Platform Security
-- for computing devices --

Jan-Erik Ekberg 1.3. 2013
Mobile vs. Personal Computing vs. Server Clouds?

Device properties are certainly not defined by processor architecture any more.

Lenovo K900
intelX86

Toshiba AC100
ARM

ARMv8
(64bit for servers)
Mobile vs. Personal Computing vs. Server Clouds?

Device interaction may not be even be determined by OS ...
... or by form factor
... or by services (everybody wants FB and Skype)
Platform Security:

a set of hardware and software mechanisms that provide enforcement, reporting and service enablement in the device

Contents:

1) Hardware enablers and roots of trust
2) Hardware security architectures:
   ARM TrustZone and TCG Dynamic Root of Trust
3) Trusted Execution Environments:
   The Global Platform TEE specification
   OnBoard Credentials
4) The Trusted Platform Module (TPM):
   History, driving use cases, v 1.2
   v2 and authorization

+ demo or NFC use case
Target setting

We have a critical piece of code and possibly secrets.

Where do we put these to be sure that they execute correctly, and don’t leak information?

(The code can operate for the user ... say a credit card .. or against the user ... say a SIM Lock)
Roots of Trust
-- a way to talk about hardware adaption --
Roots-of-Trust

- a required security function – misbehaviour cannot be detected
- Term originates from TPM, and extensively used by the Trusted Computing Group / Mobile
- RoTs are closely tied to the logic and environment on which it performs its trusted actions.

Special Publication 800-164 (Draft):
"Guidelines on Hardware-Rooted Security in Mobile Devices"

“Security components are foundational elements that can be leveraged by the device, the operating system (OS), and applications. The three required security components are

a) Roots of Trust (RoTs),
b) an application programming interface (API) to expose the RoTs to the platform, and
c) a Policy Enforcement Engine (PEnE)"

“RoTs are preferably implemented in hardware”
Root of Trust for Storage (RTS): provides a protected repository and a protected interface to store and manage keying material

Root of Trust for Verification (RTV): provides a protected engine and interface to verify digital signatures associated with software/firmware

Root of Trust for Integrity (RTI): provides protected storage, integrity protection, and a protected interface to store and manage assertions

Root of Trust for Reporting (RTR): provides a protected environment and interface to manage identities and sign assertions

Root of Trust for Measurement (RTM): provides measurement used by assertions protected

[TCG: Root of Trust for Update (RTU): provides a maintenance mechanism for RoTs]

In terms of mechanism

1. **Device Integrity**: Is there a (starting) point where we know that device integrity is assured?
2. **Isolation**: Is there a mechanism by which unintended interaction between information owners can be assured?
3. **Protected storage**: Is there a confidential, integrity- and rollback protected storage on the device at rest?
Secure Capabilities are built out of Roots-of-Trust

**Security Capabilities**

- Protected Storage
- Isolation
- Device Integrity

**Roots of Trust**

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

Picture: Andrew Regenshield: NIST/Computer Security Division
Roots-of-Trust (examples)

Trad: PC with smart card

- Device Integrity: - not for application
- Isolation: - for smart card app
- Storage: - on smart card

Issue: Attack on the UI or (bank) client SW

PC with smart card and UEFI secure boot

- Device Integrity: - software signed
- Isolation: - for smart card app - between OS apps
- Storage: - on smart card

Issue: Trustworthiness of isolation. OS big and may have flaws.

PC with smart card and UEFI secure boot and a trusted OS in parallel with big OS

- Device Integrity: - software signed
- Isolation: - for smart card app - between OS:s
- Storage: - on smart card

Issue: ?
Processor Secure Environments

- RoTs are relatively easy to implement with separate HW chips e.g. (embedded) Secure Elements (SE)
- This is an expensive approach
- Major processing core and chipset providers do provide RoT support in the main execution environment
- PSEs have been used in mobile phones for almost 10 years
- An isolated software with integrity and storage governed by a PSE is called a **Trusted Execution Environment**

Cost impact of secure environments

- Mobile phone w. PSE: \(\sim 10^{1} \) €
- Mobile phone w. eSE: \(\sim 10^{1} \) €
- Hardware security module: \(\sim 10^{3} \) €
PSE / TEE
hardware enablers
Security enablers in the chip(set)

- Trust roots and identities are **not** secret information
- Logically the difference between a trust root and an identity is scope and use: trust roots are shared between many devices and used **for validation**. Identities are device-specific and useful for **data binding** or directly for (non-cryptographic) **identification**

![Diagram showing trust root, identity, and cryptographic mechanism](image)
Secure boot is the conditional execution of further components, based on a validation based on a trust root and possibly a cryptographic mechanism.

- The boot sequence must unconditionally reach the validation step
- The validation process / calculation must be trustworthy

The collective of software booted by (recursive) application of secure boot can be called the trusted computing base.
Further identities can be assimilated by the device using identity certificates, i.e. records that can be validated by means of the trust root (using the cryptographic mechanism) and bound to the device by means of the identity present in hardware.

The validation of these further identities can be performed under the control of the TCB.
Security enablers in the chip(set)

A **Trusted Execution Environment** (TEE) is an isolated execution domain
- where **authenticated** code can be executed on-demand from the TCB or outside
- (authentication based on trust root)
- where isolation implies logical and sometimes physical **isolation of the computation** and **memory access** taking place inside the PSE.

![Diagram showing security enablers in the chip(set)]
Security enablers in the chip(set)

A **volatile register** in the TEE that stores and maintains data of the boot sequence or other environment data known by the TCB
- Can be used by code inside the PSE as an input for decision-making after boot
- Combined with secret keying, can be used for remote attestation of register contents
Security enablers in the chip(set)

The addition of a non-volatile **device secret** in the TEE domain, makes it possible to construct TEE code to

- Securely store (secret) information outside the PSE and recover it e.g. during secure boot or when needed by other TEE code
- Perform services such as DRM or SIMLock

In principle there can also be shared keys in the TEE. In that case device-specific secrets (encrypted blobs) can be bound to the device by including the device identity in them
Security enablers in the chip(set)

An external trust root can provide certificates for a **device secret** in the PSE.
- (Code) provisioning for the device or TEE is possible, even for third parties
- Device authentication is possible, and through that other forms of authentication like system/application/data attestation.
Security enablers in the chip(set)

If the secure storage is handled outside of the TEE, then rollback-protection can be achieved with a secure clock, counter or NV memory present inside the TEE. This is usable for firmware version control, PIN/PUK error entry limits etc.
ARM TrustZone
-- the de-facto PSE --
ARM TrustZone

... is an existing hardware PSE
... is included in many / most mobile phone SoCs
... is a good example of how a few, selected HW primitives can bootstrap a whole security architecture

References:

• ARM Security Technology: Building a Secure System using TrustZone® Technology (Whitepaper)
• CoreLink TrustZone Address Space Controller TZC-380 r0p1

Some pictures are from the above, non-confidential, but copyrighted material by ARM
Processor modes in TZ

**CPRS.M:** NS=0

- **Secure**
  - User
  - FIQ / IRQ
  - Supervisor
  - Monitor
    - (in secure mode independently of NS → has access to all registers in all modes)

- **Non-Secure**
  - User
  - FIQ / IRQ
  - Supervisor

**NS=1**

- **Secure**
  - User
  - FIQ / IRQ
  - Supervisor
  - Monitor

- **Non-Secure**
  - User
  - FIQ / IRQ
  - Supervisor

Banked registers (per mode):
- Link register (gen.)
- Stack pointer
- Stored program counter

1000’s of ctrl registers

NS / Privileged mode firewall for R/W

E.g. MMU setup, DMA, exception vector setup, controller conf, timers

Supervisor mode call (SMC), HW interrupts

Boot vector

Write to CPRS.M

Secure Monitor call (SMC)

Supervisor mode call (SMC), HW interrupts

Exception vector

Exception vector
ARM SoC architecture (a much simplified view)

- The information of the core state is visible on the interconnect
- Dedicated controllers can interpret the state and serve as access control filters for memory access
- The controllers are configurable, and have a "secure boot lock" i.e. all configuration can be disallowed after set-up.
Secure memory segmentation can be set up on many levels

... by the MMU translation (processor configurable)
... by configuring the external controller
... and considering where memory resides
(on the chip – externally)
1. Boot begins. Staring point secure supervisor mode

2. ROM contains a sensitive algorithm and a key. Move these to SoC RAM

3. Configure address controller to shield off SoC ROM/RAM

4. Set-up Monitor mode and Normal Supervisor

Note: Up to step 4, execution has never left the SoC!
5. Jump off “the wagon”

CPRS.M.NS := 1
JMP to bootloader

At this point is looks like SoC memory and the whole ARM TZ does not exist, it is shielded off!

An ordinary boot follows, set up MMU, load OS, drivers, rootkits etc.

A. Set up input for “the algorithm”

Leave input somewhere (on kernel stack?)

B. Execute

Compute algorithm based on input, return results, clear caches, set CPRS.M -> Supervisor
JMP -> stored PC
ARM TrustZone + Secure Boot + Secrets

... is alone almost enough to fulfill the Roots of Trust:

1. Secure boot provides verification ➞ **Root of Trust for Verification**
2. The measuring function of secure boot is the **Root of Trust for Measurement**, you can trust that it works correctly
3. A device key + code in TZ ➞ **Root of Trust for Reporting**
4. What we report is **Root of Trust for Integrity**, we can keep them in SoC RAM between the time of measurement and time of reporting
5. We do have most of **Root of Trust for Storage**: We can shield off secrets in ROM, we can use such keys in the TZ to encrypt and store on flash. Rollback protection must be arranged separately.

---

**Trusted Execution Environment (TEE)**

- **Storage**
- **Isolation**
- **Integrity**
Trusted Platform Module (TPM)
-- one functional interface to a TEE or SEE --
TCG Trusted Platform Module (TPM)
... is an application(OS) interface to secure services
... is deployed to hundreds of millions of PCs and laptops (v1.2. chip + drivers)
... Could become the way applications and OS services interact with platform security

References:

www.trustedcomputinggroup.org


• (TPM2): Trusted Platform Module Library Parts 1-3, revision 0.93, public review
TPM

- A TPM is a system component that has state that is separate from the system on which it reports
- A TPM relies on a set of (HW) RoT
- Attestation:
  - An external entity attests to a TPM with a credential for a key embedded in the TPM
  - An external entity attests to a platform to vouch for the RTM
  - The TPM produces attestations with the embedded key for possibly constrained, TPM-resident new keys, or for the state of the platform measured by RTM (integrity reporting)
- Integrity measurement: A transitive trust chain can be collected in so call Platform Configuration Registers (PCR)
- **Authorization** for functionality provided by the TPM can be required
- In terms of functionality, the TPM provides
  - Key generation and key use with TPM-resident keys
  - Secure binding with encryption, as well as non-volatile storage
  - Use as an engine for encryption / decryption and signing, also for hash algorithms and symmetric ciphers
PCRs

... A PCR aggregates measurements for eventual binding or attestation

... If RTM is correct, then a given expected PCR value can ONLY be reached by a correct extension sequence

... If measurements are actual measurements and if the root holds, then any divergence in the code causes an irrevocable change in the PCR value.

Integrity reporting: SIG(chall, PCR value)

H=H(new | old) →
H=H(m7 | H(m6 | H (m7 | ... )))
H(0) = 0
PCR binding

... Generate a TPM key. Give it a PCR value binding

.... Only if the OS is in a pretermined state can the key be used!

Usage authorization:
PCR==XYZ

Platform

measure m3
send m3 to TPM
launch code 3

measure m2
send m2 to TPM
launch code 2

measure m1
send m1 to TPM
launch code 1

Code 3

Code 2

Code 1

RTM

TPM

H=H(new | old) →
H=H(m7 | H(m6 | H (m7 | ... )))
H(0) = 0

PCR val: XYZ
TPM Extended Authorization (process)

1. Create Policy Session
   - Policy session handle: 000

2. Ex: Add some TPM internal state (value of a ctr, PCR value, ...)
   - Value: 764

3. Ex: Bind to eventual command (cmd=CMD)
   - Value: 323

4. Ex: OR policy (list of hashes)
   - If policyHash (323) is in the list
     - Reset and add H(whole list)
   - Value: 000

5. Ex: PolicyAuthorize (SIG(valX), keyhandle)
   - If policyHash (799) == valX && PubK validates SIG
     - Reset and add H(PubK)
   - Value: 643

Final: command CMD’ to operate on the object, with policy session handle

If policyHash == authVal && CMD==CMD’ && ...
- Allow operation

TPM Object with authVal = 643
Secure boot with EA

Verification Agent (trusted engine) → (in this example, the trustworthiness is irrelevant)

1) TPM2_StartAuthSession
2) TPM2_PolicyPCR
3) TPM2_PolicyCPHash

- **Constraints**: Current PCR values, target PCR (in Extend)

- **Authorization**: External signature. Note that the operation uses a ticket, that in terms of efficiency can replace internal RIM certs.

- **Operation**: Extend target PCR

Other PCR(s) → Pre-constraint PCR (set)

TPM2_Extend (incl. value to be extended)

4) TPM2_VerifySignature
5) TPM2_PolicyAuthorize

Verified PCR

booting events stored here
TPM Mobile

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

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

2) Certified boot: Secure boot with TCG authorizations that leaves a PCR trace for binding and attestation

3) Multi-stakeholder model: Platform- and Application TPMs co-exist in the TEE:
   a) Application TPMs can have an associated TEE secure application and an associated rich environment application
   b) Associated TEE secure applications can be provisioned with TPM EA formats and mechanisms
   c) Associated TEE secure applications can be automatically measured into their respective TPMs for binding
Global Platform TEE
-- the emerging standard for a trusted OS ? --
Global Platform

Most of the smart-card based ecosystems around authentication, payment and ticketing make use of Global Platform standards:

- For card interaction and provisioning protocols
- For reader terminal architecture and certification

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

References:
http://www.globalplatform.org/specificationsdevice.asp
- TEE System Architecture
- TEE Client API Specification v.1.0
- TEE Internal API Specification v1.0
The **Global Platform Device Architecture** provides the following interfaces:

- An API to communicate with the TEE (shared memory-based) application and an UUID-based identification / session control for this access.
- A standard system interface library (libc ..) for trusted applications with RPC, crypto and necessary I/O functions.

Eventually, these APIs may become the reference model for writing code for and interacting with a TEE. Missing pieces still include **provisioning** and **compliance** aspects.
Global Platform has a strong presence in standards close to security HW.
Nokia Research Snippets
OnBoard Credentials
OnBoard Credentials – one proprietary approach to 3rd-party TEE usage

- A Nokia-proprietary solution (100 mil. deployed devices 2012)

- Supports “open provisioning” akin to TCG endorsement for privacy CA:s but applied to credential programs and secrets. I.e. “anybody can provision”

- Not a trusted OS. Consists of a VM, and distributes processing between the trusted VM in the TSE and an untrusted OS driver

- Credential programs are primarily written in BASIC

- Original intent: To design a credential system that is better than browser password stores and cheaper than dedicated tokens
OnBoard Credentials – architecture

- ported to 5 processor families from 3 chipset vendors

- and runs under 4 OSs

- “survives” on 7.5 kB of TEE memory excluding crypto and signed loading support
function main()

    dim mode as integer
    dim j as integer
    dim password as array
    dim data as array

    mode = read_integer(IO_PLAIN_RW, 0)
    read_array(IO_PLAIN_RW, 1, password)
    j = alen(password)
    if j != 8
        return 0
    end
    if mode == 0
        read_array(IO_PLAIN_RW, 2, data)
        append_array(password, data)
        write_array(IO_SEALED_RW, 2, password)
    end

    if mode == 1
        read_array(IO_SEALED_RW, 2, data)
        j = array_part_match(password, data, 0, 8)
        if j == 0
            return 0
        end
        j = alen data
        j = j - 8
        copy_array_part(password, data, 8, j)
        write_array(IO_PLAIN_RW, 2, password)
    end
    return 1
end
OnBoard Credentials – provisioning

- Provisioning protocols typically focus on **user authentication** only
  - ... and also tend to focus on secrets like keys – not programs (TCG cert provisioning)
- IETF keyprov WG: Dynamic Symmetric Key Provisioning Protocol (DSKPP)
  - Allows **device authentication** as well
- We need more...
  - provision a key so that it can be accessed by **specific credential programs**
- Subject to...
  - “Anyone can provision credential secrets securely to a credential program”
  - Support for multiple versions of credential programs
  - Support for several co-operating credential programs
OnBoard Credentials – provisioning

- Provision a family root key to the device
  - using authentic device public decryption key $PK_{dev}$

- Transfer encrypted credential secrets and code
  - using family confidentiality keys CK
  - Code may often come from a different source that the secret

- Endorse credential programs for secret family membership
  - (program ID is encrypted)
  - using family integrity key IK
NFC + TEE for public transport

NY MTA trial, summer 2012
Architecture context

This presentation focuses on protocols and architecture to realize Id-based ticketing in a partially off-line setting.
Gated and non-gated transport with Ids

1) In a gated architecture gates **physically block** customers from entering the transport system. The gates **authenticate customers** and **reports** the transaction evidence to backend servers.

2) In a non-gated architecture travellers **touch terminals** (during system entry / exit) under the threat of randomly applied **ticket inspection**.

We want to **enforce** that the **traveller’s device**, not the terminal, **reports** transaction evidence.
Transaction Evidence

By transaction evidence we mean:

1) Customer **identity binding**
2) **Place** of system entry / exit
3) **Time** of system entry / exit
4) Cryptographic evidence / binding of 1) - 3). **The user is a plausible attacker in this ticketing model!**
5) Possibly other context information
Long-term rationale

- Fare systems in city public transport are **often rigid** (time-based, zones, ..)
- Assumption: Only in gated systems can more fine-grained fare calculation be performed.

**IF**

*we can get* [*id, place, time*] **reliably for all travel endpoints** and possibly along the route

**THEN**

*the fare structures can be made much more fine-grained for the benefit of the customer* as well as the **transport authority**
Self-imposed constraints

1. Gated systems have a **mass-entry / mass exit issue**. Any transaction with a gate should have a low failure rate and a delay of < 300ms

2. Non-gated systems cannot expect wiring (communication / electricity) of e.g. bus stops

3. Even in densely populated areas, mobile phones statistically have connectivity issues. **Partially off-line systems** are preferred

4. Some **realism** in support for the existing mobile client base must be maintained. Non-ideal intermediate solutions are needed.
Why mobile phones?

Gated systems are best(?) served with traditional technology.

**Claim:** we cannot bridge the gap to non-gated without handheld devices
Technical fundament

1. Usage-bound, authenticated counters
2. Bound replays of old challenges
1. Usage-bound, authenticated counters

Every counter value signed only once

challenge

\text{Sig} \ (\text{challenge, "2"})

Usage operation

Secure environment

Signing key (Id bound to this key)

Limit for remote release commitment

Key for validating counter release

Signing key

Limit for remote release commitment

Key for validating counter release
2. Bound replays of old challenges

Challenges have privacy-protected Id information for server. Secure environment can channel history back to server.
The Interface 1) + 2)

- The 4 operations of the TEE are separated to avoid attacks where one operation is used as an oracle for one of the others.
- Some signatures are symmetric (k, k2) to save space

The Interface

- **OS / NFC channel**
  - **TickCert(s)**
  - **OS / NFC channel**
  - **Secure Env**

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

- Invariant for operation: \(d \leq \text{limit}\)
- \(">\text{Read”}: \text{CHALL}, d
- \(\text{ctr}, \text{ack}, \text{Sig}_k(\text{id}, \text{ctr})\)
- \(\text{Sig}_x(“\text{READ”}, \text{CHALL}, \text{d}, \text{ctr-ack}, \text{Sig}_k(\text{id}, \text{ctr-d}))\)

**Command 2**: Sign and increment

- Invariant for operation: \(\text{ctr - ack} < \text{limit}\)
- \(”\text{Increment”}: \text{CHALL}\)
- \(\text{ctr}, \text{Sig}_x(“\text{INCR”}, \text{CHALL, ctr})\)

**Command 3**: Release commitment

- Invariant for operation: \(\text{ctrN} > \text{ack}, \text{idN} = \text{id}\)
- \(”\text{Release”: ctrN, Sig}_{k2}(\text{idN}, \text{ctrN})\)
- \(”\text{OK/Fail”}\)

**Command 4**: Sign challenge

- \(”\text{Sign”: CHALL}\)
- \(\text{Sig}_x(“\text{SIGN”}, \text{CHALL})\)

**Variables**

- \(\text{id}\) (ticketing identity)
- \(k\) (symm. key), \(x\) (priv. key)
- \(k_2\) (symm. or private key)
- \(\text{ctr}\) (counter), \(d\) (difference)
- \(\text{ack}\) (acknowledged value)
- \(\text{limit}\) (ctr commit window)

**Operation**

- (none)
- \(\text{ctr++}\)
- \(\text{ack := ctrN}\)
- (none)
Why bother with new primitives – legacy shortcomings (in non-gated mostly!)

EMV cards:
1. Typically stand-alone card -> no non-gated
2. Even as mobile phone SE/SIM, no counter/time binding → online check only, since replay (time of signing) cannot be determined
3. Current technology is slow (600ms->). Issue with gated travel.

MiFare:
1. Storage based, assumes challenger is secure and has access to keys (can be applied to non-gated, but attacks against individual bus stop cards may break the full system)
2. Can include counters. Operation however is inverted, i.e. counter updates are authenticated, not necessarily the counter response.
Non-gated operation overview

Phone app.

Server cloud

Bus stop card

Get signature for challenge and card counter (+some history)

Partial off-line border

Phone secure env.

Get server-readable challenge

Feed bus stop response as challenge, get signature for that and phone counter

All evidence to server in exchange for phone counter release, and possibly a verification ticket signed by server

Fare calculation and auditing
Non-Gated protocol

Back-end (B)

Sigx("INCR", SigR("INCR", [A, A*], ctrR), ctrX),
TickCertR, TickCertx, [time diff]

ctrN, Sigk2(id, ctrX), tickB

tickB = Sigk(idx, ctrx), aux, SigB(Sigk(idx, ctrx), aux). aux = ["NON_GATED", timestamp, tap_location]

Device (X)

(null), 0

ctrX, ack, Sigk(idx, ctrx)
Sigx("READ", CHALL, 0, ctrx-ackx, ackx, Sigk(idx, ctrx))

Terminal (R)

(A = Sigk(idx, ctrx))

Secure environment in Device X

A*, ctrR, SigR("INCR", [A, A*], ctrR)

TickCertR

A*, ctrR, SigR("INCR", [A, A*], ctrR)

Secure environment in Device X

ctrR++

A

phase 0

phase 1

phase 2

phase 3

ctrX++
Some attack prevention mechanisms

1. Customer taps but does not report
   1. Device counter prevent selective non-reporting
   2. Bus-stop cards returns tap info through “history” side-channel
   3. Customer’s counter is not released – secure side inhibits tapping
   (of course, all of this is subject to the threat of inspection – in a non-gated system with no inspection no tickets/taps are ever needed)

2. ”Relay” attacks – customer remotely taps when inspection is imminent
   - Bus stop card counters provide tap ordering (virtual time). Relayed taps can be indentified as being out of order with respect to e.g. the vehicle being inspected

3. Collusion / viral attacks
   - Auditing mechanisms from smart card fraud protection can be directly applied - committed tuples [id, time, place] will identify e.g. ”speed-of-light” movement.
Conclusions

- Platform security fundamentals converge around a well-defined set of trust roots
- A secure or trusted execution environment is likely part of the future computing architectures
- Standardization is taking place to achieve conformance across interfaces and functionality in this domain. Still, many open or partially open problems remain:
  - How is the TEE application lifecycle managed, how are they provisioned. How is the environment measured?
  - What are the direct interfaces to a TEE? How will they become visible to the user (Trusted UIs, direct connectivity)?
  - How do the technical fundamentals for TEEs evolve? Multiprocessing, new forms of non-volatile memory as mass memory. Side-channel attack prevention.