Module: Hardware Security

TRUSTED EXECUTION ENVIRONMENT (TEE)
TEE hardware realization alternatives

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

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.
TRUSTED COMPUTING GROUP
TPM / TPM2

TEE Specifications: www.trustedcomputinggroup.org
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

• Locally
  • **Generation/use** of TPM-resident keys
  • **Sealing**: Securing data for **non-volatile storage** (w/ binding)
    • **Binding**: conditions to met when unsealing the data
  • **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)

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

\[
\begin{align*}
H_0 &= 0 \\
H_1 &= H(0|m_1) \\
H_2 &= H(H(0|m_1)|m_2) \\
H_3 &= H(H(H(0|m_1)|m_2)|m_3)
\end{align*}
\]
TPM Remote Attestation

**Goal:** Check whether the prover is in a trustworthy state

**Attestation Protocol**

**Prover**

Measure software state into PCRs

"TPM Quote" \( r = \text{Sign}(SK_{AIK}, c \ || \ \text{PCR-values}) \)

**Verifier**

Challenge \( c \)

Response: \( \text{Cert}_{AIK}, r \)

Verify \( r \)

**Database of acceptable measurements**

**Attestation Identity Key (AIK)** is a unique keypair whose private key \( (SK_{AIK}) \) is *TPM-protected*

\( \text{Cert}_{AIK} \) certificate for \( PK_{AIK} \) issued by, e.g., manufacturer
Sealing

**Goal:** Bind secret data to a specific configuration

- E.g.,
  - create RSA keypair PK/SK when PCR\(_X\) is Y
  - bind private key: \(\text{Enc}_{\text{SRK}}(\text{SK}, \text{PCR}_X=Y)\)
    - SRK is known only to TPM (cf. “device key” \(K_D\))
    - “Storage Root Key” (created on TPM “take ownership” process)
  - TPM will “unseal” key iff PCR\(_X\) value is Y
    - Y is the “reference value”
Isolated Execution with TPMs

Dynamic RTM

- Dynamic PCRs (17-23) set to -1 on boot
- Special CPU instruction to
  - reset dynamic PCRs to 0
  - measure and extend a code block to PCR 17
  - launch that code
- “Late launch” of a hypervisor
- Can be used as a TEE for arbitrary code: Flicker by McCune et al:
  https://doi.org/10.1145/1352592.1352625
TPM authorization

- Authorization essential for access to sensitive TPM services/resources.
- TPMs have awareness of system state (cf., removable smartcards)
Authorization example: university admissions

Application for admission

Admission process

Top University

Admission policy: 

Exam score > 80%

Examination board
Authorization (policy) in TPM 1.2

**System**

External auth (e.g. password)

**Object invocation**

**Object authorization**

**TPM 1.2**

System state info

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

Reference values:

- "PCR selection"
- authData

**Text Content:**

Object authorization

- Reference values:
  - "PCR selection"
  - authData

TPM 1.2

System state info

**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
University admissions 2.0

Application for admission: program

Governing board:
2019: exam score > 85%

Checks:
- Portfolio meets policy?
- Documents in portfolio correct?

Collect documents into your portfolio

Policy assertions

Examination board

Application portfolio

Admission policy: Governing board

Top university

Other organizations
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

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

External authorization:
- signatures
- passwords

Object invocation (command)

Object authorization
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, $\text{PCR 1} = m_{\text{OS}}$
  – When app A in foreground, $\text{PCR 2} = m_{\text{A}}$
Enforcing the example policy

Command sequence

\[ v_{11} \leftarrow \ldots \text{some TPM2\_policyCommand} \ldots \]

\[ v_{12} \leftarrow \ldots \text{some TPM2\_policyCommand} \ldots \]

\[ v_{13} \leftarrow \ldots \text{some TPM2\_policyCommand} \ldots \]

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

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

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

\[
\text{newDigestValue} := H(\text{oldDigestValue} \mid \mid \text{commandCode} \mid \mid \text{state_info})
\]

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

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

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

- **TPM2_PolicyPCR**: PCR values
  
  update `policyDigest` with `[pcr index, pcr value]`
  
  \[
  \text{newDigest} := H(\text{oldDigest} \ || \ \text{TPM}_{-}\text{CC}_{-}\text{PolicyPCR} \ || \ \text{pcrs} \ || \ \text{digestTPM})
  \]

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

  e.g., if \textsf{counter5} > 2 then
  
  update `policyDigest` with `[ref, op, mem.area]`
  
  \[
  \text{newDigest} := H(\text{oldDigest} \ || \ \text{TPM}_{-}\text{CC}_{-}\text{PolicyNV} \ || \ \text{args} \ || \ \text{nvIndex->Name})
  \]
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!
Other policy commands

• **TPM2_PolicyOR**: Authorize one of several options:
  **Input**: List of digest values \(<D1, D2, D3, .. >\)

  \[
  \text{IF } \text{policyDigest in List} \text{ THEN}
  \text{newDigest := } H(0 \| \| \text{TPM2_CC_PolicyOR } \| \| \text{List})
  \]

• **TPM2_PolicyAuthorize**: Validate a signature on a policyDigest:
  **Input**: signature and public key
  **IF** signature validates \(\text{AND} \) signed text matches \(\text{policyDigest}\)
  **THEN**
  \[
  \text{newDigest := } H(0 \| \| \text{TPM2_CC_PolicyAuthorize } \| \| \text{H(pub)} \| \| \text{..)}
  \]
**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 || TPM2_CC_PolicyAuthorize|| H(pub) || ..)
```

Using TEEs

ANDROID KEYSTORE
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?
Android Key Store API

Android Key Store example

```
// create RSA key pair
Context ctx;
KeyPairGeneratorSpec spec = new
    KeyPairGeneratorSpec.Builder("key1", KeyProperties.PURPOSE_SIGN);
...
spec.build();

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

// make a signature
Signature sig = Signature.getInstance("SHA256withRSA/PSS");
sig.initSign(kp.getPrivate());
```

Android, Hardware-backed Keystore, 2015-2018
Key Store implementation: example

Keymaster operations
- Public key algorithms
- Symmetric key algorithms (AES, HMAC) from v1.0
- Access control, key usage restrictions
- Key attestation (from v2.0), “ID attestation” (from v3.0)
- Android Protected Confirmation (Android 9, API level 28)

Persistent storage on Normal World

Elenkov. **Credential storage enhancements in Android 4.3.** 2013
Android, **Hardware-backed Keystore,** 2015-2018
Android, **Protected Confirmation,** 2018
Android Key Store

• Available operations
  – Signatures
  – Encryption/decryption
  – Attestation, confirmation

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

• Different API abstraction and architecture needed
  • Example: On-board Credentials
  • GlobalPlatform device working group specifications
TEE hardware realization alternatives

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

TEE component

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.
TEE instances

ARM TRUSTZONE
ARM TrustZone architecture

System on chip (SoC)

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

Secure World and Normal World

- Normal world (REE)
  - Mobile OS
  - App
  - App
- Secure world (TEE)
  - Trusted app
  - Trusted app
  - Trusted OS

Device hardware

TrustZone overview

Normal World (NW)

User

Supervisor

Secure World (SW)

User

Supervisor

Monitor

SCR.NS=1

SCR.NS=0

SCR.NS := 1

SCR.NS := 0

Secure Monitor call (SMC)

Address space controllers

TZ-aware MMU

On-chip ROM

On-chip RAM

Main memory

physical address range

Legend:
MMU: memory management unit
TrustZone example (1/2)

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

Boot sequence

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

On-chip ROM
- SW NA
- NW NA

On-chip RAM
- SW RW
- NW NA

Main memory (off-chip)
- SW RW
- NW RW

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

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

trusted app and parameters
TZ-enabled CPUs

• TZ: set of ARM processor extensions
• Combined with other building blocks needed for TEEs
  – Trust root to verify code (e.g., hash of manufacturer’s code signing key)
  – Device-secret initialized during chip manufacture
  – Monotonic counter or writable secure memory
Secure state entry/exit in TrustZone

What happens during entry/exit?

- **Store/restore all shared registers**
  - Kernel: switching between processor modes
  - Secure monitor: switching between worlds
- **Validate/(un)marshal parameters**
  - TEE driver
- **Reconfigure MMU**
  - Secure monitor

Register banking: copies of registers

- **Special purpose registers (SP, LR, SPSR)**
  - Banked between modes, but not worlds
  - except at **highest privilege mode**
- **Ordinary registers are not banked**
TEE specifications:
https://www.globalplatform.org/specificationsdevice.asp

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

TEE Driver

Isolation boundary

TEE

Trusted Application

TEE Internal Core API v.1.1

Trusted Operating System

Secure Storage

Crypto

I/O

RPC

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;
TEECAllocateSharedMemory(&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);
// 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>Creates a persistent object in secure storage</td>
</tr>
<tr>
<td>TEE_ReadObjectData</td>
<td>Reads data from an object in secure storage</td>
</tr>
<tr>
<td>TEE_WriteObjectData</td>
<td>Writes data to an object in secure storage</td>
</tr>
<tr>
<td>TEE_SeekObjectData</td>
<td>Seeks an offset in an object in secure storage</td>
</tr>
<tr>
<td>TEE_TruncateObjectData</td>
<td>Truncates an object in secure storage</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>Opens a session to another TA</td>
</tr>
<tr>
<td>TEE_InvokeTACommand</td>
<td>Invokes a command on another TA</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
  – Original intent: virtual TEE platform for TA developers
    • Implements GP interfaces: TA development w/ standard Linux tooling
  – Port for Android (requested by an OEM)
  – https://github.com/Open-TEE
TEE standards and specifications

- First versions of standards already out
- Goal: easier development; better interoperability
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
TEE instances

INTEL SOFTWARE GUARD EXTENSIONS (SGX)
Intel Software Guard Extensions

- HW-supported TEE functionality in ring-3
- Enclave code/data encrypted by HW
- Supports attestation and sealing

Intel Software Guard Extensions:
Academic papers: https://software.intel.com/en-us/sgx/academic-research
How does SGX work?

sgx_create_enclave(path,..., &eid, ...)

Call func()

func(): Execute func()

Return

ECREATE, EADD, EEXTEND, EINIT

EENTER

EEXIT

OS/Hypervisor

Hardware support for enclaves

Skip to SGX attestation
SGX – Create Enclave

1. Create App
2. Create app certificate (includes HASH(App) and Dev PK)
3. Upload App
4. Create enclave
5. Allocate enclave pages
6. Load & measure enclave
7. Validate certificate and enclave integrity
8. Generate enclave K key
9. Protect enclave

Skip to SGX attestation
Enclave Creation – Details

1a. Request Enclave Pages

3b. copy

1b. Allocate EP to App

2a. ECREATE(SECS)

3a. EADD(*src, *dest)

4a. EEXTEND(*src)

5a. EINIT

5b. Update HASH

2b. Init SECS

4b. Hardware measures

EPC: Enclave Page Cache  EPCM: EPC Map  MEE: Memory Encryption Engine  MMU: Memory Management Unit  SECS: SGX Enclave Control Structure

CPU

Trusted  Untrusted

EPC

Enclave

Application

Encl. code

Enclave

EPC

EPC list

OS

RAM

SECS

MEE

MMU

Table:

<table>
<thead>
<tr>
<th>#</th>
<th>Key</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>K</td>
<td>PK</td>
</tr>
<tr>
<td>n+1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Skip to SGX attestation
Enclave Entry and Exit – Details

- **AEP**: Async Exit Point
- **EPC**: Encl. Page Cache
- **EPCM**: EPC Map
- **ISR**: Int. Service Routine
- **MEE**: Mem. Enc. Engine
- **TCS**: Thread Control Structure

**Diagram Details**
- **EENTER(TCS, AEP)**
- **ERESUME**
- **EEXIT**
- **Interrupt**
- **Stack**
- **Save context in Enclave**
- **Lock TCS, start Enclave**
- **Resume Enclave**

**Legend**
- **Trusted**
- **Untrusted**
Attestation in SGX

• Local Attestation: one enclave verifies another on the same device

• Remote Attestation: a remote party verifies an enclave
Enclave Identity

Identity of an enclave:

– Enclave’s **initial state**
– sealing identity
Initial State

• *Enclave measurement* representing:
  – Contents of enclave pages (initial code/data)
  – Relative position of enclave’s pages

• Determined during enclave creation:
  – Log activities during enclave creation
  – Digest of log contents in *MRENCLAVE*
  – Only CPU can modify the MRENCLAVE
Sealing Identity

- **Sealing authority (SA)** signs enclaves prior to distribution:
  - Signature on trusted (expected) value of initial state
  - Signature and SA’s public key sent to devices that need to run the enclave

- During enclave creation on device:
  - Signed measurement
    - verified using SA’s public key
    - compared with local measurement
    - If matched, sealing identity (hash of the SA’s public key) stored in the MRSIGNER register
Local Attestation

1. Verifier sends measurement ($m_{\text{Verif}}$) to prover
2. Prover calls $EREPORT$, with $m_{\text{Verif}}$ as parameter, to create report
3. Prover’s report (ID and MAC generated using the verifier’s report key) returned
   
   Report := $ID_{\text{Prover}}$, MAC($ID_{\text{Prover}}$, $RepKey_{\text{Verifier}}$)
4. Report transferred to verifier
5. Verifier calls $EGETKEY$ (for reports)
6. Verifier’s report key is returned
7. MAC included in Report verified using received report key
Remote Attestation

1. Create Report
2. Verify Report
3 & 4. Create Quote
5 & 6. Verify Quote
Intel Enhanced Privacy ID (EPIID)

- Group signature scheme
- Each signer
  - owns a secret key
  - belongs to a group
- Group has a public key $PK_G$
- Use $PK_G$ to verify signatures generated by any member
Sealing

• Store persistent data securely
• Enclaves get sealing keys via EGETKEY
• Two modes:
  – Sealing to Enclave-Identity
    • key derived from contents of MRENCLAVE
  – Sealing to Sealing-Identity
    • key derived from contents MRSIGNER