You will be learning:

- General model for mobile platform security
  - Key hardware security techniques and general architecture
- Two examples
  - ARM TrustZone, Trusted Platform Module
Platform security architecture

Legend
- Role
- Platform Security Component
- Third-Party Software Component
- Hardware-Security Functionality
Hardware platform security

Trusted Execution Environment

- Boot Integrity
- Secure Storage
- Device Identification
- Isolated Execution
- Device Authentication

HW Security API
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)
1. Platform integrity (“boot integrity”)  
2. Secure storage  
3. Isolated execution  
4. Device identification  
5. Device authentication

More information in the 2014 IEEE S&P article
Secure boot vs. authenticated boot

Why?

Secure boot

How will you implement a checker?
- hardcode $H(\text{boot code})$ as reference value in checker (in Firmware)?

Authenticated boot

Why?

State can be:
- bound to stored secrets (sealing)
- reported to external verifier (remote attestation)
Platform integrity

Certified by device manufacturer: $\text{Sig}_{\text{SKM}}(\text{H}(\text{boot code}))$

Device manufacturer public key: $\text{PK}_M$

Legend

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

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Boot code certificate

Boot code hash

Boot sequence

Platform integrity

Launch boot code

Stores measurements for authenticated boot

Signature verification algorithm

Device key $K_D$

Certified by device manufacturer: $\text{Sig}_{\text{SKM}}(\text{H}(\text{boot code}))$

Secure storage and isolated execution

Device identification

Device manufacturer public key: $\text{PK}_M$

Trust anchor (Code)

TEE code

External certificate

Platform integrity

Launch boot code

Certified by device manufacturer: $\text{Sig}_{\text{SKM}}(\text{H}(\text{boot code}))$

Device manufacturer public key: $\text{PK}_M$
Secure storage

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

Mobile device hardware TCB

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

Cryptographic mechanisms

- Device key K_D
- Non-volatile memory
- Base
- Encryption algorithm
- Protected memory
- Rollback protection
- Device identification

Platform integrity
- Volatile memory
- Boot sequence
- Trusted Application (TA)
- TEE management

Secure storage

Insecure Storage
Isolated execution

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory

Trusted Application (TA)

TEE management

Secure storage and isolated execution

Certified by device manufacturer

TA code certificate

TA code hash

Legend

Trust anchor (Hardware)

Trust anchor (Code)

TEE code

External certificate

Platform integrity

Controls TA execution

Boot sequence

Device key $K_D$

Non-volatile memory

Base identity

Device identification

TEE Entry from Rich Execution Environment
Device identification

Multiple assigned identities (Certified by device manufacturer)

Legend

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

Mobile device hardware TCB

Verification root

Cryptographic mechanisms

Volatile memory
Boot sequence
Device key $K_D$

Non-volatile memory

Trusted Application (TA)

TEE management

Platform integrity

Secure storage and isolated execution

Identity certificate
Base identity
Assigned identity

One fixed device identity

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 \( K_D \)**
  - **Device public key \( PK_D \)**
  - **Identity**
  - **Device certificate**
  - **External trust root**

**Platform integrity**
- **Secure storage and isolated execution**
- **Boot sequence**
- **Trusted Application (TA)**
- **TEE management**

**Sign system state in remote attestation**

**Used to protect/derive signature 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

Voluntary memory

Device key $K_D$

Base identity

Boot sequence

Authorized application

Trusted TEE layer

Secure storage and isolated execution

Device identification

Launch boot code

TEE Entry from Rich Execution Environment

Identity certificate
Base identity
Assigned identity

Boot code certificate
Boot code hash

TA code certificate
TA code hash

External trust root

Device certificate

Identity

Device pub key $PK_D$
### TEE system architecture

#### Architectures with single TEE
- ARM TrustZone
- TI M-Shield
- Smart card
- Crypto co-processor
- Trusted Platform Module (TPM)

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

*Figure adapted from: Global Platform. [TEE system architecture](http://example.com). 2011.*
TEE hardware realization alternatives

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

External Security Co-processor

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.
**TrustZone overview**

### Normal World (NW)
- **User mode**
  - SCR.NS=1
  - User

- **Privileged mode**
  - SCR.NS := 1
  - Supervisor

### Secure World (SW)
- **User mode**
  - SCR.NS=0
  - User

- **Privileged mode**
  - SCR.NS := 0
  - Supervisor
  - Monitor

#### Secure Monitor call (SMC)
- SCR.NS := 1

---

**Legend:**
- MMU: memory management unit

**Address space controllers**
- TZ-aware MMU

**Physical address range**
- On-chip ROM
- On-chip RAM
- Main memory

**Boot sequence**
- Monitor
1. Boot begins in Secure World Supervisor mode (set access control)

Boot sequence

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 (1/2)
5. Jump to Normal World Supervisor for traditional boot

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

6. Set up trusted application execution

7. Execute trusted application

SMC, SCR.NS→0

trusted app and parameters
Mobile TEE deployment

• TrustZone support available in majority of current smartphones
• Mainly used for manufacturer internal purposes  
  – Digital rights management, Subsidy lock...

• APIs for developers?
// 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

• Available operations
  – Signatures
  – Encryption/decryption

• Global Platform is standardizing TEE APIs
• Developers cannot utilize programmability of mobile TEEs
  – Not possible to run arbitrary trusted applications

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

- First versions of standards already out
- Goal: easier development; better interoperability
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 Core API Specification v1.1
- Trusted User Interface API v 1.0
GP TEE System Architecture

REE

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

Isolation boundary

TEE

- Trusted Application
  - TEE Internal Core API v.1.1
- Trusted Operating System
- Secure Storage
- Crypto
- I/O
- RPC

TEE Driver

- Trusted User Interface API v.1.0
Interaction with Trusted Application

REE App provides a pointer to its memory for the Trusted App

- Example: Efficient in place encryption
// 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);
TEE Internal Core API example

// 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:
        ....
    }
}
Storage and RPC (TEE internal Core API)

**Secure storage**: Trusted App can persistently store memory and objects

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEE_CreatePersistentObject</td>
<td>Create a persistent object</td>
</tr>
<tr>
<td>TEE_ReadObjectData</td>
<td>Read data from an object</td>
</tr>
<tr>
<td>TEE_WriteObjectData</td>
<td>Write data to an object</td>
</tr>
<tr>
<td>TEE_SeekObjectData</td>
<td>Seek position in an object</td>
</tr>
<tr>
<td>TEE_TruncateObjectData</td>
<td>Truncate an object</td>
</tr>
</tbody>
</table>

**RPC**: Communication with other TAs

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEE_OpenTASession</td>
<td>Open a TA session</td>
</tr>
<tr>
<td>TEE_InvokeTACCommand</td>
<td>Invoke a TA command</td>
</tr>
</tbody>
</table>

Also APIs for **crypto**, **time**, and **arithmetic** operations...
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
What protects hardware platform security?

A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the center of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever," said the old lady. "But it's tortoises all the way down!"

- Stephen Hawking, in A Brief History of Time
Specifications: www.trustedcomputinggroup.org

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

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

• For remote parties
  • well-defined remote attestation
  • Authorization for functions/objects in TPM
Trusted Platform Module (TPM)

• Locally
  • **Generation and use** of TPM-resident keys
  • **Sealing**: Secure **binding** with **non-volatile storage**
  • **Engine** for cryptographic operations
A TPM is NOT

- An enforcer for services outside the TPM
- An eavesdropping channel for remote monitoring

HOWEVER

Secure Boot + (TEE OR TPM) can be used as a control mechanism (can protect or violate user privacy)
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)

\[
\begin{align*}
H_{\text{new}} &= H(H_{\text{old}} | \text{new}) \\
H_0 &= 0 \\
H_3 &= H(H(H(0|m1)|m2)|m3)
\end{align*}
\]
Remote attestation

• verifier sends “chal”
• attestation is: $\text{SIG}_{\text{AIK}}(\text{chal, PCRvalue})$
  – AIK a unique key specific to that TPM (“Attestation Identity Key”)
• attests to current system configuration
Sealing

• Bind secret data to a specific configuration

• E.g.,
  – create RSA keypair PK/SK when \(PCR_x\) is \(Y\)
  – bind private key: \(Enc_{SRK}(SK, PCRx=Y)\)
    • SRK is known only to TPM (cf. “device key” \(K_D\))
    • “Storage Root Key”
  – TPM will “unseal” key \(iff\) \(PCRx\) value is \(Y\)
    • \(Y\) is the “reference value”
TPM Mobile (Mobile Trusted Module)

A TPM profile adding mechanisms for

- Adaptation to TEEs

- Multi-Stakeholder Model (MSM):
  - "Certified boot"
  - Reference Integrity Metric (RIM) certificates:
    - "if PCR\textsubscript{X} == \textit{ref}, extend PCR\textsubscript{Y} by target"
“If PCR$_x$ has value $H_{\text{old}}$, extend PCR$_y$ (from 0) by $v_{\text{new}}$
TPM authorization

- Authorization essential for access to sensitive TPM services/resources.
- TPMs have **awareness of system state** (cf., removable smartcards)
Authorization (policy) in TPM 1.2

External auth (e.g. password) → System invocating Object → Object invocation → Object authorization → TPM 1.2

TPM 1.2

- System state info
- Reference values:
  - "PCR selection"
  - authData

Object (e.g. key)
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

TPM 2.0

Object
(e.g. key)

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

Policy assertions

System state info

Other TPM objs

Object (e.g. key)

Reference values:
authPolicy
authValue

Check:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == X?
  - PCR 1 Y == Z?

External authorization:
signatures
passwords

Checks:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == X?
  - PCR 1 Y == Z?
Authorization Policy Example

• Allow app A (and no other app) to use a TPM-protected RSA keypair \( k_1 \)
  – Only when a certain OS is in use

• Assume that
  – When right OS is used, \( \text{PCR 1} = m_{OS} \)
  – When app A in foreground, \( \text{PCR 2} = m_{A} \)
Enforcing the example policy

Command sequence

\[ v_{11} \leftarrow \text{... some TPM2\_policyCommand ...} \]
\[ v_{12} \leftarrow \text{... some TPM2\_policyCommand ...} \]
\[ v_{13} \leftarrow \text{... some TPM2\_policyCommand ...} \]

RSA\_Decrypt(k1, c)

Checks:
- policyDigest == authPolicy?
- deferred checks succeed?
  - command == RSA\_Decrypt?
  - PCR 1 == mOS?
  - PCR 2 == mA?
TPM2 Policy Session Contents

\[ \langle \text{accumulated session policy value: } \text{policyDigest} \rangle \]

newDigestValue := \text{H}(\text{oldDigestValue } \parallel \text{commandCode } \parallel \text{state_info})

\[ \langle \text{Some policy commands reset value} \rangle \]

IF condition THEN
newDigestValue := \text{H}(0 \parallel \text{commandCode } \parallel \text{state_info})

\[ \langle \text{deferred policy checks at object access time.} \rangle \]
TPM2 Policy Command Examples

\[ \text{TPM2\_PolicyPCR}: \text{PCR values} \]

update \textit{policyDigest} with \texttt{[pcr index, pcr value]} \\
\texttt{newDigest} := H(\text{oldDigest} \ || \ \text{TPM\_CC\_PolicyPCR} \ || \ \texttt{pcrs} \ || \ \text{digestTPM})

\[ \text{TPM2\_PolicyNV}: \text{reference value and operation} \ (<, >, \text{eq}) \text{ for non-volatile memory area} \]

e.g., \texttt{if counter5 > 2 then} \\
update \textit{policyDigest} with \texttt{[ref, op, mem.area]} \\
\texttt{newDigest} := H(\text{oldDigest} \ || \ \text{TPM\_CC\_PolicyNV} \ || \ \texttt{args} \ || \ \text{nvIndex->Name})
TPM2 Deferred Policy Example

TPM2_PolicyCommandCode: Check command during "object invocation":

update policyDigest with [command code]

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

additionally save policySession->commandCode := command code

policySession->commandCode checked before 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, .. >
such that H(... |List) == H(... |List2)
Policy conjunction

- No explicit AND command
- AND: consecutive auth. commands → order dependence

Use OR to remove the order dependence of AND
External Authorization

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

IF signature validates **AND** signed text matches `policySession->policyDigest` THEN 

newDigest := $H(0 || \text{TPM2_CC_PolicyAuthorize}|| H(\text{pub}) || ..)$
Let’s try this out

- Developer D
  - Has TPM2-protected keypair $k1$ and Application A
  - Wants **only A** can use $k1$ via
    - $\text{TPM2\_RSA\_Decrypt (key, ciphertext)}$
- 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?
Enforcing policy

Command sequence

\[ v_{11} <- \text{PolicyPCR}(1, \text{mOS}) \]
\[ \text{// } v_{11} = h(0 || \text{CC}_\text{PolicyPCR} || 1 || \text{mOS}) \]
\[ v_{12} <- \text{PolicyPCR}(2, \text{mA}) \]
\[ \text{// } v_{12} = h(v_{11} || \text{CC}_\text{PolicyPCR} || 2 || \text{mA}) \]
\[ v_{13} <- \text{PolicyCommandCode}(\text{CC}_\text{RSA_Decrypt}) \]
\[ \text{// } v_{13} = h(v_{12} || \text{CC}_\text{PolicyCommandCode} || \text{CC}_\text{RSA_Decrypt}) \]
\[ \text{RSA_Decrypt}(k_1, c) \]

Objects

- PolicyPCR
- PolicyCommandCode
- PolicySession
- authPolicy
- PCR
- k1

System state info

- PolicyPCR 1: mOS
- PolicyPCR 2: mA

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

NOTE: We drop “TPM2_” and “TPM_” prefixes for simplicity…
Exercise 1

• What if D wants to authorize app A (PCR2=m_A) or app A’ (PCR2=m_A’)
Exercise 1

Command sequence

v11 <- PolicyPCR(1, mOS)
   // v11 = h (0 || CC_PolicyPCR || 1 || mOS)

v12 <- PolicyPCR(2, mA)
   // v12 = h (v11 || CC_PolicyPCR || 2 || mA)

v13 <- PolicyCommandCode(CC_RSA_Decrypt)
   // v13 = h (v12 || CC_PolicyCommandCode || CC_RSA_Decrypt)

RSA_Decrypt(k1, c)

What change will you make?
Exercise 2

Allow any app by $D$

Command sequence

\[ \text{RSA}_\text{Decrypt}(k_1, c) \]

? (must be independent of PCR 2 value)

Object invocation

\[ \text{RSA}_\text{Decrypt}(k_1, c) \]

Object authorization

\[ k_1 : \text{private decryption key} \]
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!
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
  – Short tutorial on TPM 2.0: Citizen Electronic Identities using TPM 2.0

• Mobiles can combine UEFI, NIST, GP and TCG standards

• Developers do not yet have full access to TEE functionality
Did you learn:

- General model for mobile platform security
  - Key hardware security techniques and general architecture
- Two examples
  - ARM TrustZone, Trusted Platform Module

Contributors: Jan-Erik Ekberg, Kari Kostiainen, Luca Davi, Ahmad-Reza Sadeghi, N. Asokan
Plan for the course

- Lecture 1: Platform security basics
- Lecture 2: Case study – Android
- Lecture 3: Mobile software platform security in general
- Lecture 4: Hardware security enablers
- Lecture 5: Usability of platform security
- Invited lecture: SE Android policies
- Lecture 6: Summary and outlook