Trusted Execution Environments on Mobile Devices

WiSec 2014
Kari Kostiainen, ETH Zurich

Jointly prepared with:
Jan-Erik Ekberg, Trustonic
N. Asokan, Aalto University and University of Helsinki
What is a TEE?

Trusted Execution Environment

Processor, memory, storage, peripherals

Isolated and integrity-protected

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

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

• A look back (15 min)
  – Why mobile devices have TEEs?
• Mobile hardware security (30 min)
  – What constitutes a TEE?
• Application development (30 min)
  – Mobile hardware security APIs, On-board Credentials

Break (15 min)

• Current standardization (45 min)
  – UEFI, NIST, Global Platform, TPM 2.0 (Mobile)
• A look ahead (15 min)
  – Challenges and summary

Tutorial slides

ACM WiSec 2014
7th ACM Conference on Security and Privacy in Wireless and Mobile Networks
Oxford, United Kingdom
July 23rd — 25th 2014

Conference Program

ACM WiSec 2014 is collocated with RFIDSec’14, and the two events are scheduled together. The list of papers accepted to ACM WiSec 2014 can be found here. The list of papers accepted to RFIDSec’14 can be found here. The calendar below shows the schedule for both events, colour-coded as follows (the same colour code as on the registration information page):

- Tutorial
- RFIDSec
- ACM WiSec
- Both ACM WiSec and RFIDSec

Scroll down...

**Tutorial 2:**

**Title:**
Trusted Execution Environments on Mobile Devices

**Lecturer:**
Kari Kostila, ETH Zurich

**Description:**
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.
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
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 ISN (where applicable) remains physically protected.

**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
Historical perspective

- **1970**: Cambridge CAP, Reference monitor
- **1980**: VAX/VMS, Protection rings
- **1990**: Java security architecture
- **2000**: Trusted Platform Module (TPM), Late launch
- **2010**: ARM TrustZone, TI M-Shield, On-board Credentials
  - Mobile hardware security architectures
  - Mobile OS security architectures
  - Mobile Trusted Module (MTM)

**GP TEE standards**
- NIST
- TPM 2.0

**Computers security**
- Mobile security
- Smart card security

**First part**

**Second part**
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
  - pass/fail

Authenticated boot

- OS Kernel measurer
- Boot block measurer
- Firmware measurer
  - state
  - aggregated hash
Platform integrity

Certified by device manufacturer

Boot code certificate
  Boot code hash

Mobile device hardware TCB

Device manufacturer public key

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Starts device boot

Platform integrity

Launch boot code

Boot sequence

Volatile memory

Verification root

Cryptographic mechanisms

Stores measurements for authenticated boot

Signature verification algorithm

Device key

Secure storage and isolated execution

Device identification

Trust anchor

TEE management

Starts device boot

Trust anchor

App

New

Device key

Secure storage and isolated execution

Device identification
Secure storage

Legend

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

Mobile device hardware TCB

- Verification root

Cryptographic mechanisms

- Volatile memory
  - Boot sequence
  - Trusted Application (TA)
  - TEE management
- Device key
  - Non-volatile memory
- Protected memory
- Rollback protection

Platform integrity

Secure storage

Device identification

Encryption algorithm

Legend

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

Legend

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

TEE Entry from Rich Execution Environment

- Platform integrity
- Secure storage and isolated execution
- Boot sequence
- Code executed in isolation
- Controls TA execution
- Device identification
- Base identity
- TEE management
- Non-volatile memory
- Device key
- Volatile memory
- Cryptographic mechanisms
- Verification root
- Mobile device hardware TCB

- TA code certificate
- TA code hash
- Certified by device manufacturer

See diagram for detailed explanation of concepts.
Device identification

Mobile device hardware TCB

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

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

Identity certificate
- Base identity
- Assigned identity

Multiple assigned identities

One fixed device identity

Platform integrity

Secure storage and isolated execution

Device identification
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

Sign system state in remote attestation

Used for signatures

Platform integrity

Secure storage and isolated execution

External trust root

Device certificate

- Identity
- Device public key

Issued by device manufacturer
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
    - Boot sequence
    - Trusted application
    - TEE mgmt layer
    - Secure storage and isolated execution
    - Device key
    - Non-volatile memory
    - Base identity

- Device identification
  - Device certificate
    - Identity
    - Public device key

- Device authentication
  - External trust root

Launch boot code

TEE Entry from Rich Execution Environment
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

**TEE component**

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

**On-SoC**

- RAM
- ROM
- Processor core(s)
- OTP Fields
- Internal peripherals
- On-chip Security Subsystem

**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)

Secure World and Normal World

TrustZone system architecture

Normal world (REE)

Secure world (TEE)

Device hardware

TEE entry

Trusted OS

Trusted app

Trusted app

Mobile OS

App

App

Memory controller

Access control hardware

Interrupt controller

Main CPU

Access control hardware

Modem

Off-chip/main memory (DDR)

Peripherals (touchscreen, USB, NFC...)

SoC internal bus (carries status flag)

On-chip memory

Boot ROM

TrustZone hardware architecture
TrustZone overview

Normal World (NW)

User mode

- User
- Supervisor

Privileged mode

- User
- Supervisor

Secure World (SW)

- User
- Supervisor

Secure Monitor call (SMC)

Boot sequence

Address space controllers

TZ-aware MMU

Physical address range

On-chip ROM

On-chip RAM

Main memory (DDR)
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
   - On-chip ROM
     - Code (trusted OS)
     - Device key
     - SW NA
     - NW NA
   - On-chip RAM
     - SW RW
     - NW NA

3. Configure address controller (protect on-chip memory)
   - Secure World Supervisor
   - Main memory (DDR)
     - SW RW
     - NW RW

4. Prepare for Normal World boot
   - Secure World Supervisor

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
   - Java

2. Mobile hardware key stores:
   - iOS Key Store
   - Android Key Store
   - Apple iOS
   - Google Android

3. Programmable TEE “credential platforms”:
   - Nokia
   - Trustonic TEE API
   - On-board Credentials
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

Persistent storage on Normal World

Android Key Store

• Only predefined operations
  – Signatures
  – Encryption/decryption

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

• (Same limitations hold for hardware protected iOS key store)
On-board Credentials goal

An *open* credential platform that enables existing mobile TEEs

Secure yet inexpensive
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

Driver

Trusted execution environment (TEE)
- ObC Interpreter
  - I/O data
  - Interpreted code
  - Interpreter state
  - Loaded trusted app

Mobile device hardware with TEE support

Centralized provisioning vs. open provisioning

Centralized provisioning
(smart card)

Open provisioning
(On-board Credentials)
Open provisioning model

1. Certified device key + user authentication
   PK
   Enc(PK, FK)

2. Provision new family
   Enc(PK, FK)

3. Provision new secrets
   AuthEnc(FK, secret)

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

Certified device key
PK

establish new security
domain (family)

install secrets, associate
them to family

install trusted apps, grant access to secrets

Service provider
User device

Pick new ‘family key’ FK
Encrypt family key
Enc(PK, FK)

Encrypt and authenticate
secrets
AuthEnc(FK, secret)

Authorize trusted
applications
AuthEnc(FK, hash(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

```python
// 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)
```

ObC trusted application extract

```python
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)


http://www.mirrorlink.com
Attestation protocol

Kostiainen, Asokan and Ekberg.
Practical Property-Based Attestation on Mobile Devices, TRUST 2011.
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

Ekberg and Tamrakar. Tapping and Tripping with NFC, TRUST 2013
Transport ticketing protocol

Authenticated counters implemented as an ObC program

**Command 1**: Read card state and counter commitment

<table>
<thead>
<tr>
<th>REE</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Read”: CHALL, d</td>
<td>(none)</td>
</tr>
<tr>
<td>ctr, ack, Sigk(id, ctr)</td>
<td>ctr++</td>
</tr>
<tr>
<td>Sigx(“READ”, CHALL, d, ctr-ack, Sigk(id, ctr-d))</td>
<td>ack := ctrN</td>
</tr>
</tbody>
</table>

**Command 2**: Sign and increment

<table>
<thead>
<tr>
<th>REE</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Increment”: CHALL</td>
<td>(none)</td>
</tr>
<tr>
<td>ctr, Sigx(“INCR”, CHALL, ctr)</td>
<td></td>
</tr>
</tbody>
</table>

**Command 3**: Release commitment

<table>
<thead>
<tr>
<th>REE</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Release”: ctrN, Sigk2(idN, ctrN)</td>
<td></td>
</tr>
<tr>
<td>“OK/Fail”</td>
<td></td>
</tr>
</tbody>
</table>

**Command 4**: Sign challenge

<table>
<thead>
<tr>
<th>REE</th>
<th>TEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Sign”: CHALL</td>
<td></td>
</tr>
<tr>
<td>Sigx(“SIGN”, CHALL)</td>
<td></td>
</tr>
</tbody>
</table>

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
See you in 15 minutes...

BREAK
Outline

• A look back (15 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 (15 min)

• Current standardization (45 min)
  – UEFI, NIST, Global Platform, TPM 2.0 (Mobile)
• A look ahead (15 min)
  – Challenges and summary
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

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

Key management for update

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

Key Exchange Keys

Platform Key (Pub/Priv)

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 defined user disabling
  - Policy vs. mechanism
Hardware-based Trust Roots for Mobile Devices

NIST
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

“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 Measurement

RoT Integrity

RoT Reporting

RoT Verification

RoT Storage

Verification root

Cryptographic mechanisms

Volatile memory

Boot sequence

Trusted application

TEE mgmt layer

Secure storage and isolated execution

Device key

Non-volatile memory

Base identity

Device certificate

Identity

Public device key

Device authentication
Roots of Trust in Current Smartphones

Many existing smartphones support **secure boot** and **TrustZone TEE**

1. Secure boot $\rightarrow$ Root of Trust for **Verification**
2. Measuring in secure boot $\rightarrow$ Root of Trust for **Measurement**
3. Device key + code in TZ TEE $\rightarrow$ Root of Trust for **Reporting**
4. TEE secure memory $\rightarrow$ Root of Trust for **Integrity**
5. Device key + TEE $\rightarrow$ Most of Root of Trust for **Storage**.

*No easy rollback protection!*

*Limited technical novelty... Basis for security level evaluation?*
GLOBAL PLATFORM

Trusted Execution Environment (TEE) specifications
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

[Link to specifications]

- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal API Specification v1.0
- Trusted User Interface API v 1.0
// 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);
Interacting with Trusted Application

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

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

1. Interaction with TEE Driver
2. Interaction with Trusted Application
// 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

```c
TEE_CreatePersistentObject(TEE_STORAGE_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

```c
TEE_OpenTASession(TEE_UUID* destination, ..., paramTypes, params[4], &session);
TEE_InvokeTACCommand(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
Global Platform User-centric provisioning

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

• Open TEE
  – Virtual TEE platform for prototyping and testing
  – Implements GP TEE interfaces
  – https://github.com/Open-TEE
TRUSTED COMPUTING GROUP

TPM 2.0 (Mobile)
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
  - Requires a root of trust for measurement (RTM)

Authenticated boot

\[
H_{new} = H(new \mid H_{old})
\]

\[
H_{0} = 0
\]

\[
H_{3} = H(m3 \mid H(m2 \mid H(0\mid m1)))
\]
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 2.0

- Recent specification, in public review
  - Algorithm agility
  - New authorization model
  - “Library specification”
    - Defines interface, not physical security chip
    - Intended for various devices (not only PCs)

- Our focus
  - TPM 2.0 relation to mobile devices
  - Authorization model (secure boot)
TPM 2.0 Mobile Reference Architecture

“Protected Environment”
- “the device SHALL implement Secure Boot”
- “the Protected Environment SHALL provide isolated execution”
TPM 2.0 on Mobile Devices

• Trusted application on TrustZone TEE likely

• Other alternatives
  – Embedded secure element (smart card)
  – Removable secure element (microSD card)
  – Virtualization
Authorization (policy) in TPM version 1

System

TPM 1

Object

External auth (e.g. password)

Object invocation

Object authorization

System state info

Object (e.g. key)

ruleset
TPM2 Policy Session

- More expressive policy definition model
- Various policy preconditions
- Logical operations (AND, OR)
- A policy session accumulates all authorization information
Authorization (policy) in TPM2

System

Commands to include (system) state in policy validation

external auth

Object invocation ("policy command")

Object authorization

TPM2

System state info

policySession: policyDigest

Other TPM objs

Object (e.g. key)

reference value: authPolicy
Advanced Secure Boot example

1. RTM 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 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

Platform A kernel
 measurement → PCR 2

AND

OR

Platform B kernel
 measurement → PCR 2

AND

TEE OS driver loaded
measurement → PCR 3

TEE successfully loaded
measurement → PCR 1

IF

AND

Rollback protection...
Advanced Boot Policy

OS driver for TEE will be measured and launched

Assumptions

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

IF

AND

 Ext. sign.
 Platform A kernel
 measurement → PCR 2

Platform B kernel
 measurement → PCR 2

OR

TEE OS driver loaded
 measurement → PCR 3

TEE successfully loaded
 measurement → PCR 1

CTR5 > 2

PolicyPCR(2, H(...)) → V1
PolicyPCR(2, H(...)) → V2

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

{Check: Eventual command == PCRExtend}
Standards summary

• Global Platform Mobile TEE specifications
  – Sufficient foundation to build trusted apps for mobile devices
  – More open developer access still needed

• TPM 2.0 library specification
  – TEE interface for various devices (also Mobile Architecture)

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

Challenges ahead and summary
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
Novel mobile TEE architectures

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

- **Issues to resolve**
  - When one core enters TEE mode, what others do?
  - Possible to have separate TEEs for each core?

- **SICE**
  - Architecture for x86 that assigns *one or more cores for each TEE*
  - Other cores can run REE simultaneously
  - Leverages System Management Mode (SMM)
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
  – Remote attestation and isolated execution at minimal hardware cost
  – Custom access control enforcement on memory bus
  – [Defrawy et al. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. NDSS’12.](#)
Issues of open deployment

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

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

• Trustworthy user interaction needed for many use cases
  – Provisioning
  – User authentication
  – Transaction confirmation

• Technical implementation possible
  – TrustZone supports needed interrupt handling

• But how does the user know?
  – Am I interacting with REE or TEE?
Trustworthy user interaction

• Personalized security indicator
  – Example: a figure chosen by the user
  – Protected by the TEE secure storage

• Secure attention sequence (SAS)
  – Control-Alt-Del in Windows
  – Example: double click smartphone home button to start TEE interaction

Dhamija and Tygar. The Battle Against Phishing: Dynamic Security Skins. SOUPS’05.
Trustworthy user interaction

• Do security indicators work?
  – Previous studies show that people tend to ignore indicators
  
  – Recent studies show that warnings can be effective (in some cases)

• No existing studies for smartphones

• Applications where user interaction not needed
  – Location verification for payments
  – Marforio et al. Smartphones as Practical and Secure Location Verification Tokens for Payments. NDSS’14.
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 Mobile Architecture

• Better developer access still needed

• Open research problems remain

Additional slides

TPM 2.0 Authorization model
TPM2 Policy Session Contents

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

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

\[ \text{Can contain optional assertions for } \textit{deferred policy checks} \text{ to be made at object access time.} \]
TPM2 Policy Command Examples

- **TPM2_PolicyPCR**: Include PCR values in the authorization

  update policyDigest with [pcr index, pcr value]

  \[
  \text{newDigest := H(oldDigest || TPM_CC_PolicyPCR || pcrs || digestTPM)}
  \]

- **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]

  \[
  \text{newDigest := H(oldDigest || TPM_CC_PolicyNV || args || nvIndex->Name)}
  \]
TPM2 Deferred Policy Example

TPM2_PolicyCommandCode: Include the command code for later “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!
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
**TPM2_PolicyAuthorize:** Validate a signature on a policyDigest:

**IF** signature validates **AND** `policySession->policyDigest` in signed content **THEN**

newDigest := $H(0 \ || \ TPM2_{CC}\_PolicyAuthorize || H(pub) || ..)$
Example policy: Simple Secure Boot

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

- Sequence of commands to ensure secure boot
  - V1 <- PolicyPCR (2, mA)
  - V2 <- PolicyCommandCode (PCRExtend)
  \[ \rightarrow \text{PCRExtend}(5, mA) \]

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

NOTE: We drop “TPM2_” and “TPM_” prefixes for simplicity…
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. RTM 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)
   - *Conditional to previous steps*
Advanced Boot: example policy

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

IF

OS TEE driver will be measured and launched

Assumptions

Driver supplier can change policy later

Policy applies only to PCR update

OR

Platform A kernel

Platform B kernel

External signature

AND

AND

Rollback protection...

AND

CTR5 > 2

TEE successfully loaded

measurement → PCR 2

measurement → PCR 2

measurement → PCR 1

measurement → PCR 3

measurement → PCR 5

measurement → PCR 5

measurement → PCR 2
Advanced Boot Policy

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

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

OS driver for TEE will be measured and launched

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

Ext.sign.

Platform A kernel
- measurement → PCR 2
- Rollback protection ..
- TEE successfully loaded

Platform B kernel
- measurement → PCR 2
- CTR5 > 2

IF

AND

TEE OS driver loaded
- measurement → PCR 3

OR

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

OS driver for TEE will be measured and launched

make sure PCREx tend is used (not, e.g., PCRReset)

PolicyCommandCode or PolicyCPHash

Platform B kernel

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

TEE successfully loaded

TEE OS driver loaded

measurement → PCR 3

measurement → PCR 2

measurement → PCR 1

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

OS driver for TEE will be measured and launched

IF

AND

Platform A kernel

AND

Platform B kernel

OR

Ext.sign.

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

AND

Rollback protection ...

IF

AND

TEE OS driver loaded

measurement \rightarrow PCRS

TEE successfully loaded

measurement \rightarrow PCRI

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

{Check: Eventual command == PCRExtend}
Example policy

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

To bind a PCR value:

\[
\text{PolicyPCR}(\text{index}(3), \text{value(expected meas.)})
\]

\[
\begin{align*}
Z &\rightarrow \text{PolicyCommandCode(PCRExtend)} \rightarrow Y \rightarrow \text{PolicyAuthorize}((\text{Sig}_A(Y)) \rightarrow X \\
\{\text{Check: Eventual command == PCRExtend}\}
\end{align*}
\]
Example policy

IF

AND

Assumptions

Driver supplier can change policy later

Policy applies only to PCR updates

OS driver for TEE will be measured and launched

measurement → PCR5

Platform A kernel

measurement → PCR 2

Ext. sign.

Platform B kernel

measurement → PCR 2

OR

measurement → PCR 3

TEE OS driver loaded

AND

CTR5 > 2

TEE successfully loaded

W → PolicyPCR(3, meas.) → Z

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

{Check: Eventual command == PCRExtend}
Example policy

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

PolicyOR ({V1, V2})

OS driver for TEE will be measured and launched

AND

Ext.sign.

measurement → PCR 2

OR

Platform A kernel

Platform B kernel

AND

measurement → PCR 3

measurement → PCR 2

Rollback protection..

CTR5 > 2

TEE successfully loaded

measurement → PCR 1

W → PolicyPCR(3, meas.) → Z

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

**Platform A kernel**
- IF AND
- Ext. sign.
- Measurement \(\rightarrow\) PCR 2

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

**Assumptions**
- Rollback protection..

**TEE OS driver loaded**
- TEE OS driver loaded
- measurement \(\rightarrow\) PCR 3

**OS driver for TEE**
- will be measured and launched
- measurement \(\rightarrow\) PCR 5

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

**V2 \(\rightarrow\)**
- PolicyCommandCode(PCRExtend) \(\rightarrow\) Y \(\rightarrow\) PolicyAuthorize(SigA(Y)) \(\rightarrow\) X

**Check:** Eventual command == PCRExtend

**Z \(\rightarrow\)**
- TEE successfully loaded
- measurement \(\rightarrow\) PCR 1
OS driver for TEE will be measured and launched

Assumptions
- Driver supplier can change policy later

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(Sig_B(…))

TEE OS driver loaded
- measurement → PCR 3

Platform B kernel
- Measurement → PCR 2
- TEE successfully loaded
- measurement → PCR 1

V1 → PolicyOr({V1, V2} → W → PolicyPCR(3, meas.) → Z
V2 →
Z → PolicyCommandCode(PCRExtend) → Y → PolicyAuthorize(Sig_A(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

**IF**

**AND**

**Platform A kernel**

- Ext. sign.
- Measurement $\rightarrow$ PCR 2

**OR**

**Platform B kernel**

- **Assumptions**
  - Rollback protection
  - CTR5 $>$ 2

- TEE OS driver loaded
- Measurement $\rightarrow$ PCR 3

- PolicyPCR(3, H(...)) $\rightarrow$ V1 $\rightarrow$ PolicyPCR(3, meas.) $\rightarrow$ Z
- PolicyPCR(3, H(...)) $\rightarrow$ V2 $\rightarrow$ 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}