

Trusted Mobile Platforms

(a.k.a. hardware-assisted
platform security in
handsets)

Jan-Erik Ekberg, Nokia Research Center
4.11 2009

Contents

- Introduction
 - How do handsets differ from PC's
 - Business and usability motivators for security
- The TPM and MTM specifications
- [ObC – another approach to credentials]
- Conclusions

State-of-the art

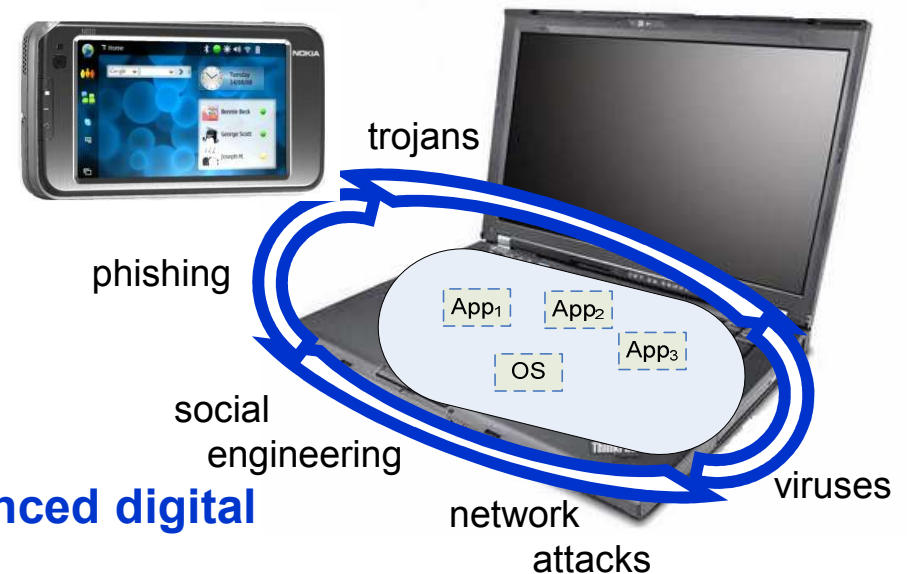
Traditional OS/system security approach is failing (always is...)

A **perimeter defence** may be appropriate for well-managed network servers, but for personal and embedded devices

- Security is not the user's primary interest
- The users' administrative talent is negligible
- The software is simply too complex to be bug-free

while users' are increasingly relying on advanced digital services for everyday use

- Internet banking, payments, and ticketing
- Electronic voting
- wireless service packages with communication cost
- communication (VoIP, messaging, e-mail ← ISDN, fax and snail mail)



The (mobile) security legacy

The business environment has for years motivated handset manufacturer's to consider security from the ground up:

- | | | |
|-------------------------------------|-----|----------------------|
| - open SW platforms | vs. | + regulation, safety |
| - 3 rd party development | | + operator binding |
| | | + user expectation |

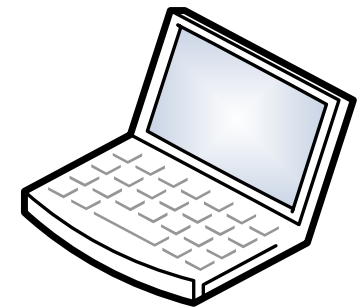
-> secure boot, mandatory access control, validated installation,
secure storage e.g. for radio parameters, secure execution environments

whereas the PC industry (esp. laptops) has endorsed the
Trusted Platform Module (TPM) chips and standard by TCG

-> trusted boot, remote attestation, user privacy, key storage



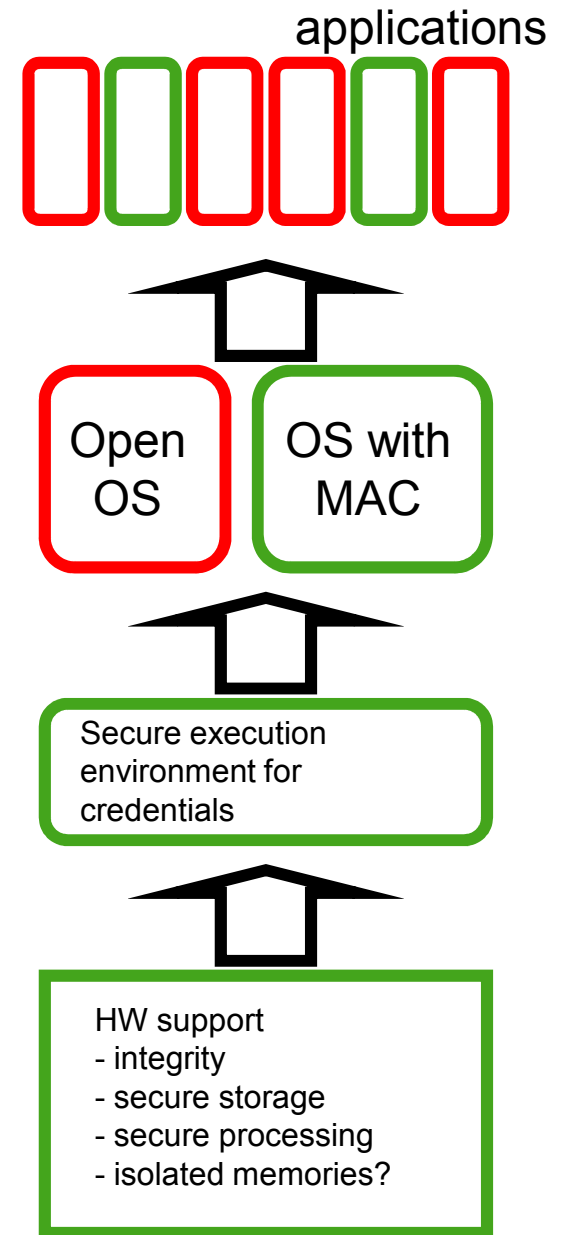
TI M-shield,
ARM TrustZone;
Symbian capabilities ...



TPM

Platform Security

- To achieve secure processing in heterogenous environments, **trust roots** are needed
- Without **enforcement**, guarantees are hard to give
- A good security infrastructure leaves room for un-trusted 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.



Security from a handset *business* perspective

- 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)
 - IMEI lock / Subsidy lock (i.e., SIM-lock)
- But also for
 - Device stability
 - Malware protection
 - General trustworthiness of the platform
 - Theft and copy “management”

Security from a handset *usability* perspective

- Starting point:
 - 3rd party applications cannot be trusted.
 - Not even the company's "own" applications
- **Applications** want to feel free and have **all the services / resources in their use**. Environment should be "open" (Java, VMs).
- **Users should not be disturbed** – they will not, and should not have to care. Also, compared to PC users, handset users are on average far less computer-educated, and are hampered by the limited (size) UI.
- **Development and distribution of apps** should be easy for "hobbyists".
- **Legacy systems and applications** have to be supported due to overall digital convergence (e.g. FAT filesystem on memory sticks, WLANs ..)
- For some / many services, the **malicious adversary is the owner/user** of the device.

Hardware security features (in mobiles)

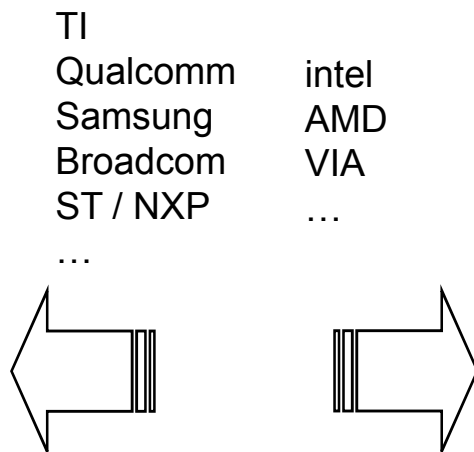
Processing in embedded devices (typical)

Embedded

Processor with
ARM 7/9/11/Cortex A8
core(s)

Often mostly a SoC with
- 3G/GSM digital logic cores
- I/O support
- DSPs

Custom ASIC:s are common,
the integrator is part of the
design process.



P C

X86-compatible processor

Powerful auxiliary chipset
(North/Southbridge, TPM,
GPU, ...)

“The processor” is standard
w.r.t. the integrator -> similarly
behaving products

Power management features are **typically** emphasized more on the embedded side

Processor clock speeds are **typically** 3-10 times higher on the PC side

(but in practice the digital convergence is eating away the differences and fast)

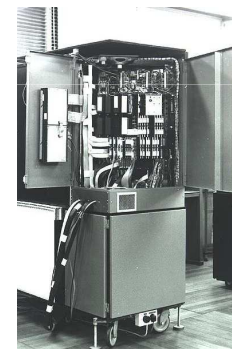
A plethora of security solutions in the embedded space

- Handsets have required HW security (due to openness) since around 2001
- Even before that, embedded controllers had security needs in industrial setting.
- Since the “penalty” of adding a new ASIC/chip only for security is high both in terms of cost, battery and real-estate, SoC- security solutions dominate.



Architecting a “ring-1” – solution for security within the processor is not a new one.

- Multics 6180 (sec. HW extensions) (1972)
- The Cambridge CAP computer (ca 1976)



The CAP computer
(Wikimedia)

TrustZone[®]
Security Foundation by ARM[®]

PadLock
VIA



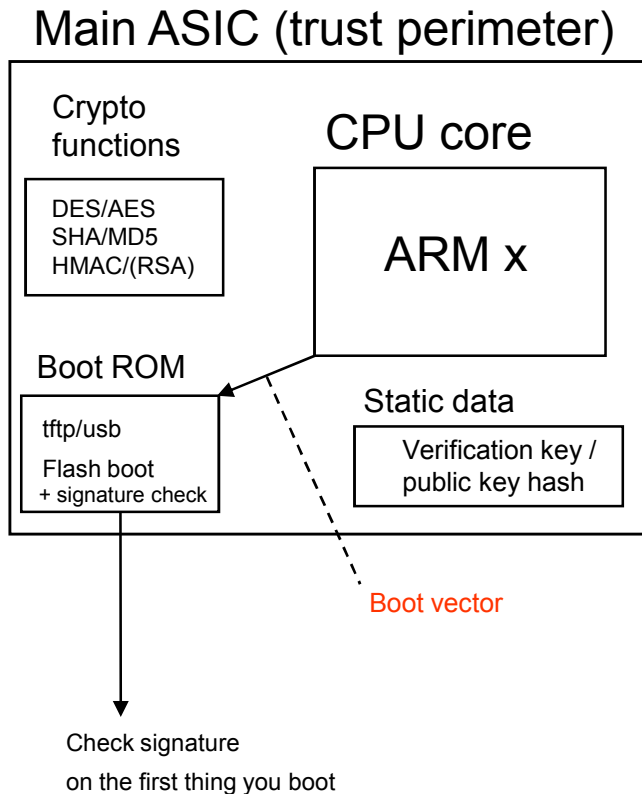
SecureMSM[™]
(QualComm)

To introduce the “typical” processor security setup, consisting of

- a) Secure boot
- b) Device secrets
- c) a trusted execution environment
(sometimes achieved by virtualization)

let's look at the security mechanisms in an incremental fashion ...

The absolute basics (“accelerator” solution)



Assuming you have

1. on-chip ROM and
2. point the boot vector to it and
3. have the possibility to store a public key and
4. have the sign. verification algorithm handy

simply enforce signature checking on the first piece of code you stumble upon. Fail = abort.

Important point: no secret info!!!

(The ASIC typically also have other cores related to e.g. communication, but these are ignored here)

Come the eFUSES...

- Main ASICs do not typically have enough
 - voltage or
 - silicon layersto contain flash storage (re-writable non-volatile storage)

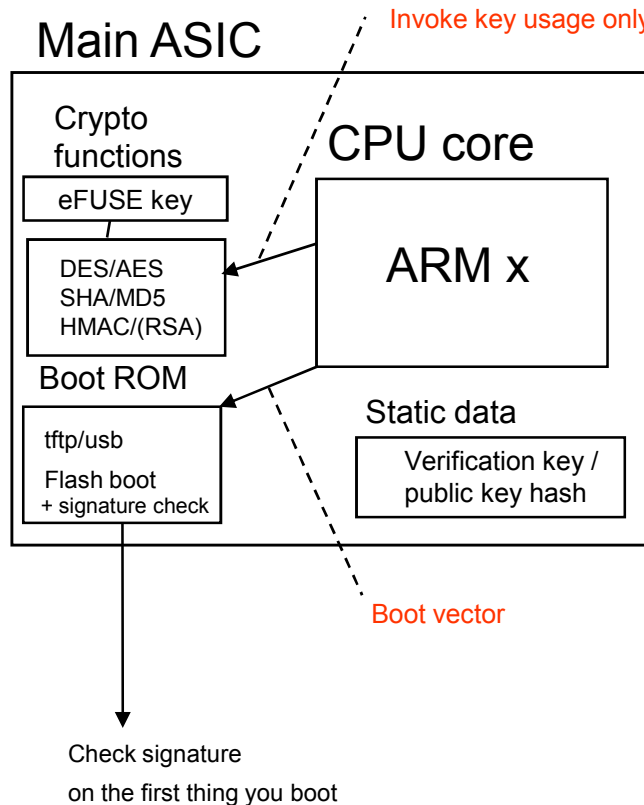
However, there is a technology originally by IBM called eFUSES.

An eFUSE behaves exactly like a fuse in an electrical circuit, in can be “blown” by a suitable SW invocation, and the value can thereafter be read.

- An array of, say 128 e-fuses forms a 16-byte value
- These values can be programmed at a factory
- These values fit on a 4-layer(?) ASIC and need no energy pumps
- The value could e.g. hold a **SECRET** key!

(See <http://en.wikipedia.org/wiki/EFUSE>)

The next-to-trivial (“hide the key” solution)



If you have a secret, it is of no value if everybody can read it.

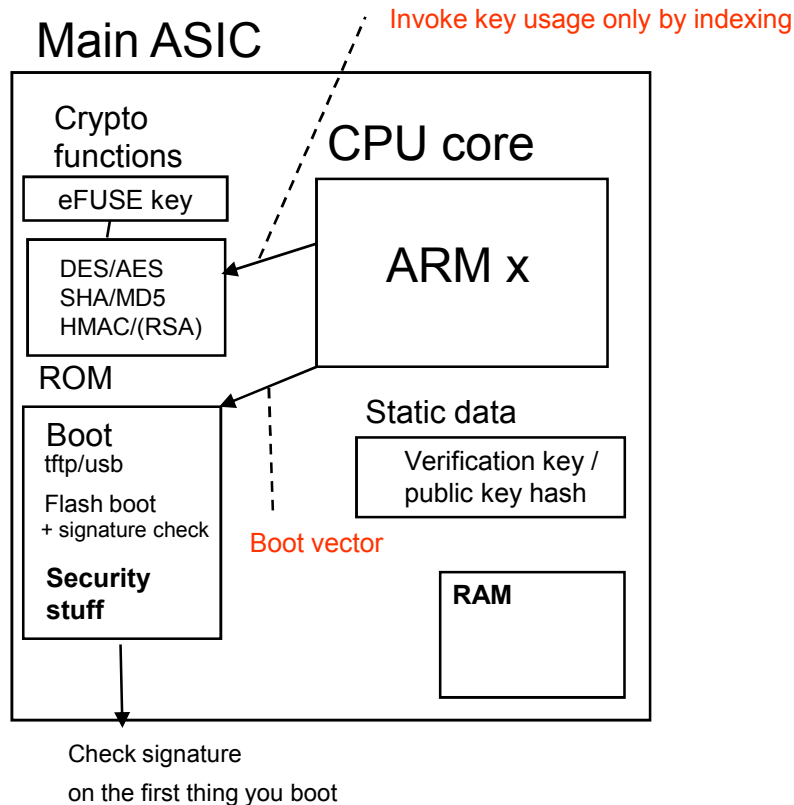
However, given an accelerator

- as a **HW implementation**
- **directly wired** to the crypto

we can transform the public information into an oracle, which is much better

... and all kinds of semi-weird access control protection system can be hardcoded in the accelerator

Anything more already requires some thought



A larger context is best explained in context of ARM TrustZone, but that architecture is by no means the first

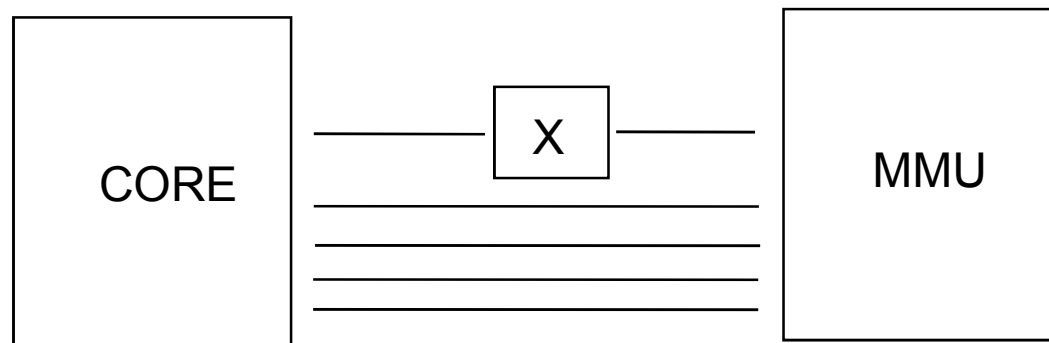
We add some ROM and RAM with the intention to make a secure execution environment. Should we make this only visible to processor privileged mode or what???

No, not good enough...

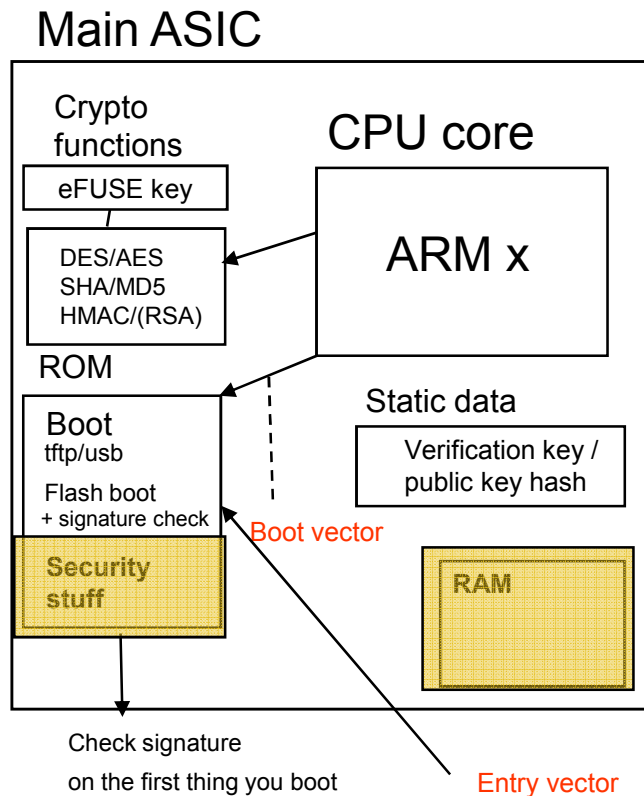
- OS:es are also attacked (vulnerabilities)
- DMA is a problem, interrupt contexts also

So we add a new processor mode for accessing secure memory. How?

- Memory is typically addressed
- The address space could e.g. easily be extended by one more “HIGH” bit
- Maybe the MMU will only accept the “HIGH” bit when we are secure
- Maybe the secure ROM and RAM have addresses with the “HIGH” bit set
- If the processor core is not in our control (read ARM), then ‘x’ can be controlled by external logic on the memory bus ..



... and how do we control the access to set the new “secure mode”?



Let's define a unique entry point

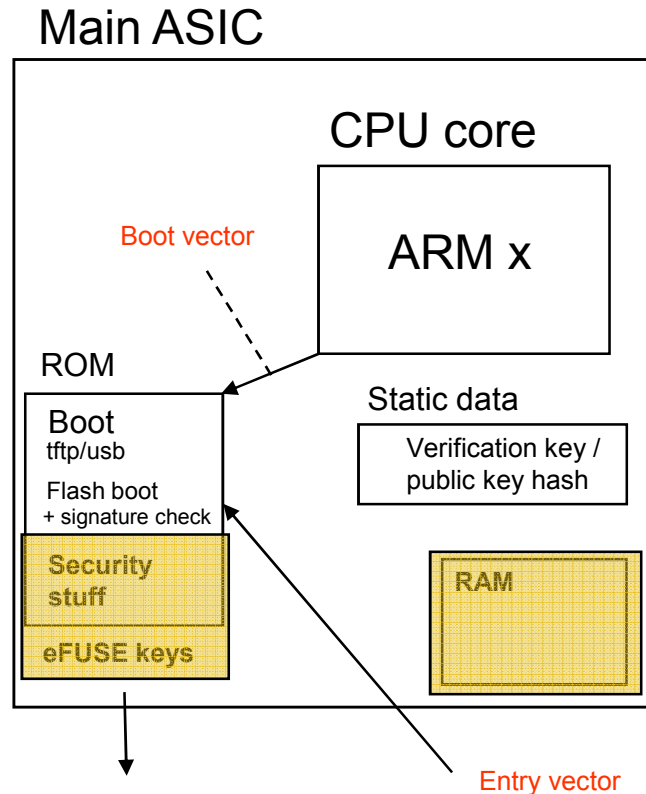
- A new interrupt
- A memory address on on-chip ROM

On entry some magic happens

- we check **entry constraints**
 - **interrupts off?**
 - **caches flushed?**
 - ...
- On the secure side, we enforce a function interface (API), maybe we can even upload signed code?

And only then we can access our ***precious, precious*** memory. We are in a ***Trusted Execution Environment (TrEE)***

So we end up in a rough architecture like



And with this it is up to the integrator

- To decide ROM contents →
- What is the RAM used for, and how

Additionally

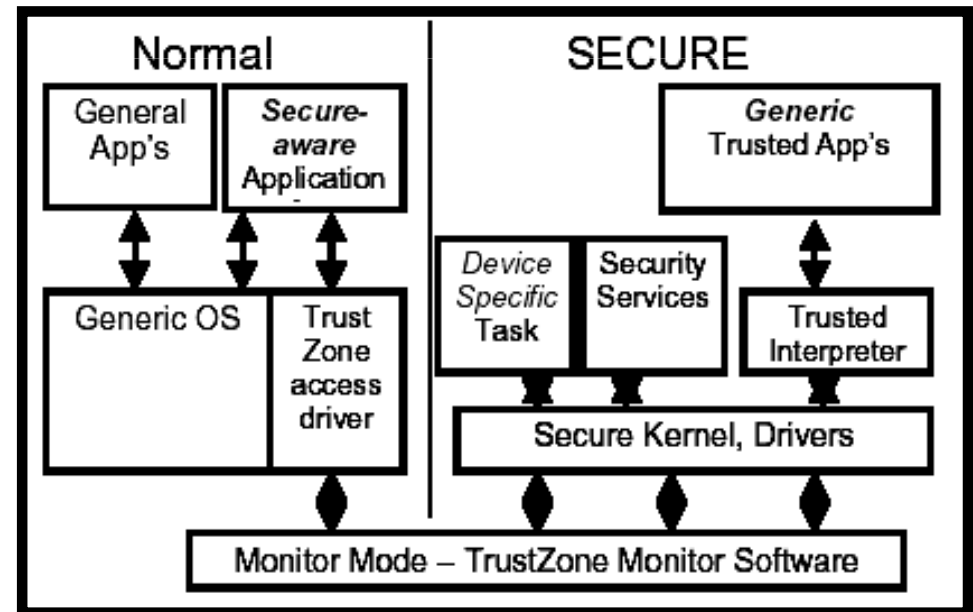
- the interrupts can be re-enabled also in the “closed place”
- there can be versioning, code uploads to RAM ...

As always there are (research) problems. Of types manufacturers rarely wish to talk about.

In practice the same design is a recurring theme, at least, say for systems like M-shield and TrustZone (and even TPM Late Launch is close in principle)

HW security solutions wrap-up

- Trustworthy software security is solely based on secure hardware services
- A whole core cannot typically be dedicated to security (cheaply enough)
- Core-external add-on providing the services
 - E.g., OMAP1710++ (TI- M-shield), or
- Integrated into the core
 - E.g., ARM TrustZone (ARM1176 / OMAP25xx) - based solutions
- Features:
 - Verifies the boot image before loading it
 - Provide a trusted “monitor” for validating entry into a secure execution environment
 - Provides some limited non-volatile storage for permanent keys, hashes etc.
- Provides basis for, e.g.,
 - Monitor runtime integrity
 - Manage cryptographic keys
 - Commit all actions requiring the use of private keys
 - Secure storage
 - A very secure execution environment for sensitive logic

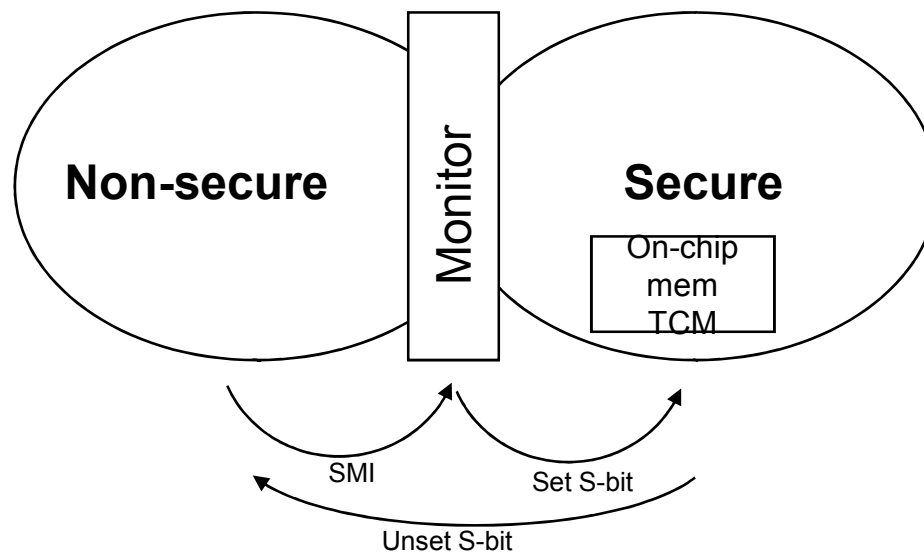


Briefly about ARM TrustZone architecture

http://www.arm.com/pdfs/DDI0301D_arm1176jzfs_r0p2_trm.pdf

The S bit

- When the S-bit is 0, the system (core) is in the “non-secure world”
- When the S-bit is 1 OR when the system is in the monitor mode, we are in the “secure world”




The Monitor mode

Thumb state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined	Secure monitor
r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und	SP_mon
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und	LR_mon
PC	PC	PC	PC	PC	PC	PC

Thumb state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und	SPSR_mon

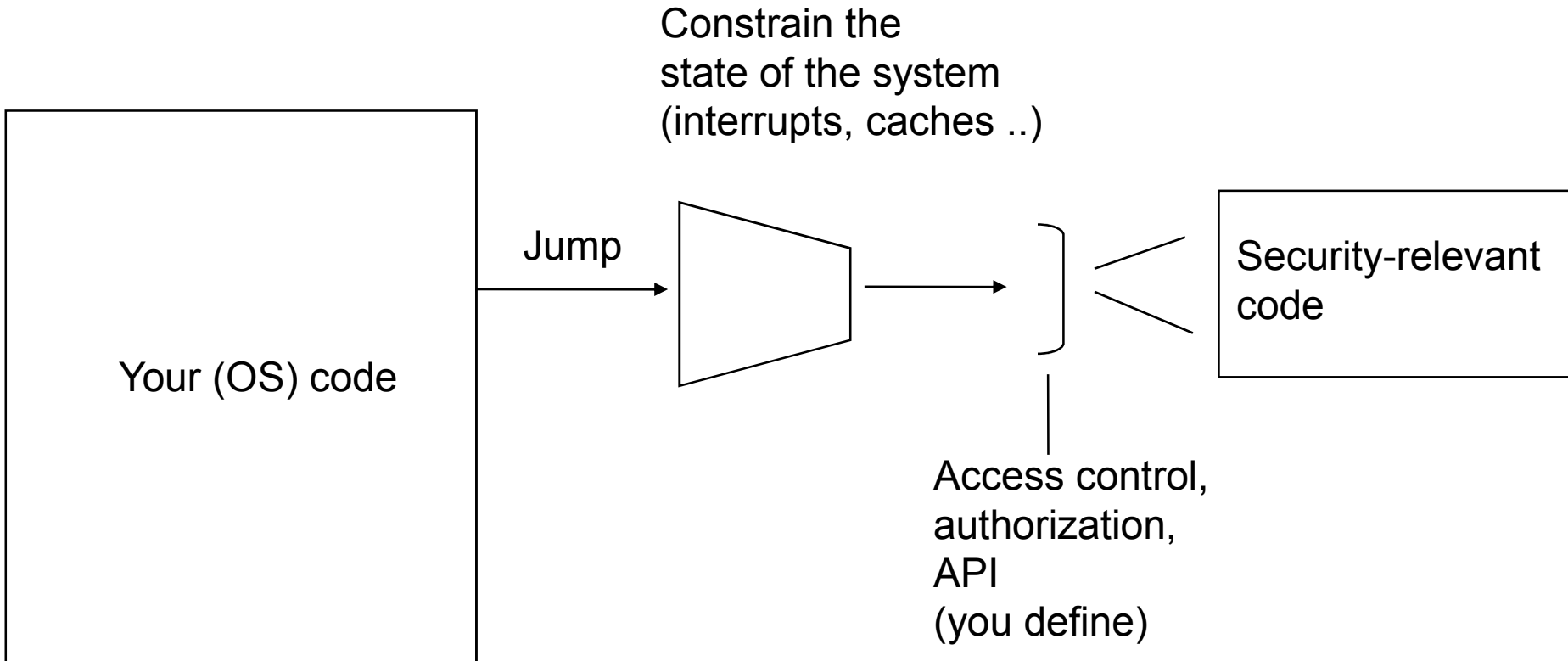
 = banked register

- The SMI interrupt will
 - Store the return address
 - Set the monitor mode
 - Disable interrupts, aborts.
Switch to ARM mode (from Thumb, Jazelle)
 - Pass a 16-bit parameter to the monitor
 - PC = Monitor_Base_Address + 0x00000008

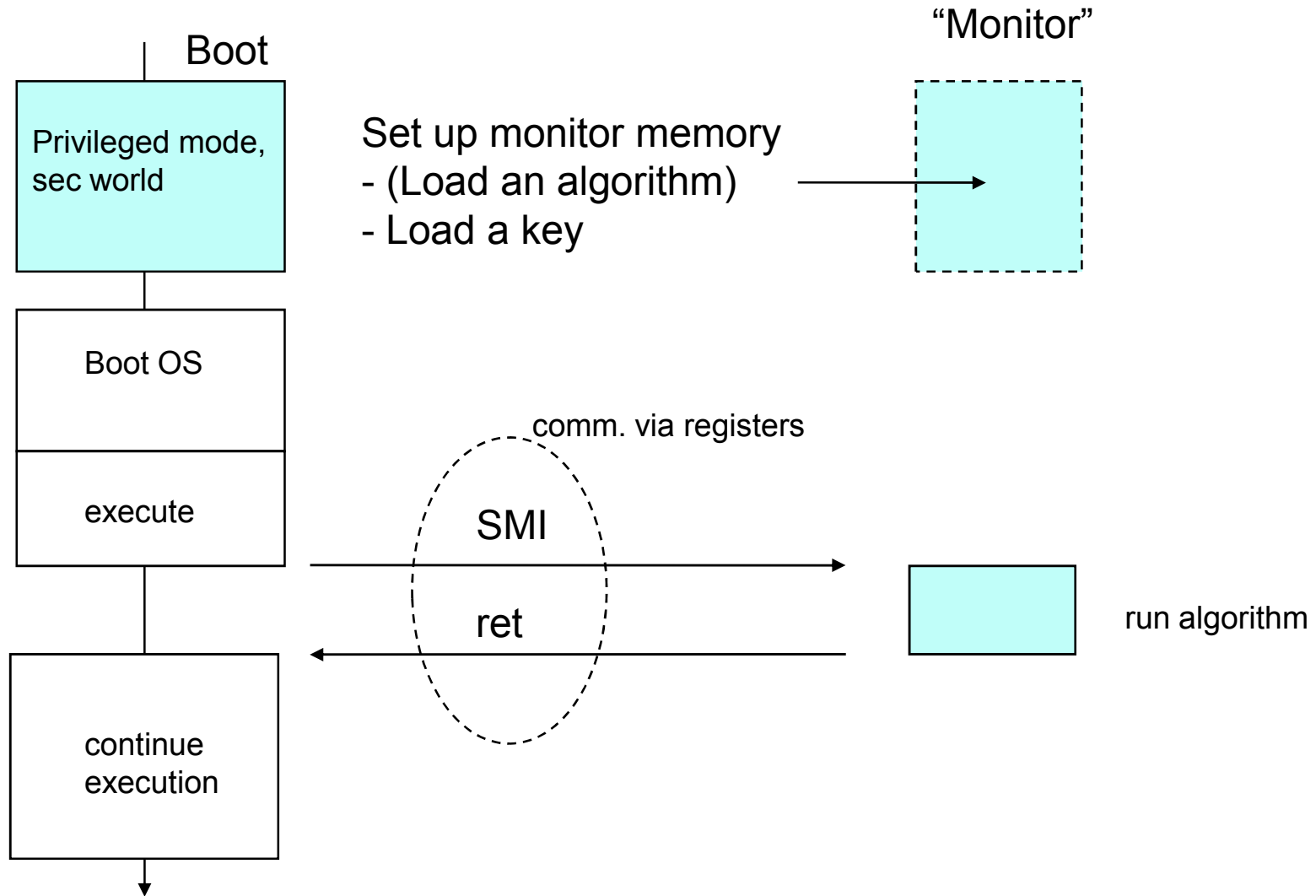
In the secure world privileged modes, you can switch to monitor mode at will (e.g. MSR instruction), in all privileged modes you can use the SMI (Software Monitor interrupt) to get there.

Only in Monitor mode may the S-bit be set.

Once more, the principle



TZ - Example 1



Other TZ stuff ...

- A (Not) secure bit is added for the MMU page table entries.
 - > The MMU mapping will hide some parts of the memory
- as well as a “edit right” bit for the page table (line) itself
- The same information is duplicated in the cache (TLB)
- The AxProt signal (NS/S) is also visible for external logic (on the internal processor bus)
- existing interrupts can be configured to enter Secure Monitor
- The DMA controller has TZ additions constraining DMA transfers based on NS/S

Trusted Mobile Platforms

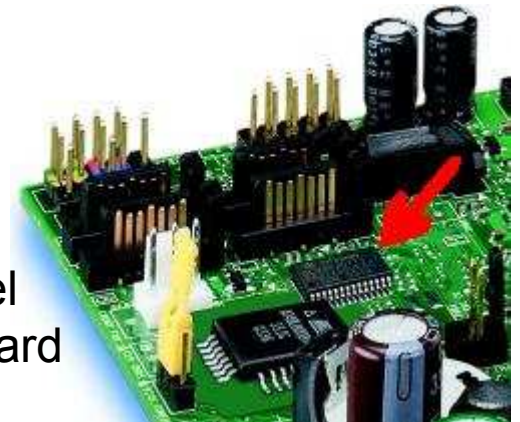
Jan-Erik Ekberg, Nokia Research Center
7.11 2007

TPM (introduction)

See www.trustedcomputinggroup.org for specification

What is a Trusted Platform Module?

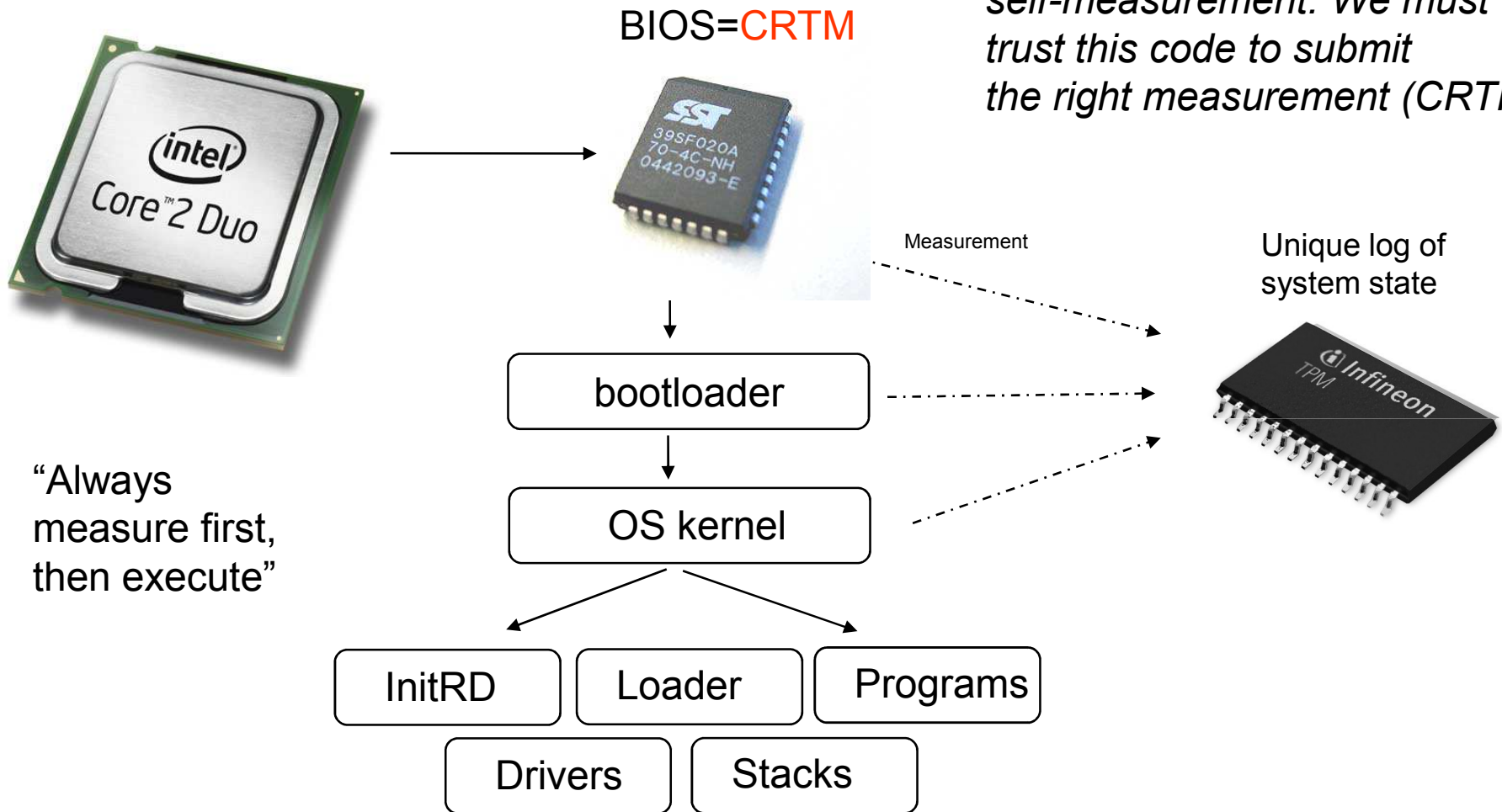
- TPM is a separate security chip soldered on the computer motherboard (most if not all business laptops have it already)
- TPM provides the system with security services,
 - Trusted boot
 - Sealing
 - Binding
 - Secure storage
 - Remote Attestation
 - (True) random number generator
- TPM requires CRTM support (Core Root of Trust for Measurement)
- First wide-scale use case: Windows Vista Bit-locker
- Standardized by the Trusted Computing Group



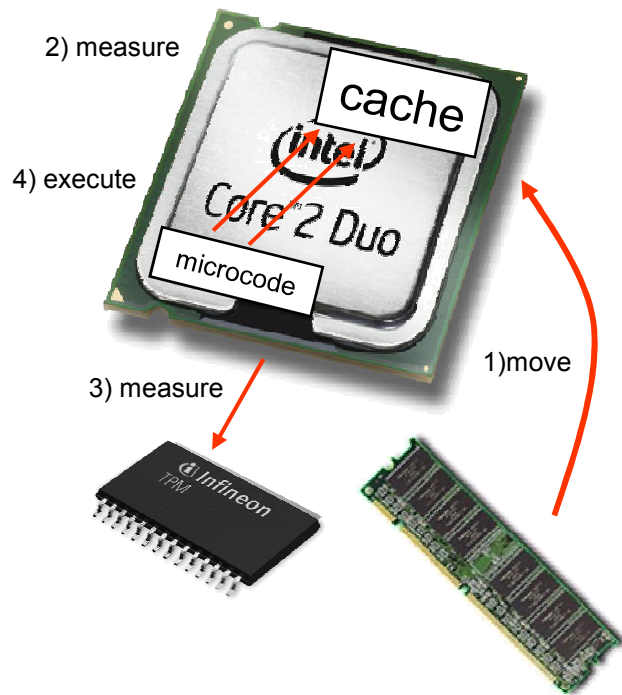
Apple Intel motherboard

TPM (trusted boot principle)

The very first measurement is a self-measurement. We must thus trust this code to submit the right measurement (CRTM)

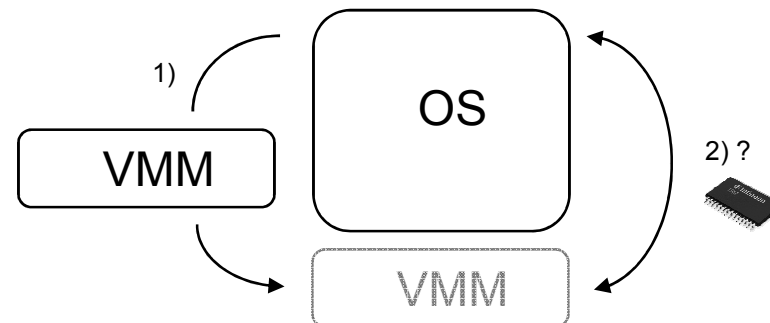


TPM – Dynamic root of trust



During execution the (Intel/AMD) processor can

- Be given a piece of (VMM) code
- Processor disables security-critical activities (multi-core, interrupts..)
- Code is
 - put into cache (processor local storage)
 - measured
 - measurement stored in TPM
 - executed



Platform configuration registers (PCRs)

- All system measurements are not stored in a list, but in one of 16 (32) or more **PCRs**.
- A PCR update (extend) to a PCR x with a new value *newval* consists of the operation

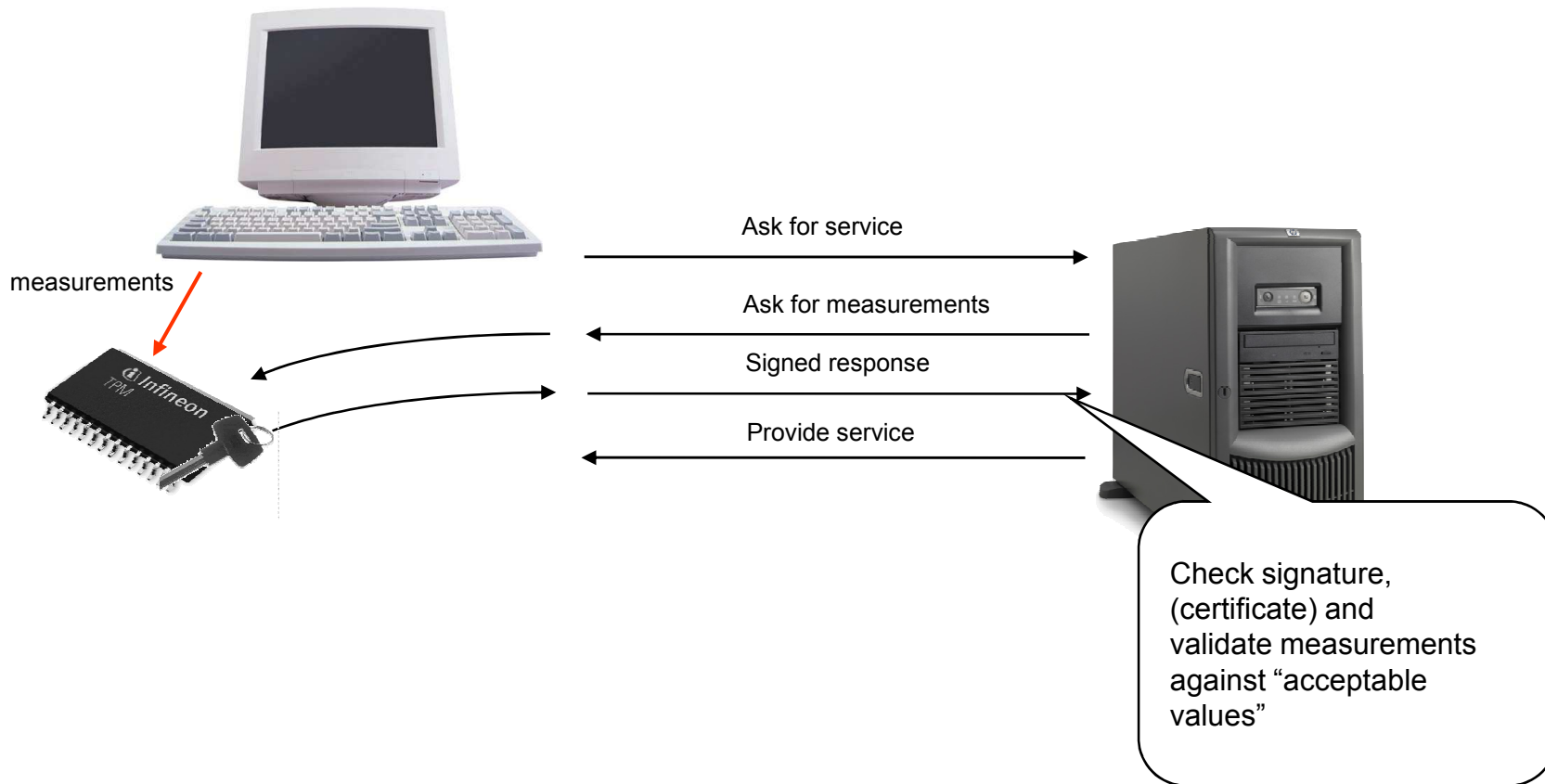
$$x' = \text{SHA-1}(\text{oldval}(x) \parallel \text{newval})$$

- The initial state of a PCR register is either 0x000...000 or 0x1111...111
- The system (OS) keeps a list of all extends (*newval*:s).
- If one knows the list of *newval1*, *newval2*, *newval3* ... and the TPM states the current value of the register to be x''

→ then

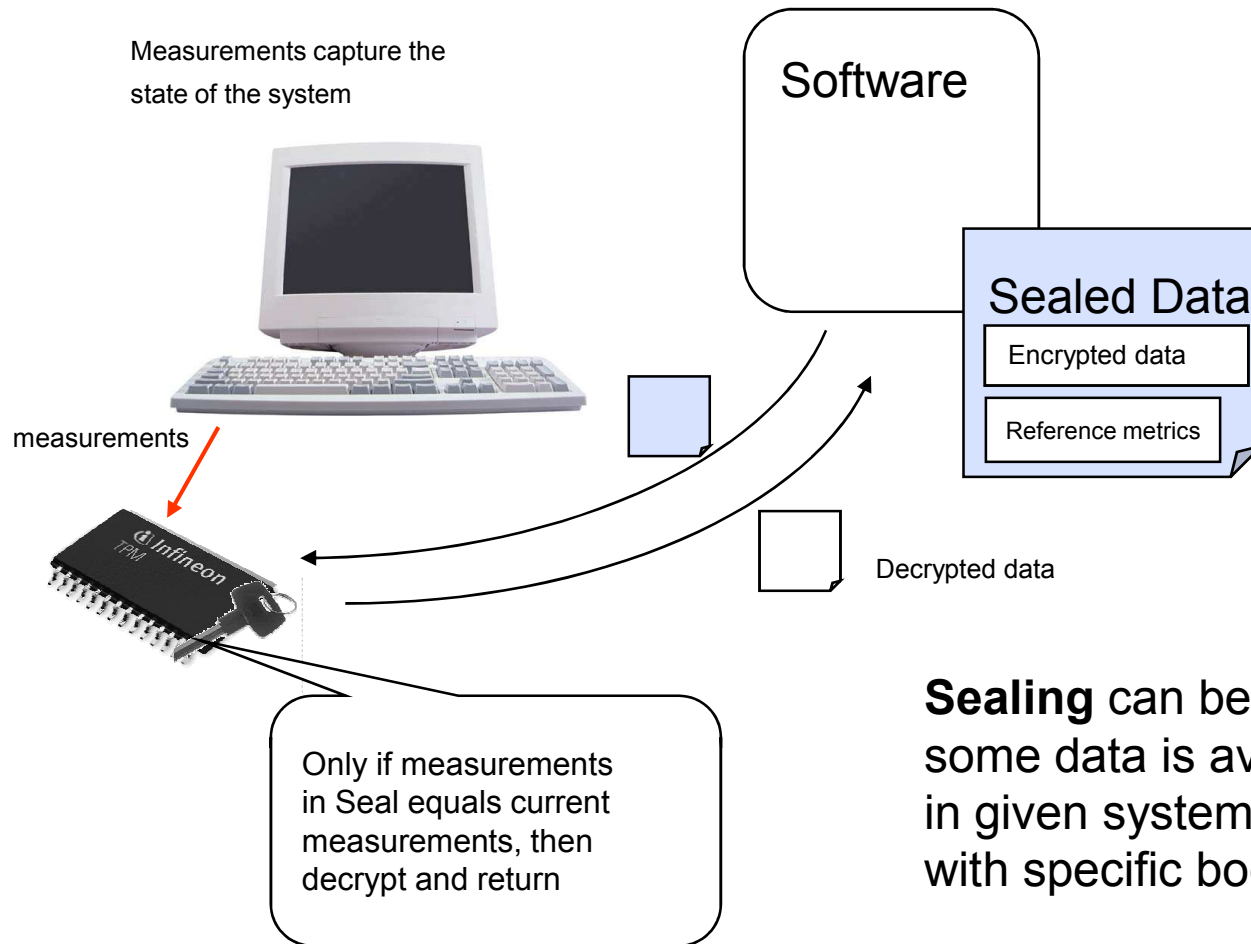
TPM use 1: Remote attestation

Measurements capture the state of the system



Remote attestation provides a means for an external party to validate the software state of a computing device

TPM use 2: Sealing



Sealing can be used to guarantee that some data is available to software only in given system states (e.g. when booting with specific bootloader and OS)

Binding is like sealing, but relates to key use for RSA key-pairs generated in TPM

Mobile TPM = MTM

www.trustedcomputing.org/groups/mobile

MTM In Brief

- Supports (mandates) most core functionalities of TPMv1.2:
 - Binding and sealing
 - Signing and key certification
 - Attestation

-> but delegation, migration, DAA, memory services are at large optional
- MTM adds
 - Secure boot (wrong measurement -> boot is aborted)
 - (SW) Functionality rather than HW
 - The concept of MTM instances

Another way to look at the difference TPM v.s. MRTM

TPM v.1.2 (max)

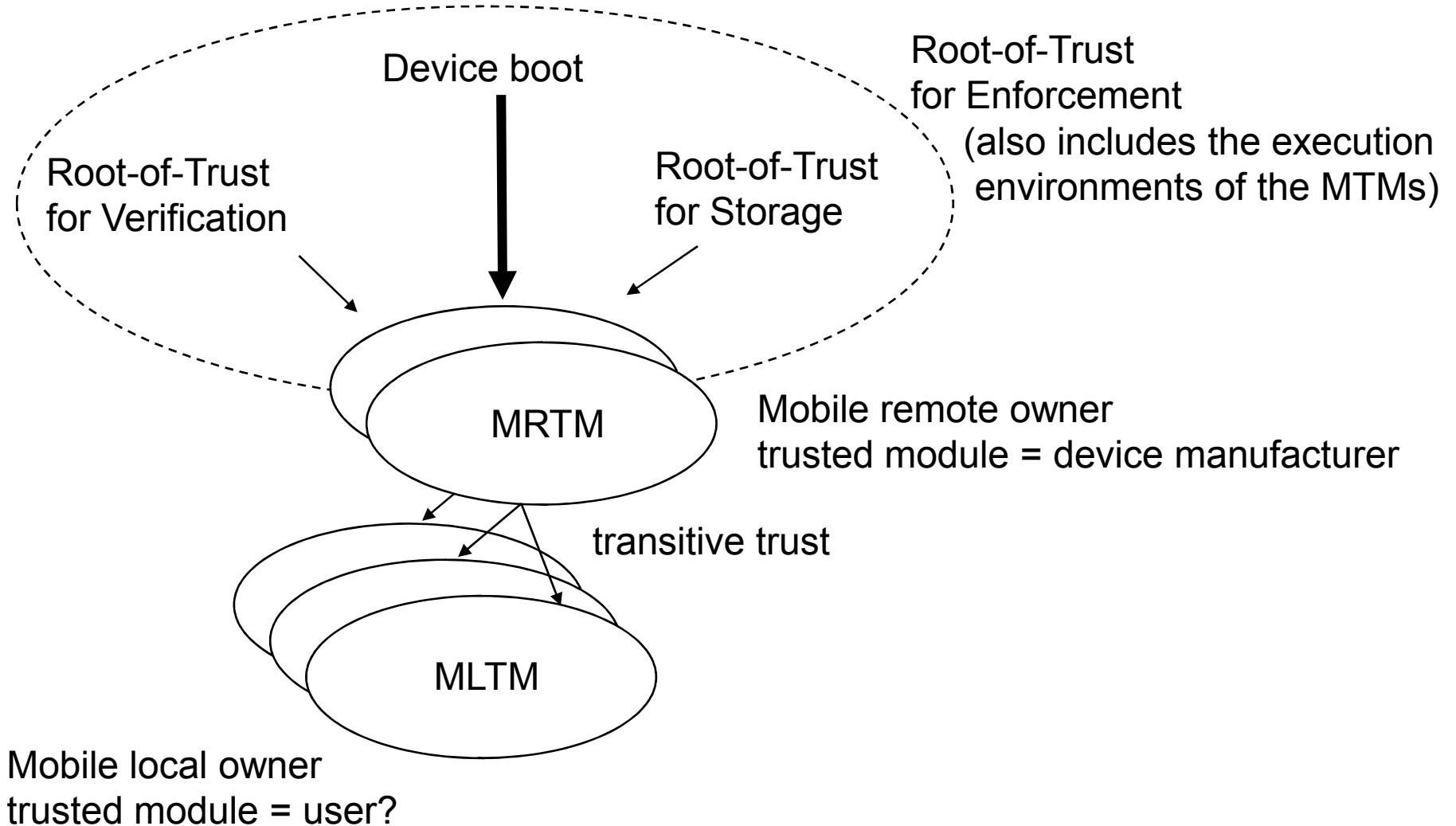
TPM_Init, TPM_Startup, TPM_SaveState, TPM_SelfTestFull TPM_ContinueSelfTest TPM_GetTestResult.
TPM_SetOwnerInstall, TPM_OwnerSetDisable TPM_PhysicalEnable TPM_PhysicalDisable,
TPM_PhysicalSetDeactivated TPM_SetTempDeactivated TPM_SetOperatorAuth TPM_TakeOwnership
TPM_OwnerClear, TPM_ForceClear TPM_DisableOwnerClear. TPM_DisableForceClear TSC_PhysicalPresence
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_ConvertMigrationBlob,
TPM_AuthorizeMigrationKey, TPM_MigrateKey, TPM_CMK_SetRestrictions, 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, **TPM_Sign**, TPM_GetRandom, TPM_StirRandom, **TPM_CertifyKey**,
TPM_CertifyKey2, TPM_CreateEndorsementKeyPair, TPM_CreateRevocableEK, TPM_RevokeTrust,
TPM_ReadPubek, TPM_OwnerReadInternalPub, TPM_MakeIdentity, TPM_ActivateIdentity, **TPM_Extend**,
TPM_PCRRead, **TPM_Quote**, TPM_PCR_Reset, TPM_Quote2, TPM_ChangeAuth, TPM_ChangeAuthOwner,
TPM_OIAP, **TPM_OSAP**, TPM_DSAP, TPM_SetOwnerPointer, TPM_Delegate_Manage,
TPM_Delegate_CreateKeyDelegation, TPM_Delegate_CreateOwnerDelegation,
TPM_Delegate_LoadOwnerDelegation, TPM_Delegate_ReadTable, 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,
~~optionally with migration, locality~~ TPM_OwnerReadPubek, TPM_DisablePubekRead, TPM_LoadKey

add: **MTM_InstallIRIM**, **MTM_LoadVerificationKey**, **MTM_LoadVerificationRootKeyDisable**, **MTM_VerifyRIMCert**,
MTM_VerifyRIMCertAndExtend, **MTM_IncrementBootstrapCounter**, **MTM_SetVerifiedPCRSelection**

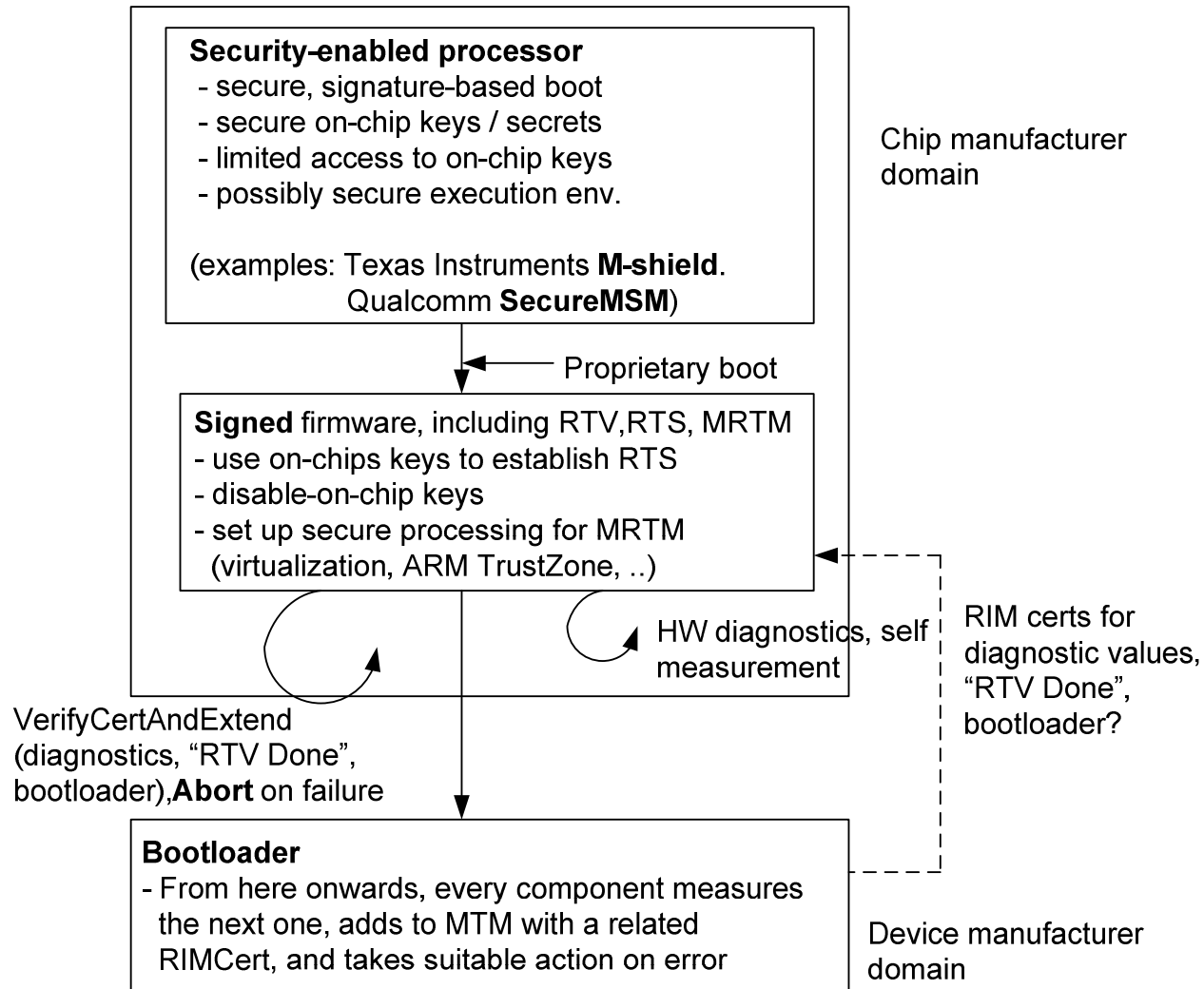
Use cases according to specification (2005):

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

MTM as an architectural component



The boot-up



Static data in MTMs

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

PCR info allocation

- PCR 0: HW Platform
- PCR1: Roots-of-Trust
- PCR2: Engine load events
- PCR3-6: MRTM proprietary measurements (platform, code-specific)
- PCR7: OS measurement
- PCR8-: “Platform specific”

The reference architecture defines an event language for facilitating interoperability, e.g.,

Diagnostic: “MRTM1”:”/boot/mrtm.bin”:**0234B8269CC672EF27352**
(engine) (object) (image)

RIM certificate (Reference Integrity Metric)

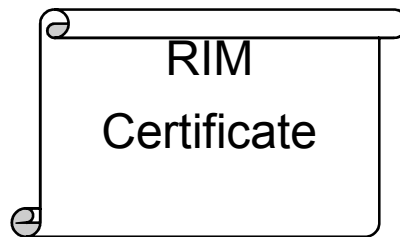
Constraints:

Counter value

- bootstrap
- RIMProtect

PCR composite value

Verification key
(authorization)

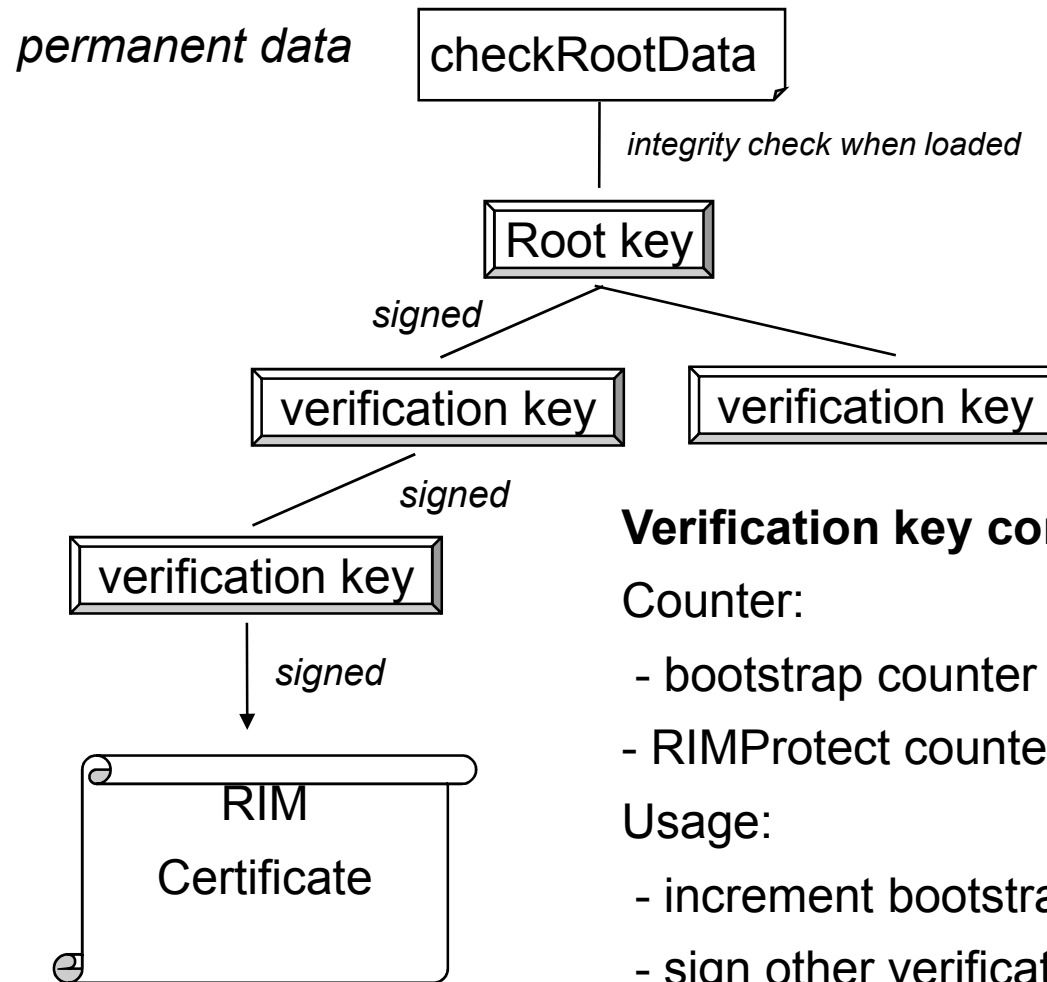


Target:

Measurement value
to be added to given
PCR

Signed by
a verification key

Verification keys



Verification key constraints:

Counter:

- bootstrap counter
- RIMProtect counter

Usage:

- increment bootstrap counter
- sign other verification keys
- sign RIM certificates

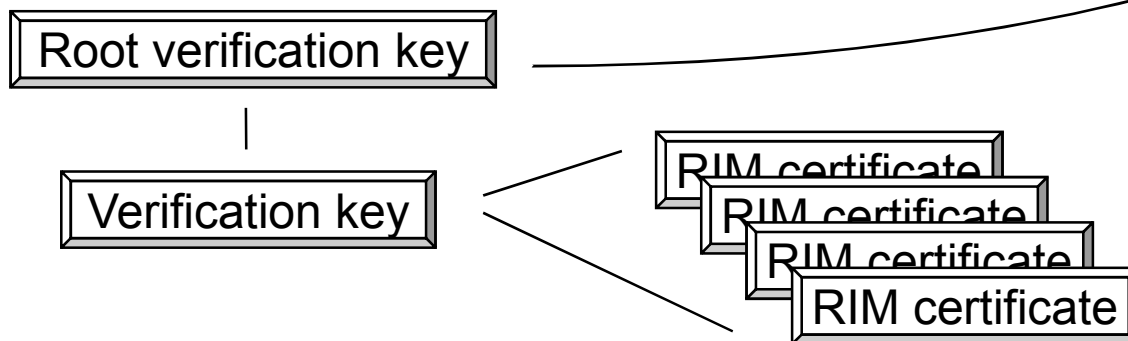
(+ a possibility for vendor extensions)

Example setup (MRTM)

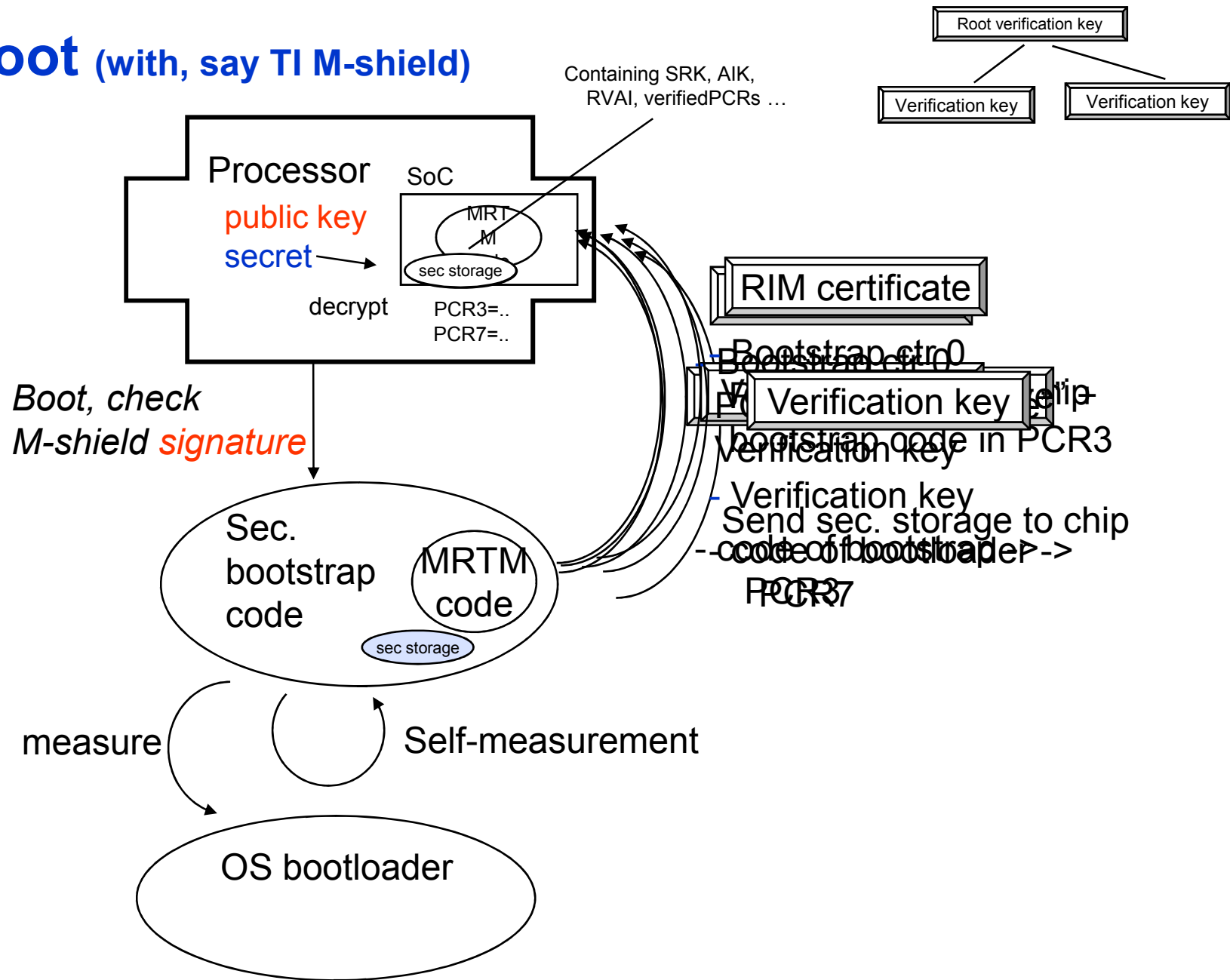
Fixed data

- verifiedPCRs {0-3}
- loadVerificationKeyMethods {root data, auth:s}
- integrityCheckRootData {0x7817..} hash of root verification key
- *loadVerificationRootKeyEnabled = FALSE*
- *Bootstrap counter value 0*
- AIK {0xAAE824..}
- SRK {0x9837923..} factory preset

Fixed data on storage:



Boot (with, say TI M-shield)

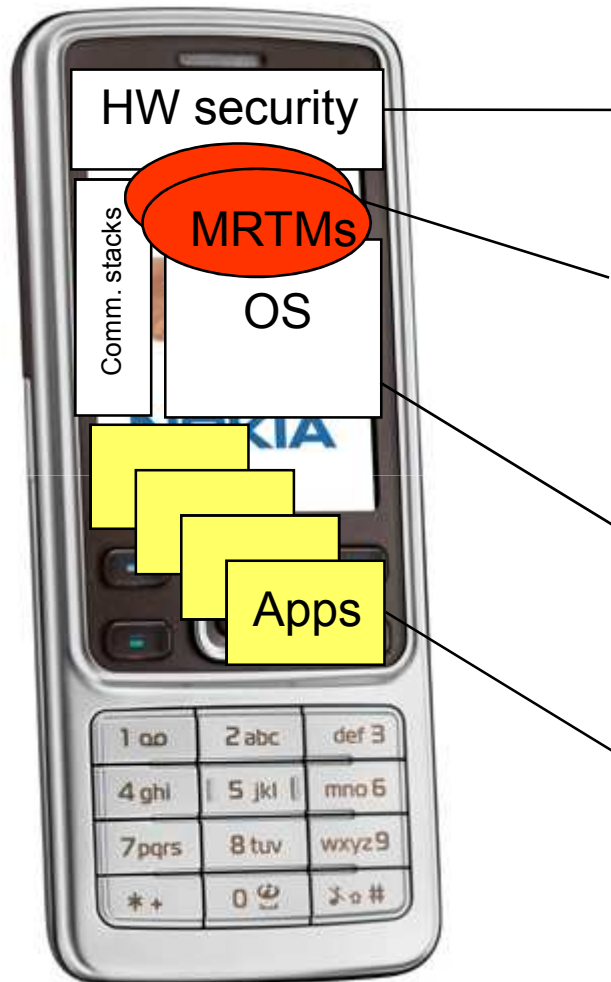


Additional MTM components

- Certificate (revocation) lists for both RIM certificates and verification keys validity lists -> whitelist
- Lists of MTMs that are to be started at boot (error -> abort)
(a local owner may also have MLTM as part of secure boot)
- External and Internal RIMs

- Counters
 - The Bootstrap counter is only guaranteed for 32 steps
 - The RIMProtect counter is only guaranteed for 4096 steps
 - The security level of the counters is more or less unspecified

How do the pieces fit together in an embedded device?



Chip-vendor specific solutions will dominate (secure boot, secure storage, secure execution?)

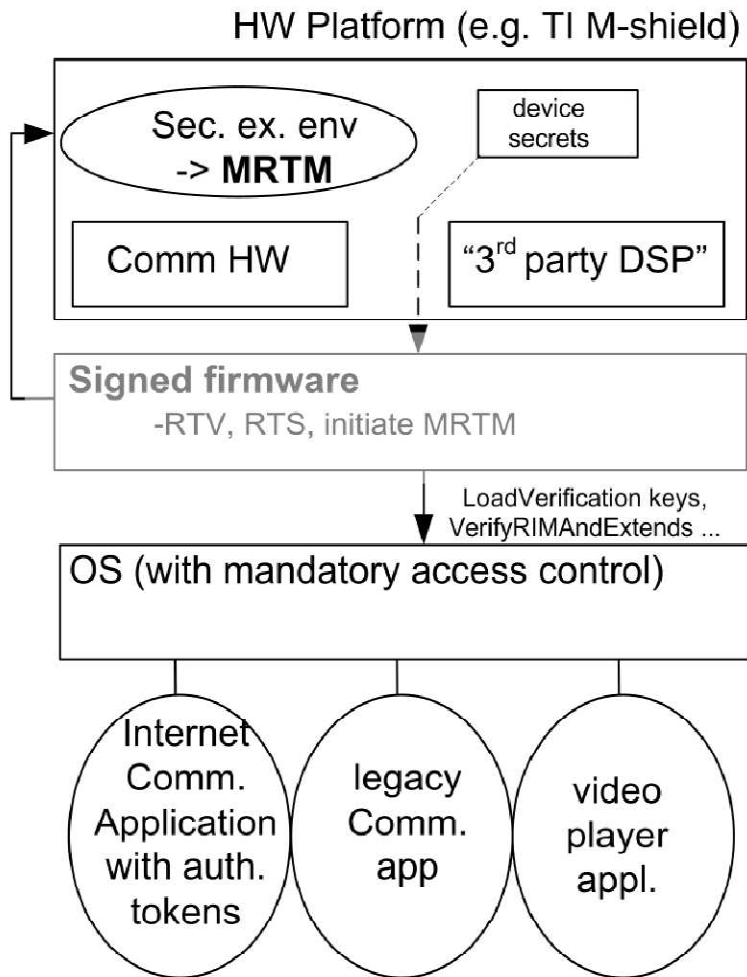
MTM provides a unified approach for

- defining and enforcing secure boot properties
- a defined level of security
- (an API for security services)

OS security (MAC) can bootstrap from known interfaces (integration not necessary)

Some apps may use MTMs e.g. for secure storage, attestation, ...

... and how to protect interfaces, if needed



Motivation:

- coherent way to map a conditionally available property

Approach:

- map a PCR register (value) to the enabling / disabling of a (HW) resource
- as MTM can be software, this is only some extra logic
- no extra drivers are needed in enabling software, Only the conditional application of RIM certificates
- can be used for post-deployment activation

... and please get your hands dirty

MTM emulator

- Done at Nokia Research Center Helsinki in 2007 (a few fixes have dropped in since then)
- A free-for-all (GPL) MTM emulator done as patch to Mario Strasser's TPM emulator
- Get it at <http://mtm.nrsec.com>
- Compiles and runs at list on Linux and Cygwin

Conclusions

- Mobile Terminals have a long tradition of HW & SW security
- Regulatory approval and service cost motivate inclusion of security
- The history in embedded devices stems from legacy (HW, SW, security)
- The MTM specification
 - Gives a way to quantify secure boot in a commonly agreed way
 - May be used as bridge between HW & OS security features
 - Can be used for some typical mobile services
- The legacy TrEE:s are more versatile than e.g. the TPM/MTM specifications
 - Can be used to support architectures like On-Board Credentials
 - ... and vice versa: Could TPM / MTMs develop towards more general service support (include the notion of a TrEE:s)?

Thank you!
Any questions?

On-Board Credentials

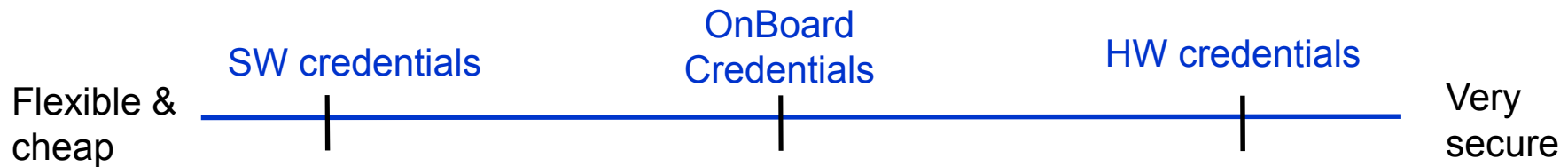
(R&D project at Nokia)

Traditional credential types

- Credential: “credential secret” + a “credential program”
- Software credentials
 - Virtual: e.g., passwords remembered by browser (extension)
 - Cheap and flexible
 - Insecure against malware and device loss
- Hardware credentials
 - Physical: e.g., SIM cards and SecurID tokens
 - More secure and more intuitive
 - No trusted path to user, more expensive, less flexible



OnBoard Credentials



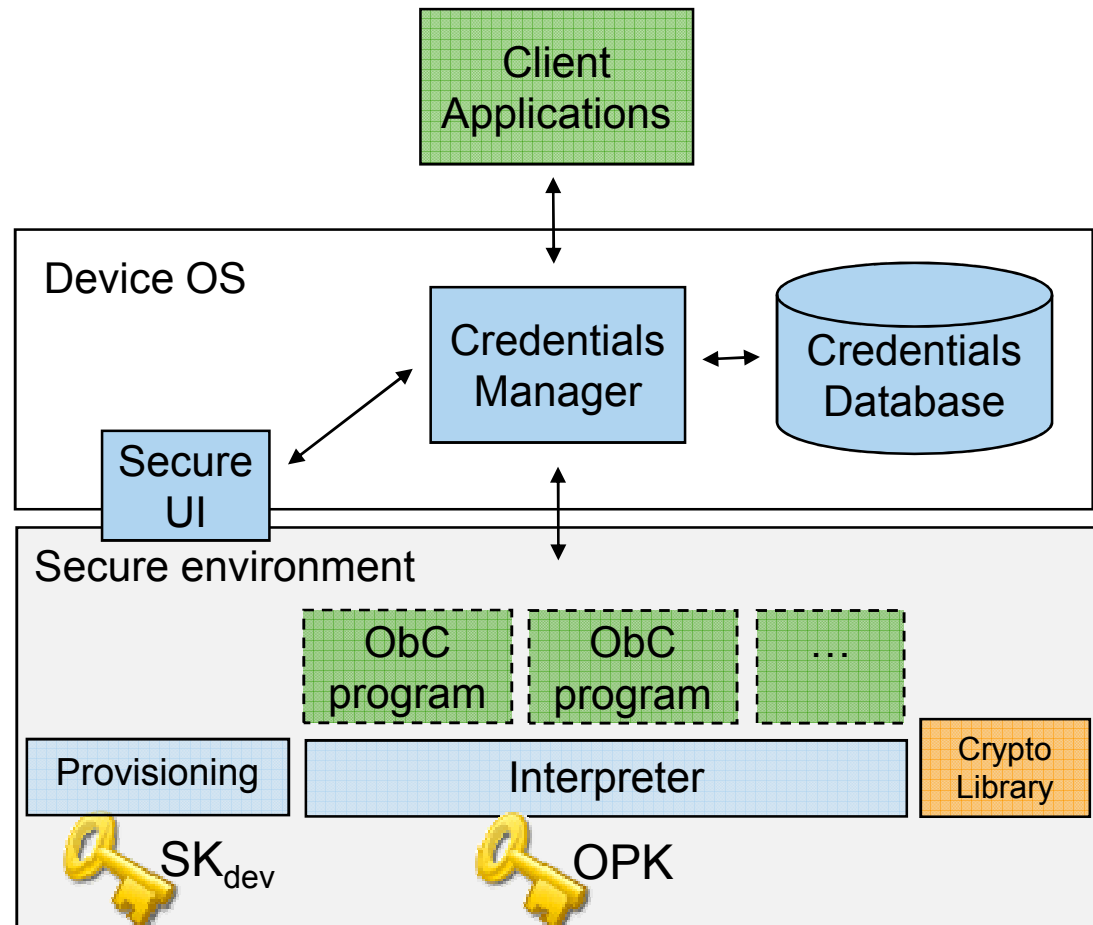
- OnBoard Credentials (ObCs): Virtual credentials based on trusted HW
 - More secure than traditional software credentials
 - Cheaper and more flexible than traditional hardware credentials
- Not much prior work on this topic (academic or otherwise)
 - Systems based on TPM; Fixed credential type (e.g. passwords for browsing)
- Similar in spirit to multi-application smartcards
 - But without issuer control

Design goals

- Credential programs can be **executed securely**
 - Use the secure execution environment
- Credential **secrets can be stored securely**
 - Use a device-specific secret in secure environment for secure storage
- Creating **new types of credentials** (i.e., new credential programs) is easy
- **Anyone can create and use** new credential types
 - Need a security model to strongly isolate credential programs from one another
 - Avoid the need for certification of credential programs
- **Anyone can provision credential secrets** securely to a credential program
 - Needs a mechanism to create a secure channel to the credential program
 - (certified) device keypair; unique identification for credential programs

ObC architecture

- Secure environment based on trusted hardware
 - **Secure storage**, using a device-specific ObC Platform Key (OPK)
 - **Secure execution**
 - TPM, M-Shield, ...
 - **Certified device keypair**
 PK_{dev}/SK_{dev}
- **Interpreter** running in secure environment (Lua)
 - Credential programs implemented as **scripts**
 - code hash as unique ID
- Secure UI for user interaction



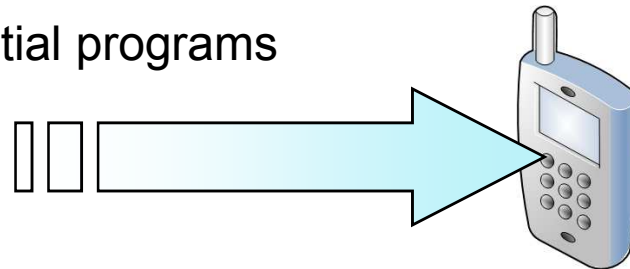
Longer Tech Report at <http://research.nokia.com/tr/nrc-tr-2008-001>

Isolation of Credential 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 = f (ObC Platform Key, ObC program ID)
 - ... and every group of interdependent ObC programs
 - Sealing-key = f (ObC Platform Key, ObC program group ID)

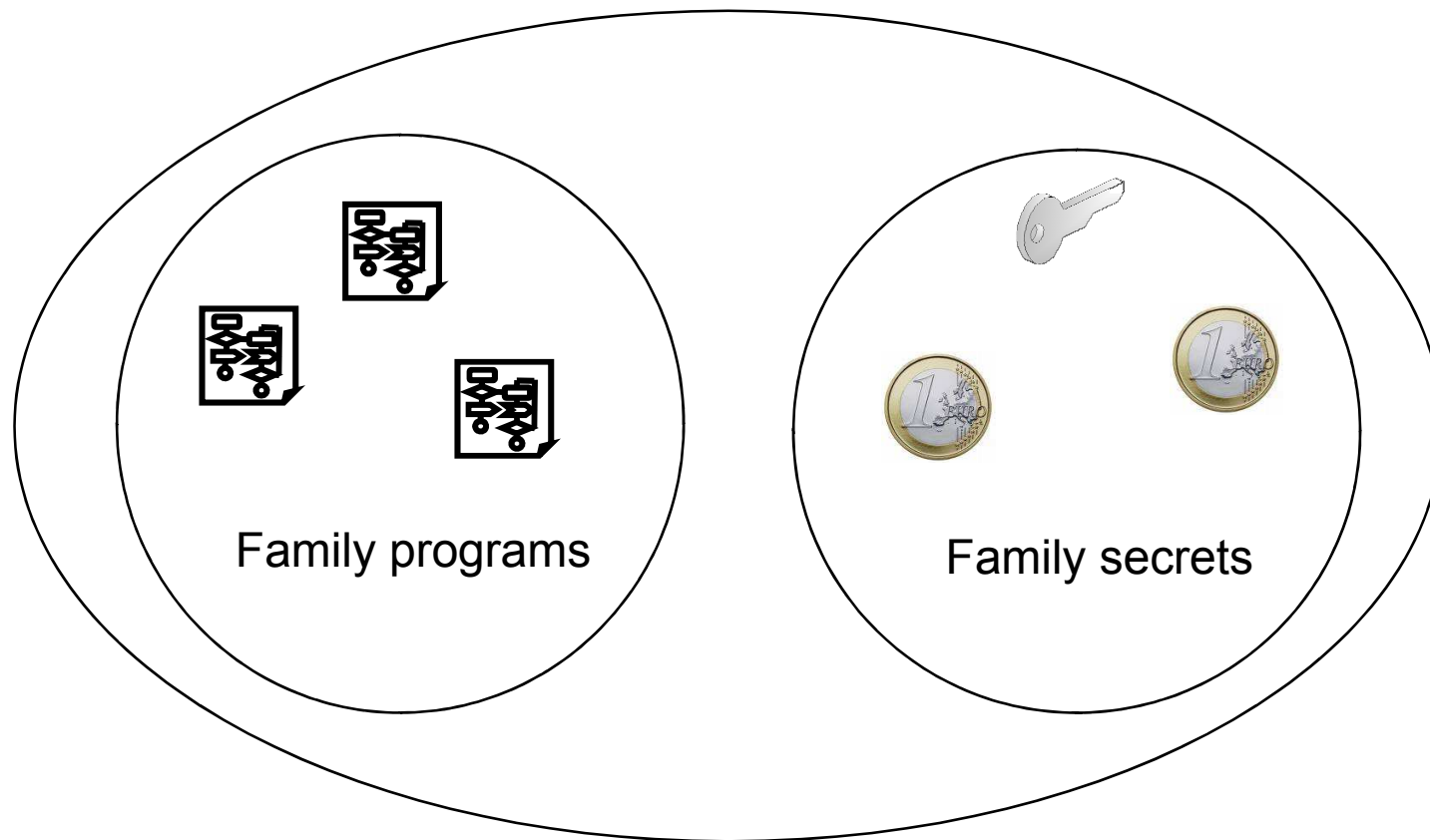
Requirements for Provisioning Credentials

- Provisioning protocols typically focus on **user authentication** only
 - ... and also tend to focus on secrets like keys – not programs
- 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



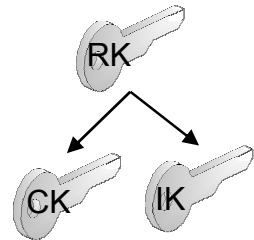
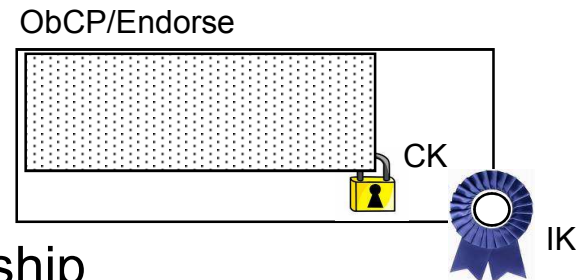
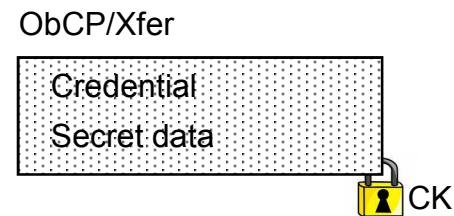
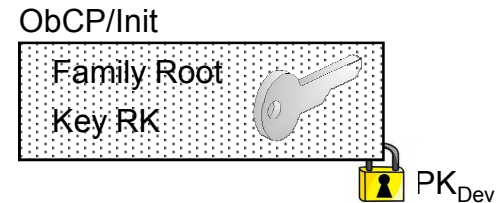
Provisioning credential secrets (1/3)

Basic Idea: the notion of a **family** of credential secrets and credential programs endorsed to use them



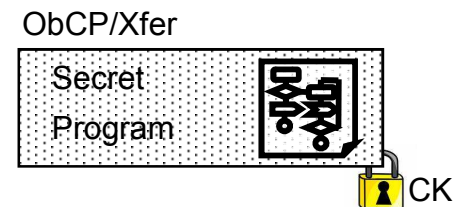
Provisioning credential secrets (2/3)

- Provision a family **root key** to the device
 - using **authentic device public key** PK_{dev}
- Transfer encrypted credential secrets
 - using family **confidentiality key** **CK**
- Endorse credential programs for family membership
 - (program ID is encrypted)
 - using family **integrity key** **IK**



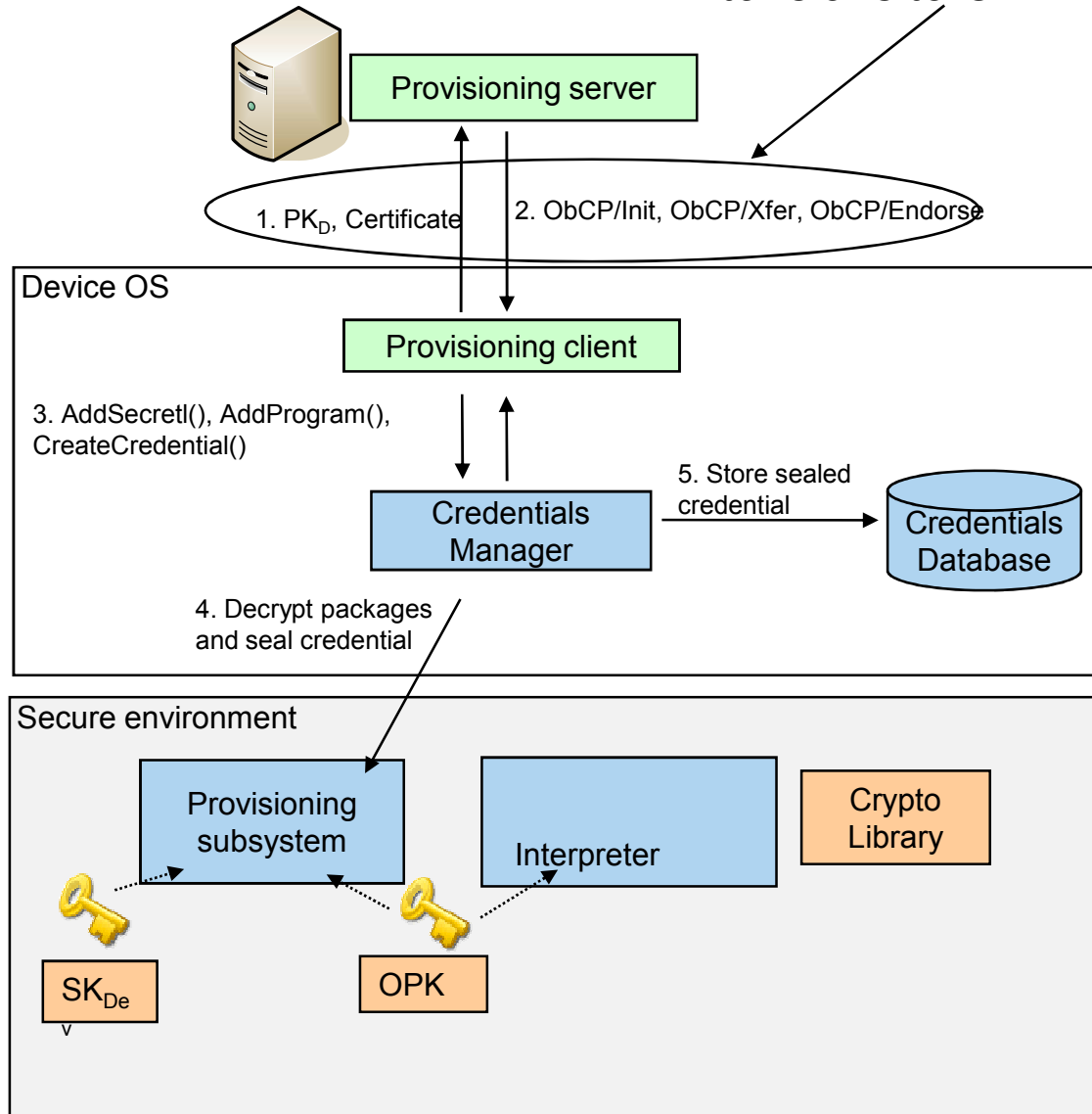
Provisioning credential secrets (3/3)

- 2-pass provisioning
 - Get (certified) device public key and validate it
 - Send ObCP/Init, ObCP/Endorse and ObCP/xfer
 - CT-KIP 2-pass extensions defined by Magnus Nyström (RSA)
- Anyone can define a family by provisioning a root key
- Multiple credential secrets and programs can be added to a family
- Credential Programs can be encrypted as well



Device view: ObC Provisioning

Extensions to CT-KIP 2-Pass



- Credentials protected by hardware secure environment
 - Provisioned data encrypted with PK_D
 - Not accessible to Device OS
 - Cannot be copied between devices
 - Hardware attack is typically destructive and device-specific
- All Credential Manager data can be backed up

Conclusions

- Mobile Terminals have a long tradition of HW & SW security
- Regulatory approval and service cost motivate inclusion of security
- The history in embedded devices stems from legacy (HW, SW, security)
- The MTM specification
 - Gives a way to quantify secure boot in a commonly agreed way
 - May be used as bridge between HW & OS security features
 - Can be used for some typical mobile services
- The legacy TrEE:s are more versatile than e.g. the TPM/MTM specifications
 - Can be used to support architectures like On-Board Credentials
 - ... and vice versa: Could TPM / MTMs develop towards more general service support (include the notion of a TrEE:s)?

Thank you!
Any questions?

Software security (extra slides)

Security before Symbian 9:



In the mobile domain, the “open OS:s” are already hardened

- Symbian OS 9.0+ mandates the use of OS Security features
- Major building blocks of the Symbian OS PlatSec
 - Capability framework
 - Rights of executables
 - Trust levels of libraries
 - Data Caging
 - Protected directories
 - Process Identification
 - Trusted Computing Base (TCB)
 - Kernel, Software installer, File server, Loader
 - Secure UI

Symbian security basics

- Security boundaries
 - Process boundary = memory protection boundary = security boundary
 - Therefore all code within a process has the same trust level
- Security policy checking
 - Done at process boundary – when inter-process communication is done
 - Policy checks by resources themselves (single class)
 - Policy based on capabilities
 - Capabilities are permissions to do something
 - SID and VID identify the process and the group it belongs to
- Minimum permissions principle
 - Processes are given only those capabilities that they need
 - Even all system servers do not have all capabilities
- Local storage
 - Program-specific private directory

Capability framework

- 40% of Symbian OS APIs protected with a capability
- To use a service requiring a capability, the application needs to have it
 - Normal case:
 - Application has passed certain tests and is signed against a certificate (Symbian signing)
 - A signed installation package contains the list of capabilities the application has
 - A self-signed application has no capabilities
 - user can grant user capabilities
 - Blanket (installation time)
 - One-shot (when the requiring action takes place)
- Capabilities for a dynamic library
 - For an application to load a library, the library always has to have a superset of the capabilities of the application for the load to succeed
- Certification authority creates a single-use key pair and certificate and signs the software package. It can now be distributed
 - This way, each software package is signed with a unique key
 - Revocation of a key revokes (blacklists) one software package

Capability assignment

	One-shot	Blanket
Unsigned - Sandboxed	NetworkServices Location	LocalServices UserEnvironment
Signed	Premium Billable events. (Not capability related)	LocalServices UserEnvironment NetworkServices Location ReadUserData WriteUserData ReadDeviceData WriteDeviceData SWEvent ProtSrv PowerMgmt SurroundingsDD

Capability framework – Basic/User capabilities

LocalServices	Grants access to local network services that usually do not incur a cost
NetworkServices	Grants access to remote network services that may incur a cost. The capability to access selected APIs can be granted on a one-shot basis if the application is unsigned-sandboxed
UserEnvironment	Grants access to live confidential information about the user and their immediate environment (e.g. audio, video or biometric data)
WriteUserData	Grants write access to data that is confidential to the phone user. The capability to access selected APIs can be granted on a one-shot basis if the application is unsigned-sandboxed
ReadUserData	Grants read access to data that is confidential to the phone user. The capability to access selected APIs can be granted on a one-shot basis if the application is unsigned-sandboxed

More constrained capabilities (system software)

- MultimediaDD Grants access to multimedia device drivers
- PowerMgmt Grants the right to power off unused peripherals, switch the phone into/out of standby state and power phone down
- ReadDeviceData Grants read access to phone confidential settings or data
- WriteDeviceData Grants write access to phone confidential settings that control the phone's behaviour
- NetworkControl Grants access or modification rights to network protocol controls
- SwEvent Grants the right to generate software key and pen events
- TrustedUI Grants the right to create a trusted UI session, and therefore to display dialogs in a secure UI environment
- ProtServ Grants the right to a server to register with a protected name. Protected names start by a "!".
- DRM Grants access to protected content
- SurroundingsDD Grants access to the surroundings device driver

Really powerful capabilities

TCB	Grants write access to executables and shared read-only resources
AllFiles	Grants read access to entire file system; grants write access to other process' private directories
CommDD	Grants access to communications device drivers
DiskAdmin	Grants access to specific disk administration operations

The capability mechanism is

- Easy to understand
- Easy to control

But still, the Symbian security

- **combines policy with enforcement**
- and is not extendable.

Process Identification

- Secure Identifier (SID)
 - Guaranteed to be locally unique
 - Helps in limiting access to APIs to specific applications
 - *Symbian Signed* applications will have their SID from a protected range
- Vendor Identifier
 - Signed applications can have a unique Vendor identification

Data Caging

- Fixed filesystem structure
- /sys/
 - Restricted system area, accessible only to programs with TCB capability
 - Executables are placed in /sys/bin/ (executables are not run from other places)
- /private/
 - /private/<SID>/ contains the private data for each program
 - Backup and Restore will need active participation of the owning process
- /resource/
 - Contains public data
 - Read-only for programs without TCB capability
- Everything else has public r/w rights
- Executing from removable media (which might have been altered)
 - An initial hash of a program is stored in the /sys/ -directory, which is recomputed and checked whenever the program is run.