^*$
- driver supplier $A$ can authorize any value $Y$ as policy for PCR 5

* more precisely $H(0 \ || \ PolicyAuthorize \ || \ PK_A \ || \ ...)$

$Y \rightarrow PolicyAuthorize(Sig_A(Y)) \rightarrow X$
Example policy

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

**IF**

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

**AND**

**Platform A kernel**
- Ext.sign.
- measurement → PCR 2
- TEE OS driver loaded
- measurement → PCR 3

**OR**

**Platform B kernel**
- measurement → PCR 2

**AND**

**CTR5 > 2**
- TEE successfully loaded
- measurement → PCR 1

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

**Assumptions**

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

**IF**

- Platform B kernel
  - Measurement → PCR 2
  - CTR5 > 2
  - TEE successfully loaded

**OR**

- OS driver for TEE will be measured and launched

**AND**

- PolicyCommandCode or PolicyCPHash
- make sure PCRExtend is used (not, e.g., PCRReset)
- TEE OS driver loaded
  - measurement → PCR 3

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

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

\{Check: Eventual command == PCRExtract\}

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

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

- Platform A kernel
  - Ext. sign. → PCR 2
  - CTR5 > 2
- Platform B kernel
  - Measurement → PCR 2
  - TEE successfully loaded
  - TEE OS driver loaded

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

OS driver for TEE will be measured and launched

platform A kernel AND

measurement → PCR5

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

AND

Platform A kernel

Ext. sign.

TO bind a PCR value:

PolicyPCR (index(3), value(expected meas.))

TEE OS driver loaded

measurement → PCR 3

TEE successfully loaded

measurement → PCR 1

measurement → PCR 2

Z → PolicyCommandCode(PCRExtend) → Y → PolicyAuthorize(Sig_A(Y)) → X

{Check: Eventual command == PCRExtend}
Example policy

**Assumptions**

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

**IF**

**AND**

**Assumptions**

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

**OS driver for TEE**

**will be measured and launched**

**Platform A kernel**

**Platform B kernel**

**Ext. sign.**

**OR**

**AND**

**Checker**

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

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

PolicyOR ({V1, V2})

Platform A kernel

Platform B kernel

Assumptions

Driver supplier can change policy later

OS driver for TEE will be measured and launched

AND

OR

TEE OS driver loaded

measurement → PCR 3

TEE successfully loaded

measurement → PCR 1

CTR5 > 2

AND

Rollback protection...

Policy OR ({V1, V2})

PolicyCommandCode(PCRExtend) → Y → PolicyAuthorize(SigA(Y)) → X

{Check: Eventual command == PCRExtend}

W → PolicyPCR(3, meas.) → Z

Z → PolicyCommandCode(PCRExtend) → Y → PolicyAuthorize(SigA(Y)) → X
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**
- Ext. sign.
- Measurement \(\rightarrow\) PCR 2

**Platform B kernel**
- Measurement \(\rightarrow\) PCR 2

**IF**
- **AND**
  - CTR5 > 2

**OR**
- **AND**
  - TEE OS driver loaded
    - Measurement \(\rightarrow\) PCR 3
  - TEE successfully loaded
    - Measurement \(\rightarrow\) PCR 1

**V1 \(\rightarrow\)** PolicyOr({V1,V2}) \(\rightarrow\) W \(\rightarrow\) PolicyPCR(3, meas.) \(\rightarrow\) Z

**V2 \(\rightarrow\)** PolicyOr({V1,V2}) \(\rightarrow\) W \(\rightarrow\) PolicyPCR(3, meas.) \(\rightarrow\) Z

**Z \(\rightarrow\)** PolicyCommandCode(PCRExtend) \(\rightarrow\) Y \(\rightarrow\) PolicyAuthorize(Sig_A(Y)) \(\rightarrow\) X

{Check: Eventual command == PCRExtend}
Example policy

OS driver for TEE will be measured and launched

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

Possibly more authorizations needed (e.g., PolicyNV)

or

OSB provider updates PCR2 with result of some PolicyAuthorize(SigB(...))

Assumptions

Driver supplier can change policy later

V1 -> PolicyOr({V1,V2} -> W -> PolicyPCR(3, meas.) -> Z

V2 -> Z -> PolicyCommandCode(PCRExtend) -> Y -> PolicyAuthorize(SigA(Y)) -> X

{Check: Eventual command == PCRExtend}
Example policy

OS driver for TEE will be measured and launched

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

AND

Platform A kernel

Platform B kernel

OR

TEE OS driver loaded

And

 measurement \rightarrow PCR 5

measurement \rightarrow PCR 2

PolicyPCR(3, H(...)) \rightarrow V1 \rightarrow PolicyOr\{V1,V2\} \rightarrow W \rightarrow PolicyPCR(3, meas.) \rightarrow Z

Z \rightarrow PolicyCommandCode(PCRExtend) \rightarrow Y \rightarrow PolicyAuthorize(SigA(Y)) \rightarrow X

{Check: Eventual command == PCRExtend}

TEE succesfully loaded

measurement \rightarrow PCR 1

TEE succesfully loaded

measurement \rightarrow PCR 1
Sequence of TPM commands (1/2)

Assume PCR2 will have value mB if a kernel authorized by provider B (such as platform B kernel was booted, and PCR1 will have mN if the correct TEE driver N was loaded

\[< V1 \leftarrow \text{PolicyPCR}(2, mB) \]

\[< W \leftarrow \text{PolicyOR} \{V1, V2\} \]

\[< Z \leftarrow \text{PolicyPCR}(1, mN) \]

\[< Y \leftarrow \text{PolicyCommandCode} \text{ (PCRExtend)} \]

\[< X \leftarrow \text{PolicyAuthorize} \text{ (sig), where } sig = \text{Sig}_A (Y) \]

→PCRExtend(5, measurement value)

authPolicy for PCR5 is X
Sequence of TPM commands (2/2)

\[ V1 = h(0 || PolicyPCR || 2 || mB) \]
\[ W = h(0 || PolicyOR || (V1 || V2)) \]
\[ Z = h(W || PolicyPCR || 1 || mN) \]
\[ Y = h(Z || PolicyCommandCode || PCR_Extend) \]
\[ X = h(0 || PolicyAuthorize || PK_A) \]
Standards summary

• Global Platform Mobile TEE specifications
  – Sufficient foundation to build trusted apps for mobile devices

• TPM 2.0 library specification
  – TEE interface for various devices (also Mobile Architecture)
  – Extended Authorization model is (too?) powerful and expressive

• Mobile deployments can combine UEFI, NIST, GP and TCG standards

• Developers do not yet have full access to TEE functionality
Challenges ahead and summary

A LOOK AHEAD
Open issues and research directions

1. Novel mobile TEE architectures
2. Issues of more open deployment
3. Trustworthy TEE user interaction
4. Hardware security and user privacy

Skip to Summary
Novel mobile TEE architectures

- Multiple cores?
- Low-cost alternatives?
TEE architectures for multi-core

- Issues to resolve
  - Possible to have separate TEEs for each core?
  - Can other cores run REE, while TEE active on one?

- SICE
  - Architecture for x86
  - Assigns **one or more cores for each TEE**
  - Other cores can run REE simultaneously
Low-cost mobile TEE architectures

- Can mobile TEEs made cheaper?
  - Low-end phones and embedded mobile devices

- TrustLite
  - Execution aware memory protection
  - Modified CPU exception engine for interrupt handling

- SMART
  - Attestation and isolated execution at minimal hardware cost
  - Custom access control enforcement on memory bus
Issues of open deployment

• Certification and liability issues?
  – Especially application domains like payments

• Credential lifecycle management
  – Device migration becomes more challenging in open model
  – Hybrid approach: open provisioning and centralized assisting entity
Trustworthy user interaction

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

• Technical implementation possible

• But how does the user know?
  – Am I interacting with REE or TEE?
Hardware security and user privacy?

- Secure boot **can** be used to limit user choice
  - Common issue of mechanism vs. policy

- Allows new opportunities for attackers
  - Vulnerabilities in TEE implementation → rootkits
Summary

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

• TEE functionality and interfaces are being standardized
  – Might help developer access
  – Global Platform TEE architecture
  – TPM 2.0 Extended Authorization and Mobile Architecture

• Open research problems remain
Further reading


- **Citizen Electronic Identities using TPM 2.0,** To appear in ACM CCS TrustED workshop (2014)