Trusted computing

Aurélien Francillon

Secappdev 24/02/2015

Contents

- Goal of trusted computing
- Software (attempts) at trusted computing
- Trusted computing with hardware
  - Static root of trust
  - Dynamic root of trust

Trusted computing goals

We always need to trust computers or devices

It is impossible to tell whether a computer is compromised or not by merely “looking” at it

- Even harder remotely

Looking at a computer is not enough

- Malware
- Backdoors …

  … used to be annoying and noisy but don't have to !

Arms race

- Detecting malware is possible from a higher privileged level
  - So malware tries to get there
  - Fight between anti-virus and malware to get to higher privileged modes
- Many examples of very powerful malware
  - Kernel rootkits
  - VMM “blue pill” rootkits
  - Boot sector virus
  - Disk backdoors

The malware problem

- Code Red and nimda worms
- Note from Bill Gates in 2002
  - « Trustworthy Computing »
  - A new started for trusted computing

Software Attempts at Trusted computing

There must be clever ways to do it in software !

- Anti-viruses have been fighting malware for a while
  - But they work best when analyzing software before it gets executed
- Can we detect a compromised computer ?

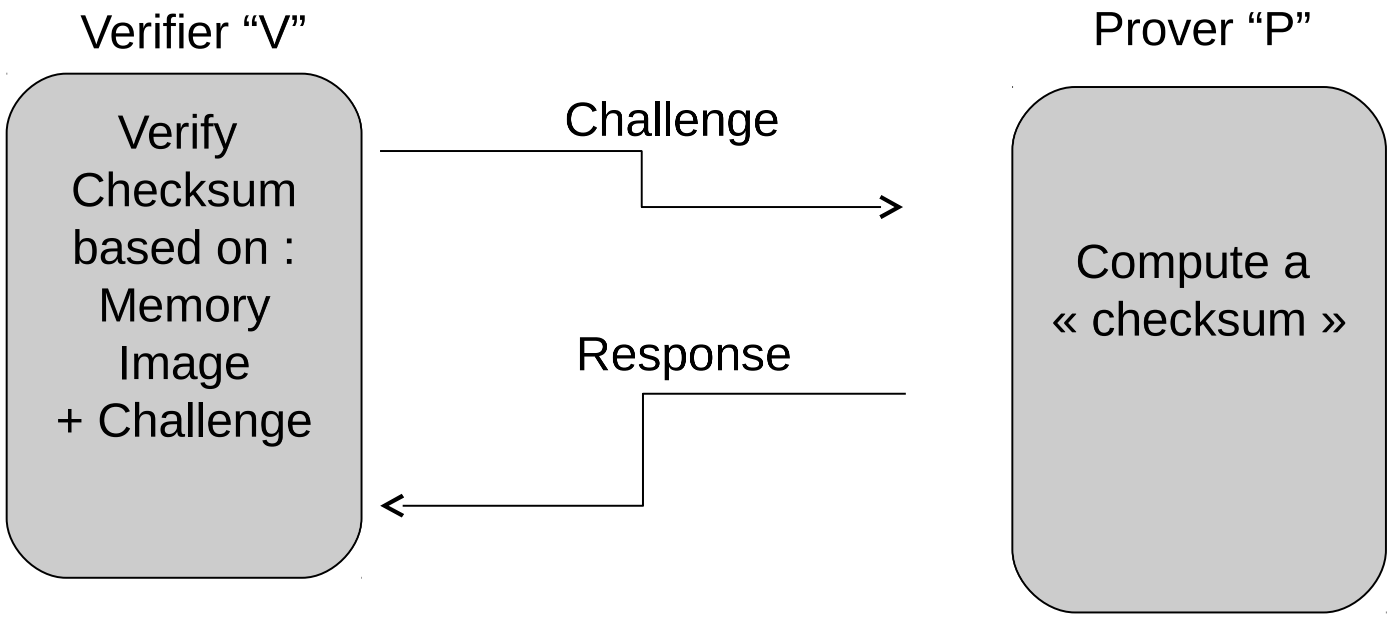
- Computers are very complex, lets start with a very simple device

Lets try to do Remote attestation (in software)

- We want to remotely verify the integrity of a system
  - We know the hardware, and it is simple
  - We know the software running on it
  - We want to verify that the device is not compromised
- Problem 1: What is the sign of a compromise ?
  - Code integrity ?

Remote attestation

- Un-trusted Prover “P” Trusted Verifier “V”
- V knows P memory contents
- V sends a Challenge with a nonce to P
- The prover computes a “checksum” of its memory
- Verifier checks the validity of the checksum



Devices where software based attestation was done

Tow typical devices are Micaz and Telosb:

- Both use IEEE 802.15.4 radio
- Zigbee compatible
- Both have a MCUat 8Mhz

Micaz

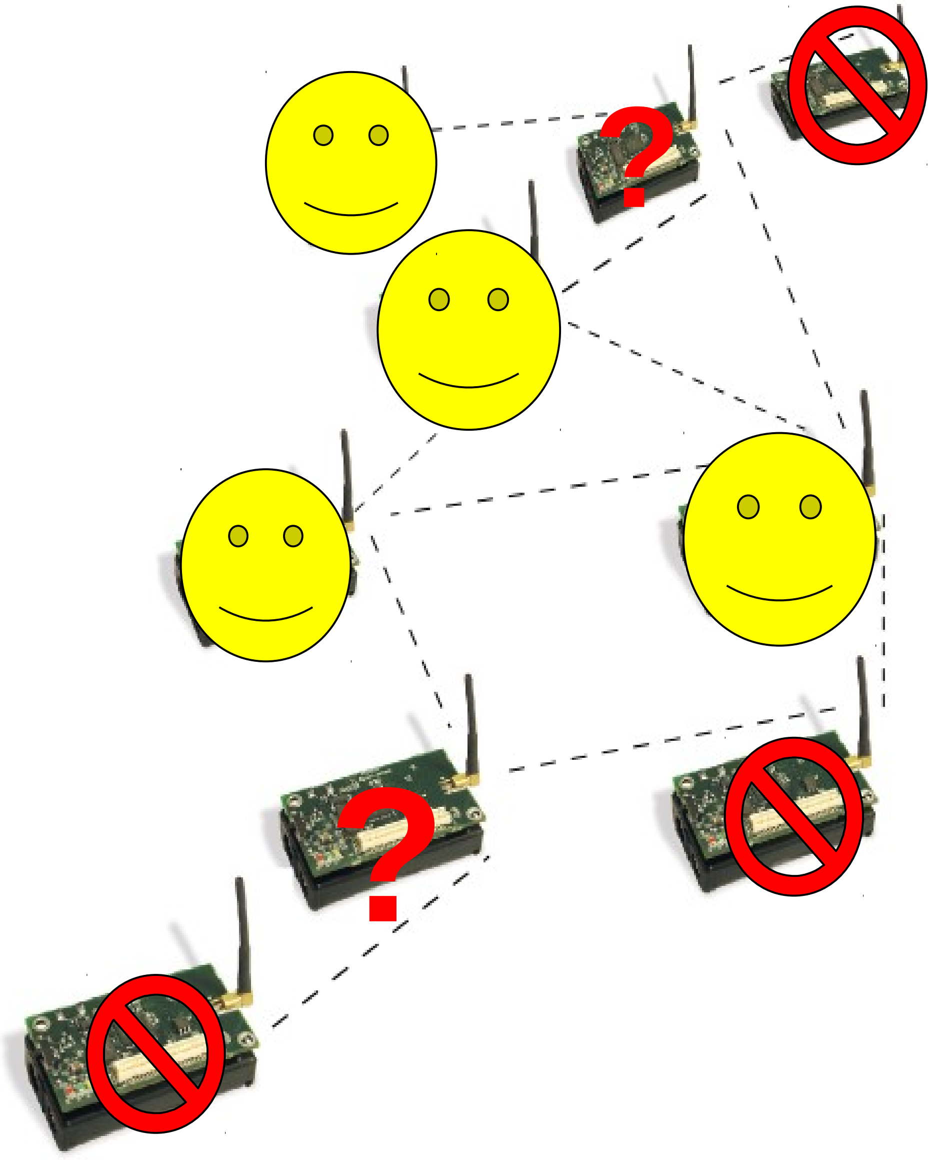
- 4kB sram
- 128kB flash
- 512kB external flash

Telosb

- Texas Instruments msp430
- 10kB sram
- 48kB flash
- 1MB

What remote attestation tells us

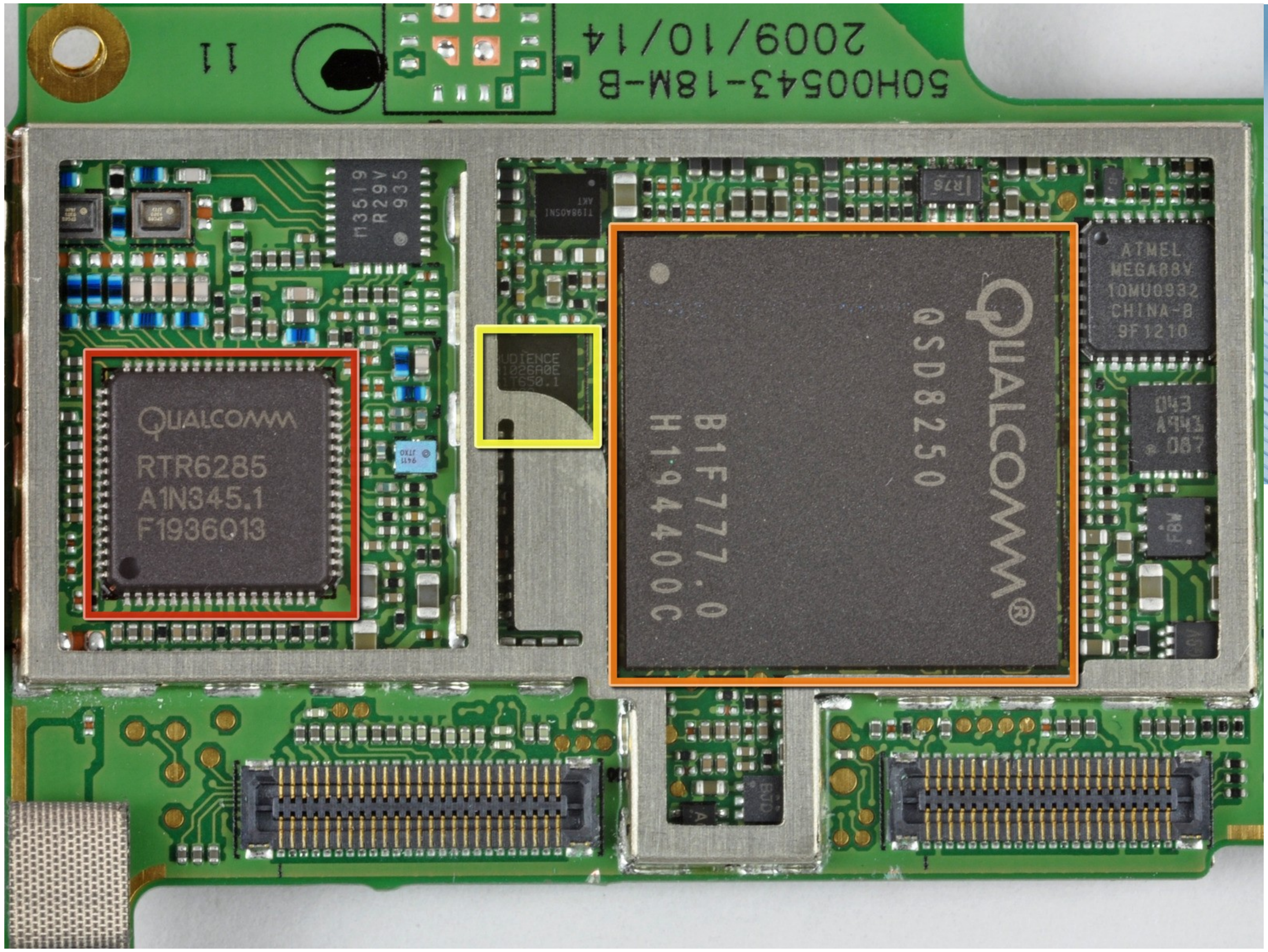
- Positive result
  - Correct memory Contents
  - Good device
- Negative result:
  - Malfunctioning device
  - Malicious device
- No Result
  - Malfunctioning device or
  - Malicious device



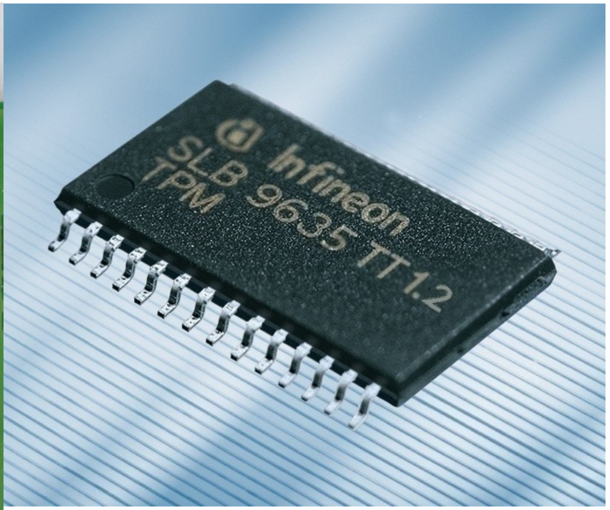
Software based attestation

- Called « Software-based Attestation »
  - Software-based Remote Attestation
  - Verifying software by software (???)
  - Similar to self checking software (obfuscation, tampering protection)
    - User land programs can be easily defeated with MMU tricks
- Motivation
  - Embedded systems without support for this
  - General purpose computers without a TPM

Motivation : size/integration



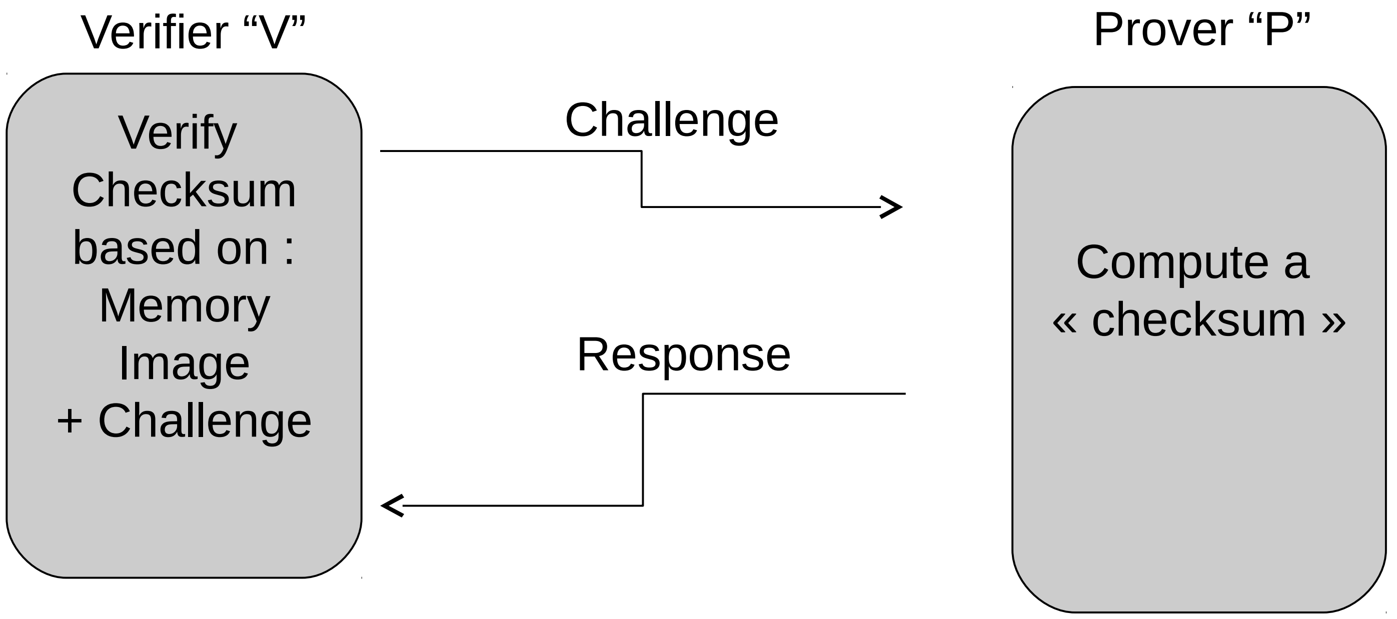
Google Nexus One Main PCB



Infineon TPM
10*7 mm

Computing the « Checksum »

- Standard cryptography not possible !
- We want to prevent a compromised system to cheat with its own memory contents
- We need additional tricks



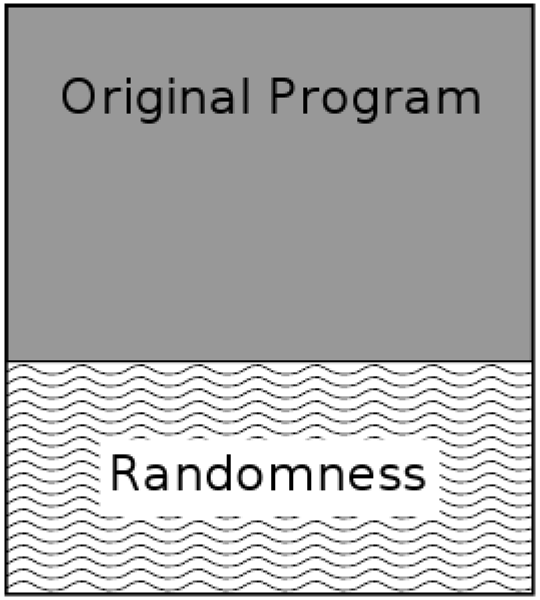
Cheating with the memory contents

- How can we do this ?

- To prevent “cheating” rely on specific constraints:
  - Time
  - Memory
  - Code obfuscation

Approach 1 : fill free memory

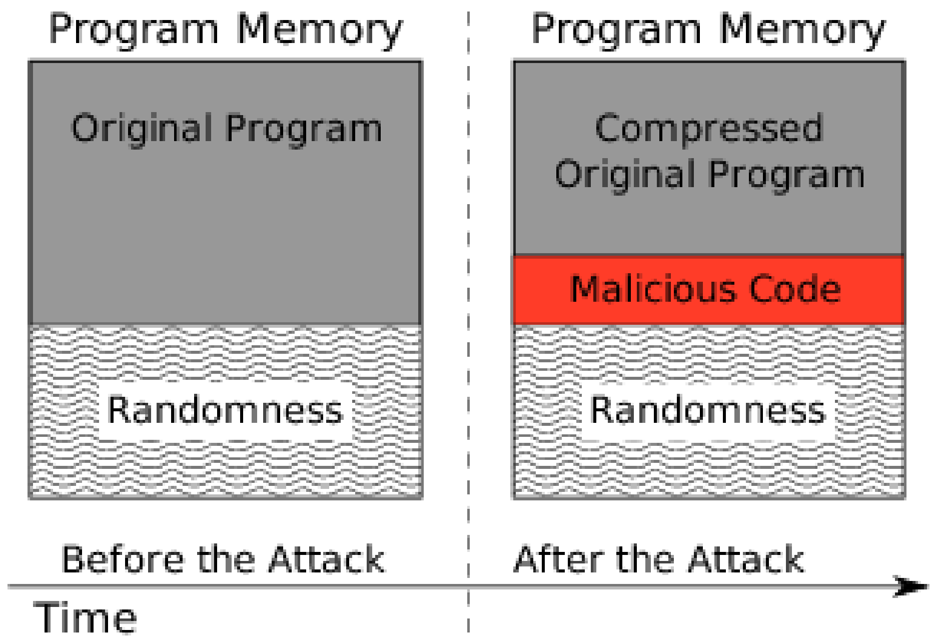
- Randomness-based
  - Fill free space with randomness
  - Checksumming all memory
  - No room for malicious code
  - No strong time constraints



E.g., “Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks” Park and Shin, IEEE TMC, 2005

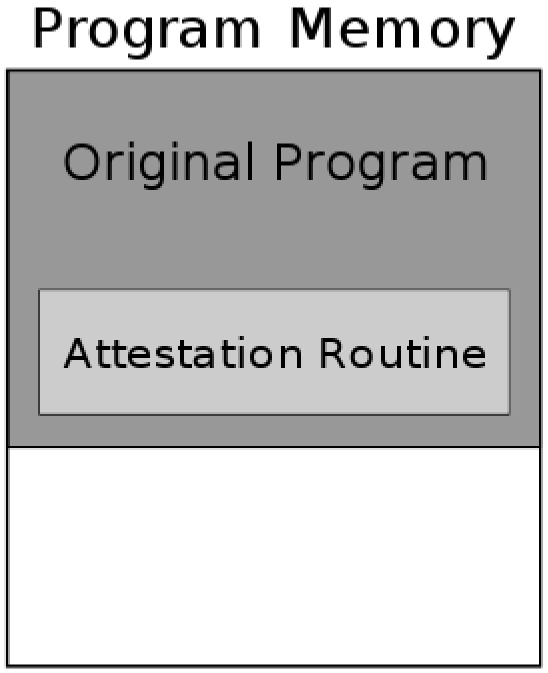
Attacks on randomness based attestation

- Compress the original code
  - This frees space for malicious code
  - When attestation request is received by malicious code
  - It decompresses the original program
  - Live computation of the checksum

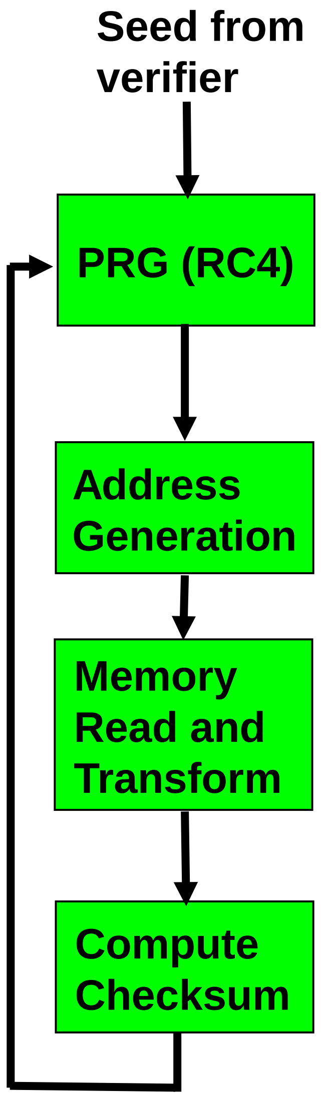


Option 2: timing-based !

- Attestation is optimized to be performed in a constant time
  - Code modifications are detected by additional delay
  - Random memory accesses
  - Possibly relies on cache behaviors
    - Cache hit / Cache miss
- Strong Time constraints, not for:
  - Multi hop/unreliable networks



SWATT Assembly Code



```
Generate ith member of random sequence using RC4
zh   = 2          ldi zh, 0x02
r15 = *(x++)     ld r15, x+
yl   = yl + r15      add yl, r15
zl   = *y         ld zl, y
*y   = r15        st y, r15
*x   = r16        st x, r16
zl   = zl + r15      add zl, r15
zh   = *z         ld zh, z
Generate 16-bit memory address
zl   = r6         mov zl, r6
Load byte from memory and compute transformation
r0   = *z         lpm r0, z
r0   = r0 xor r13   xor r0, r13
r0   = r0 + r4   add r0, r4
Incorporate output of hash into checksum
r7   = r7 + r0   add r7, r0
r7   = r7 << 1   lsl r7
r7   = r7 + carry_bit   adc r7, r5
r4   = zh         mov r4, zh
```

NB: This is just an example I'm not expecting you to understand the code
(Slide from A. Perrig)

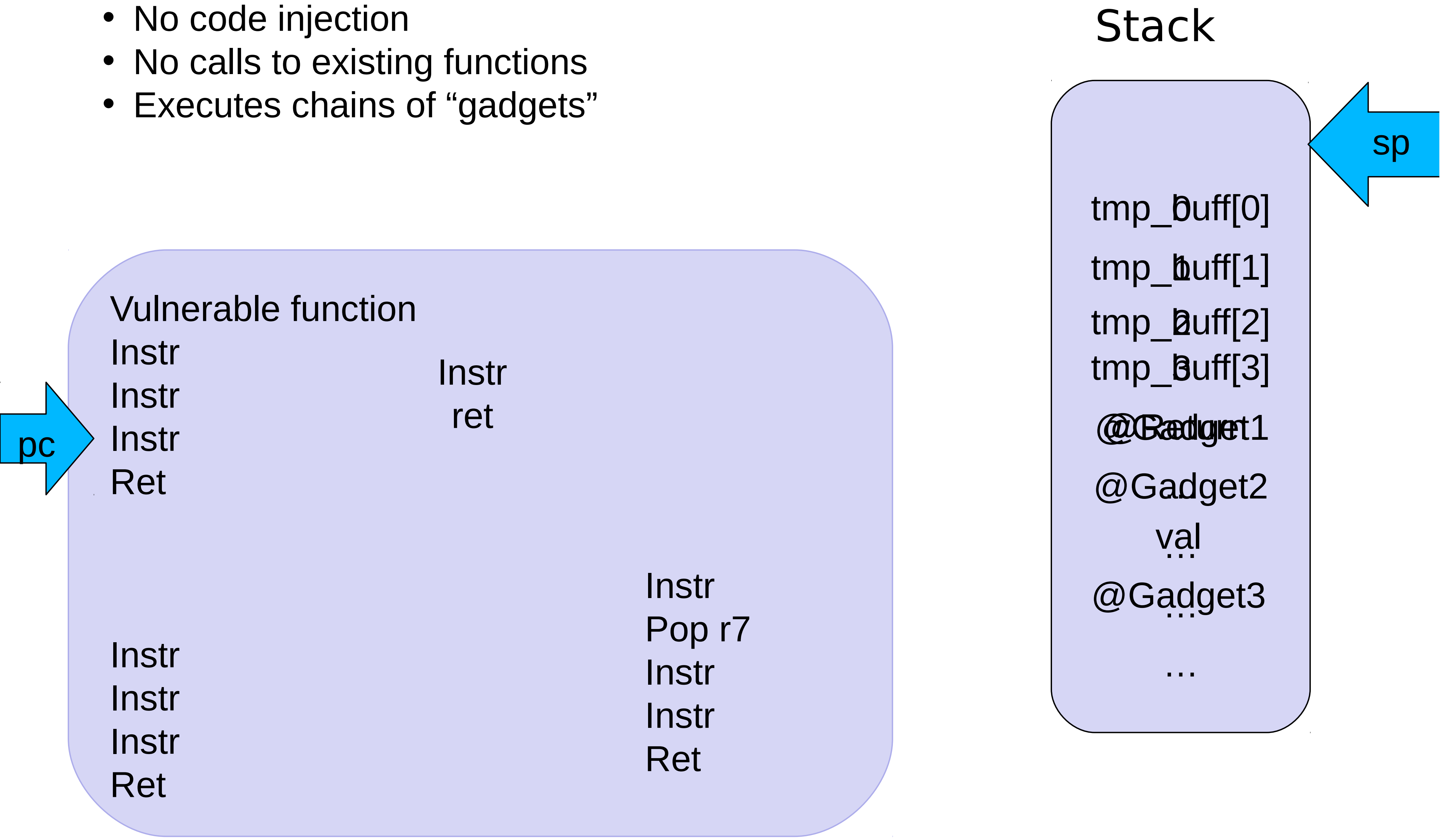
Timing based attestation

- Difficult to have :
  - Optimal code
  - Know the best attack
  - This is very important to be able to distinguish between attacks and network delays

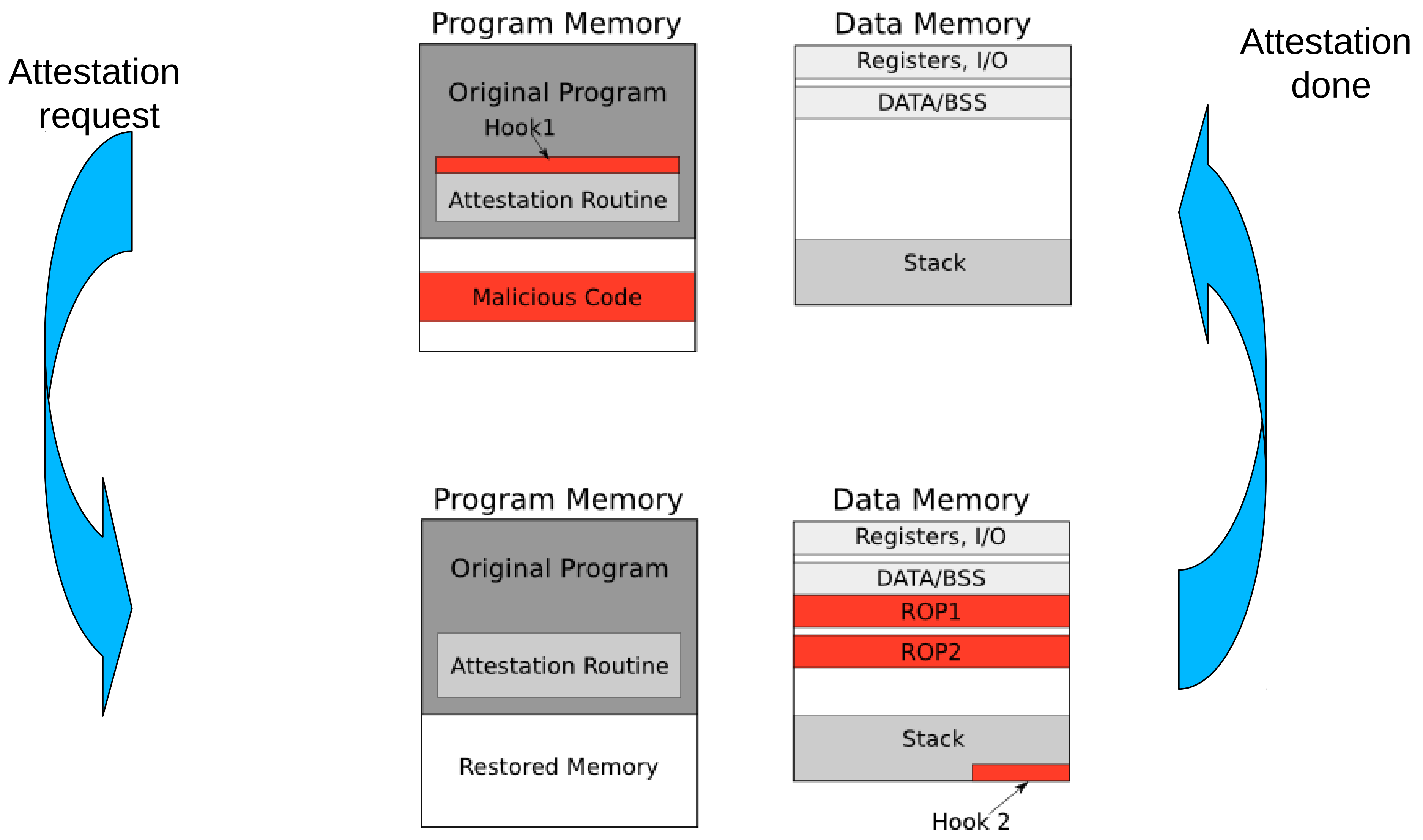
- A bigger problem : return-oriented programming
  - ROP Allows to perform arbitrary computations by manipulating only data memory

Return Oriented Programming in one slide !

- No code injection
- No calls to existing functions
- Executes chains of “gadgets”



ROP Rootkit


Attestation request
Program Memory
Original Program
Hook1
Attestation Routine
Malicious Code
Data Memory
Registers, I/O
DATA/BSS
Stack
Attestation done
Program Memory
Original Program
Attestation Routine
Restored Memory
Data Memory
Registers, I/O
DATA/BSS
ROP1
ROP2
Stack
Hook 2

ROP Rootkits

- Very powerful attacks
  - ROP is now standard exploitation technique
  - Difficult to prevent (arms race)
- Several examples of ROP rootkits
  - e.g., was done for windows kernel rootkits
- Not sufficient to verify integrity of code only:
  - Verifying the integrity of code
  - Or integrity of the complete platform.
- We will see that dynamic root of trust can help here

Software based attestation

- Software based attestation difficult
  - If even possible
  - Like crypto experts advise against designing your own cipher, I would advise you against designing your own software based attestation
- If you really have no other option that's better than nothing
  - Could be seen as an obfuscation
  - Not perfect but will slow down attackers
  - I would be happy to learn about your scheme
    - Chances I can break it :)
- Belief : We need some hardware support

Trusted computing
With hardware support

HW based Trusted Computing

- Rely on dedicated hardware to provide strong guaranties e.g.:
- Prevent booting modified image (Secure boot)
- Proving integrity of running software (Attestation)
- Protecting secrets from a modified OS (Sealing)
- Proving identity (Authentication)

Static Root of Trust

Static Root of Trust (a.k.a SRTM)

- Provides a measurement of the code at loading time

SRTM Example 1: TPM v1.1

- Hashes code before loading
- Stores the hash in TPM registers (“extend”)
- Platform Configuration Registers PCR
- Resulting hash can be used to prove the load time status of the system

Static Root of Trust

SRTM Example 2: Secure Boot

- Very common on embedded systems
- A fixed bootloader (e.g., in ROM)
- Contains a public key
- Loads code
- Verify signature of the code
- if valid executes it otherwise stops execution (otherwise Brick!)

Secure Boot: Smartphone Example

Smartphones are usually locked down

Operators wants to:

- Protect their network from abuse
- Prevent subsidized phones but want them to be used on their network (simlock)
- Provide value added services
- Comply with regulations

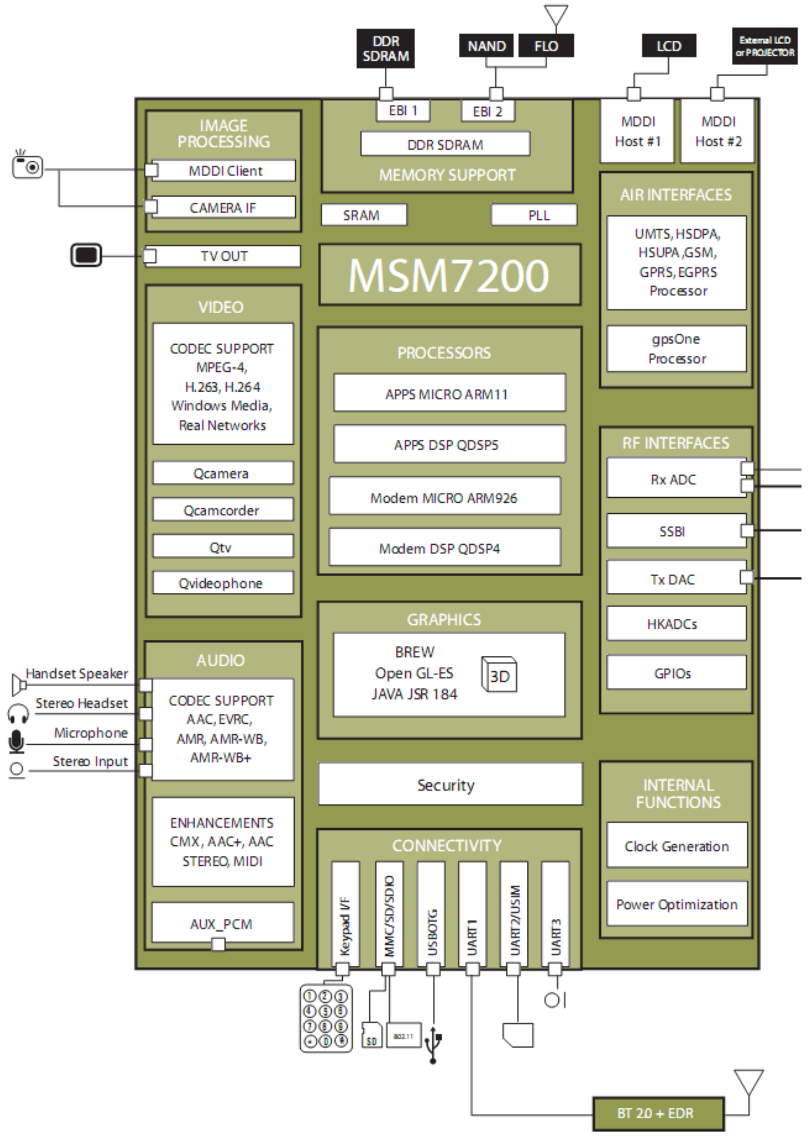
Manufacturers mainly sell the phones to operators :

- This is becoming less true

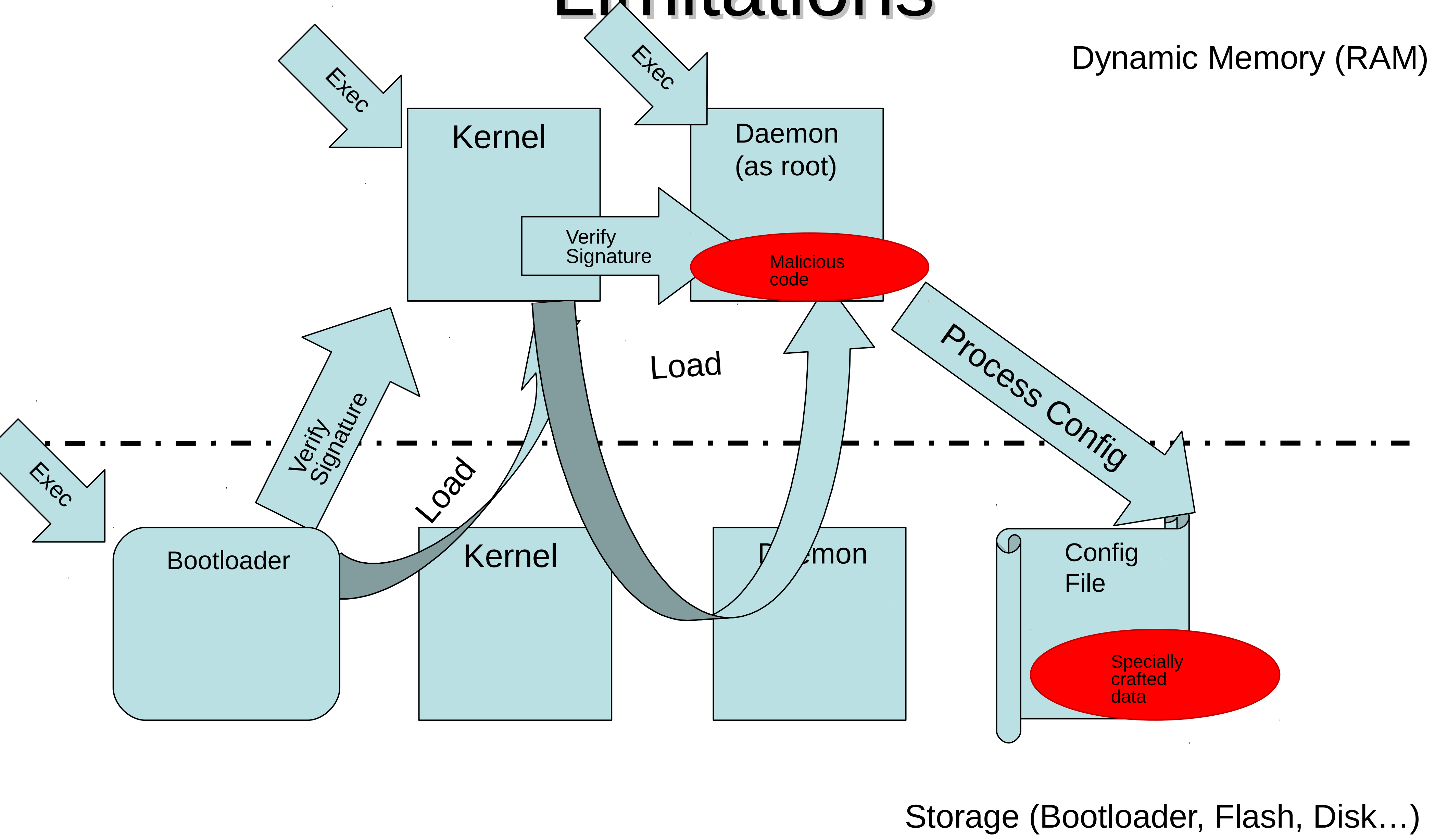
Static Root of Trust : Problems

- Verifies only static information
  - Code at initial loading time
- Difficult to know the runtime status of a device
  - Exploits at runtime can compromise the system state
- Long running applications
  - Should we reboot them before doing sensitive operations ?
- Even reboot not sufficient, example : permanent “jailbreaks” on the iPhone
  - The system code is loaded with secure boot
  - Only “secure code” is executed
  - Configuration file loaded at boot runtime exploit a bug in a root daemon

Secure Boot: Smartphone Example

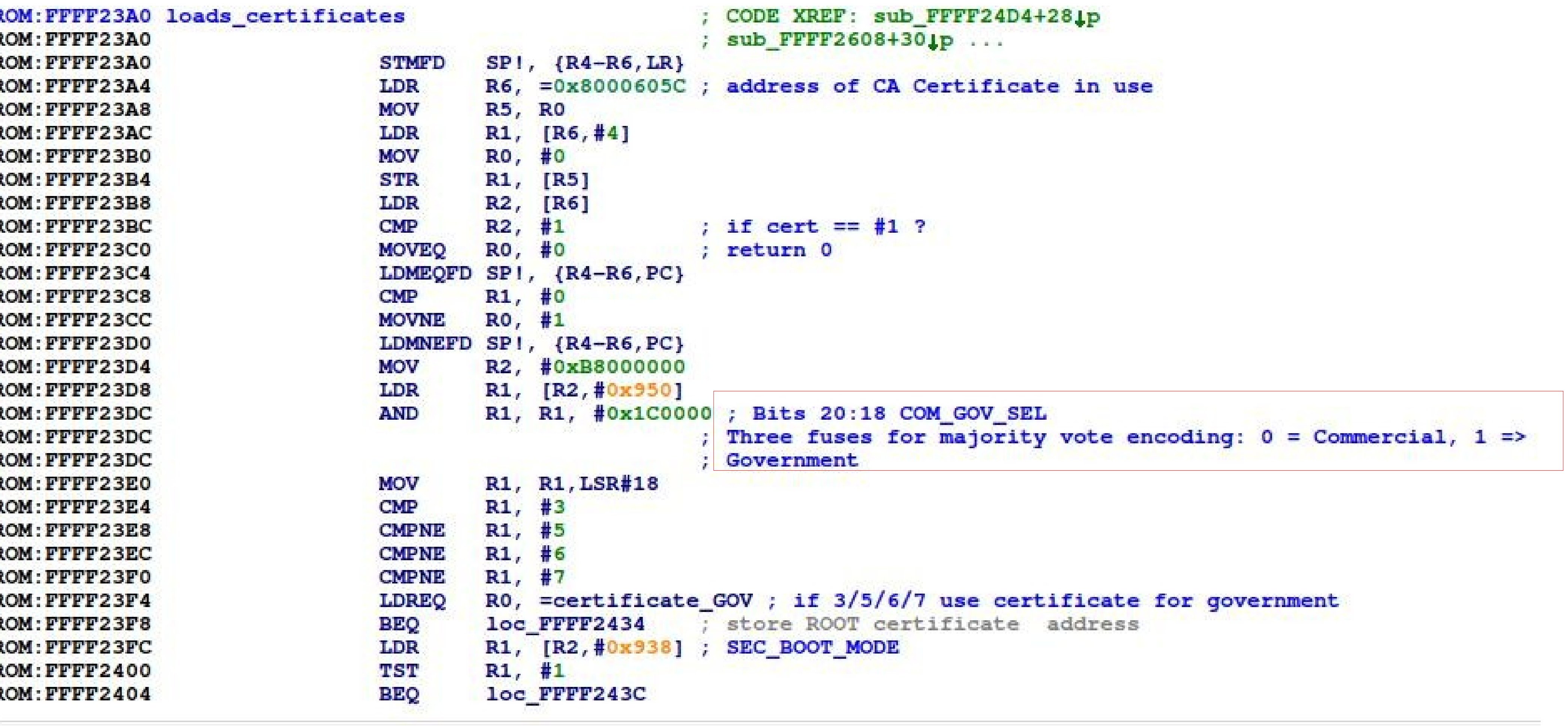


Schematic View of a SRTM Limitations



From an actual smartphone chip

- ROM memory



```
OM:FFFF23A0 loads_certificates                   ; CODE XREF: sub_FFFF24D4+28↓p
OM:FFFF23A0                                      ; sub_FFFF2608+30↓p ...
OM:FFFF23A0              STMFD   SP!, {R4-R6,LR}
OM:FFFF23A4              LDR     R6, =0x8000605C ; address of CA Certificate in use
OM:FFFF23A8              MOV     R5, R0
OM:FFFF23AC              LDR     R1, [R6,#4]
OM:FFFF23B0              MOV     R0, #0
OM:FFFF23B4              STR     R1, [R5]
OM:FFFF23B8              LDR     R2, [R6]
OM:FFFF23BC              CMP     R2, #1          ; if cert == #1 ?
OM:FFFF23C0              MOVEQ   R0, #0          ; return 0
OM:FFFF23C4              LDMEQFD SP!, {R4-R6,PC}
OM:FFFF23C8              CMP     R1, #0
OM:FFFF23CC              MOVNE   R0, #1
OM:FFFF23D0              LDMNEFD SP!, {R4-R6,PC}
OM:FFFF23D4              MOV     R2, #0xB8000000
OM:FFFF23D8              LDR     R1, [R2,#0x950]
OM:FFFF23DC              AND     R1, R1, #0x1C0000 ; Bits 20:18 COM_GOV_SEL
OM:FFFF23DC                                      ; Three fuses for majority vote encoding: 0 = Commercial, 1 =>
OM:FFFF23DC                                      ; Government
OM:FFFF23E0              MOV     R1, R1,LSR#18
OM:FFFF23E4              CMP     R1, #3
OM:FFFF23E8              CMPNE   R1, #5
OM:FFFF23EC              CMPNE   R1, #6
OM:FFFF23F0              CMPNE   R1, #7
OM:FFFF23F4              LDREQ   R0, =certificate_GOV ; if 3/5/6/7 use certificate for government
OM:FFFF23F8              BEQ     loc_FFFF2434    ; store ROOT certificate  address
OM:FFFF23FC              LDR     R1, [R2,#0x938] ; SEC_BOOT_MODE
OM:FFFF2400              TST     R1, #1
OM:FFFF2404              BEQ     loc_FFFF243C
```

Secure boot

- Good to prevent booting untrusted image
- In general under the control of the owner of the keys
  - In general the manufacturer
- Does not tell much about the runtime status of the system
  - Can be defeated after (every) boot

TPM-based Trusted Computing

TCPA/ TCG

Trusted Computing Platform Alliance (TCPA, a.k.a. “Palladium”)

- Now Trusted Computing Group (TCG)
- IBM, Intel, Microsoft…

Promoting standard for more secure computing

- Relying on an additional chip on the motherboard
- Trusted platform module (TPM)

Microsoft Vista incorporates support for these chips, and uses them as the basis for certain novel security functions.

- Open source software also exists that is capable of exploiting this hardware.
- However, the full potential of the hardware remains to be exploited

Trusted Computing vs Secure Boot

Secure boot authorize signed software to execute

Trusted boot with TPM, take measurements of executed software

- E.g. can let do the attestation by a trusted third party
- Attested Boot, Verified Boot

Windows 8 moved from attested boot to secure boot:

- How to run Linux on windows 8 certified hardware ?
- Secure boot can be manually deactivated in the bios
- But not on ARM platforms!

Fundamental Requirements

To achieve trust we need to have a way make sure that the operating system has booted correctly.

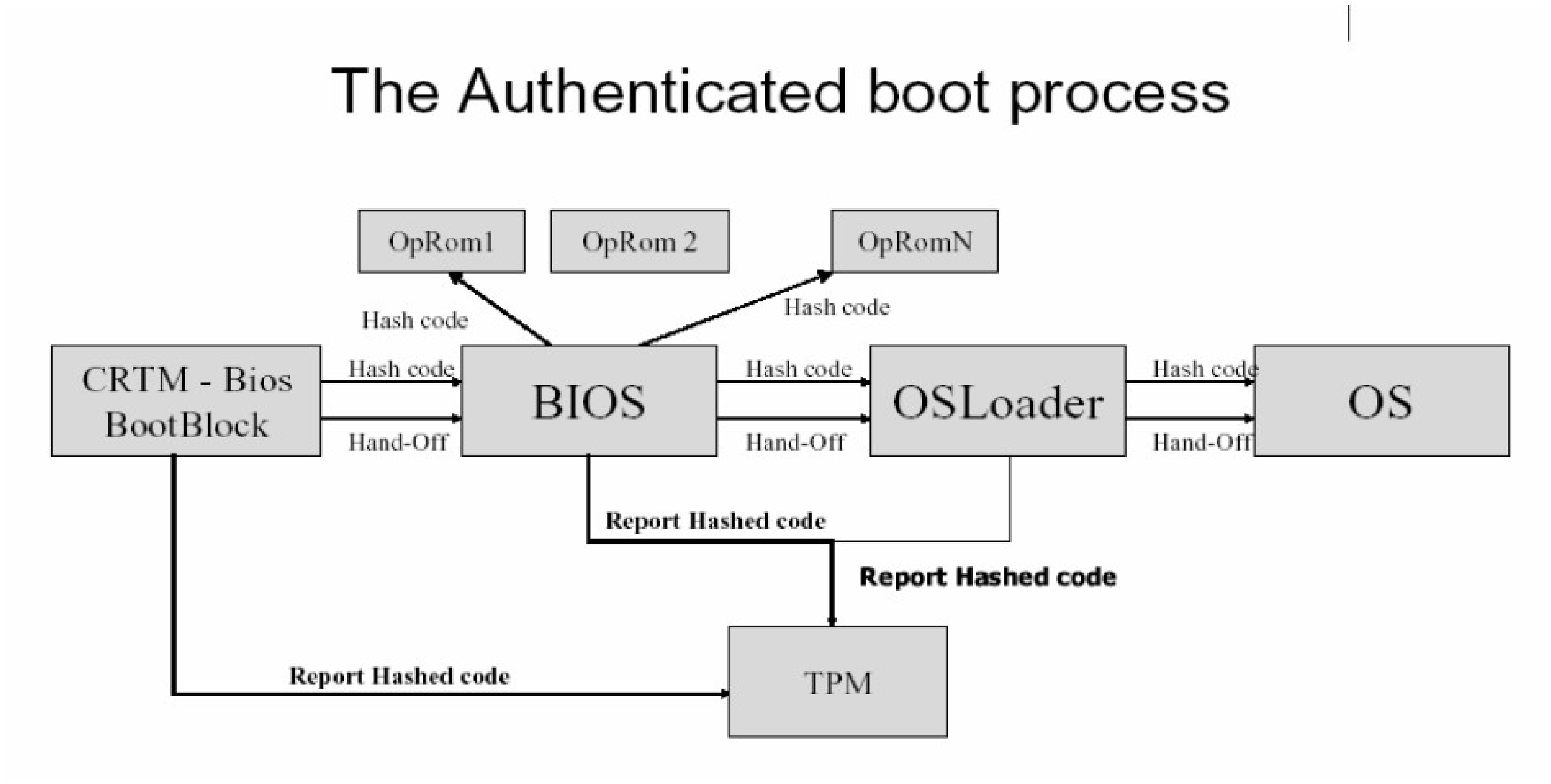
This requires assuming that the PC hardware has not been modified

- embedding key functions in a dedicated chip – the Trusted Platform Module (TPM).

We need a way of monitoring the boot process.

- initial boot measured by the ‘Core Root of Trust’ (ROM)
- loaded software has been measured it can measure the next software to be loaded, iterating the process

Authenticated boot



TPM registers

- Platform Configuration Registers (PCRs).
  - They are used to store platform software integrity metrics.
  - A TPM has several PCRs (16 min) and uses them to record different aspects of the state of the trusted platform.
  - Each PCR has a length equal to a SHA-1 digest, i.e., 20 bytes.

PCR contents/ Extend

- Each PCR holds a value representing a summary of all the measurements presented to it since system boot:
  - This is less expensive than holding all the individual measurements in the TPM;
  - This means that an unlimited number of results can be stored.
- A PCR value is computed using the “extend” operation:
  - **PCR=SHA-1(previous PCR value || latest measurement result)**
  - A PCR must be a TPM shielded location, protected from interference and prying.
  - The **measurement results** are provided by software.

Reporting on integrity

- Measurements reported to the TPM during or after the boot process cannot be removed or deleted until reboot.
- The attestation identity keys are used to sign integrity reports.
- The recipient can then evaluate the trustworthiness of the:
  - Signed integrity measurements, by examining the platform identity certificate;
  - Software configuration of the platform, using the reported measurements.
- This is a “quote” operation the TPM returns :

Sign(PCRs, Nonce)

Authenticated vs Secure Boot

- The above measures provide **authenticated boot**,
  - i.e., a means by which a third party can verify that a certain set of software has booted.

- Does not guarantee **secure boot**,
  - i.e., guarantee that only a particular set of software is able to boot.

Sealing Data

- Sealing uses a “Storage Root Key”
  - The private key is in the TPM
  - Only the TPM can decrypt data
- Seal operation: Data + list of PCR are encrypted
- Data is un-sealed only when the PCR are in the same status
  - i.e., the system is “clean”
- Usually the sealed data is an encryption key (BitLocker)

What TC will not do

- Software vulnerabilities result from both:
  - design errors;
  - coding errors;
- TC technology will not prevent such errors, or aid in the development of secure software without vulnerabilities.
- Vulnerabilities can be abused at any time
  - TC will not prevent the exploitation of such vulnerabilities.

- Viruses/malicious code – TC will not stop them being written or circulated.
- TC ensures that modifications to the software stack are detected

TC Limitations

Where to stop verification of code ?

- BIOS, Bootloader, Kernel
- OS libraries
- Applications
- Shell scripts ?
- Just In Time compiled code ?
- JavaScript ?
- Interpreted data
- All data ?

Integrity Measurement Architecture

- Problem with extending all software
  - Same code run PCR values different

- PCR extend (bios)
- PCR extend (bootloader)
- PCR extend (kernel)
- …
- PCR extend (mail app)
- PCR extend (web app)

- PCR extend (bios)
- PCR extend (bootloader)
- PCR extend (kernel)
- …
- PCR extend (web app)
- PCR extend (mail app)

Integrity Measurement Architecture

- IMA Integrity measurement architecture (IBM)
- In addition to attesting the boot /BIOS/kernel all the running software is extended to the PCR’s
  - The (attested Linux) kernel sends the hashes to the TPM before running applications
  - The ordered list of run software is returned during attestation together with the hashes
- This allows for easy verification of the PCR values given known valid hashes
  - Re computation of the PCR
- Example IMA response.xml
- Now being standardized by TCG, in mainland Linux kernel

Runtime Status

- Lets go back to the « running state » problem
  - Even with IMA we do not verify runtime integrity
  - Is it realistic to assume the system has full integrity ?
  - ROP ?

- Intel TXT Is a technology

Intel TXT

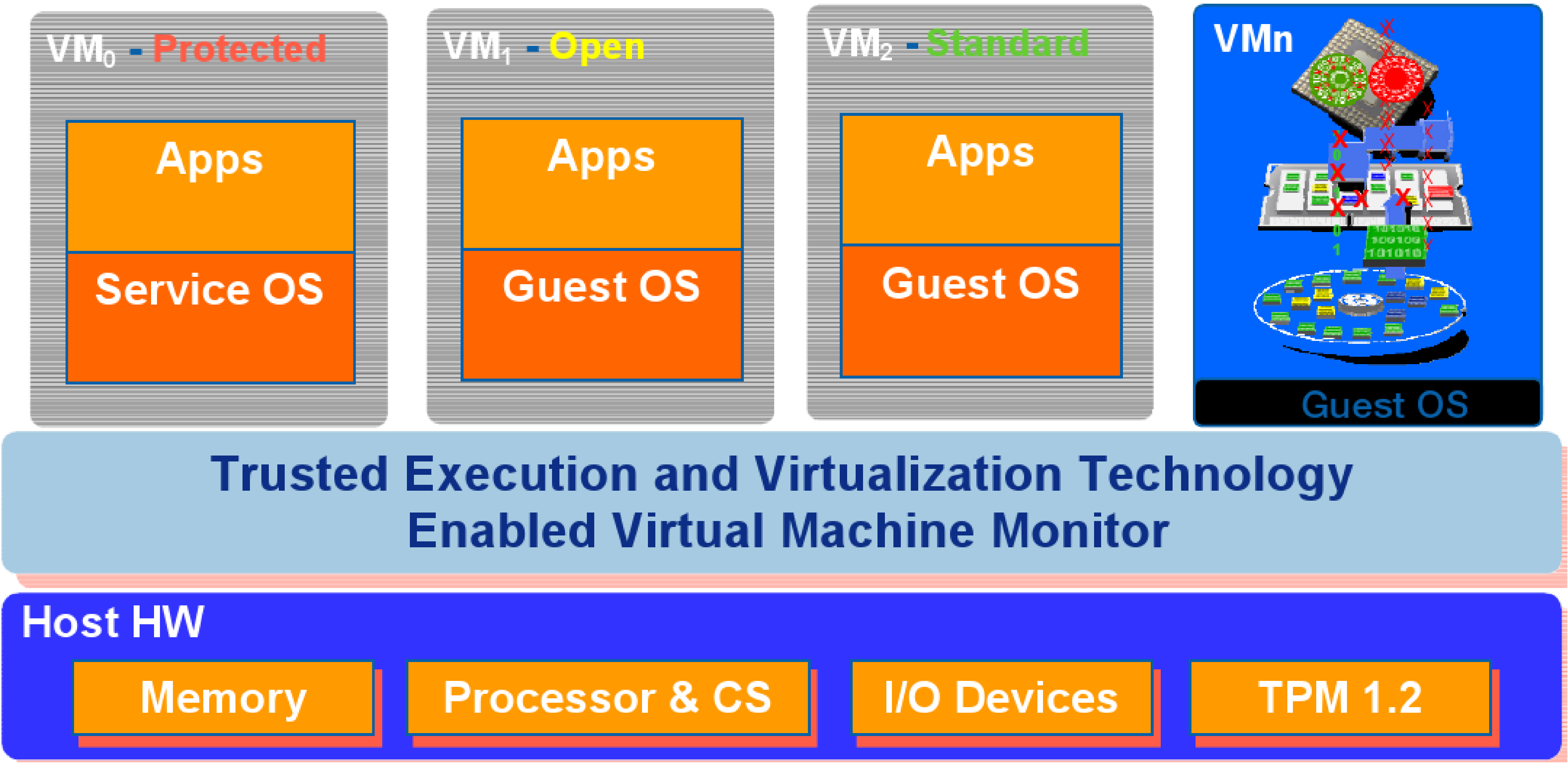
Intel **Trusted eXecution Technology**

- launch a secure environment on an untrusted system

A **Dynamic Root of Trust Measurement** (DRTM)

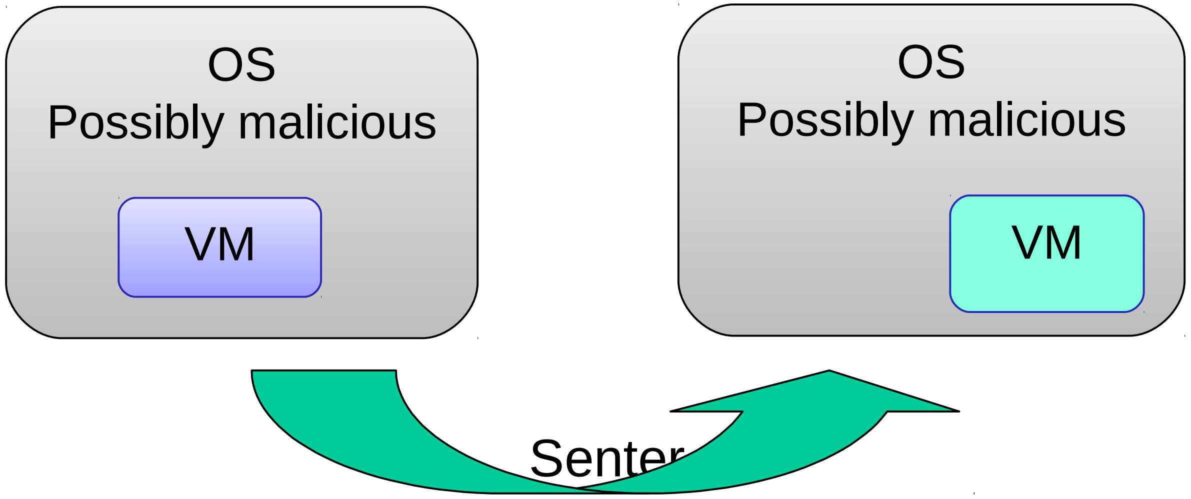
- Removes the BIOS/OptionROM/Bootloaders from the chain of trust
- When launching e.g.,
  - a Virtual machine
  - a sensitive operation

Intel TXT vision



TC Applications: Intel TXT

- Sinit code is executed, configures platform/memory protections (DMA …)
  - PCR >= 17 reset
  - PCR 17 holds the measurement of the Sinit code, PCR 18 “VM”
  - OS can’t tamper with VM anymore
- The TPM :
  - Can unseal keys for the VM only (PCR > 17)
  - Remote attestation can be done on the status of the VM



TXT: example usage

- A document/keys must be accessible only in a secure environment
  - Use TPM Seal to attach the key with a SW configuration
  - The document will be unsealed only if platform PCR are in a given configuration
  - Only the “late launched” environment needs to be attested
- Hardware protections guarantees that the “Virtual machine” accessing the document is unmodified.
  - Company environment closely controlled and isolated from personal mail/applications
  - E.g. vpn connection / access to files
- TXT ideally also needs protected graphics, input (vision...)
- Looking forward to the TXT enabled malware !

Problems with TXT

- One implementation “Flicker”
- Performance issue
  - Launch is slow
- Exclusive access to hardware
  - Cannot be interrupted
  - Not scheduled
- Impossible to do the “protected VM”
- No trusted access to hardware

Intel Software Guard Extensions (SGX)

- New mechanism
  - Papers, datasheets but not in real CPUs (yet?)
  - New set of instructions and CPU features
- Compared to TXT:
  - Enclaves can communicate with the rest of the system and can be scheduled
  - Would allow to use trusted VMs in untrusted clouds
- Trust based on internal TPM
  - Not anymore a separate chip

TC applications

- Not many applications :(
- BitLocker Drive Encryption is one of the few.
  - provides full volume encryption of the Windows volume, which helps protect data on a lost or stolen machine against compromise.
  - Trusted Platform Module (TPM) 1.2 can be used to store the keys that encrypt and decrypt the Windows volume.

TPM Reset attacks

At boot the TPM PCR are clean

- While booting the the chain of trust is built
- The PCR holds the hash of the measured environment
- This is used for e.g., attestation/unseal

When the computer reboots the PCR registers are cleaned to be able to restart the procedure from the beginning

TPM Reset attacks

Hardware attack

- The TPM is connected to a bus (LPC) on the motherboard
- This bus is used for several devices (serial, parallel port controllers, fan controller …)
- One pin of the bus is “reset”
- Holding this pin to ground resets all the devices on the bus
- HW reset demo (video)

TCG assumes that hardware attacks are out of scope of the Trusted Computing platform specification

  – Nevertheless fixed it in new releases (more complex reset)

Other TPM Attacks

- Some SRTM are not that static
  - Could be reprogrammed (+not signed !)
  - Malicious SRTM can break the trust chain
- Another reset attack : set a “reset” bit into a control register
  - Resets the PCR’s (undisclosed TPM model)
- BIOS bugs ?

TXT Attacks

System Management Mode (SMM) is used for power management

- Runs specific code in a separate execution space
- Has access to all physical memory / IO
- Is the most privileged mode on the CPU

SMM Memory is protected by the chipset. However, on some chipsets, bugs allows to bypass those protections

- Using cache coherency attacks
- No cache flush when SMI is launched
- Some ways to change the address of the SMM Memory pages

Modifying SMM memory before late larch

- allows to execute arbitrary code in the context of the “late launched” VM
- Sinit code, launched

Dynamic root of trust in embedded systems

TEE

- Trusted Execution Environment
  - A protected place to execute in a system
  - Execution cannot be tampered
- Various forms of it
  - ARM TrustZone

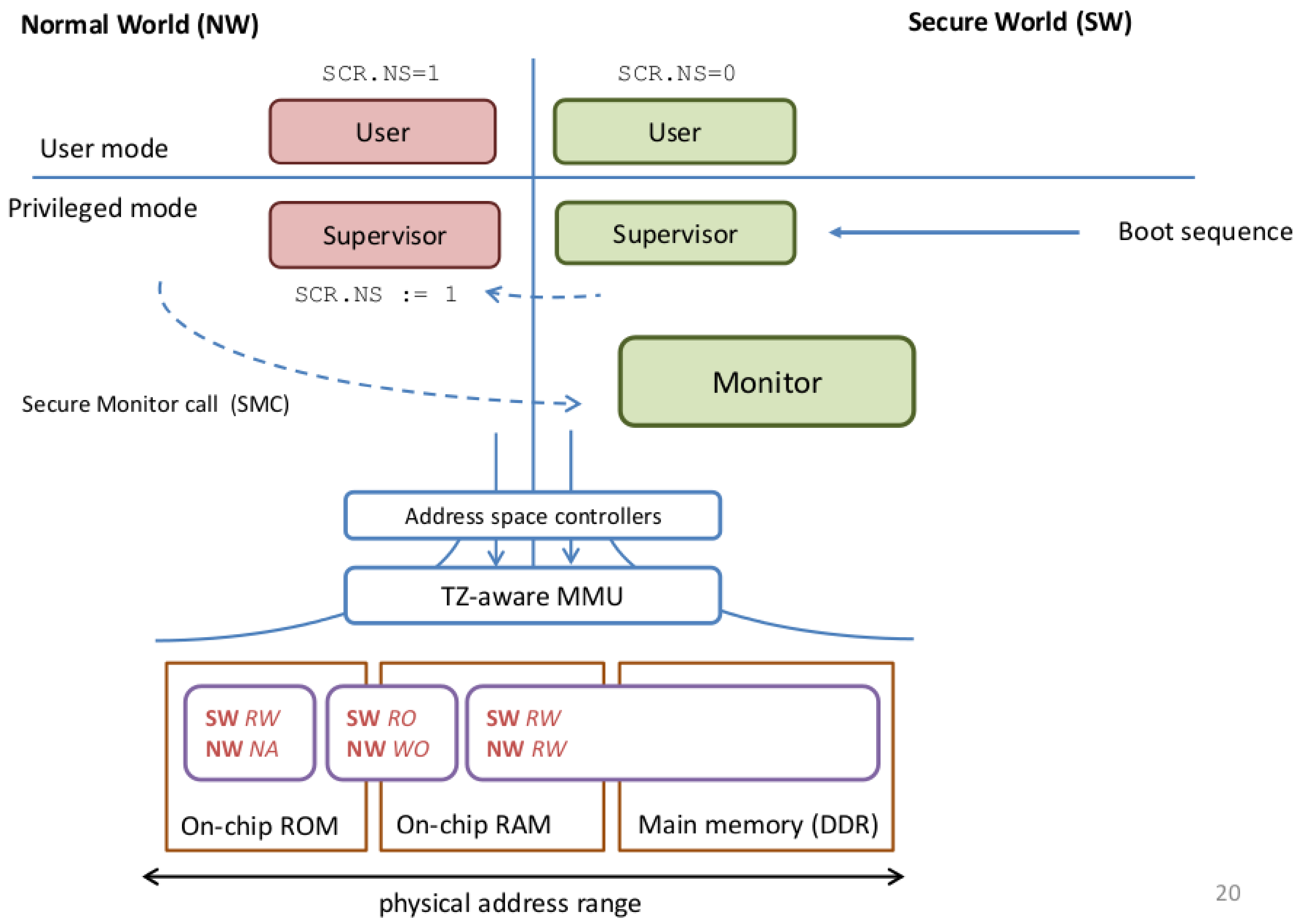
TrustZone

- A new mode of execution of the processor
  - “Secure mode”
- Hypervisor like instructions
- Banked registers
- Very flexible system (for the silicon manufacturer!)

Next slides from “Trusted Execution Environments on Mobile Devices” J.-E. Ekberg, K. Kostiainen, N. Asokan, ACM CCS 2013 tutorial

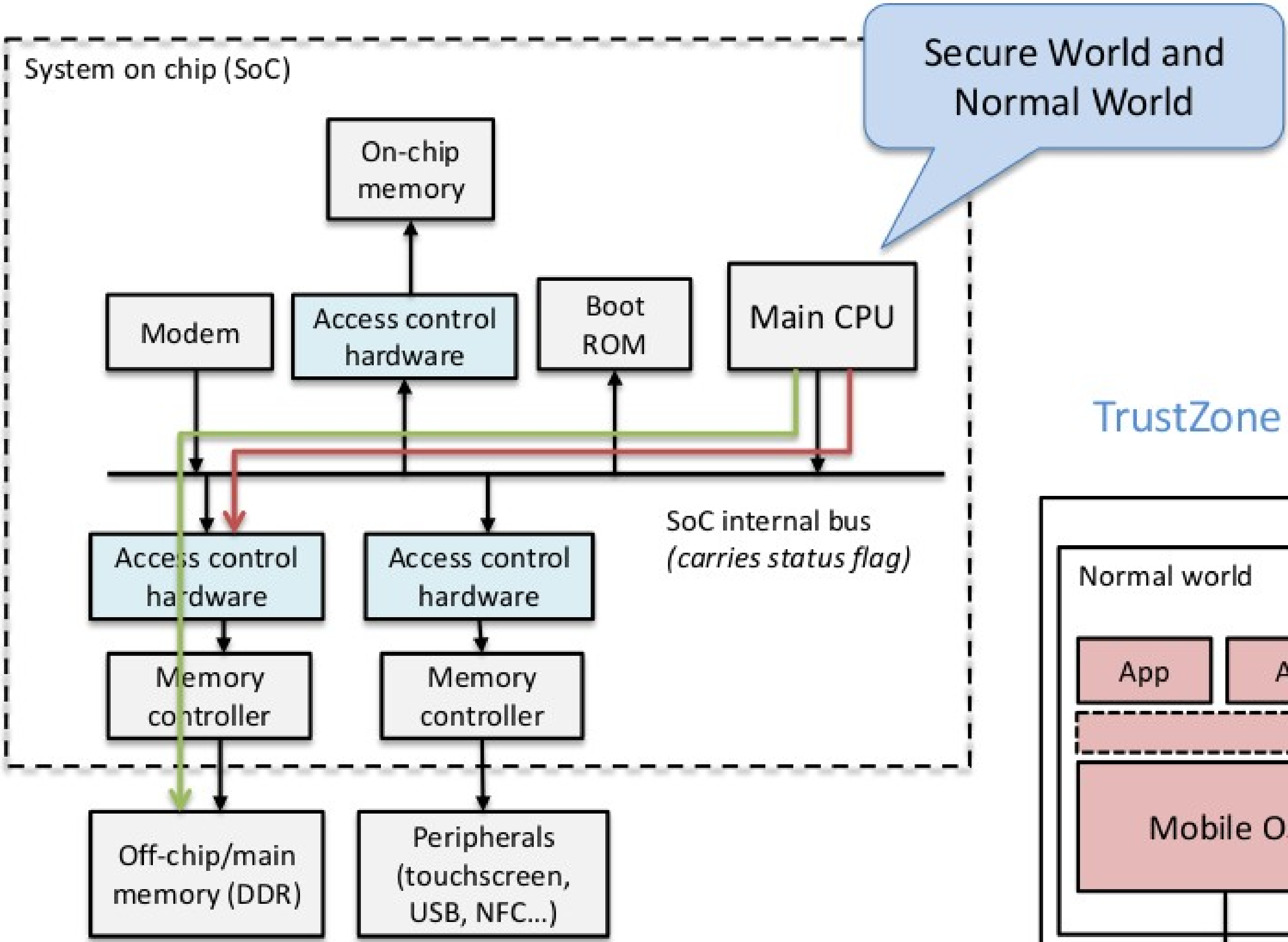
- used with permission (ordering and mistakes mines ...)
- Full original set of slides available online.

TrustZone overview



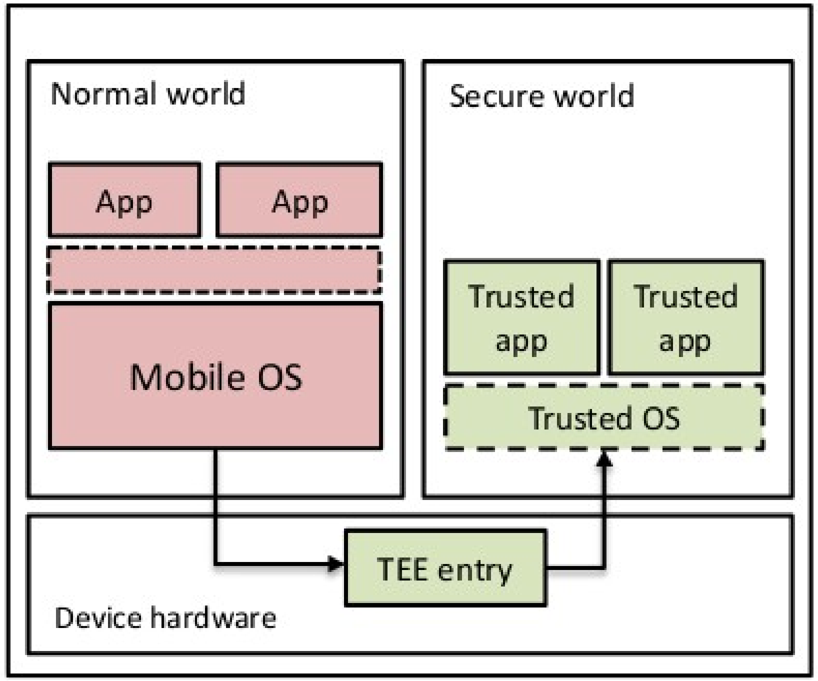
Except from “Trusted Execution Environments on Mobile Devices” Ekberg, Kostiainen, Asokan. Used with permission

ARM TrustZone architecture



TrustZone hardware architecture

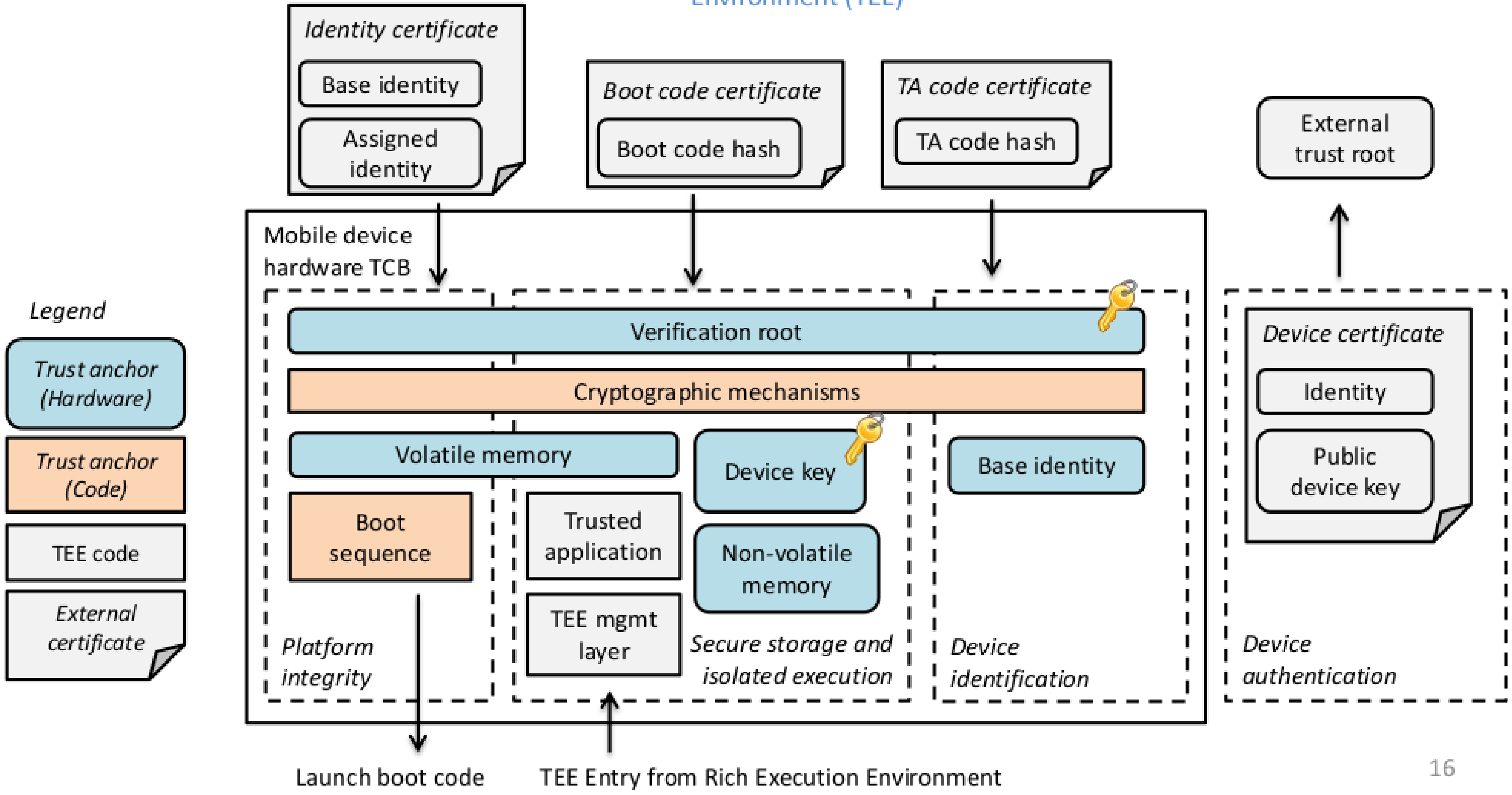
TrustZone system architecture



Except from “Trusted Execution Environments on Mobile Devices” Ekberg, Kostiainen, Asokan. Used with permission

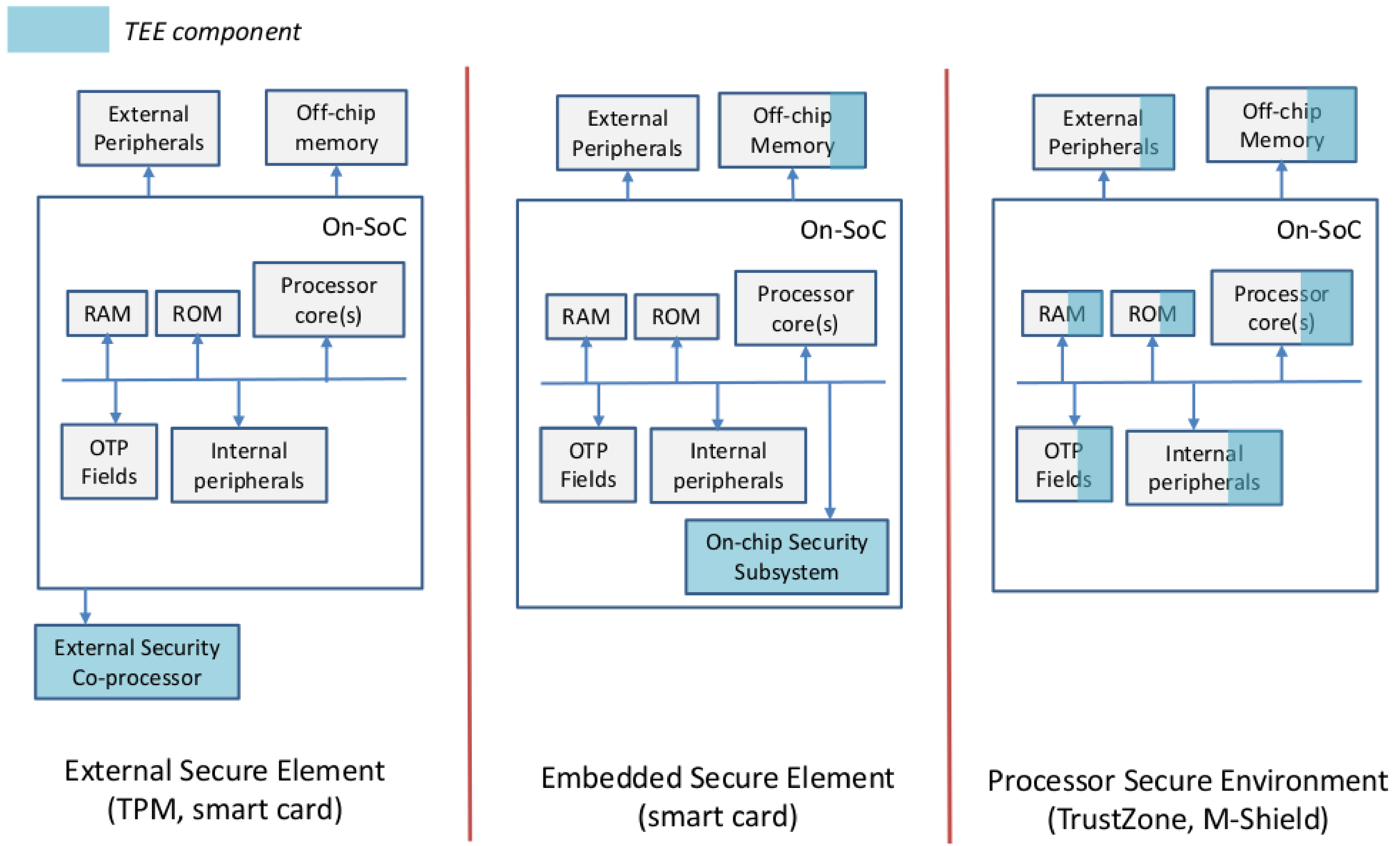
Hardware security mechanisms (recap)

1. Platform integrity
   - Secure boot
   - Authenticated boot
2. Secure storage
3. Isolated execution
   - Trusted Execution Environment (TEE)
4. Device identification
5. Device authentication
   - Remote attestation



Except from "Trusted Execution Environments on Mobile Devices" Ekberg, Kostiainen, Asokan. Used with permission

TEE hardware realization alternatives

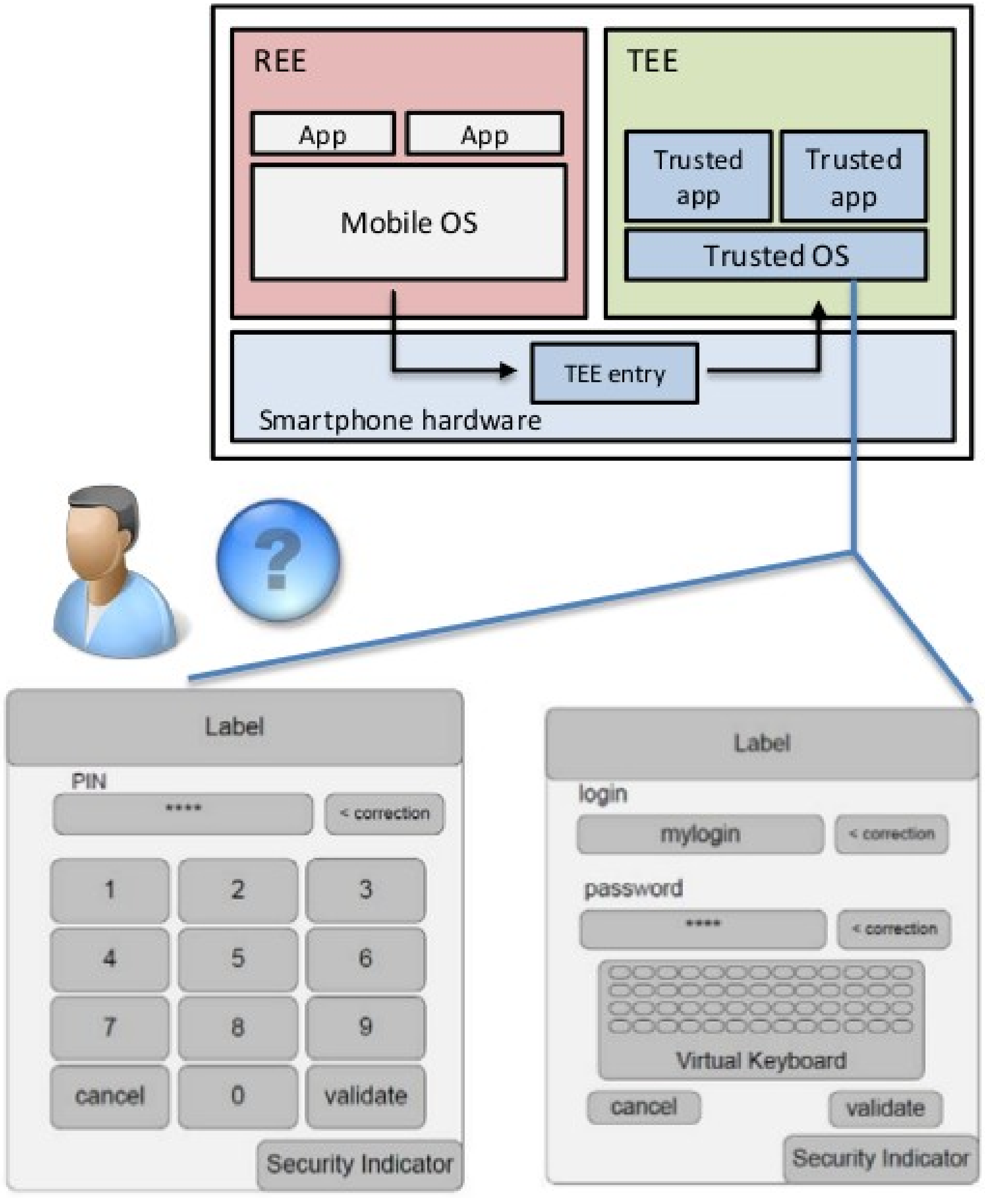


External Secure Element (TPM, smart card) | Embedded Secure Element (smart card) | Processor Secure Environment (TrustZone, M-Shield)

Figure adapted from: Global Platform. TEE system architecture. 2011.

Trusted path to user (GP)

- Trustworthy user interaction needed
  - Provisioning
  - User authentication
  - Transaction confirmation
- Trusted User Interface API 1.0:
  - Set up widget structures
  - Call TEE_TUIDisplayScreen
  - Collect results
- Only for I/O directly wired to to the trusted OS



Except from “Trusted Execution Environments on Mobile Devices” Ekberg, Kostiainen, Asokan. Used with permission

In even smaller systems

Dynamic root of trust in smaller systems ?

- Possibly in some smart cards ?
  - ARM SecureCore ?
  - Other vendors ?

- Possible Custom modification of insecure micro controller core
  - SMART: a flexible small dynamic root of trust
  - Implemented on MSP430 and AVR cores

TAMPRES

- **SMART Goal: providing external verification**
  - Device authentication
  - Attestation
  - Trusted Execution
- **With minimal modifications to a CPU Architecture**
  - Simple modification to the HW
  - Most of the logic in software

- **SMART prototype**
- **Small series chips (MPW)**
- **Working prototype**
  - Interest from industry
- **Next steps at:**
  - Proving properties
  - Verification Techniques

Conclusion

- Relatively simple solutions can provide a quite good level of security
- You should probably not do it in software (or not expect too much)
- Hardware-aided security is not bullet proof
  - Software attacks are still possible and an overall careful design is required
  - Yet those techniques “raises the bar”: attacks are difficult in practice
  - Usually provides limited resistance to Hardware attacks
- Is it largely used in practice ?
  - Do you use it?

References Software based attestation

- Establishing the Genuinity of Remote Computer Systems Kennell and Jamieson, Usenix Security Symposium, August 2003
- Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms Seshadri, Luk, Shi, Perrig, van Doorn, Khosla.
- Side Effects Are Not Sufficient to Authenticate Software Usenix Security U. Shankar, M. Chew, and J.D. Tygar
- On the Difficulty of Validating Voting Machine Software with Software EVT 2007 : R. Gardner, S.Garera, A. Rubin
- A Generic Attack on Checksumming-Based Software Tamper Resistance IEEE S&P: G. Wurster, P. C. van Oorschot, A. Somayaji
- "SWATT: SoftWare-based ATTestation for Embedded Devices.“ In Proceedings of the IEEE Symposium on Security and Privacy IEEE S&P 2004 : A. Seshadri, Luk, Shi, Perrig, van Doorn, Khosla.
- SCUBA: Secure Code Updates by Attestation in Sensor Networks Wise 2006: A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla.
- “On the difficulty of software-based attestation of embedded devices” CCS 2009: Castelluccia, Francillon, Perito, Soriente
- “Memory Corruption Attacks The (almost) Complete History”, Haroon Meer, BlackHat USA 2010
- Paul C. van Oorschot, Anil Somayaji, Glenn Wurster:
- Hardware-Assisted Circumvention of Self-Hashing Software Tamper Resistance. IEEE Trans. Dependable Sec. Comput, 2005

TPM/TXT

- "Bootstrapping Trust in Commodity Computers." B. Parno, J. M. McCune, A. Perrig IEEE S&P 2010
- “OSLO: Improving the security of Trusted Computing” Bernhard Kauer
- Trusted Computing Group http://www.trustedcomputinggroup.org
- System Management Mode : “SMM reloaded ? “, Can Sec West, Loic Duflot
- Attacks on TXT: The invisible things labs http://theinvisiblethings.blogspot.com/
- Users fears about trusted computing well represented on the wikipedia page http://en.wikipedia.org/wiki/Trusted_computing
- FLICKER: MINIMAL TCB CODE EXECUTION https://sparrow.ece.cmu.edu/group/flicker.html
- Intel® SGX for Dummies (Intel® SGX Design Objectives)

  https://software.intel.com/en-us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx