

SYSTEM SECURITY III: TRUSTED AND CONFIDENTIAL COMPUTING

TDDD17 Information Security, Second Course
Ben Smeets

Ericsson Research Security / Lund University

Goal of this lecture

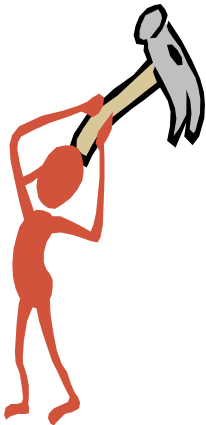
- Understand trusted and confidential computing and its purpose
- Threats to computing HW/infrastructure
- Get a basic insight in technologies to achieve trusted computing in devices, servers, and cloud infrastructure
- Meet technical approaches to build trustworthy ICT systems
 - In the first part you already saw approaches used in operating systems and VMs with access control and the use of memory protection

Overview

- Why trusted computing?
 - Intuitive model for trusted computing
 - Confidential computing – what is it
 - Roots of trust
 - Hardware versus software
 - Confidential computing - how
- CPU secured execution environment:
 - TrustZone,
 - SGX
 - (AMD SEV)

New Security Challenges

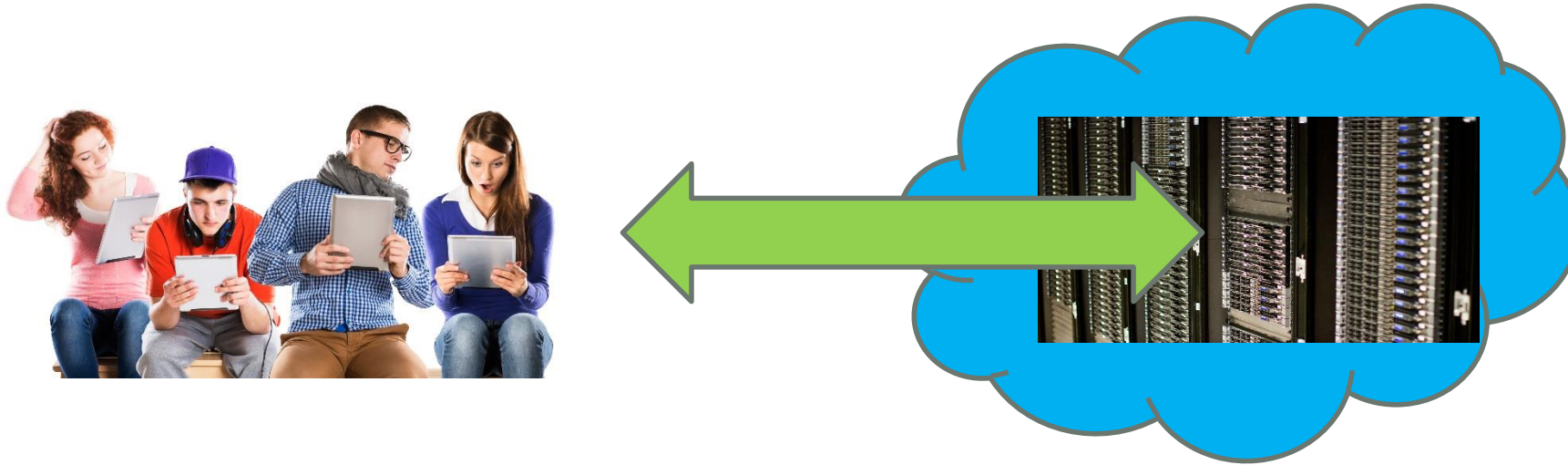
- Computing devices are becoming **distributed, unsupervised**, and **physically exposed**
 - Computers on the Internet (with untrusted owners)
 - Embedded devices (cars, home appliances)
 - Mobile devices (cell phones, PDAs, laptops)
 - Base stations and wireless access points
- Cloud computing
 - Virtualization, containers
 - Web technologies - microservices
- Attackers may **physically tamper** with devices
 - Invasive probing
 - Non-invasive measurement
 - Install malicious software



The main security question from a user's perspective

USER(S)

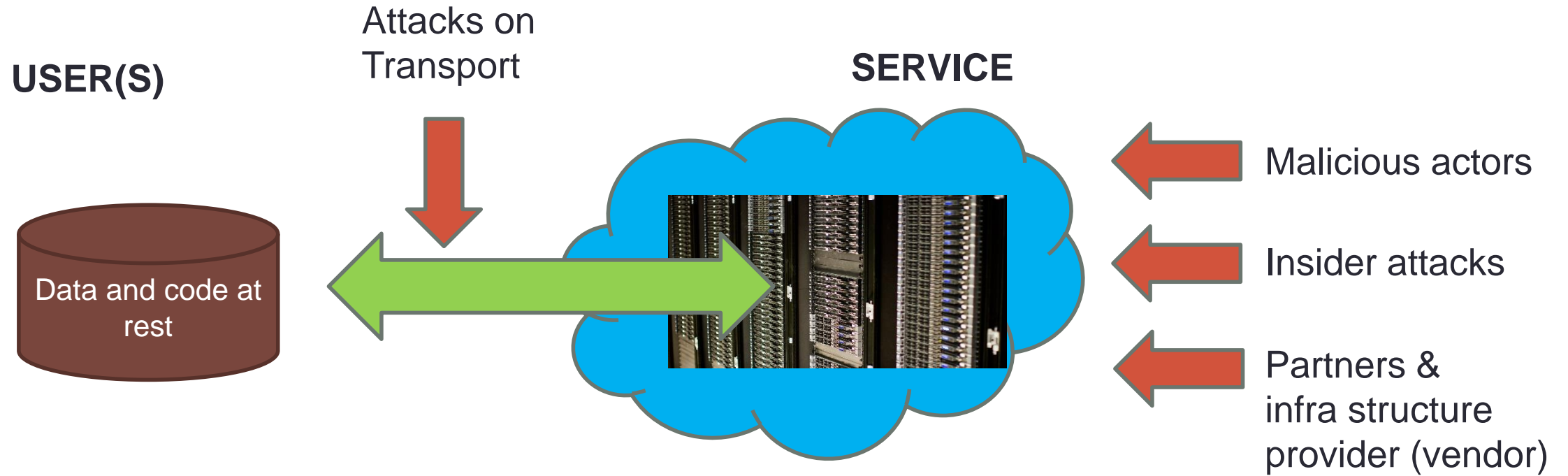
SERVICE



How can we trust the service I'm interacting with?

(we ignore here the questions related to the trustworthiness related to the semantics of data exchanged and processed)

Threats



Protection needed for

- Transport
- When data and code is processed

Important aspects to consider

Zero Trust principle

- Is it really the right service/server I'm interacting with?
- Is the service/server in a proper state so
 - I dare to interact sensitive information?
 - Privacy or Business Data
 - Store keys of (web) services
- Does it comply to business or regulatory requirements?
- **Zero Trust principle:** Instead of assuming trust across a system (perimeter), entities should only engage with other entities after proper verification. See NIST SP 800-207

The typical problems we want to address

- How can we, inside a device/computer, protect sensitive data (and thus also keys)?
- How can we securely insert a key in a remote server for setting up a secure TLS connection?
- How can we do confidential computing, say of patient information, on a remote systems?
 - Protect against threats from malicious actors, insiders, partners and vendors

Trusted Computing

- Trusted computing is a notion for computing where we can provide answers to our first two problem questions.
- There are different approaches to this and there is no well-established agreed precise definition of its properties.
- Other closely related notions are that of
 - Trusted Execution Environments (TEEs),
 - Trusted Platforms, and
 - Confidential Computing

Confidential Computing

- Takes trusted computing a step further to include that data and code can be confidentiality/integrity protected – i.e., we get an answer to the third problem question
- Hence the remote site must have a kind of place where the data and code executes but is confidentiality protected, e.g., through HW security mechanisms,

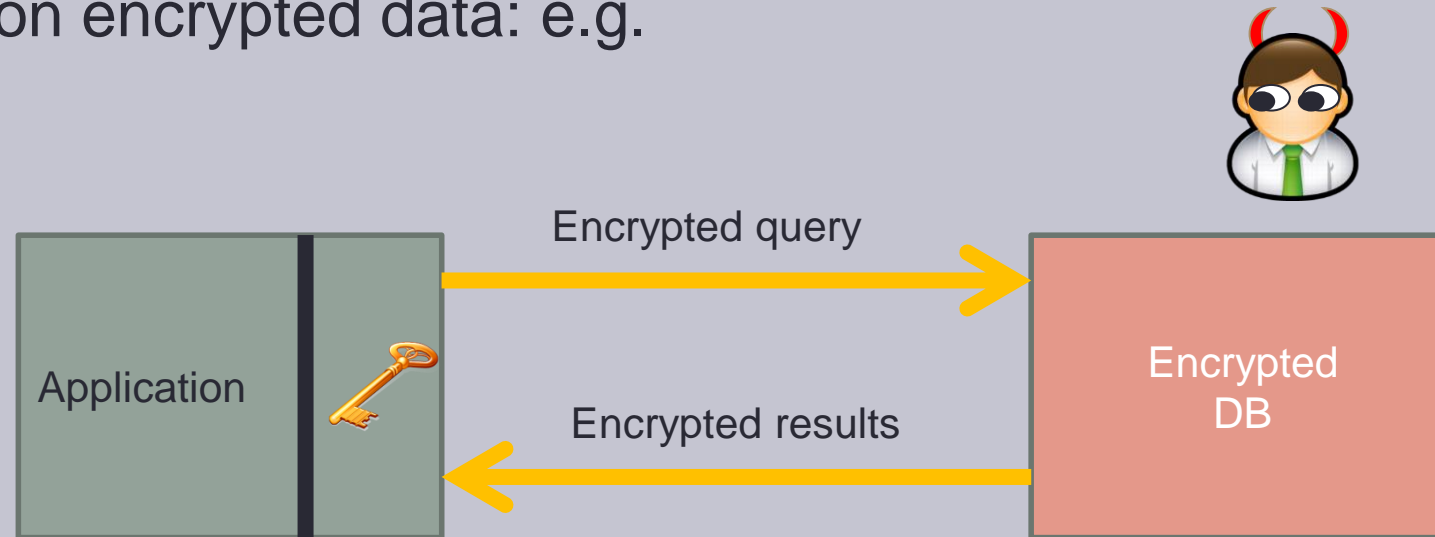
Alternative to trusted computing/platforms

- Secure multi-party computation and homomorphic encryption can be alternatives but, except for special cases these are slow!
- The idea is that the remote server only gets encrypted data and never the original data

Unfortunately secure multi-party computation and Homomorphic encryption is still not practical except for some special (use) cases.

Homomorphic encryption - Processing on encrypted data

- Send data encrypted to remote server and devise a method that does the processing on encrypted data: e.g.



- For example database operations

See <http://css.csail.mit.edu/cryptdb/> Not completely homomorphic encryption based, however

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. [CryptDB: Protecting Confidentiality with Encrypted Query Processing](http://css.csail.mit.edu/cryptdb/).
In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, October 2011.

Trusted vs Trustworthy

What are we after, a trusted or trustworthy platform?

Trusted: Trust is mental view/perception of a system but the question is: IS it trustworthy?

Trustworthy: The system fullfills the requirements defined by a methodology, e.g., evaluation, compliance test.

However: Is the methodology then trustworthy (and we get a recursion) or we just trust the methodology.

Recall: Using Common Criteria a system that is successfully evaluated at level EALx is considered to be trustworthy.

Common Criteria as basis for trustworthiness

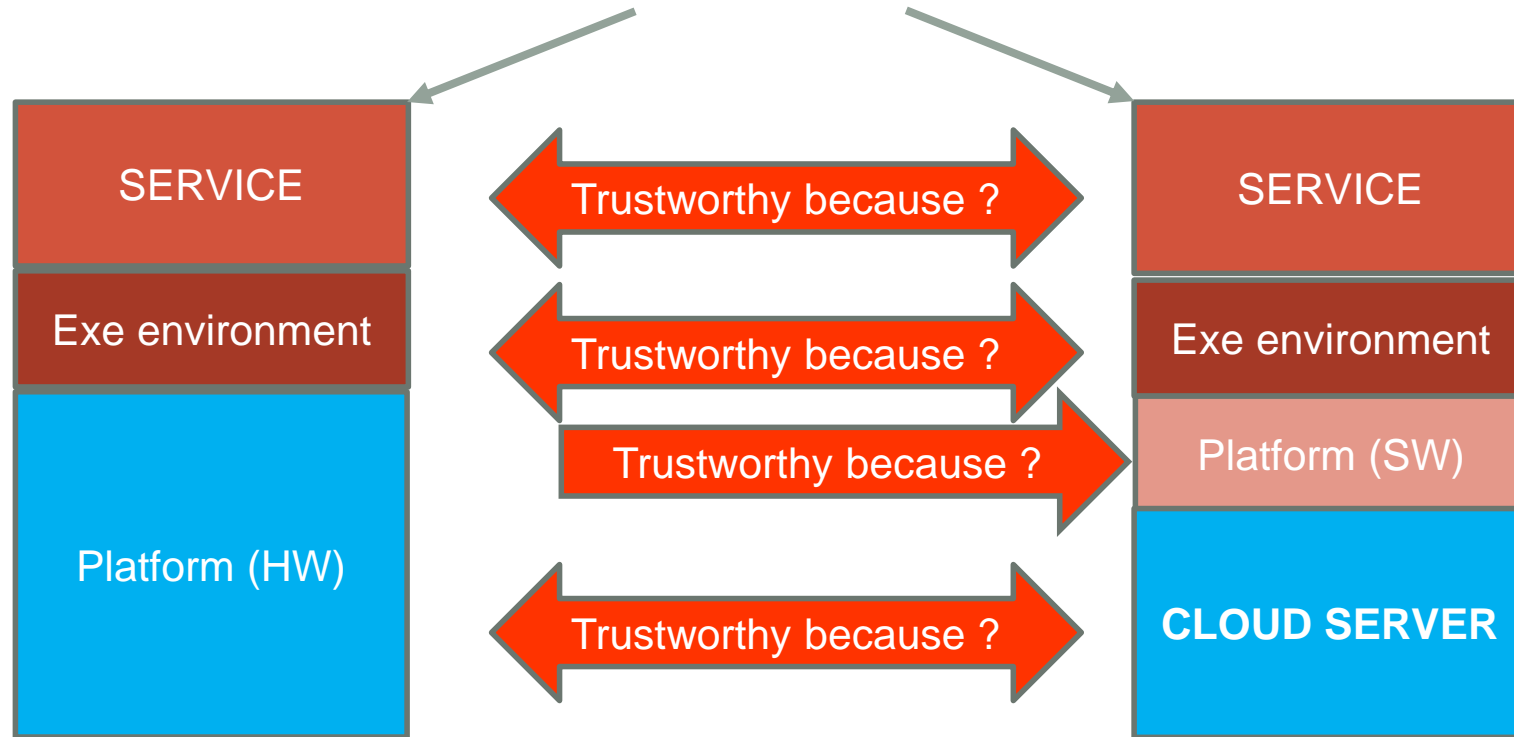
- Common Criteria (CC) is an ISO standard of a methodology to evaluation and certify products according an agreed target set of (security related) requirements
-
- It is used for smart cards, crypto libraries, crypto HW, servers, etc.
- Certification is done via approved certification bodies and an CC certificate holds in any country that accepts the CC scheme.
- In Sweden, see FMV/CSEC
<https://www.fmv.se/verksamhet/ovrig-verksamhet/csec/>

How to obtain trustworthiness ?

Traditional realization

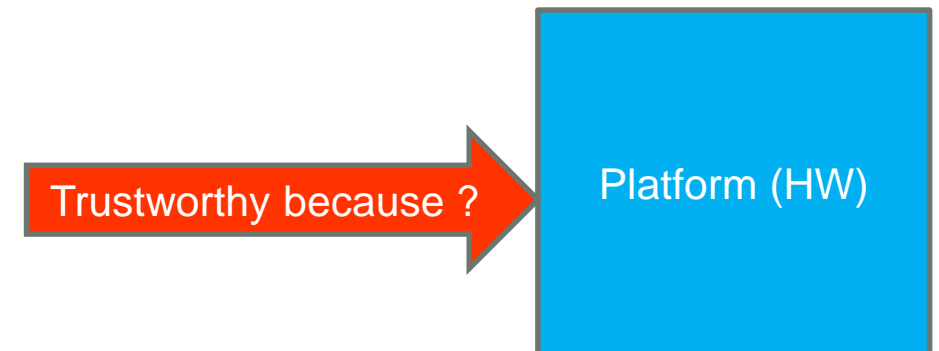
How to deal with
the differences between
cloud and traditional?

Cloud realization

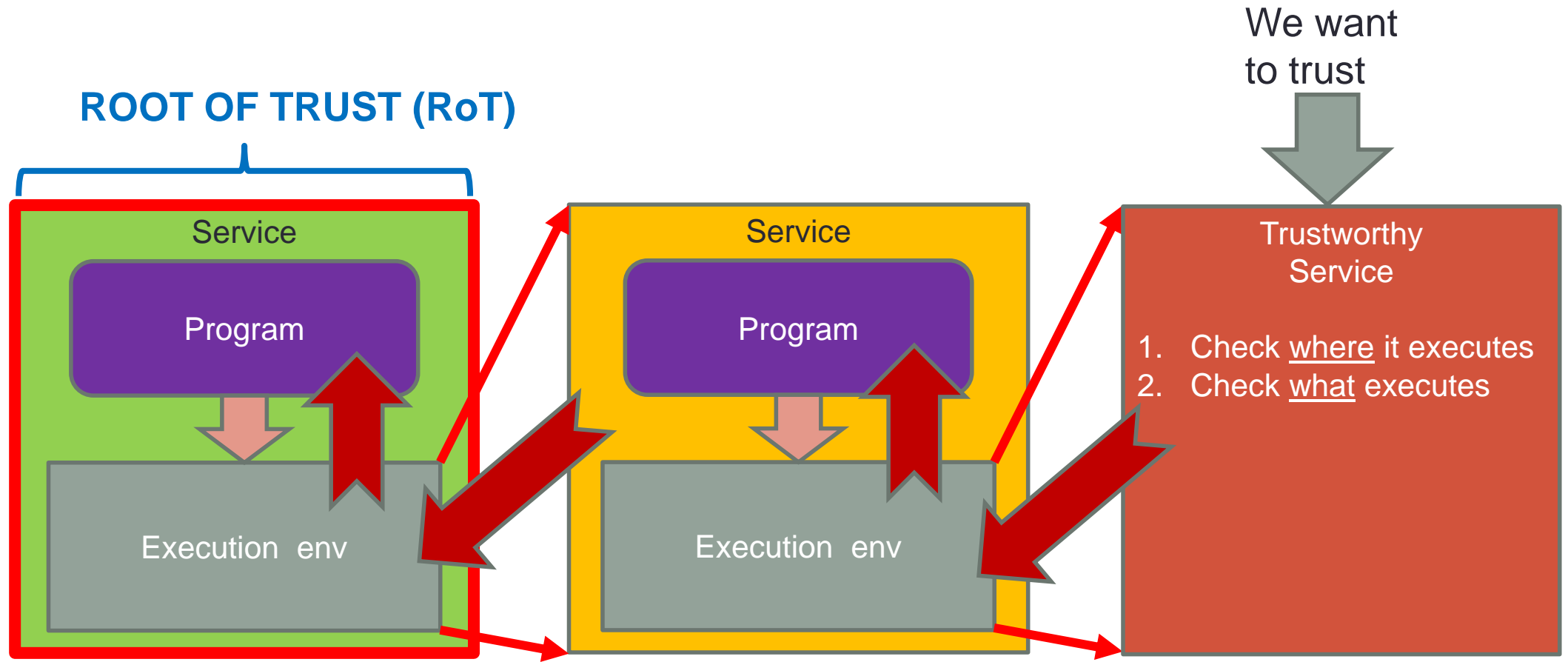


E.g. How & why trust HW

- Trust by reputation vendor (e.g., made by Sectra)
- Trust by relying on a third party (e.g. recommendation)
- Assurance of design
 - Review
 - Proofs (by modeling of HW)
- Assurance of production
 - HW is produced according to design



Start of trust chain – Root of Trust(RoT)



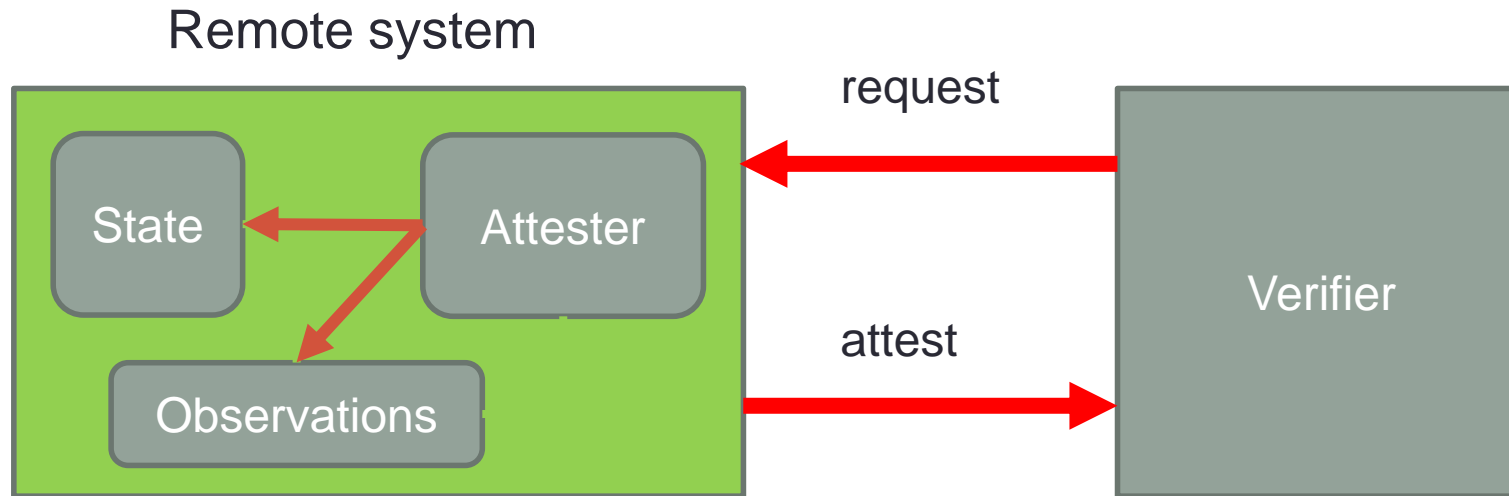
Recursion must stop at a service we trust/have to trust, e.g. Intel HW.

Note: RoT is not only data (e.g. keys) but also logic, therefore we say that a RoT is an engine.

Trustworthy at distance: Remote attestation

Purpose is

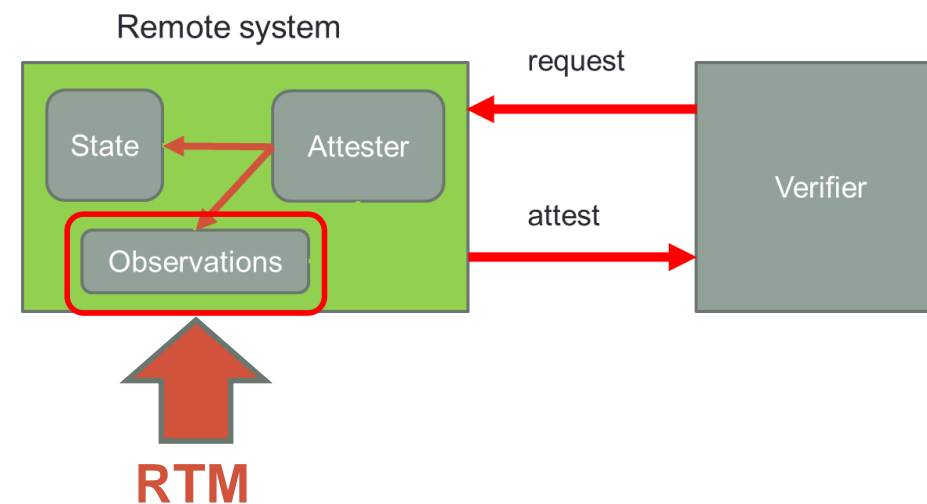
- to establish a trust relation(e.g. a secure channel) to a specific remote processing system and
 - to know it is running the correct code/data
- Provide secure information of a system's state to a remote party



Note: similarity to a challenge-response based authentication

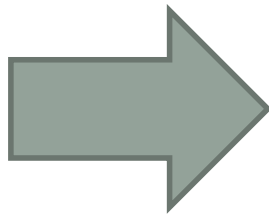
RTM – Root of Trust of Measurement

- If we collect the “observations” which are called measurements in a secure way and keep them secured (so they cannot be manipulated) we have built a Root of Trust of Measurement
- The RTM can be used to create attestation reports based on these measurements



Trustworthy: Hardware vs Software

- Functionality in Hardware
 - hard/costly to change
 - high performance possible
- Functionality in Software
 - Easy to change
 - Difficult to hold private keys



The general view is that HW is more trustworthy than SW realizations

Trustworthy Systems in Software

- **Possible to do** but we have limitations
 - owner of the device on which software runs should not be an attacker (he/she and the device "work together"/"have the same interests")
 - Does not work when the device is in the "enemy's territory"
- But "software only" is sometimes the only implementation option: e.g. virtual platforms

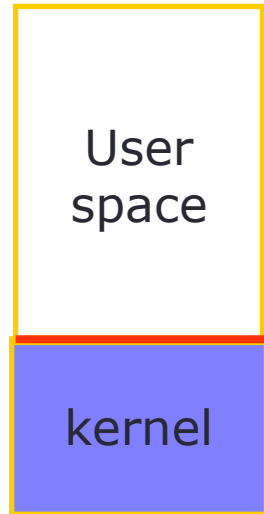
Trusted Execution Environments(TEE)

- Solutions to have best of both, using soft- and hardware protection mechanisms
 - Hypervisor (also called Virtual Machine Monitor (VMM))
 - attestation through virtual device
 - Modify OS
 - try to create isolation (VMs, Containers or OS features)
 - Dockers, SystemD, SE Linux

Our focus

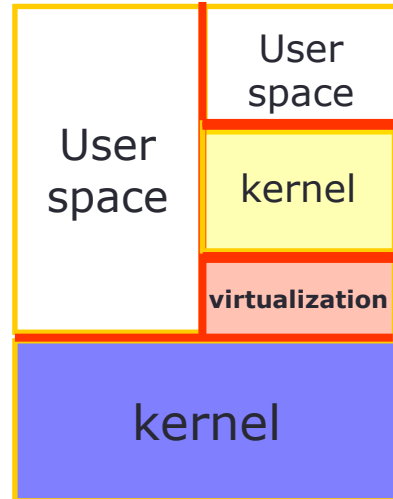
- Modify existing hardware (CPU, memory controllers, etc)
 - attestation done by hardware module
 - add secure execution mode to CPU

Execution environment setups for a trustworthy platform



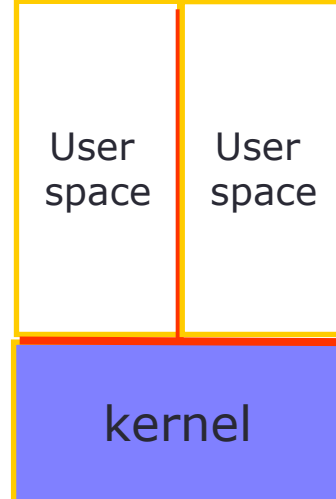
Normal OS

Windows, Linux
SE Linux,
Android
iOS



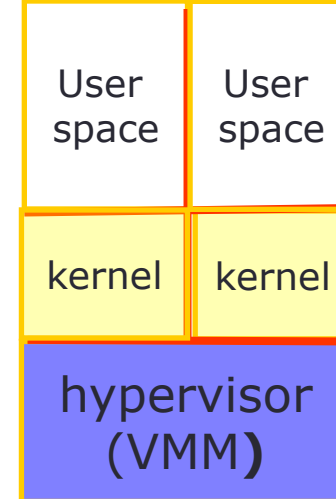
Virtual Machine

VMWare, KVM,
Virtualbox,
Java VM



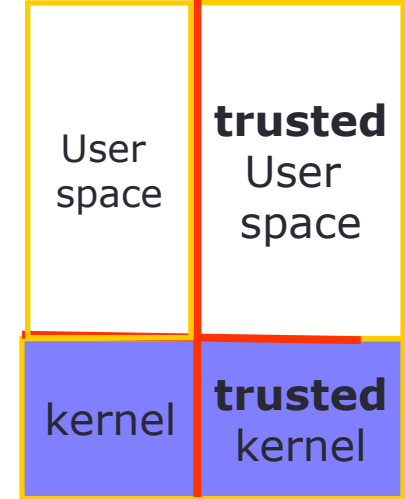
Containers

Docker, LXC
systemd



Hypervisor/VMM

Xen, VMware ESXi,
Microsoft Hyper-V
(L4)



**CPU with
trusted mode**

e.g. TrustZone
and Intel SGX

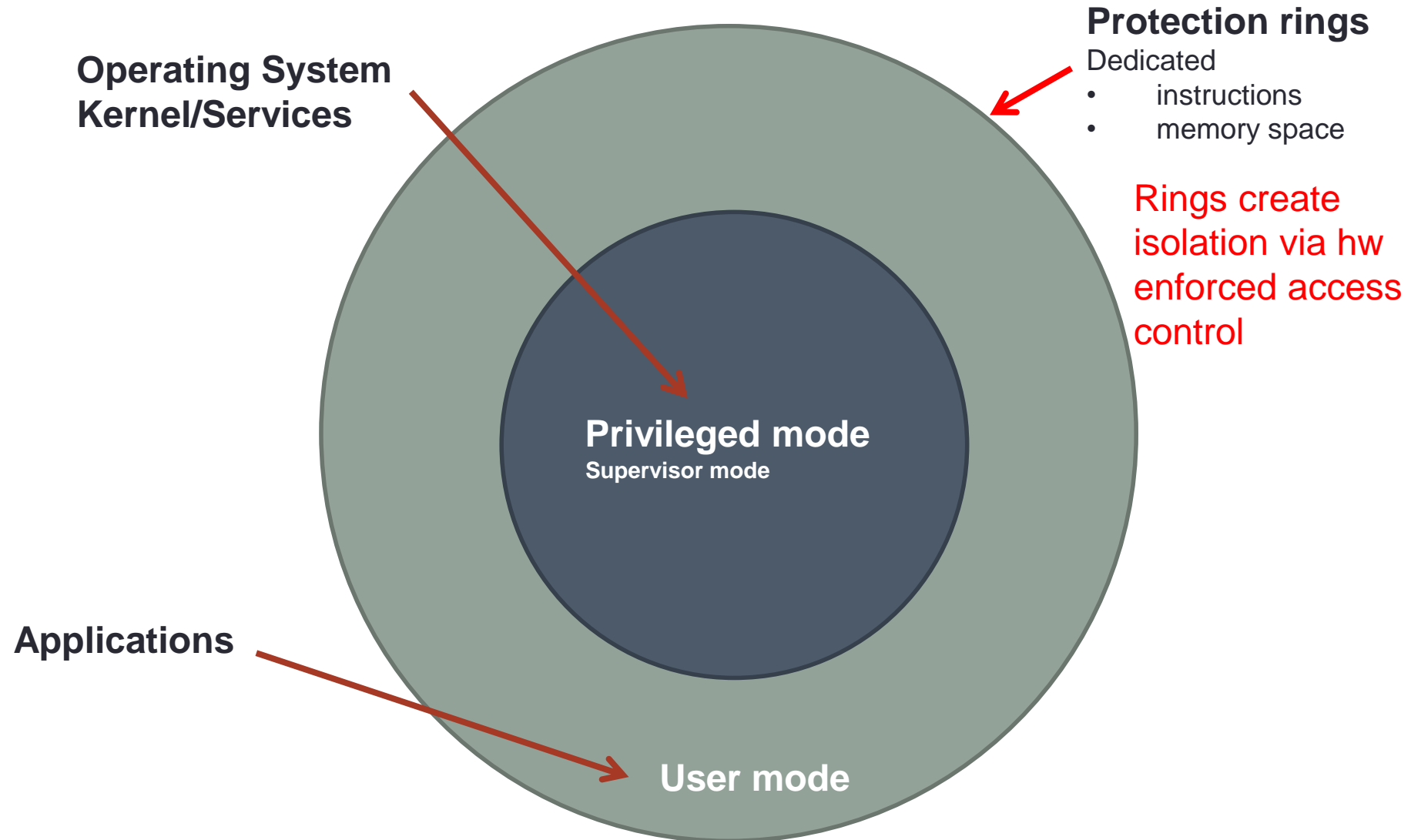
Examples of approaches to CPU/HW supported trusted computing

- ARM TRUSTZONE
 - Basic idea of TZ
 - Trustzone use
 - Trustzone shortcomings
- Intel SGX
 - Basic ideas and concepts of SGX enclaves
 - Secure key delivery
 - Local and remote attestation
 - Two examples where SGX is used
 - SGX shortcomings

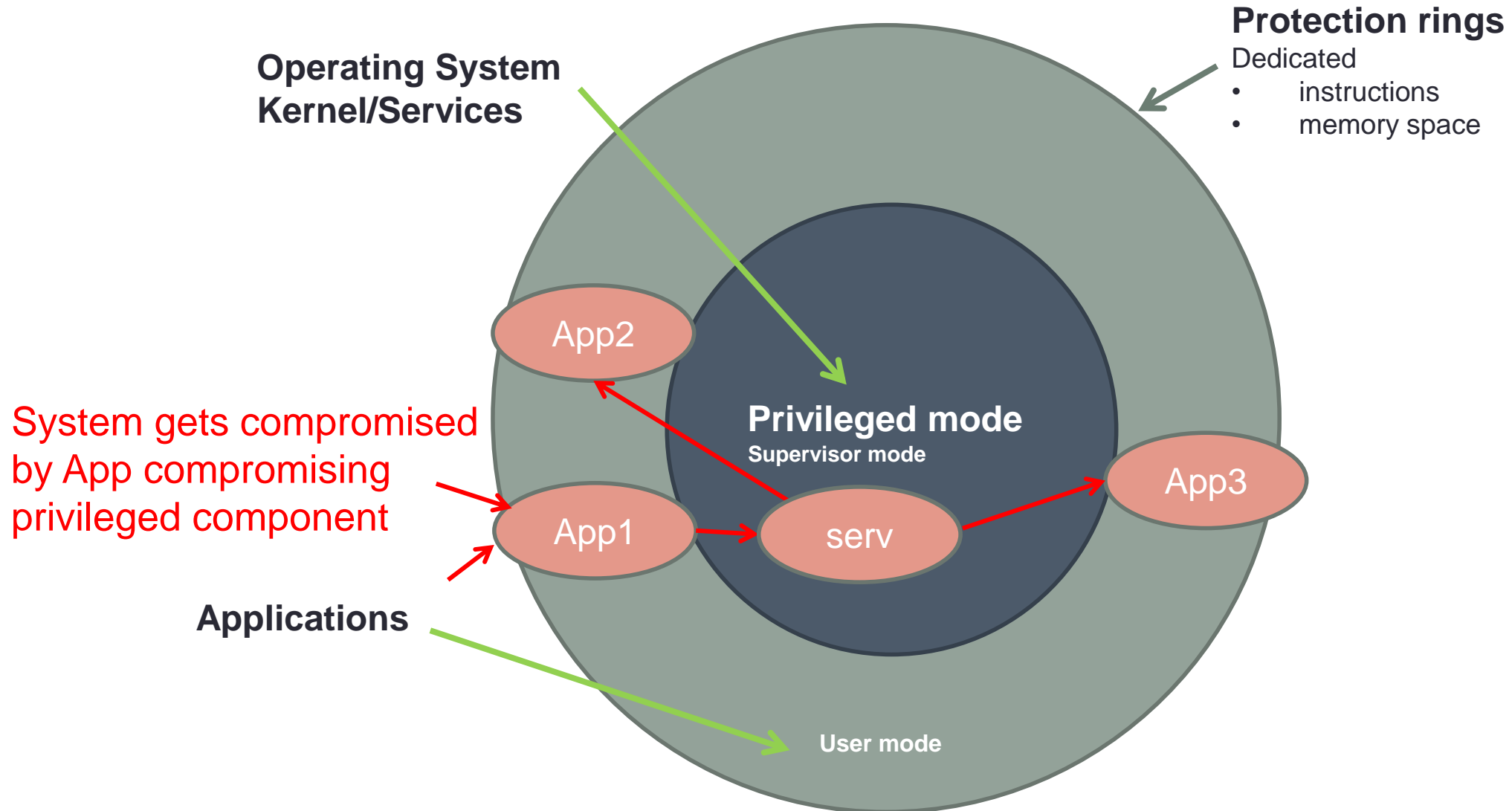
ARM TRUSTZONE

TrustZone is a set of security extensions added to ARMv6 processors and greater, such as ARM11, CortexA8, CortexA9, CortexA15 and now Cortex-M. To improve security, these ARM processors can run a secure operating system (secure OS) and a normal operating system (normal OS) at the same time from a single core.

ARM standard approach



Security problem for applications

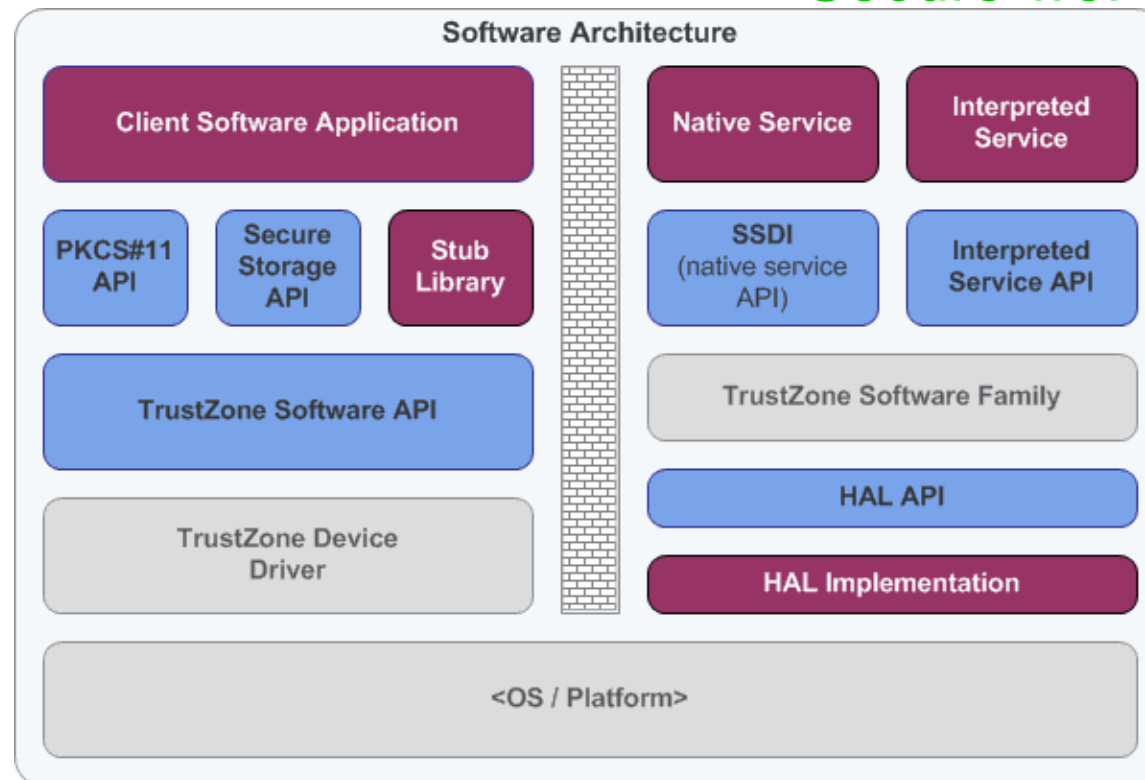


ARM TrustZone

- A special mode of operation for the ARM11 processor
- Divides the SoC into “normal world” and “secure world”

Normal world

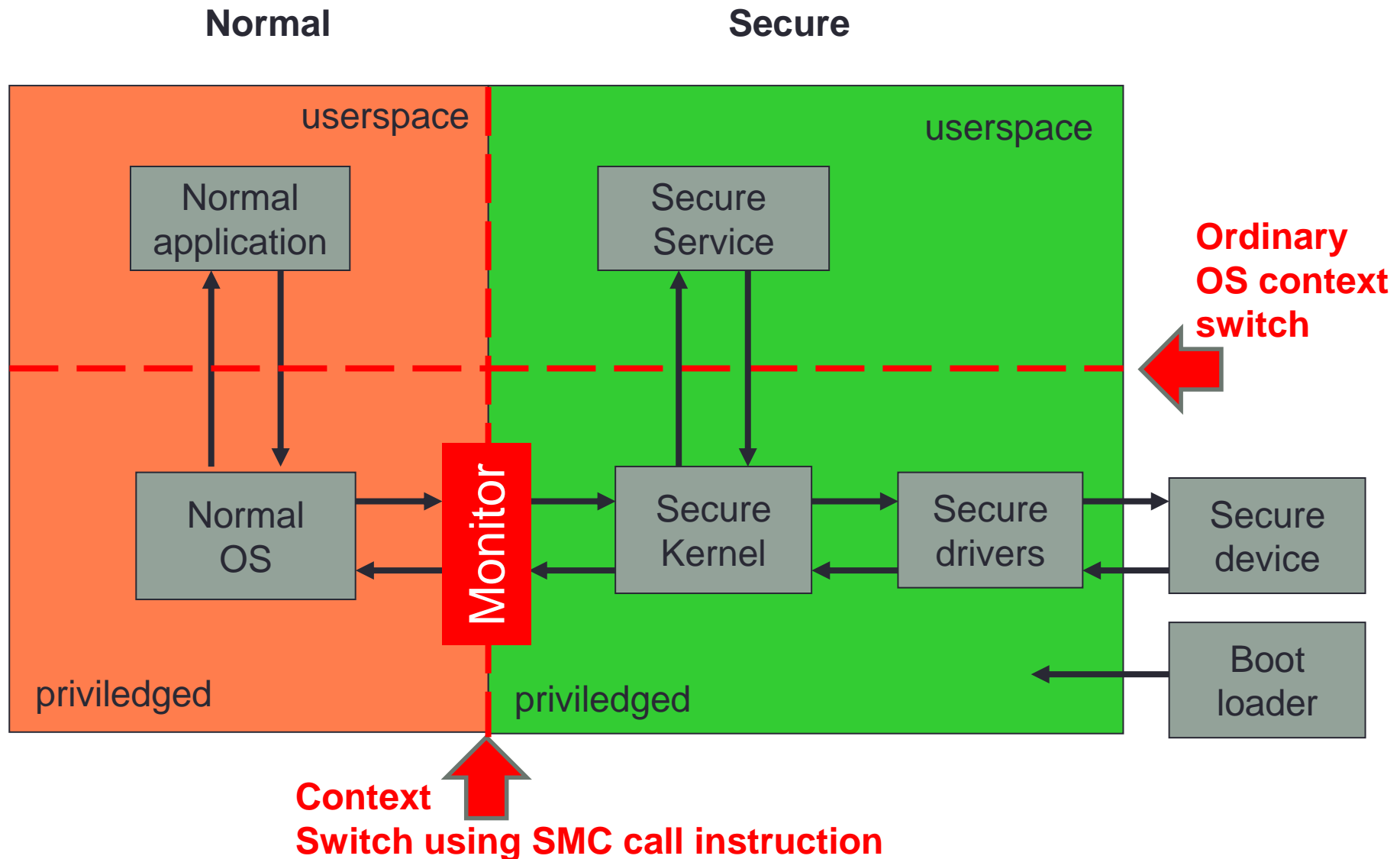
Secure world



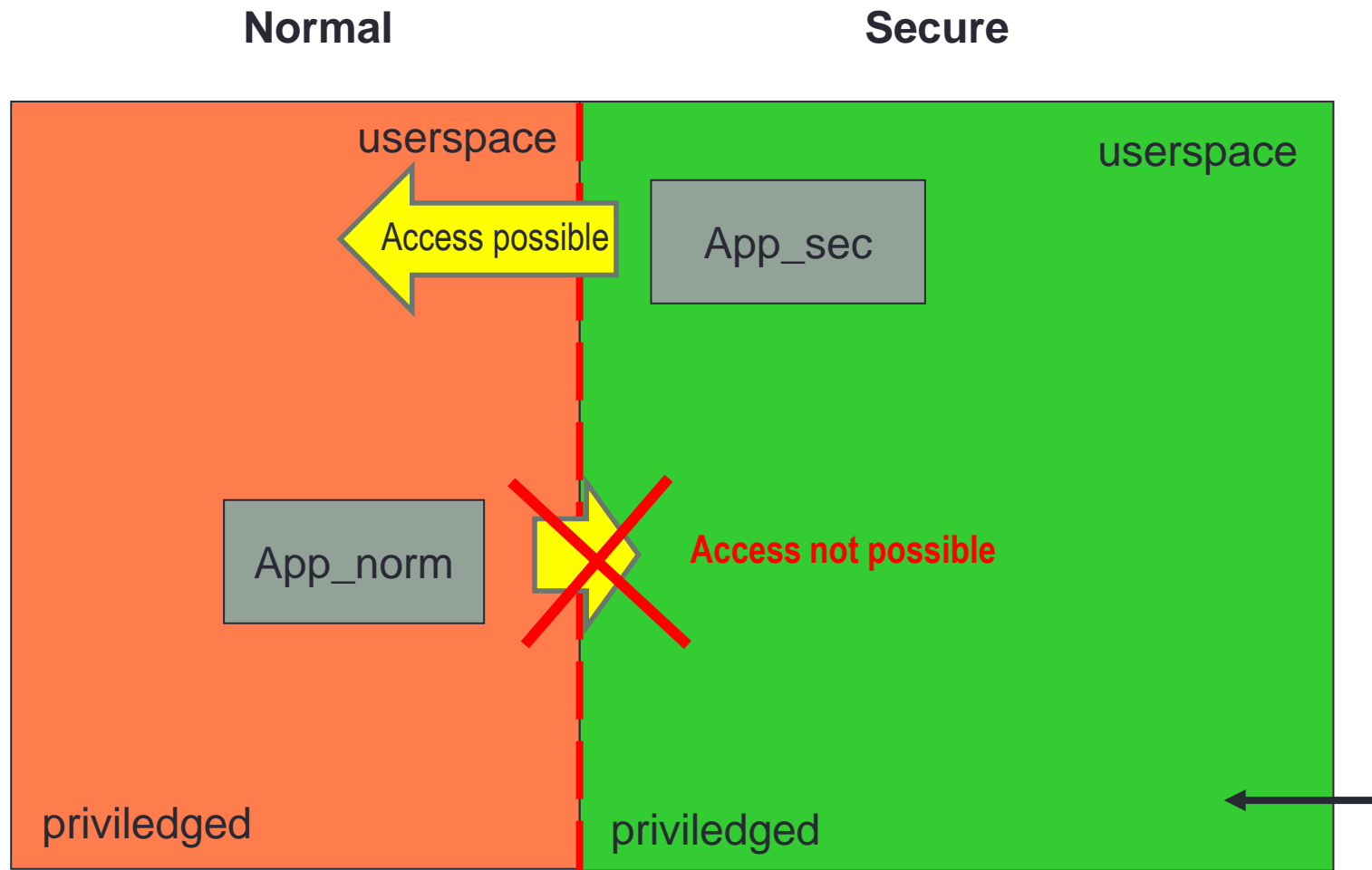
Basic idea

- **Introduce an NS-bit**
 - use this bit to **tag** secure data throughout system
 - Buses, cache, pages
- **Monitor**
 - manages the NS-bit
 - manages transition in & out of security mode
 - Small fixed API (so we can better check/verify the code)
- **Isolation**
 - HW enforced
 - Processes in normal world cannot access/use data/resources that are tagged as belonging to the secure world
 - Processes in secure world can access normal world but ring protection is still present
- **Secure interrupt**
 - that forces execution to proceed in secure world

Switching from Normal to Secure: monitor



Isolation in TrustZone

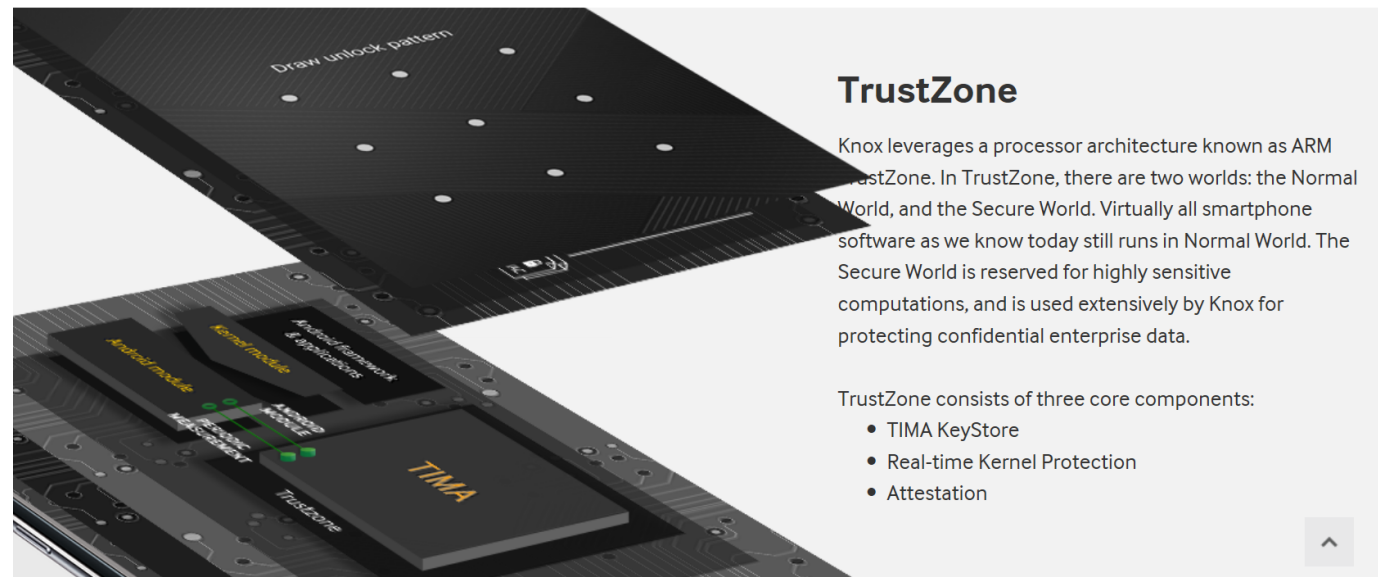


TrustZone use

Widespread in use in smartphones using Qualcomm and Samsung chipsets

Forms a core of Samsung's KNOX solution

- <https://www.samsungknox.com/en/secured-by-knox>



Shortcomings of Trustzone

- Since the TZ system is not an isolated part on the ASIC it is practically impossible to get high EAL levels in the Common criteria framework nor in the US NIST security levels for HW , FIPS 184-2, Security Requirements For Cryptographic Modules
- Isolation of multiple apps in secure world and handling of multiple threads ???
- Secure boot of system and thus the setup of the TZ system is not part of the TZ solution and must be addressed by the chip maker that used TZ in his ASICS and the final device vendor (e.g. Samsung, Sony)

SGX - ENCLAVES

Software Guard eXtensions

SGX is a new technology introduced in Intel chipsets
SGX architecture includes 17 new instructions, new processor structures and a new mode of execution (additional extensions for servers are upcoming).

Overview - SGX characteristics

The new Intel CPU HW features:

- Include loading an enclave into protected memory, access to resources via page table mappings, and scheduling the execution of enclave enabled application. Thus, system software still maintains control as to what resources an enclave can access.
- An application can be encapsulated by a single enclave or can be decomposed into smaller components, such that only security critical components are placed into an enclave.

Enclave technologies

We will look specifically at Intel SGX enclave technology. SGX has its specifics pros and cons but we use it here as a generic example of TEE.

Other technologies are

- AMD SEV (secure encrypted virtualization) (of which there several variants)
 - Intel TDX (trust domain extension)
-
- These technologies are not technically equivalent but can be viewed as equivalent when viewed as generic TEEs

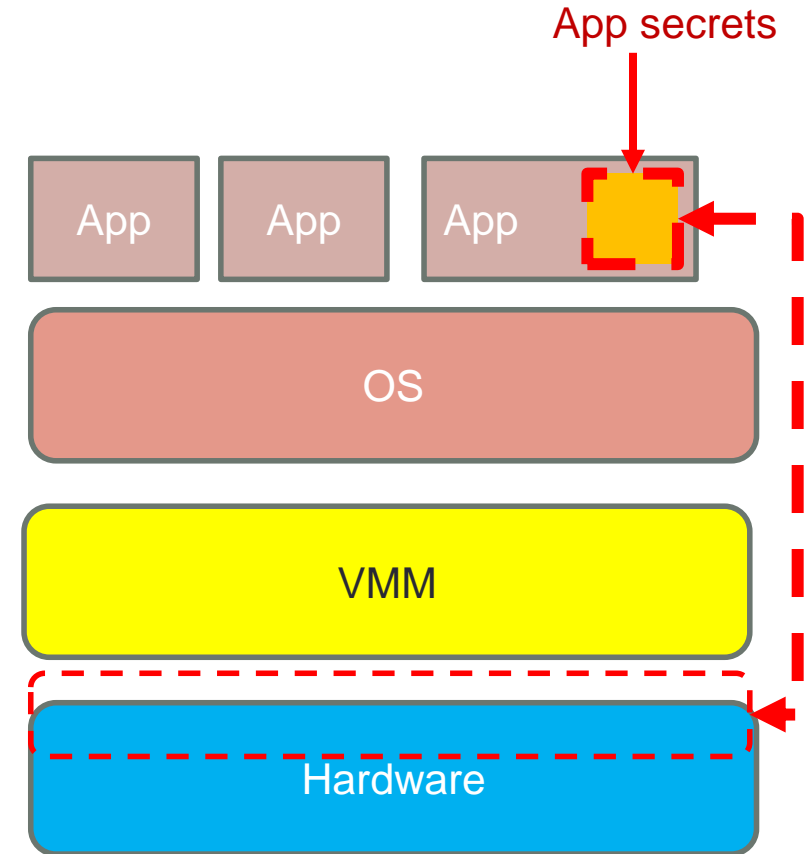
Enclaves

- Enclaves are isolated memory regions of code and data
- One part of physical memory (RAM) is reserved for enclaves and is called Enclave Page Cache (EPC)
- EPC memory is encrypted in the main memory (RAM)
- EPC is managed by OS or VMM
- Trusted hardware consists of the CPU Die only

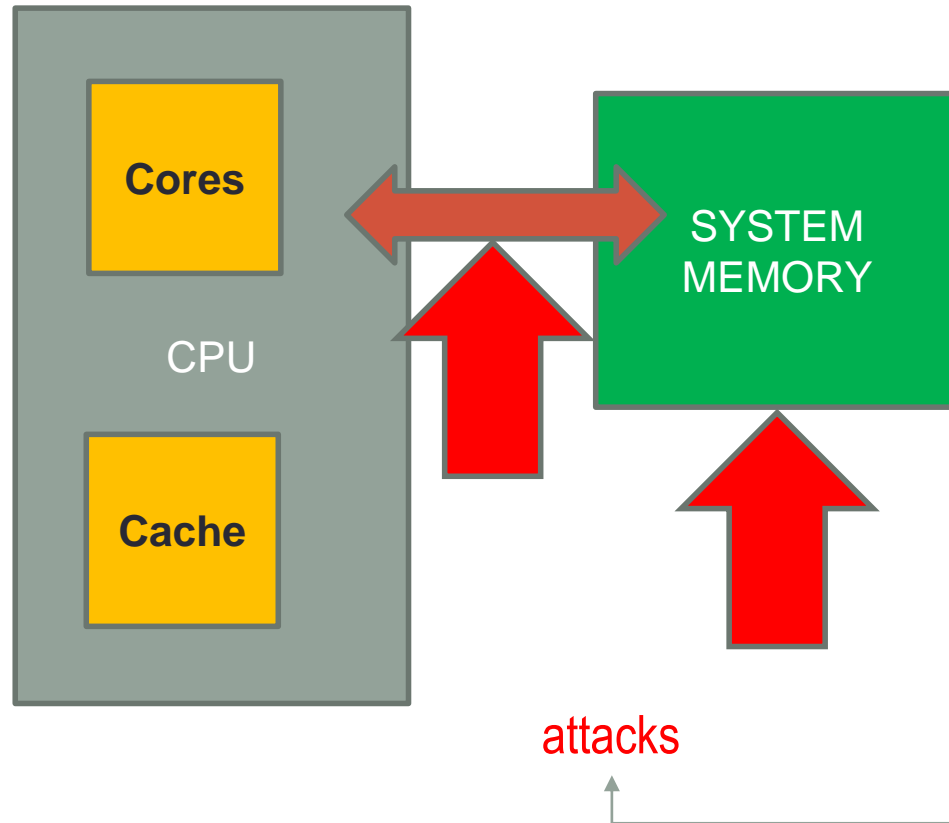
More info see this good overview (but very detailed) paper:
Victor Costan and Srinivas Devadas, SGX explained:
<https://eprint.iacr.org/2016/086.pdf>

Reduced attack surface with SGX

- Application gets ability to defend its own secrets
 - Smaller attack surface (App enclave+processor)
- Malware that subverts OS or VMM, BIOS, drivers cannot steal app secrets
- Insiders in OS/VMM gain no access to secrets

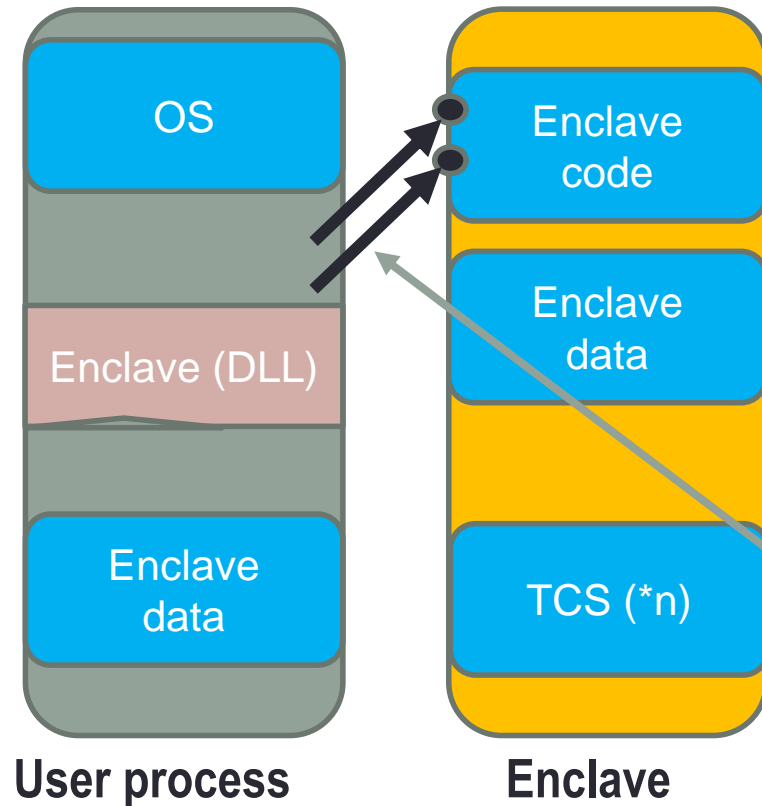


Protection against Memory Snooping



1. Security perimeter is the CPU package boundary
2. Data and code unencrypted inside CPU package
3. Data and code outside CPU package is encrypted/integrity protected,
4. External memory reads and bus snoops tapping gives access to encrypted

SGX Programming Environment



**Protected execution environment
embedded in a process**

- With its own code and data
- Provide confidentiality and integrity protection
- Support for multiple threads
- With full access to app memory
- Dedicated controlled entry (call) points into enclave (ecalls)

TCS= Thread Control Structure

ECALL and OCALL

Interactions with enclaves goes via what Intel defined as ECALLs and OCALLs:

- **Enclave Calls (ECALLs)**

(calls from applications **into the enclave**)

- The application can invoke a pre-defined function inside the enclave, passing input parameters and pointers *to shared memory within the application*.

- **Outside Calls (OCALLs)**

(calls from **enclave to its application**)

- When an enclave executes, it can perform an OCALL to a pre-defined function in the application. Contrary to an ECALL, an OCALL cannot share enclave memory with the application, *so it must copy the parameters into the application memory* before the OCALL.

Some insights how SGX practically works

- SETUP, we have:
 - An SGX enabled HW
 - An Independent Software Vendor (ISV) that delivered applications with enclaves
- IMPORTANT NOTIONS
 - Launch Authority
 - SGX Keys
 - MRENCLAVE (TEE instance/Enclave fingerprint)
 - MRSIGNER (SW in TEE fingerprint)
 - Attestation

Launch Authority

To launch (create) an enclave on an ASIC it must be authorized by a so-called Launch Authority.

- New SGX servers have various way to realize this (and use BIOS to select mode)
 - By Intel
 - By Infrastructure owner, (e.g. Microsoft Azure cloud)
 - By Actor running a dedicate LA application

Also test mode available

MRENCLAVE (TEE instance fingerprint)

- **Enclaves identity is defined by a SHA-256 hash digest of its loading activity procedure.**
 - This includes the information of enclave's code and data, as well as meta-data (i.e. relative locations of each page in enclave's stack and heap regions, its attributes and security flags, et cetera).
- **This cryptographic log of enclave's creation process forms a unique measurement called MRENCLAVE**
 - that represents a specific enclave identity.
 - The MRENCLAVE typically tells info about
 - HW version,
 - ASIC/SGX Firmware version,
 - enclave configuration params

Measurement is basically a recorded cryptographic hash

MRSIGNER

- **MRSIGNER is a notion that reflects enclave's sealing authority.** The sealing authority signs the enclave sw
- This value is represented by a hash over sealing authority's public key (it is part of so-called enclave's SIGSTRUCT certificate).



SGX keys

- The SGX system needs various keys. Some are programmed (by fuses) into the HW and others are derived as needed via EGETKEY calls

- HW

- Root Provisioning Key (RPK)
 - Root Sealing Key (RSK).

- EGETKEY:

- Symmetric Key for sealing
 - Symmetric Key for reporting

based on a process that combines

- enclave parameters and
 - system parameters.

} Unique for
Enclave instance

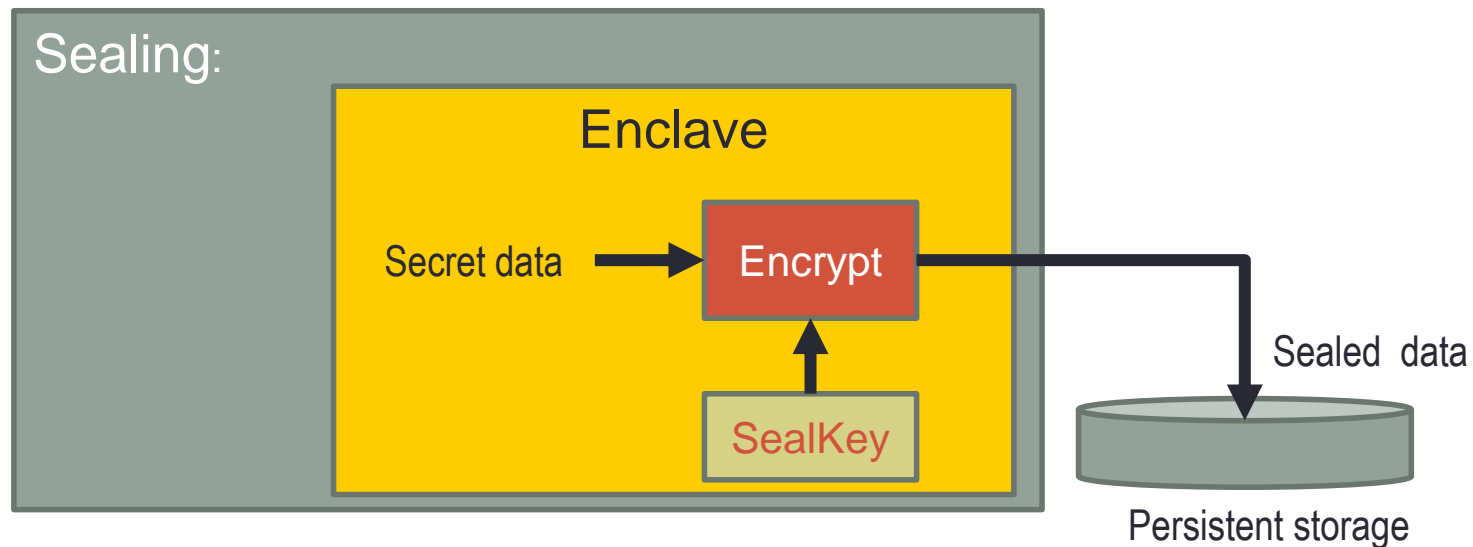
Not so relevant anymore

In SGX1.0 Intel computes the RPK as an EPID type key. For newer SGX versions there will be alternatives. Intel maintains a database of issued RPKs to facilitate a proof that an SGX ASIC is genuine.

Intel claims they have no knowledge of the RSK

Sealing (of secret data)

- Sealing is the process of encrypting enclave secrets for persistent storage to disk. Encryption is performed using a **private Seal Key that is unique to that particular platform and enclave**, and is unknown to any other entity (also other enclaves!)

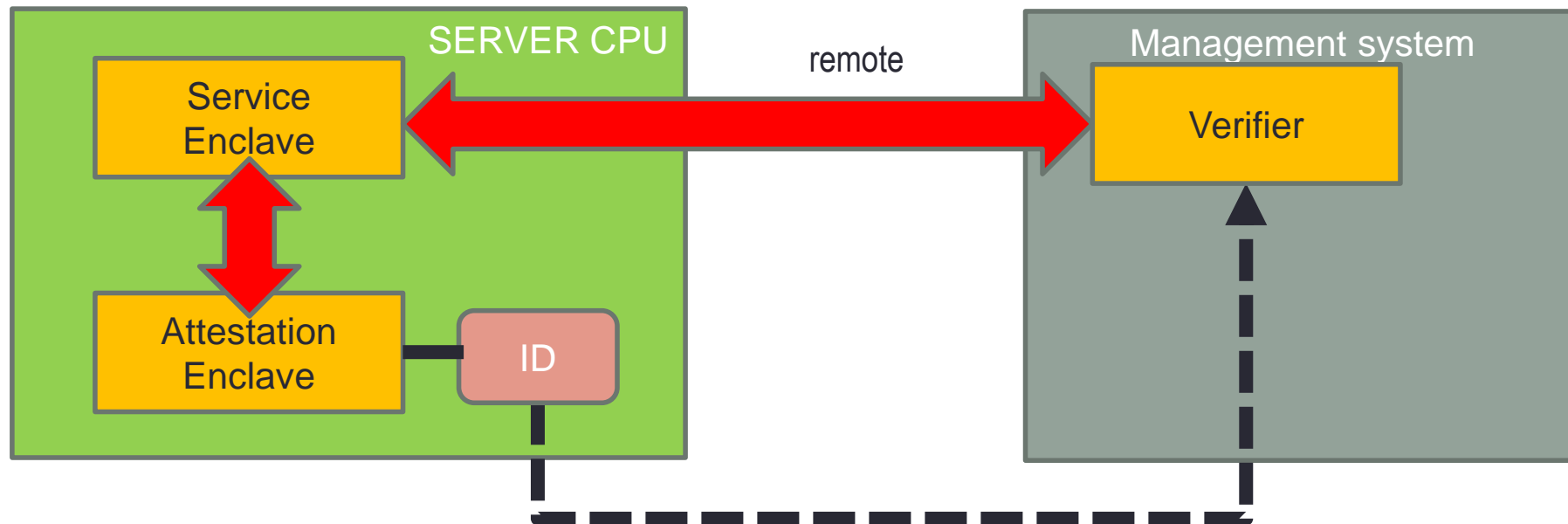


Pros and Cons ?

Sealkey is derived via EGETKEY

Attestation

- SGX supports also attestation of enclaves using the reporting key BUT does so indirectly when doing remote attestation

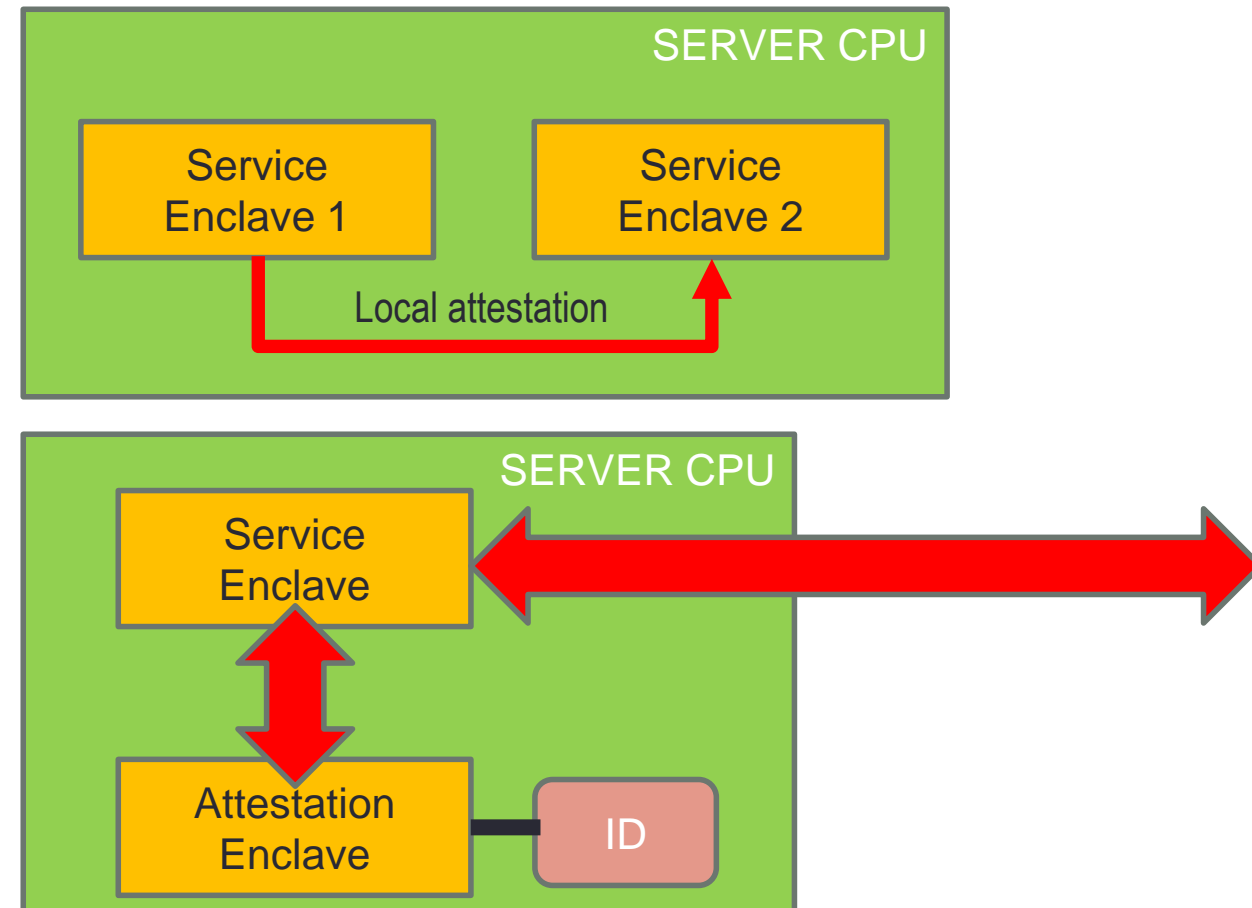


RoT anchor (e.g. certificate link to ID credentials in server HW)

Trust via Out-Of-band (e.g. root certificate)

How does attestation work in SGX

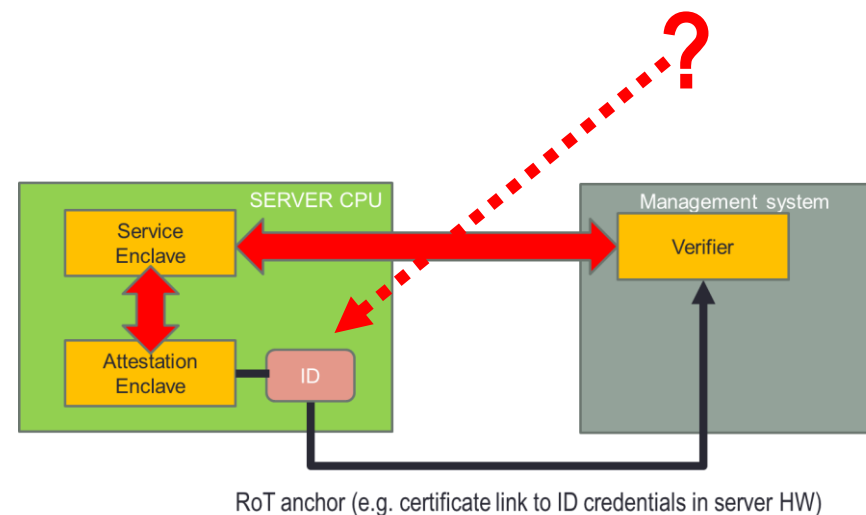
- SGX has two kinds of attestation
 - **Local** attestation
(on the same CPU)
 - **Remote** attestation



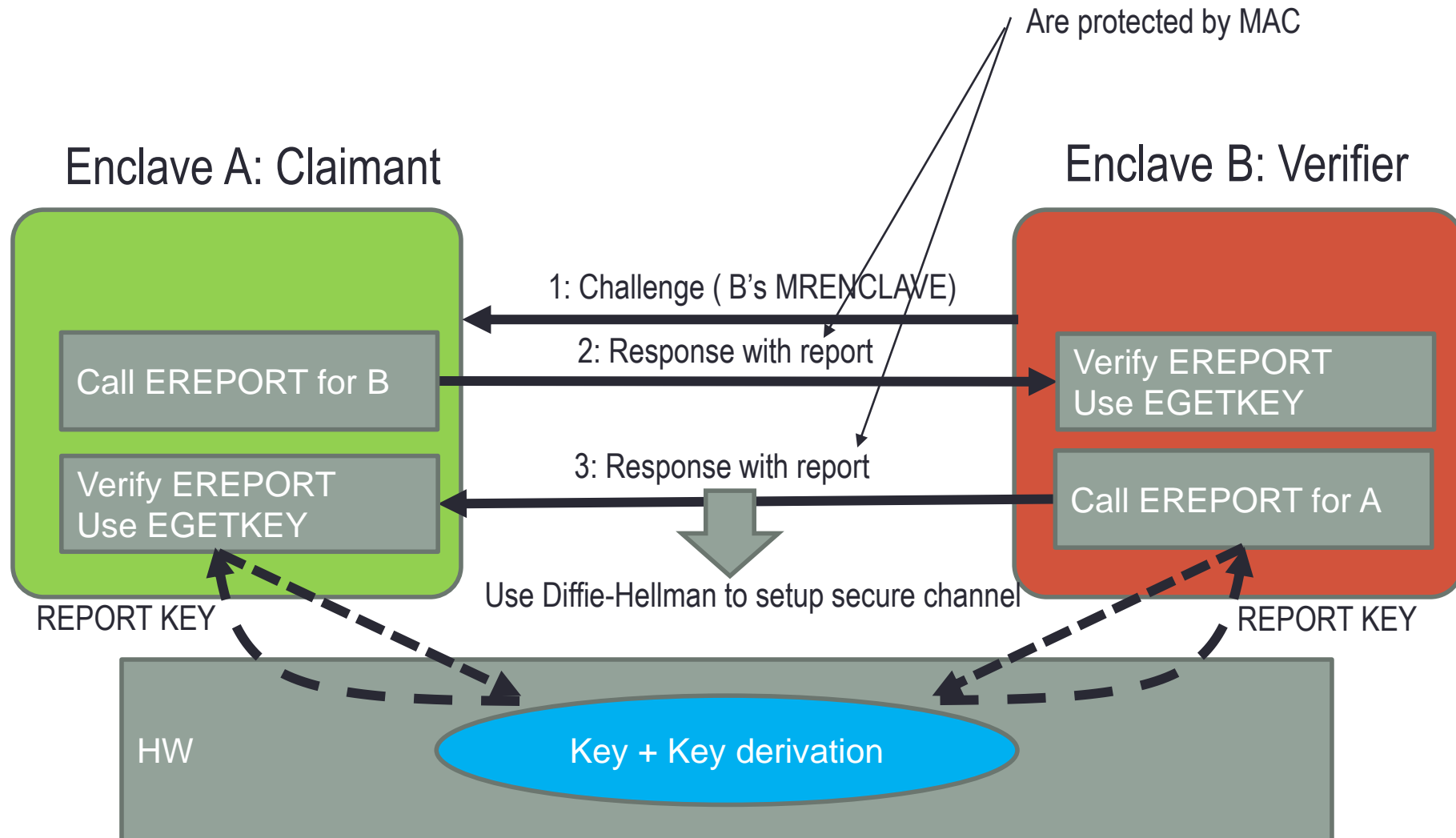
Attestation: which identity?

For Intel SGX there are in essence two solutions

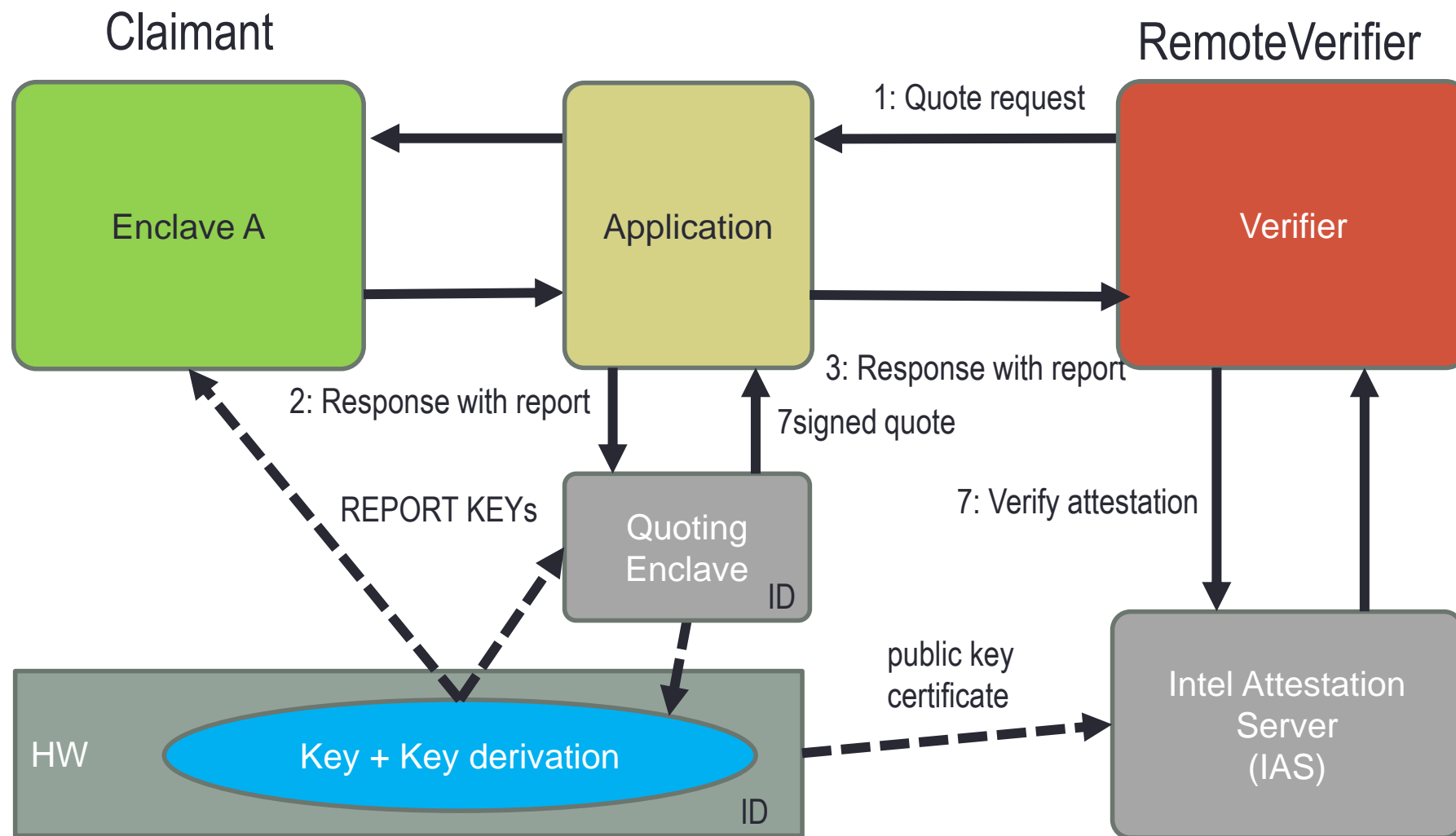
- An Intel programmed ePID identity
 - ePID is a cryptographic group key with privacy preserving features.
- An Intel programmed Platform Certification Key (PCK)
 - Plain ECDSE key
Elliptic Curve based scheme



Local attestation



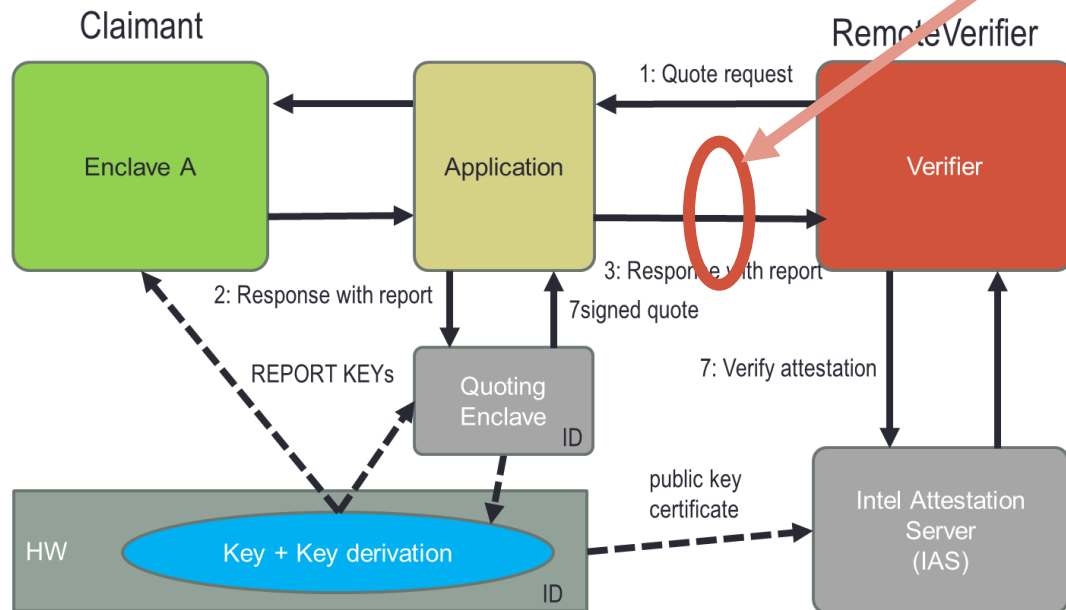
Remote attestation



Remote attestation – the verifier

The report contains information about

1. the enclave A instance and the HW, and
2. the code loaded/running in enclave A

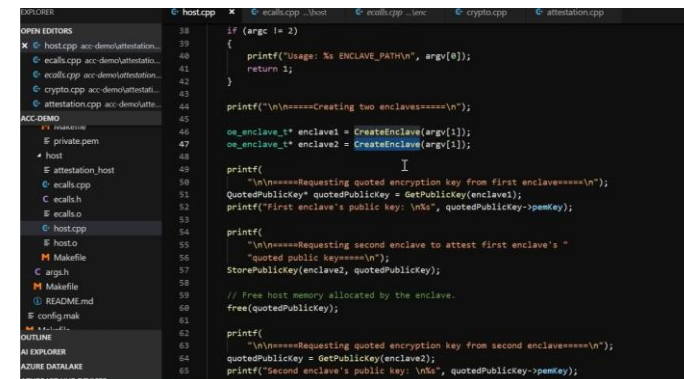


1: Quote reference values from ASIC supplier

2: Quote reference values from code developer

Current use of SGX

- Cloud use
 - Microsoft in Azure uses Openenclave as one of the current initiatives to implement confidential computing using enclave technology
 - Google has a similar initiative called Asylo
 - Baidu, has developed MesaTEE using SGX
- Many products/projects
 - Fortanix
 - Scone
 - Graphene (experimental still)



```
38 if (argc != 2)
39 {
40     printf("Usage: %s ENCLAVE_PATH\n", argv[0]);
41     return 1;
42 }
43
44 printf("\n\n====Creating two enclaves====\n\n");
45
46 oe_enclave_t* enclave1 = CreateEnclave(argv[1]);
47 oe_enclave_t* enclave2 = CreateEnclave(argv[1]);
48
49 printf(
50     "\n\n====Requesting quoted encryption key from first enclave====\n\n");
51 QuotedPublicKey* quotedPublicKey = GetPublicKey(enclave1);
52 printf("First enclave's public key: \n\n", quotedPublicKey->pubKey);
53
54 printf(
55     "\n\n====Requesting second enclave to attest first enclave's "
56     "quoted public key====\n\n");
57 StorePublicKey(enclave2, quotedPublicKey);
58
59 // Free host memory allocated by the enclave.
60 free(quotedPublicKey);
61
62 printf(
63     "\n\n====Requesting quoted encryption key from second enclave====\n\n");
64 quotedPublicKey = GetPublicKey(enclave2);
65 printf("Second enclave's public key: \n\n", quotedPublicKey->pubKey);
66
```

<https://github.com/Microsoft/openenclave>

Two use cases of SGX

- **Protecting Machine-Learning models:**

- ML models are trained on valuable data and as such one often wants to keep the model confidential. SGX can be used to perform ML based computation in cloud without “loosing” the model.

- **Blockchain with SGX:**

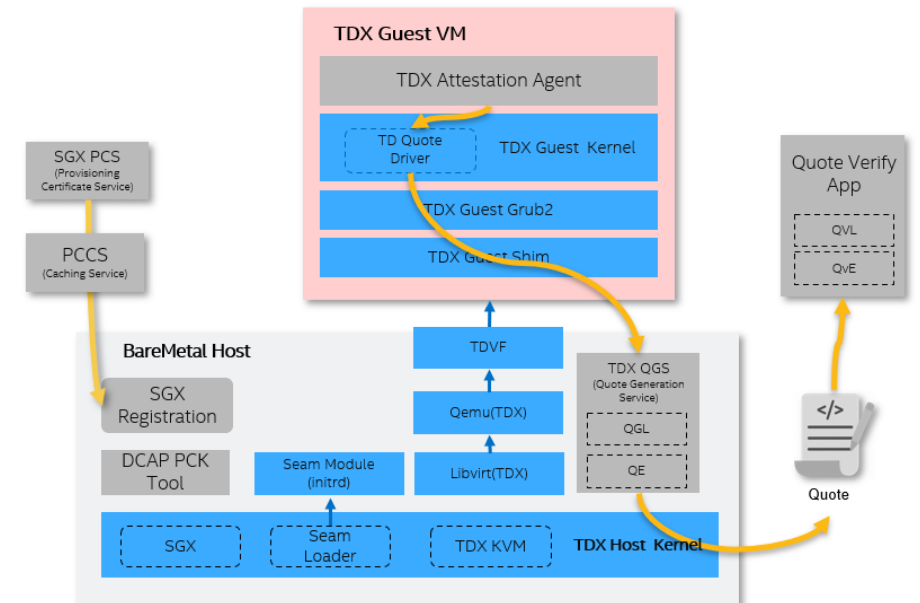
- Instead of Proof-of-work SGX enclaves are use to realize a trustworthy consensus scheme.
- The data and computations that demand privacy can be selectively placed inside an enclave protected from untrusted blockchain node access.
- Then the blockchain data can be kept in encrypted form until it is needed for a transaction. It is then decrypted in the secure enclave where permitted participants can view it.

SGX 1.0 shortcomings

- Use of EPID and requirement of IAS gives a too hard connection to Intel which is not acceptable in many uses cases (is remedied in next generation SGX 2.0)
- Enclave size EPC is too small and SGX not really works well with virtualized systems. (remedied in SGX 2.0)
- SGX leaks information – attacks have been found.

Intel TDX

- SGX was primarily designed to protect part of an application which implicitly means that one had in mind that applications had to be designed for using SGX and the enclave – hangs – on the application part that runs in the normal OS world.
 - This complicates things for using existing software – some solutions exist that work but still it is not ideal
- TDX creates a TEE where a whole VM can run.
 - TDX and SGX exist in parallel



AMD SEV-(SNP)

- SEV = Secure Encrypted Virtualization
 - SEV-SNP = + Secure Nested Paging
- } Two variants of SEV

SEV VMs control whether a memory page is private or shared using the enCrypted bit (C-bit) in the guest page tables. The location of the C-bit is implementation defined and may be the top physical address bit as shown in Figure 1.

Shared (unencrypted) memory is marked C=0 by the VM, indicating it does not have to be encrypted with the VM's memory encryption key.

Private (encrypted) memory pages are for the exclusive use of that VM and are marked as C=1

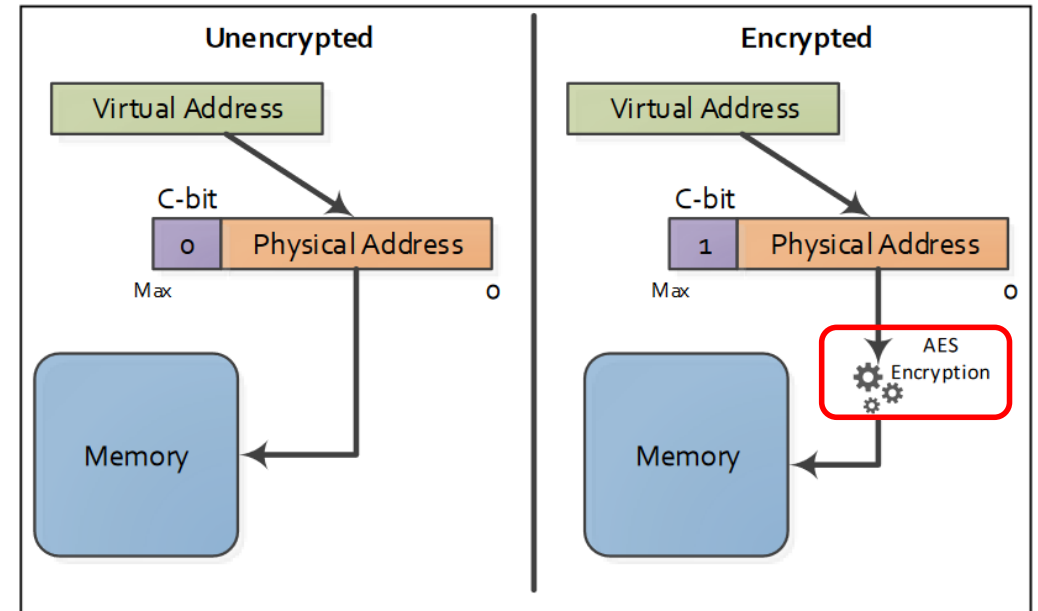
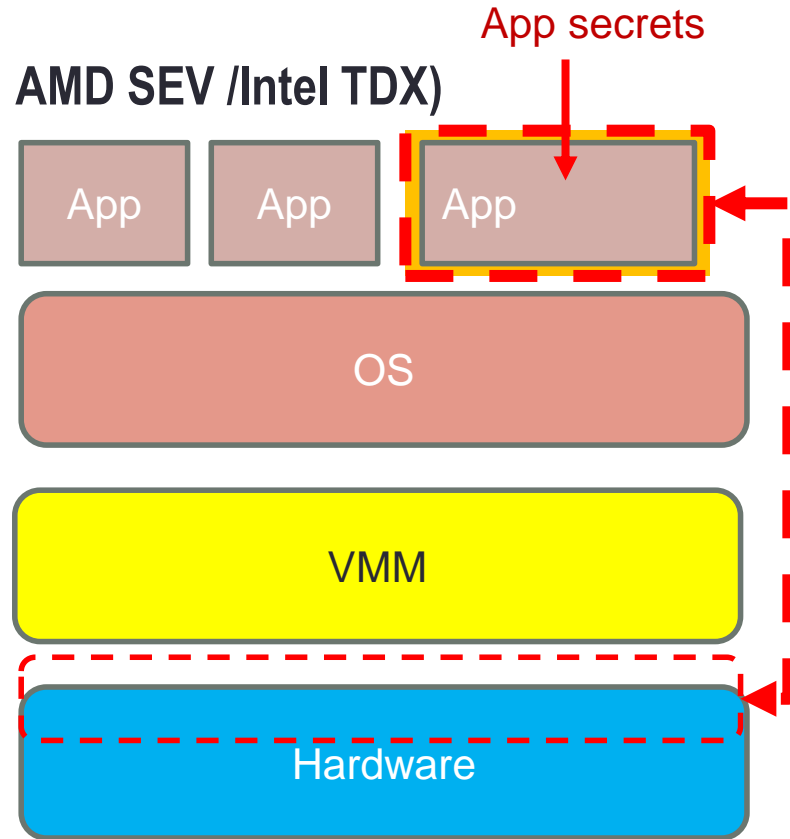


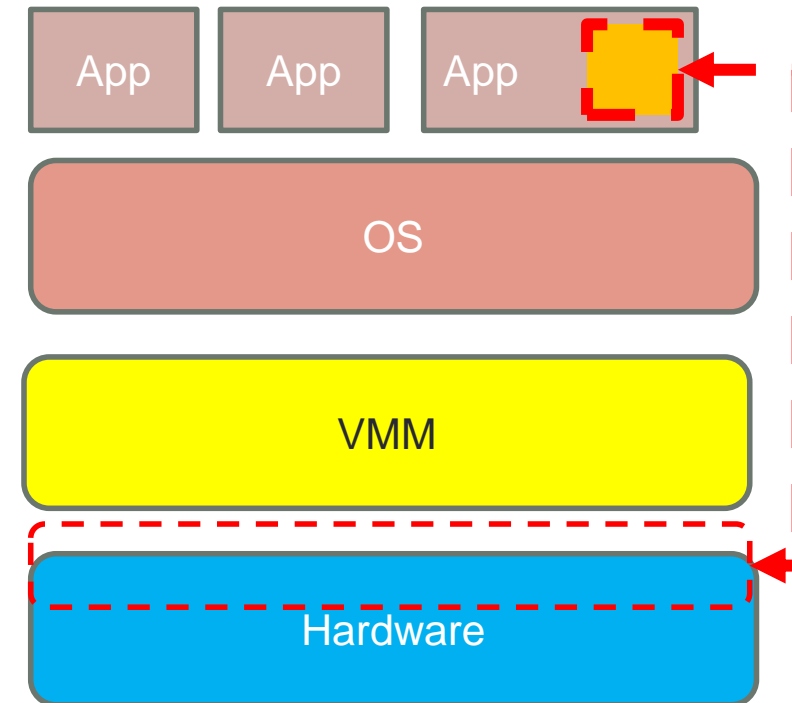
FIGURE 1: ENCRYPTION CONTROL

AMD SEV vs SGX

Entire application (a VM or container) runs in protected TEE



Intel SGX



AMD SEV

Compared to Intel SGX, AMD SEV

- requires no changes of application to run it in encrypted space
- can already be used in virtualized systems
- lacks integrity protection (partly fixed in new release)
- Has also reported weaknesses, e.g. <https://thehackernews.com/2018/05/amd-sev-encryption.html>

Confidential compute frameworks

- There are industrial activities ongoing to establish frameworks that are technology neutral (i.e. cover SGX as well as SEV) that allow the
 - Execution of code in an HW protected environment
 - Code and data is encrypted
 - Environment can be verified through remote attestation

E.g.

- Fortanix: [Fortanix products](#)
- Redhat: [Enarx](https://enarx.dev/): <https://enarx.dev/>
- Microsoft: [Azure Confidential Computing](#)
- Google: [Google Cloud](#)

General problem of TEEs

- Side-channel attacks
 - Side-channel attacks
 - and more side channel attacks
-
- ASICS are very complex and often have not been designed from the start to deal with the threat of side-channels. So, it is very hard to be sure that – also in TEE solutions – side channels do not exist.
-
- And frankly HW are rather clear they do not claim their solutions to be free of these
 - Do not expect miracles from confidential computing and always be prepared to deal with situations when “sh** hits the fan” so to speak.

END

Slides that follow are only for reference and do not belong to the mandatory course material

STUDY QUESTIONS

1

- Explain why trustworthiness is a preferable notion over trusted when we talk about compute platforms?
- What is the purpose of a remote attestation wrt to trustworthiness?
- What is the main principle of Zero Trust
- What is a RoT and give three different types of RoTs.
- What is the purpose of an RTM?
- What can Common Criteria be used for wrt to the trustworthiness of a platform?
- To what extend can I make all parts of an ICT system trustworthy?
- Under which conditions can we rely on SW to have a trustworthy PC?

2

- What is virtualization and what is its security relevans?
- Is type I virtualization more secure than type II virtualization? discuss arguments.
- Give at least three examples of HW based trusted (trustworthy) computing?
- Describe the isolation between processes in a running TrustZone enabled system that are located in normal or secure world.
- How can one prevent in a TrustZone system that a virus scanner is never executed?
- What is the purpose of the monitor in a Trustzone system?

3

- Why is there a need to have ecalls and ocalls in SGX?
- SGX lacks a secure interrupt what does that imply wrt to starving an enclave by the OS?
- Explain the role of MRENCLAVE and MRSIGNER
- Does SealKey always depend on MRENCLAVE / MRSIGNER?
- Can I just program an application with an enclave and make it execute? Give pros and cons for such capability.
- Why can the local attestation not be used for remote attestation?