Mobile Trusted Computing

Fruct 2011
7.11 2011
Jan-Erik Ekberg

Partly joint work with N.Asokan, K Kostiainen, E Reshtova
Platform Security

- To achieve secure processing in heterogeneous environments, **trust roots** are needed.
- Without **enforcement**, guarantees are hard to give.
- A good security infrastructure leaves room for untrusted components without sacrificing overall security.
- Compared to perimeter security, the **trusted computing base** is minimized.
- Software vulnerability analysis is still an integral part of the system, but not at run-time, security is achieved by “updates only”.
- **Privacy** needs to be also guaranteed by policy, not just by mechanism. This is common-place, e.g. in communication networks.
Enforcement mechanisms have a history

- Hardware-support for platform security
  - Cambridge CAP etc. (~1970s) / Multics
  → Extended to Trusted Execution Environments

- Hardware-assisted secure storage

- Secure and authenticated boot
  - TCPA and TCG (late 1990s)
  - Academic research projects (mid 1990s)
  → Extended (private secure storage for applications)
  → Adapted (normal vs. developer mode in MSSF)

- Permission-based platform security architectures
  - VAX /VMS privileges for user (~1970s)
  → Adapted for applications
  - Code signing (mid 1990s)
  → Borrowed for application installation
HW Enablers / ASIC support
(high-level view)
Hardware support for platform security

TCB for platform software

Public key hash

Trust root

Base identity

Crypto Library

Boot sequence (ROM)

E.g., serial number

Start of boot code

Basic elements in immutable storage

Nokia Research Center 2011
Secure bootstrapping

1. **Trust root**
2. **Crypto Library**
3. **Boot sequence (ROM)**
4. **TCB for platform software**
5. **Launch platform boot code**

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

**Validate and execute**

Ensure only authorized boot image can be loaded
Identity binding

- Trust root
- Base identity
- Crypto Library
- Secure boot code
- Code certificate
  - Boot code hash
- Assigned identity
- Identity certificate
  - Base identity
  - Assigned identity

E.g., IMEI, link-layer addresses, ...

Validate and accept assigned ID

TCB for platform software

Securely assign different identities to the device

Launch platform boot code

Nokia Research Center 2011
Trusted execution

Identity certificate
  - Base identity
  - Assigned identity

Code certificate
  - Boot code hash
  - TrEE code hash

TrEE
  - Device key
  - TrEE code

Basis for secure external storage

Validate and execute

Isolated execution

Authorized code execution, isolated from the OS

TCB for platform software

Launch platform boot code

TrEE API

Crypto Library

Boot sequence (ROM)

Secure boot

Base identity

Trust root

Nokia Research Center 2011
Secure state

Identity certificate
- Base identity
- Assigned identity

Code certificate
- Boot code hash
- TrEE code hash

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

TrEE
- Configuration register(s)
  - Device key
  - TrEE code

TCB for platform software
- Launch platform boot code
- TrEE API

Integrity-protected state within the TrEE

Rollback protection for persistent secure storage

Securing TrEE sessions, authenticated boot
Device authentication

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

- **Code certificate**
  - Boot code hash
  - TrEE code hash

- **Device certificate**
  - Identity
  - Public device key

- **External trust root**

- **TrEE**
  - Configuration register(s)
  - Device key
  - Non-vol. memory or counter
  - TrEE code

- **TrEE API**

- **Launch platform boot code**

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

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

- **Prove device identity or properties to external verifier**

- **Device authentication, secure provisioning, attestation**

Nokia Research Center 2011
Summary of hardware mechanisms

• **Secure boot**: Ensure only authorized boot image can be loaded
• **Authenticated boot**: Measure and remember loaded image

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

• **Secure storage**: Protect confidentiality and integrity of data
• **Isolated execution**: Run authorized code isolated from OS

• **Device authentication**: Prove device identity to external verifier
• **Remote attestation**: Prove device configuration to verifier
Hardware security architectures (mobile)

• TI M-Shield and ARM TrustZone

• Augments central processing unit
  − “Secure processor mode”

• Isolated execution with on-chip RAM
  − Very limited (<10kB)

• Secure storage
  − Typically with write-once E-fuses

• Usually no counters or non-volatile memory
  − Cost issue
Hardware security architectures (TCG)

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

• Mobile Trusted Module (MTM)
  − Mobile variant of TPM
  − Can be implemented using e.g. TrustZone or M-Shield
  − Discussed further...
Uses of hardware security

• Recap from features
  − Secure/authenticated boot
  − Identity binding/device authentication
  − Secure storage
  − Remote attestation

• Uses of hardware security (device manufacturer)
  − Device initialization
  − DRM
  − Subsidy lock

• How can developers make use of hardware security?
SW/OS support
Adoption of security mechanisms

Operators

Regulators

End users

Hardware-based mechanisms

~2002

TrustZone®

Security Foundation by ARM®

~2001

~2005

~2008

~2010

~2011

Software-based mechanisms

J2ME

Symbian OS

Platform Security

Android

Windows Phone

MeeGo™
Open mobile platforms

• Java ME ~2001
  – For “feature phones”
  – 3 billion devices!
  – Not any more supported by the latest smartphones

• Symbian ~2004
  – First “smartphone” OS
  – App development in C++ (Qt)

• Android ~2007
  – Linux-based OS
  – App development in Java

• MeeGo ~2010
  – Linux-based OS
  – App development in C (Qt)
  – MSSF

• Windows Phone ~2010
  – Windows (CE) based kernel
  – End-user app development in Silverlight / .NET
Mobile platform security model

• 3.5 phases
  1. Distribution
  2. Installation
  3. Run-time enforcement
  4. ++ off-line enforcement

• Common techniques
  – Code signing
  – Permission-based access control architecture
  – (User involvement in assigning permissions)
  – App R&D by local “installation” / “opened device”
Distribution

- Developer produces a software package
  - Code
  - Manifest

- May submit to a signer for a trusted signature

- Distributed to device via on-line stores (typically)
Installation

• Installer consults local policy and trusted signature
  - Identify application
  - Grant requested privileges

• Installer may prompt user
Run-time enforcement

• Monitor checks if subject has privileges for requested access

• Resource may perform additional checks

• User may be prompted to authorize access
1. **OS bootstrapping**

Is hardware security used to secure OS bootstrapping?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure boot</td>
<td>Not applicable</td>
<td>No?</td>
<td>Authenticated boot: “Normal mode” vs “Developer mode”</td>
<td>No?</td>
</tr>
</tbody>
</table>
## 2. Application identification

### How are applications identified at install and runtime?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install and runtime:</td>
<td>Install:</td>
<td>Install:</td>
<td>Install:</td>
<td>Install:</td>
</tr>
<tr>
<td>• Protected range SID and</td>
<td>• Signing key</td>
<td>• Signing key</td>
<td>• Software source</td>
<td>• Software source</td>
</tr>
<tr>
<td>VID (managed)</td>
<td>• Midlet attributes</td>
<td>• Unix UID</td>
<td>(signing key)</td>
<td></td>
</tr>
<tr>
<td>• UID (unmanaged)</td>
<td></td>
<td>• Package name (locally unique)</td>
<td>• Package name</td>
<td>No IPC between installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>applications</td>
</tr>
<tr>
<td></td>
<td>Runtime:</td>
<td></td>
<td>Runtime:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Unix UID</td>
<td></td>
<td>• Software source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Package name</td>
<td></td>
<td>• Package name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(locally unique)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Nokia Research Center 2011
### 3. Application update

**How is a new version of an existing application verified?**

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected SID, VID:</td>
<td>Signed midlets:</td>
<td>“Same origin” policy</td>
<td>“Same or higher origin” policy</td>
</tr>
<tr>
<td>• trusted signature</td>
<td>• same-origin policy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rest:</td>
<td>Unsigned midlets:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• no controls</td>
<td>• user prompt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Permission granularity

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed set of “capabilities” (21)</td>
<td>Fine-grained permissions (many)</td>
<td>Fine-grained permissions (112)</td>
<td>Fine-grained resource-tokens</td>
<td>Fixed set of capabilities (8++) Four “chambers”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux access control</td>
<td>Linux access control</td>
<td></td>
</tr>
</tbody>
</table>

How finely is access control defined?

Android and MSSF: Each application is installed under a separate Linux UID
## 5. Permission assignment (basis)

### Basis for granting permissions?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 categories</td>
<td>Trusted signatures for protection domains</td>
<td>4 protection levels</td>
<td>Trusted signatures</td>
<td>Chambers map to stakeholders (MS / OEM / operator / user)</td>
</tr>
<tr>
<td>Trusted signature (also user prompts)</td>
<td>4 permission modes</td>
<td></td>
<td>Local policy file</td>
<td>Capabilities defined by developer (least-privilege prot.)</td>
</tr>
</tbody>
</table>

**User, System, Restricted, Manufacturer**

**Blanket, Session, One-shot, No**

**Least Privilege Chamber**
- Standard Rights
- Elevated Rights
- Trusted Computing Base

**Normal (automatic)**
- Dangerous (user-granted)
- Signature (developer-controlled)
- SystemOrSignature (Google-controlled)
6. Permission assignment (user prompting)

<table>
<thead>
<tr>
<th></th>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td>21 capabilities</td>
<td>Function group</td>
<td>Permission group</td>
<td>Capability description</td>
</tr>
<tr>
<td>description</td>
<td></td>
<td>description</td>
<td>description</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 groups</td>
<td>11 groups</td>
<td></td>
</tr>
</tbody>
</table>

**What is shown to the user?**

- E.g., LOCATION, NETWORK, ACCOUNTS, ...
- E.g., NetAccess, PhoneCall, Location, ...

**E.g.,** Read user data, Use network, Access positioning, ...
7. Permission assignment (timing)

When are permissions assigned to a principal?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install-time assignment</td>
<td>Run-time prompts</td>
<td>Install-time assignment</td>
<td>Install-time assignment Run-time privilege shedding possible</td>
<td>Install-time assignment</td>
</tr>
</tbody>
</table>

Symbian and MSSF: Permissions of app loading a DLL is a subset of permissions of DLL
8. Application integrity

How is the integrity of installed applications protected?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated directory</td>
<td>Java sandboxing</td>
<td>Java sandboxing</td>
<td>IMA, Smack</td>
<td>Chamber-dependent sandboxing (.NET, OS permissions)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linux access control</td>
<td>Offline protection with EVM and TrEE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Integrity Measurement Architecture (IMA)

- Store hash of file (in extended attribute security.ima) and verify on launch

Extended Validation Module (EVM)

- Store MAC of all extended attributes (in security.evm) and verify on access
9. Access control policy

How does a resource declare the policy for accessing it? How is it enforced?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare in code</td>
<td>[System resources] Enforced by VM</td>
<td>Declare in manifest</td>
<td>Declare in manifest</td>
<td>Declare in manifest</td>
</tr>
<tr>
<td>Enforced by IPC framework or code</td>
<td>Enforced by VM</td>
<td>Enforced by VM</td>
<td>Enforced by Smack or via libcreds</td>
<td>Enforced by VM</td>
</tr>
</tbody>
</table>

Nokia Research Center 2011
10. Application data protection

How can applications protect the confidentiality and integrity of their data?

<table>
<thead>
<tr>
<th>Symbian</th>
<th>Java ME</th>
<th>Android</th>
<th>MSSF</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime:</td>
<td>Runtime:</td>
<td>Runtime:</td>
<td>Runtime:</td>
<td></td>
</tr>
<tr>
<td>• private directory</td>
<td>• private record stores</td>
<td>• dedicated UID</td>
<td>• fine-grained data caging</td>
<td></td>
</tr>
<tr>
<td>Off-line:</td>
<td>Off-line:</td>
<td>Off-line:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• private secure storage</td>
<td></td>
<td>• private secure storage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Nokia Research Center 2011
GP TEE introduction

TEE System Architecture 0.4
TEE Internal API Specification 0.27
Architecture pictures (from spec.)

**REE:** Rich Execution Environment

**TEE:** Trusted Execution Environment
Architecture, software view

TEE internal API spec. domain
Implementation can be
- A processor environment
- A separate core on a chip
- A co-processor (?)
Specification overview

The TEE API defines the driver interface for
- send code to the secure environment
- execute code that has been sent to the secure environment
- includes concepts for control (sessions, instances ..)

The TEE Internal API defines an API for
- interfacing secure code (TA:s) to the framework
  - application session control, instance, lifecycle
- STDLIB – like functionality
- Cryptographic subroutines
  - RSA
  - AES in many modes, e.g. authenticated encryption
  - Hash functions
  - randomness
- Secure storage
MTM introduction
Requirements (v.1):

- Platform Security is an **enabler**
- Required by
  - Regulatory approval (for “open” platforms)
  - IMEI lock / Subsidy lock (i.e., SIM-lock)
  - Media consumption and protection (DRM)
  - Confidential data management (user, corporation)
  - Remote Attestation (RA) / Corporate access (VPN)
  - Application authentication, authorization, accounting
  - Reliable PKI (key management, usage, etc)
- But also for
  - Device stability
  - Malware protection
  - General trustworthiness of the platform
  - Theft and copy “management”

**MTM1 use cases (2005)**

- Platform and/or Application Integrity
- Device Authentication
- Robust DRM Implementation
- SIMLock / Device Personalization
- Secure Channel between Device and UICC
- Secure Software Download
- Device Owner Data Protection and Privacy
- Mobile Ticketing / Mobile Payment

---

In TCG / TPM, by tradition, **user control** and **user privacy** is in focus. In the “mobile” domain these considerations are important, but not in dominance!
The 3(4) main targets of MTM

1) Allow MTM to be implemented using a legacy trust environment (TEE)

2) Add secure boot. Authenticated boot is not enough for the majority of the MTM v1 use cases

3) Make the MTM footprint small. [Consequence of (1)]

An evident requirement is to maintain as much of TPM compatibility as possible. PC/Mobile convergence in essence motivated the whole MTM work.
MTM v1 overview

- Mandates only core functionalities of TPMv1.2: [target: small size]
  - Binding and sealing
  - Signing and key certification
  - Attestation (AIK may be fixed)
    -> but delegation, migration, DAA, memory services are at large optional

- MTM v1 adds:
  [targets: integrator/manufacturer control, deployability]
  - Secure boot (wrong measurement -> boot is aborted)
  - (SW) Functionality rather than HW. Device binding through roots of trust
  - The concept of MTM instances. For stakeholders: Integrator, Operator, User
MTM vs. TPM commands

**TPM_Init**, **TPM_Startup**, **TPM_SaveState**, **TPM_SelfTestFull** **TPM_ContinueSelfTest** **TPM_GetTestResult**.
**TPM_SetOwnerInstall**, **TPM_OwnerSetDisable** **TPM_PhysicalEnable** **TPM_PhysicalDisable**,
**TPM_PhysicalSetDeactivated** **TPM_SetatTempDeactivated** **TPM_SetOperatorAuth**
**TPM_TakeOwnership**
**TPM_OwnerClear**, **TPM_ForceClear** **TPM_DisableOwnerClear**.
**TSC_ResetEstablishmentBit**, **TPM_GetCapability**
**TPM_SetCapability** **TPM_GetCapabilityOwner**
**TPM_GetAuditDigest** **TPM_GetAuditDigestSigned** **TPM_SetOrdinalAuditStatus** **TPM_FieldUpgrade**,
**TPM_SetRedirection**, **TPM_ResetLockValue**, **TPM_Seal**, **TPM_Unseal**
**TPM_UnBind**
**TPM_CreateWrapKey**, **TPM_LoadKey2**, **TPM_GetPubKey**
**TPM_Sealx** **TPM_CreateMigrationBlob**
**TPM_AuthorizeMigrationKey**
**TPM_MigrateKey**
**TPM_CMK_ApproveMA** **TPM_CMK_CreateKey** **TPM_CMK_CreateTicket** **TPM_CMK_CreateBlob**
**TPM_CMK_ConvertMigration**, **TPM_CreateMaintenanceArchive**
**TPM_LoadMaintenanceArchive**, **TPM_KillMaintenanceFeature**
**TPM_LoadManuMaintPub**, **TPM_ReadManuMaintPub**, **TPM_SHA1Start**, **TPM_SHA1Update**
**TPM_SHA1Complete**, **TPM_SHA1CompleteExtend**
**TPM_Sign**, **TPM_GetRandom**
**TPM_StirRandom**, **TPM_CertifyKey**, **TPM_CertifyKey2**
**TPM_CreateEndorsementKeyPair**
**TPM_CreateRevocableEK**, **TPM_RevokeTrust**, **TPM_ReadPubek**
**TPM_OwnerReadInternalPub**
**TPM_MakeIdentity**, **TPM_ActivateIdentity**, **TPMExtend**
**TPM_PCRRead**, **TPM_PCRReset**
**TPM_PCR_Reset**, **TPM_PCR_Reset**, **TPM_ChangeAuth**, **TPM_ChangeAuthOwner**, **TPM_OIAP**
**TPM_OIAP**, **TPM_OSAP**, **TPM_DSAP**
**TPM_SetOwnerPointer**, **TPM_Delegate_Manage**
**TPM_Delegate_CreateKeyDelegation**
**TPM_Delegate_CreateOwnerDelegation**, **TPM_Delegate_LoadOwnerDelegation**
**TPM_Delegate_UpdateVerification**, **TPM_NV_DefineSpace**
**TPM_NV_WriteValue**, **TPM_NV_WriteValueAuth**
**TPM_NV_ReadValue**, **TPM_NV_ReadValueAuth**
**TPM_KeyControlOwner**, **TPM_SaveContext**, **TPM_LoadContext**
**TPM_FlushSpecific**, **TPM_GetTicks**, **TPM_TickStampBlob**
**TPM_EstablishTransport**, **TPM_ExecuteTransport**
**TPM_ReleaseTransportSigned**, **TPM_CreateCounter**
**TPM_IncrementCounter**, **TPM_ReadCounter**
**TPM_ReleaseCounter**, **TPM_ReleaseCounterOwner**
**TPM_DAA_Join**, **TPM_DAA_Sign**, **TPM_EvictKey**
**TPM_Terminate_Handle**, **TPM_SaveKeyContext**
**TPM_LoadKeyContext**, **TPM_SaveAuthContext**, **TPM_LoadAuthContext**
**TPM.DirWriteAuth**, **TPM.DirRead**, **TPM_ChangeAuthAsymStart**
**TPM_ChangeAuthAsymFinish**
**TPM_Reset**
**TPM_OwnerReadPubek**, **TPM_DisablePubekRead**
**TPM_LoadKey**

Add: **MTM_InstallRIM**, **MTM_LoadVerificationKey**, **MTM_VerifyRIMCert**
**MTM_VerifyRIMCertAndExtend**, **MTM_IncrementBootstrapCounter**
MTM implementation size

- The sizes are w.o. crypto primitives (e.g. RSA)
- The size excludes platform-dependent, but sizable code parts, including upload code verification, state decryption / encryption, ..
- As a “monoblock”, the code amounts to around 20kB compiled

<table>
<thead>
<tr>
<th>Command</th>
<th>Size (bytes)</th>
<th>Command</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ChangeAuth</td>
<td>1774</td>
<td>TPM_IncrementCounter</td>
<td>514</td>
</tr>
<tr>
<td>TPM_CertifyKey</td>
<td>1606</td>
<td>TPM_FlushSpecific</td>
<td>454</td>
</tr>
<tr>
<td>TPM_CreateWrapKey</td>
<td>1534</td>
<td>MTM_VerifyRIMCert</td>
<td>432</td>
</tr>
<tr>
<td>TPM_LoadKey2</td>
<td>1430</td>
<td>MTM_IncrementBootstrapCounter</td>
<td>410</td>
</tr>
<tr>
<td>MTM_LoadVerificationKey</td>
<td>1422</td>
<td>TPM_GetPubkey</td>
<td>394</td>
</tr>
<tr>
<td>TPM_SeaI</td>
<td>1112</td>
<td>TPM_Extend</td>
<td>292</td>
</tr>
<tr>
<td>TPM_Unseal</td>
<td>980</td>
<td>TPM_ReadCounter</td>
<td>224</td>
</tr>
<tr>
<td>MTM_VerifyRIMCertAndExtend</td>
<td>898</td>
<td>TPM_OIAP</td>
<td>212</td>
</tr>
<tr>
<td>TPM_Quote</td>
<td>792</td>
<td>TPM_PCRRead</td>
<td>144</td>
</tr>
<tr>
<td>TPM_Sign</td>
<td>772</td>
<td>TPM_GetRandom</td>
<td>138</td>
</tr>
<tr>
<td>TPM_OSAP</td>
<td>580</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MTM performance with a TEE

+ Logic runs at full processor speed (332 MHz, ARM 9)
- Invocation time of the secure environment, validation of “collection” (1ms)
- Penalty caused by state protection (~10ms)  
  (further optimization possible)

<table>
<thead>
<tr>
<th></th>
<th>TPM_OIAP</th>
<th>TPM_OSAP</th>
<th>TPM_PCRRead</th>
<th>TPM_Extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRC MRTM/M-Shield</td>
<td>11.6ms</td>
<td>12.0ms</td>
<td>11.5ms</td>
<td>11.6ms</td>
</tr>
<tr>
<td><strong>Ubuntu Linux 9.04 with 2.6.24-19-generic kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atmel AT97SC3201</td>
<td>12.02ms</td>
<td>25.00ms</td>
<td>11.36ms</td>
<td>11.29ms</td>
</tr>
<tr>
<td>Atmel AT97SC3202</td>
<td>35.94ms</td>
<td>35.99ms</td>
<td>35.41ms</td>
<td>35.42ms</td>
</tr>
<tr>
<td>Broadcom BCM5755</td>
<td>24.01ms</td>
<td>24.00ms</td>
<td>23.34ms</td>
<td>23.30ms</td>
</tr>
<tr>
<td><strong>Ubuntu Linux 9.04 with vanilla 2.6.24 kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atmel AT97SC3201</td>
<td>6.01ms</td>
<td>20.94ms</td>
<td>6.11ms</td>
<td>8.05ms</td>
</tr>
<tr>
<td>Atmel AT97SC3202</td>
<td>17.99ms</td>
<td>23.99ms</td>
<td>17.26ms</td>
<td>17.06ms</td>
</tr>
<tr>
<td>Broadcom BCM5755</td>
<td>12.58ms</td>
<td>12.59ms</td>
<td>12.00ms</td>
<td>12.00ms</td>
</tr>
</tbody>
</table>
MTM device binding and configurations

- **Device boot**
  - Root-of-Trust for Verification: code integrity
    (Root-of-Trust for measurement)
  - Root-of-Trust for Storage: state integrity, rollback protection
  - Root-of-Trust for Reporting: remote attestation

**Mobile remote owner trusted module**
- Management from outside the device (manufacturer / operator)

**Transitive trust**

**Mobile local owner trusted module**
- Management at the device using TPM commands (user, owner)
MTM static data

- `counterStorageProtectId` – storage protect counter
- `counterRIMProtectId` – protect RIM certs
- `counterBootstrap` - initial boot version
- `verifiedPCRs` - PCRs only modifiable by RIM certs
- `loadVerificationKeyMethods` – (root load, integrity check root data, auth(s))
- `InternalVerification key` - key for InstallRIM - certs
- `verificationAuth` (auth. For InstallRIM)
- `loadVerificationRootKeyEnabled` *(in STANY_FLAGS)*
- `integrityCheckRootData` - hash of root verification key
- `AIK preconfigured` - (attestation key, if preconfigured)
- `SRK` - storage root key

Rollback protection
Constrained PCRs (RIM update only)
Secure boot support
Trust root
Keys
MTM verification keys

Public-key certificates, used for the “stakeholder model”, i.e. external authorization for PCR updates (secure boot) and counter increments

Validated inside the MTM, on load.

Verification key “permissions” usable when key is loaded:
- Increment bootstrap counter
- Validate other verification keys
- Validate RIM certificates
- (vendor extensions)

Verification key constraints = validated at load:
- Signed by parent key
- Counter binding
- (vendor extensions)
- Note: No PCR binding

checkRootData (RVAI)

“root” verification key

verification key

verification key

RIM certificate (attribute cert)

In persistent state

Binding, e.g. public key hash

Binding = RSA sign.
MTM RIM certificates

RIM = reference integrity metric

Attribute certificates, used for the explicit activity of constrained PCR updates (secure boot).

Validated inside the MTM, on use. The successful verification of a RIM certificate typically implies a PCR update

RIM certificate “action” = implied on successful load:
- Increment given PCR
- RIM certificate includes PCR extension value
- (vendor extensions)

RIM certificate constraints = validated at load:
- Signed by parent key
- Counter binding
- PCR (composite) binding
- (vendor extensions)
MTM secure boot overview with TEE (1)

“First”, the root of trust for enforcement sets up the MTM with the legacy TEE

1) Install (signed) code, provide “self-measurements”

2) Install data for MTM code, decrypt inside TEE

MTM NV state

Crypt for the TEE

Execution environment

Trust root

Secret

1) Install (signed) code, provide “self-measurements”

2) Install data for MTM code, decrypt inside TEE
“Second”, secure boot with MTM commences

0) Measure

1) Find the verification keys and RIM cert matching the measurement

2) Set up trust chain for validation

3) Update PCR using RIM cert.

4) On successful PCR update, launch the measured code, on error “abort”

Verification keys and RIM certificates can be stored / received on-demand, integrity guaranteed by external authorization.

Next component to boot
MTM going forward

Environment

- TEE API standardization is ongoing in Global Platform Device Committee (internal and external TEE APIs). This was not available in 2005.

- TPM specification is transitioning to TPM2

- MTM (and TPM) are not really being used – MTM is not even widely deployed like TPMv1.2.

Additional use cases and requirements have been publicised

- Most new MTM use cases have a strong legacy aspect: Banking, Vending machines, Id:s, Health care…

- Implication: There is a **need to support legacy security algorithms** for applications. MPWG has distilled this to support for (open) remote provisioning of algorithms. Execution of algorithms is also needed, as is algorithm use of MTM/TPM secrets and state, especially platform binding (PCRs)
ObC & TEEs & eSE:s & NFC
On-board Credentials

Welcome to eBanking

Welcome Karl Timo Juhani Kostilainen
Please verify your identity:
Security card number: 4600134200
Key number: 6137
Security number: 
You have 23 numbers left on your security card.
On-board Credentials architecture

Secure environments overview - mobile

Overview of the current “development state” of HW-originated platform security services – And NFC may be wave that breaks the status quo

- **Operator and SIM services**
  - OMA
  - ETSI

- **Provisioning & use, key mgmt**
  - ISO 7816

- **VM**
  - JavaCard
  - Proprietary, or GP Card Spec.
  - Proprietary

- **OS**
  - Proprietary, e.g. MultOS

- **Hardware**
  - Smart Cards / Secure Elements (SE)

- **Security enablers / service API**
  - OS APIs like PKCS#11, PC/SC
  - GP Card specification
  - GP Card spec <-> MTM2
  - GP TEE Client API
  - Proprietary (ObC) → JavaCard?
  - GP Device Committee (future spec.)
  - None, proprietary, hypervisor
  - Processor Secure Environments (TEE)

- **Service API**
  - TSS
  - TCG “TLV”

- **Function implementation varies - Smart card technology often used?**

- **ETSI**

- **OMA**

- **Operator and SIM services**
NFC and mobile secure environments

UI Applications

LLCP
R/W

NFC stack

NFC HW

Embedded SE
SIM SE
TEE

JSR-177 / PKCS#11 / MTM / PC/SC / Credential manager

SWP/HCI
... so can we make this work?
Conclusions

• Mobile phone security
  – Requirements, regulations, user expectations
  – Early adaptation of hardware security mechanisms
• Platform security architecture
  – Many features borrow or adapted. Code Signing & Permissions
  – No direct SW/HW APIs. MTM / TPM?
• Future
  – Security services (Credit Cards, Banking, Ticketing, Loyalty) are driving a new wave of interest, mostly driven by NFC
  – The intersection between HW & SW is still clearly underdeveloped