What is a TEE?

**Trusted Execution Environment**

- Processor, Memory, Storage, Peripherals
- Isolated & Integrity-protected
- from the “normal” execution environment (aka Rich Execution Environment)

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

A brief look back: How did this come to pass?
Hardware security features on mobile devices
On-board Credentials: opening up TEEs for app developers
A look ahead: standardization and beyond
A Look Back

Why do most mobile devices today have TEEs?
Platform security for mobile phones

Mobile network operators;
1. Subsidy locks → immutable ID
2. Copy protection → device authentication, app. separation
3. ...

Regulators;
1. RF type approval → secure storage
2. Theft deterrence → immutable ID
3. ...

End users;
1. Reliability → app. separation
2. Theft deterrence → immutable ID
3. Privacy → app. separation
4. ...

Closed → Open
Different Expectations compared to the PC world
Early adoption of platform security

Both IMSI and IMEI require physical protection.

Physical protection means that manufacturers shall take necessary and sufficient measures to ensure the programming and mechanical security of the IMEI. The manufacturer shall also (where applicable) ensure protection against unauthorized changes to the SIM.

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:

widespread use of hardware and software platform security

~2001  
M-Shield™ Mobile Security Technology

~2002  
Symbian OS Platform Security

~2005  
TrustZone® Security Foundation by ARM®

~2008  
Android
Hardware Security Features

What is in a TEE?
Hardware support for platform security

- **Trust root**
- **Base identity**
- **Crypto Library**
- **Boot sequence (ROM)**

- Public key hash
- E.g., serial number

TCB for platform software

Basic elements in immutable storage
Secure bootstrapping

- Trust root
- Base identity

Crypto Library
- Boot sequence (ROM)

- Boot code hash
- Validate and execute

Launch platform boot code

Secure boot

TCB for platform software

Boot authorized code only
Identity binding

- **Identity certificate**
  - Base identity
  - Assigned identity

- **Code certificate**
  - Boot code hash
  - E.g., IMEI, link-layer addresses, …

- **Trust root**
- **Base identity**

- **Crypto Library**
- **Boot sequence (ROM)**

- **Secure boot**
- **TCB for platform software**

- **Launch platform boot code**

- **Securely assign different identities to the device**

- **Validate and accept assigned ID**
Authorized execution of arbitrary code, isolated from the OS; access to device key
Secure state

- Identity certificate
  - Base identity
  - Assigned identity
- Code certificate
  - Boot code hash
  - TEE code hash
- Trust root
- Base identity
- Crypto Library
- Boot sequence (ROM)
- Secure boot

TCB for platform software

Authenticated boot

Launch platform boot code

TEE API
Secure boot vs Authenticated boot

1. **BIOS**
   - **Boot block**
   - **OS Kernel**
   - **Checker**
   - **Pass/Fail**

2. **OS Kernel**
   - **Measurer**
   - **State**
Secure state

- **Identity certificate**
  - Base identity
  - Assigned identity

- **Code certificate**
  - Boot code hash
  - TEE code hash

- **Secure boot**
  - Trust root
  - Base identity
  - Crypto Library
  - Boot sequence (ROM)

- **TEE**
  - Configuration register(s)
  - Device key
  - TEE code

- **Integrity-protected state within the TEE**

- **Authenticated boot, Securing TEE sessions**

- **Rollback protection for persistent secure storage**

- **TCB for platform software**

- **Launch platform boot code**

- **TEE API**
Device authentication

- Identity certificate
  - Base identity
  - Assigned identity
- Code certificate
  - Boot code hash
  - TEE code hash
- Device certificate
  - Identity
  - Public device key
- External trust root

Secure boot
- Secure boot (ROM)
- TCB for platform software
- Crypto Library
- Boot sequence

Launch platform boot code
- TEE code
- Device key

Configuration register(s)

Prove device identity or properties to external verifier

Non-vol. memory or counter

Device authentication, secure provisioning, attestation
Architectural options for realizing TEEs

External Secure Element

Embedded Secure Element

Processor Secure Environment

Figures taken from “GlobalPlatform Device Technology, TEE System Architecture”, Version 1.0, December 2011
Hardware security architectures (mobile)

TI M-Shield, ARM TrustZone

Augments central processing unit: “Secure processor mode”

Isolated execution with on-chip RAM: limited (e.g., < 20kB)

Access to memory locations can be restricted based on mode

Secure storage: e.g., using write-once restricted-read e-fuses
Processor modes in TrustZone

Secure World
SCR.NS=0

Normal World
SCR.NS=1

User

Supervisor

Monitor
(in secure mode independently of NS → has access to all registers in all modes)

Address space controllers

TZ-aware MMU

Secure Monitor call (SMC)
CPSR.M := '10110'

SCR.NS := 1,
CPSR.M := '10011'

NS / Privileged mode firewall for R/W

For more information, TrustZone pages at ARM
Using a TEE: the TrustZone example

All Protected Applications are equal
No open developer API on REE side
Hardware security architectures (TCG)

Trusted Platform Module (TPM)
- Standalone processor on PCs
- Isolated execution for pre-defined algorithms
- Arbitrary isolated execution with DRTM (“late launch”)
- Platform Configuration Registers (PCRs)
- Monotonic counters

TPM Mobile (previously known as MTM)
- Mobile variant of TPM
- Defines interface
- Implementation options: TrustZone, M-Shield, software

External Secure Element
Multiple implementation options
Hardware platform security features: summary

**Secure boot**: Ensure only authorized boot image can be loaded

**Authenticated boot**: Measure and remember what boot image was loaded

**Identity binding**: Securely assign different identities to the device

**Secure storage**: Protect confidentiality/integrity of persistent data

**Isolated execution**: Run authorized code isolated from the device OS

**Device authentication**: Prove device identity to external verifier

**Remote attestation**: Prove device configuration/properties to external verifier
Uses of hardware security

For device manufacturer and operator:

Immutable ID
- secure boot, identity binding

Copy protection
- secure boot, identity binding, device authentication, secure storage, isolated execution

Subsidy lock
- secure boot, identity binding, secure storage, isolated execution

... 

How can developers make use of hardware security?
On-board Credentials
Opening up TEEs for App developers
On-board Credentials (ObCs)

An open credential platform that leverages on-board trusted execution environments.

Secure yet inexpensive
On-board Credentials (ObCs)

SW-only credentials
- Easy, cheap, flexible
- Insecure

Dedicated HW credentials
- Secure, intuitive
- Expensive, inflexible, single-purpose

Open (provisioning): Like multi-application smartcards, but without issuer control.
Issuer-centric provisioning for smartcards

card management (personalization, application provisioning)

End User

Card Issuer
Trusted Service Manager

service providers
On-board user credentials: design goals

Credential programs can be executed securely

Credential secrets can be stored securely

Anyone can create and use new credential types
  Security model to strongly isolate credential programs from one another
  Avoid need for centralized certification of credential programs

Anyone can provision credential secrets securely to a credential program
  Need a mechanism to create a secure channel to the credential program
  (certified) device keypair; unique identification for credential programs

Protection of asymmetric credentials is attestable to anyone
  Anyone can verify that a private key is protected by the TEE
**Design constraints**

- No fine-grained access control within TEE
- Small on-chip memory (too small for standard interpreters)
- Open provisioning implies no fixed trust domains/hierarchies
ObC Architecture

Build on any TEE that supports:

• Secure execution (within TEE)
• Secure storage (secret key OPK in TEE)
• Certified device keypair (PK_D/SK_D in TEE), CERT_D
• Source of randomness
ObC Architecture

Build on any TEE that supports:

- Secure execution (within TEE)
- Secure storage (secret key OPK in TEE)
- Certified device keypair (PK_D/ SK_D in TEE), CERT_D
- Source of randomness

```
function main()
    read_array(IO_PLAIN_RW, 0, data)
    read_array(IO_SEALED_RW, 1, key)
    aesenc(cipher, data, key)
    write_array(IO_PLAIN_RW, 0, cipher)
    return 0
end
```

“On-board Credentials with Open Provisioning”, Kostiainen et al, ASIACCS ‘09
Isolation of ObC Programs

Isolating the platform from ObC programs
- Constraining the program counter, duration of execution, …

Isolating ObC programs from one another
- Only one ObC program can execute at a time
- An ObC program can “seal” data for itself
  - Sealing key is different for every independent ObC program
    Sealing-key = KDF (OPK, program-hash)
  - A program can invoke functions like “seal(data)” (unsealing happens automatically on program loading)
Idea: a **family** of credential secrets + credential programs endorsed to use them

“family” = dynamic trust domain; **same-origin** authorization policy
Provisioning credential secrets (2/3)

- Provision a family **root key** to the device
  - using *authentic device public key* $\text{PK}_D$

- Transfer encrypted credential secrets
  - using authenticated encryption (AES-EAX) with RK

- Endorse credential programs for family membership
  - Program ID is a cryptographic hash of program text
  - using authenticated encryption (AES-EAX) with RK
Anyone can define a family by provisioning a root key ("Same Origin" policy)

Multiple credential secrets and programs can be added to a family

Credential Programs can be encrypted as well
Asymmetric ObCs

Cert_D (Device certificate) Certificate for PK_D issued by manufacturer

SKAE (Subject Key Attestation Evidence) for PK: Signature on PK by SK_D, attesting: SK is within TEE and who can use SK

“Key Attestation from Trusted Execution Environments”, Kostiainen et al, TRUST 2010
- M-Shield/TrustZone secure boot used for validation of OS
- Interpreter, provisioning system are PAs
  - Use on-chip RAM
- OPK from chip-specific secret
- Device key pair
  - generated by Prov. PA
  - protected by OPK
  - [certified by manufacturer]
ObC on Lumia WP8/TrustZone (2011-2013)

- TZ secure boot for OS validation
- OPK from chip-specific secret
- Device key pair
  - protected by OPK
  - certified during manufacture
- Previous instantiations for
  - different OSs: Symbian, MeeGo, Linux
  - different TEEs: M-Shield, TPM

Skip to ObC examples
ObC Features

Custom Credentials
- Secure key/code provisioning

Built-in Credentials
- Key attestation or Secure key Provisioning

Device Certification
- Validate device platform

Device Authentication

Application Authentication

Content attestation

Secure user credentials

Platform authentication
Example usage scenarios: Platform Authentication

Prove to a third party (e.g., external server)

**Device authentication**: identity of device
E.g., CAPTCHA-avoidance

**Application authentication**: identity of application/process
E.g., Extended Web Service APIs for trusted apps

**Content attestation**: type of content
E.g., Enforcing driver distraction rules in MirrorLink
MirrorLink Remote Attestation

Smartphone | Car head unit

User input

Content for user output

Enforcement of Driver distraction regulations

"Head unit can use attestation for enforcing driver distraction"

www.cs.helsinki.fi

http://mirrorlink.com/
Attestation protocol

Attestation protocol

TEE

Attestation service

Attested application

Verifier

Pick random nonce n

Pick property p to attest

Attest(n, p, PKA)

Check application identifier
Verify property p

sig ← Sign(SK_D, n || p || Hash(PKA))

sig, Cert_D

p, sig, Cert_D, PKA

appSig ← Sign(SK_A, appData)

appData, appSig

Verify Cert_D and sig
Check property p
Save PK_A

Verify appSig

Application Identifier

App1

App2

...

Property

P1, P2

P3

...

“Practical property-based attestation”, Kostiainen et al, TRUST 2011
Example usage scenarios: User Credentials

Provision and store user credentials to user’s personal device

User benefits:

“no need to a bunch of different security tokens”;
“digital credentials provisioned easily” (http, e-mail, …)

Phone-as-smartcard: use device-resident credentials from legacy PC apps (e.g., browsers, Outlook, VPN clients)

NFC Transport ticketing

“Soft” tokens: embedded SIM, embedded SecurID

…
Phone as smartcard (PASC)

Applications use public key (PK) cryptography via standard frameworks

- Crypto API (windows), Cryptoki (Linux, Mac), Unified Key/cert store (Symbian)

Agnostic to specific security tokens or how to communicate with them

→ Any PK-enabled smartcard can be used seamlessly with PK-aware applications!

What if mobile phone can present itself as a PK-enabled smart card?

“Can hand-held computers still be better smartcards?”, Tamrakar et al, INTRUST 2010
NFC Transport Ticketing

- Support ticketing in both gated and non-gated public transit systems
- Ticketing based on itinerary and identity verification
- Trial in NY MTA Long Island Rail Road
TEE support for transport ticketing

Support for managing authenticated counters implemented as an ObC program

REЕ/NFC channel

Command 1: Read card state and counter commitment
Invariant for operation: d <= limit

```
"Read": CHALL, d
ctr, ack, Sig_k(id, ctr)
Sig_x("READ", CHALL, d, ctr-ack, Sig_k(id, ctr-d))
```

Command 2: Sign and increment
Invariant for operation: ctr - ack < limit

```
"Increment": CHALL
ctr, Sig_x("INCR", CHALL, ctr)
```

Command 3: Release commitment
Invariant for operation: ctrN > ack, idN==id

```
"Release": ctrN, Sig_k2(idN, ctrN)
```

Command 4: Sign challenge

```
"Sign": CHALL
Sig_x("SIGN", CHALL)
```

---

\[id\] (ticketing identity)
\[k\] (symm. key), \[x\] (priv. key)
\[k2\] (symm. or private key)
\[ctr\] (counter), \[d\] (difference)
\[ack\] (acknowledged value)
\[limit\] (ctr commit window)

---

"Mass transit ticketing with NFC Mobile Phones", Ekberg & Tamrakar, INTRUST 2011
"Tapping and Tripping with NFC, Ekberg & Tamrakar", TRUST 2013
Embedded SecurID token

Joint research project with RSA security (2008)
ObC Status

Available on off-the-shelf Nokia WP8 and Symbian devices

Development environment for ObC programs in BASIC

Credential Manager and interfaces (native, C#)

Available from Nokia under limited license agreement for research and testing [http://obc.nokiareserach.com](http://obc.nokiareserach.com)

More information in two dissertations:

2. 2013, Jan-Erik Ekberg: Securing Software Architectures for Trusted Processor Environments
Limitations

Open provisioning model

What about liability and risk management?
Is intuitiveness diminished?
  e.g. User interaction for credential migration (lifecycle management)

Certification and tamper resistance

Not comparable to high-end smart cards?

A powerful tool, but not a silver bullet
Will open-provisioning emerge as an alternative to centralized provisioning?
A Look Ahead
Standardization and Beyond
Commercial offerings starting to appear

Provide crypto API using TEE-protected keys
No open developer API for trusted execution, provisioning, attestation etc.?

http://www.trustonic.com/
Standardization in Global Platform

Open Issues:
- Provisioning
- TEE Functional API

http://www.globalplatform.org/specifications/device.asp
Issuer-centric provisioning for smartcards

Card Issuer
Trusted Service Manager

End User

service providers

Recap

Card management
(personalization, application provisioning)
GP: Consumer-centric Provisioning vision

“A New Model: The Consumer-Centric Model and How It Applies to the Mobile Ecosystem”,
GP White Paper, March 2012

---

Trust token providers
Trusted Service Managers

“Security verification (optional)”

application provisioning

service providers

End User

<table>
<thead>
<tr>
<th>token provisioning</th>
<th>personalization?</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential lifecycle management?</td>
<td></td>
</tr>
</tbody>
</table>

Skip to conclusions
Best of both worlds: GP and TPM Mobile?

TEE Functional API: Extended TCG Software Stack (TSS) with support for
- provisioning
- trusted applications
  making use of TPM 2 features like fine-grained access control
Summary

Hardware-based TEEs are widely deployed on mobile devices

But access to app developers has been limited

ObC: a proprietary solution to open up on-board TEE to developers

Global Platform: standardizing TEE functionality and interfaces

Will consumer-centric / “open” provisioning succeed?

“What is sauce for the goose...” Next generation mobile rootkits, BlackHat EU 2013