SYSTEM SECURITY: TRUSTED COMPUTING I

TDDE62 Information Security: Privacy, System and Network Security

Ben Smeets

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

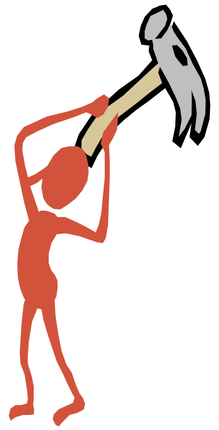
Overview

- Why trusted computing?
  - Intuitive model for trusted computing
  - Confidential computing
  - Roots of trust
  - Hardware versus software
  - Confidential computing - how

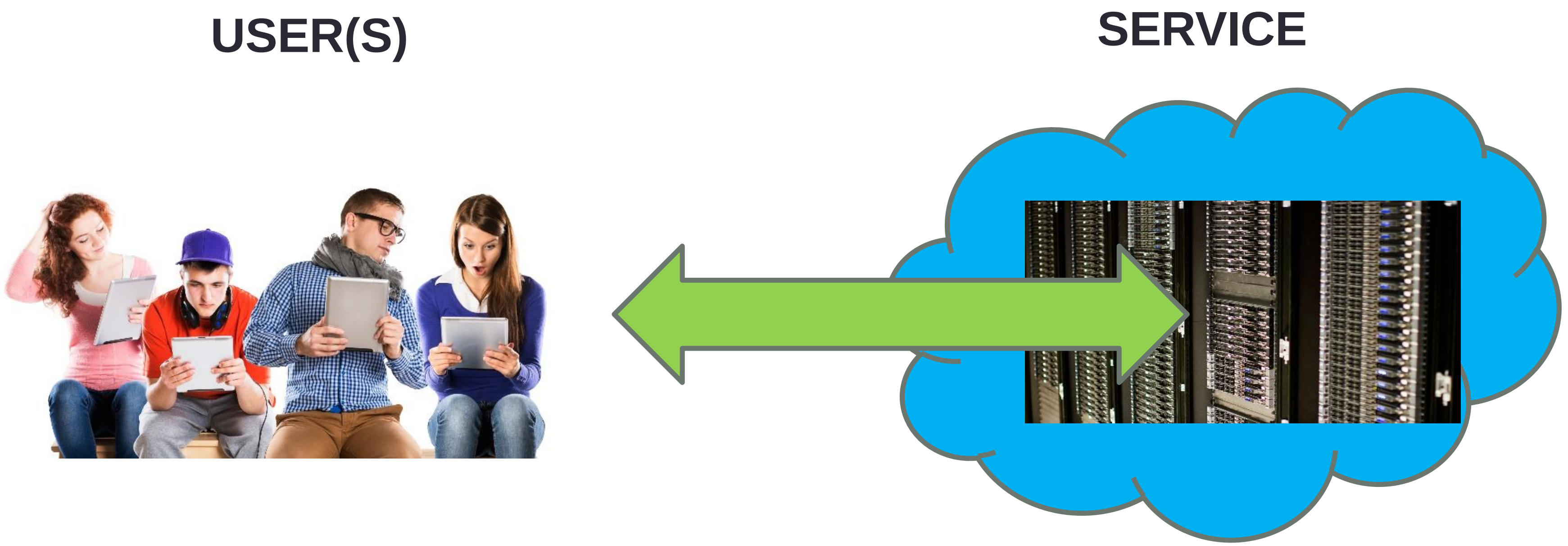
- CPU secured execution environment:
  - TrustZone,
  - SGX
  - (AMD SEV)

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



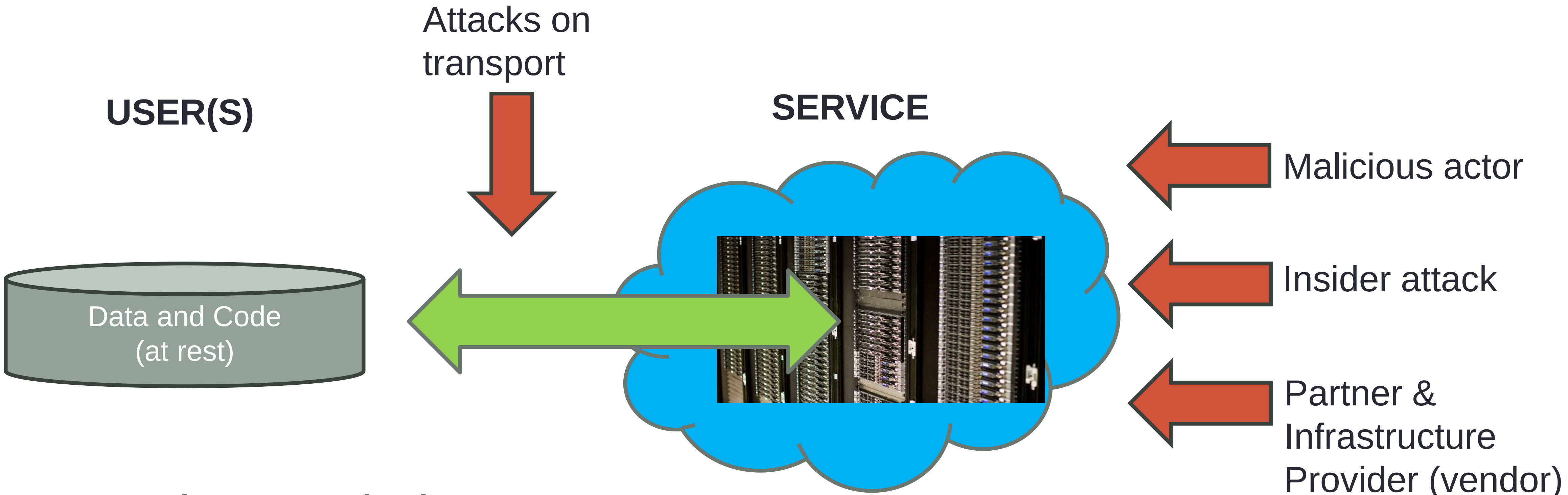
The main security question from a user's perspective



How can I 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 is processed
- When code is loaded/used

Important general aspects to consider
Zero Trust principle(s)

- Is it really the right service/server I’m interacting with ?
- Is the service/server in a proper (secure) state so
  - Do I dare to interact/exchange sensitive information?
    - Privacy or Business critical data
    - Store keys for (e.g. web) services
- Does the service/server comply to business or regulatory requirements?
- **Zero Trust Principles:** Instead of assuming trust across a system (perimeter), entities should only engage with other entities after proper verification (authentication and authorization).

  See NIST SP 800-207

What are typical problems we want to address

1. How can we, inside a device/computer, protect sensitive data (and thus also keys)?

2. How can we securely insert a key in a remote server for setting up a secure (e.g., TLS) connection?

3. 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 three 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.

  - Consequence: 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.

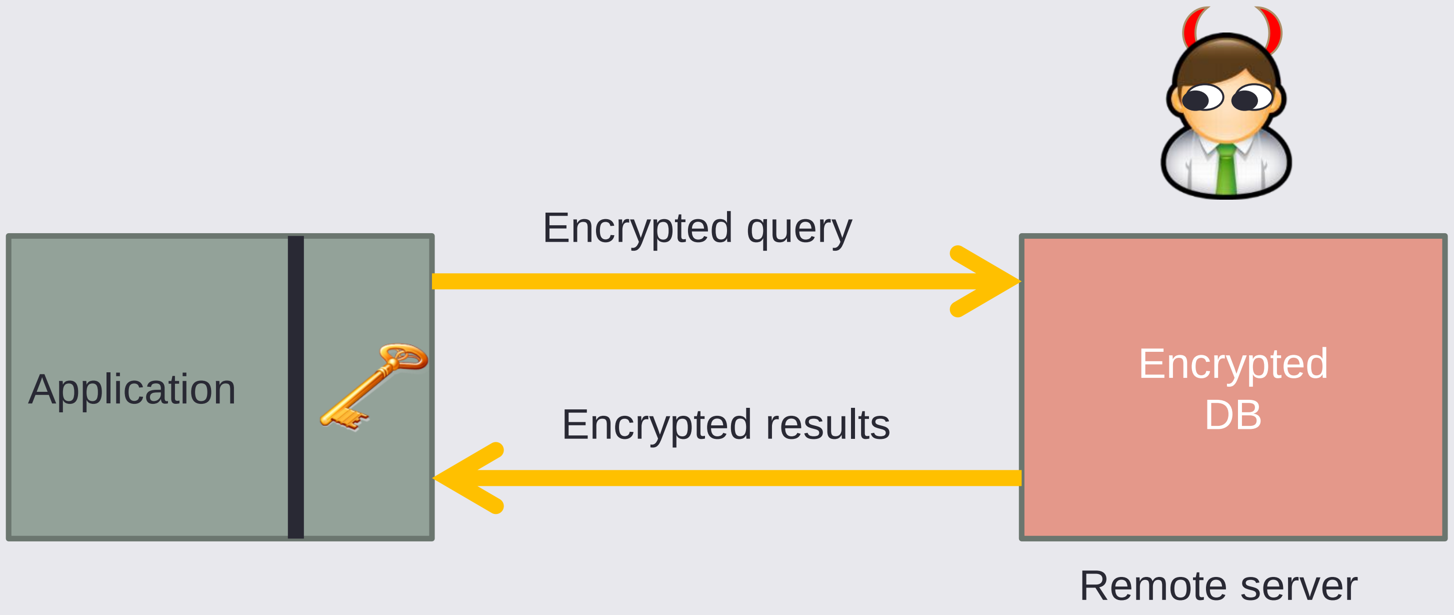
Alternative to trusted computing/platforms

- 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 (cleartext) 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 remove server and devise a method (algorithm) that does the processing on encrypted data and produces the result in encrypted form



- For example database operations

See http://css.csail.mit.edu/cryptdb/ Not completely homomorphic encryption based

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. 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 of a system, but the question is “IS it trustworthy?”

Trustworthy: The system fulfills 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.

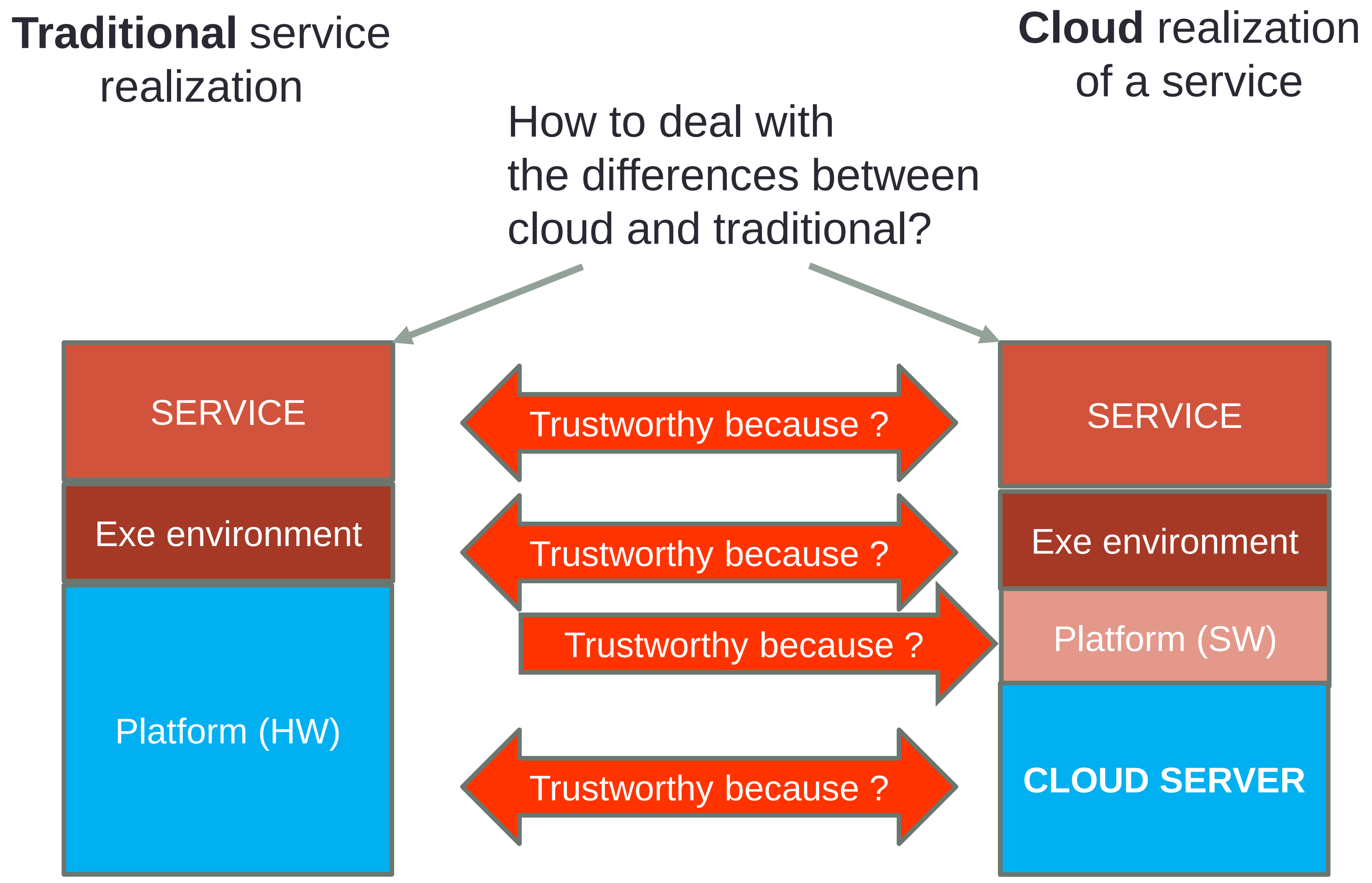
Recall: Using Common Criteria a system that is successfully evaluated at level EALx (x say =4) is considered to be trustworthy.

Common Criteria as abasis 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, severs, 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

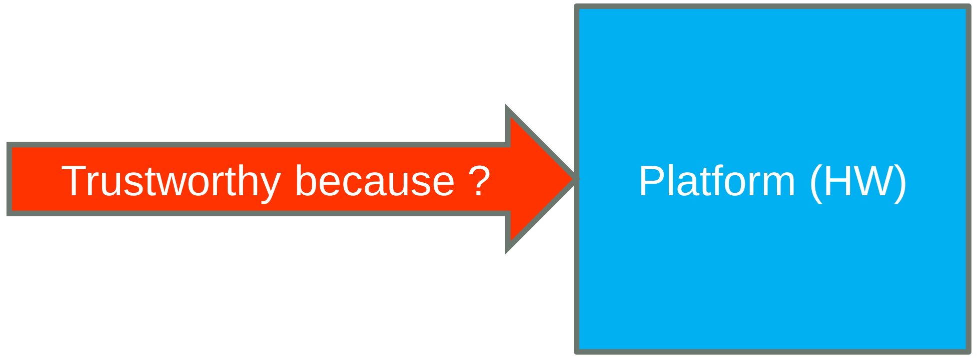
  https://www.fmv.se/verksamhet/ovrig-verksamhet/csec/ --Sveriges Certifieringsorgan för IT-säkerhet (CSEC)

How to obtain trustworthiness (to a service)?

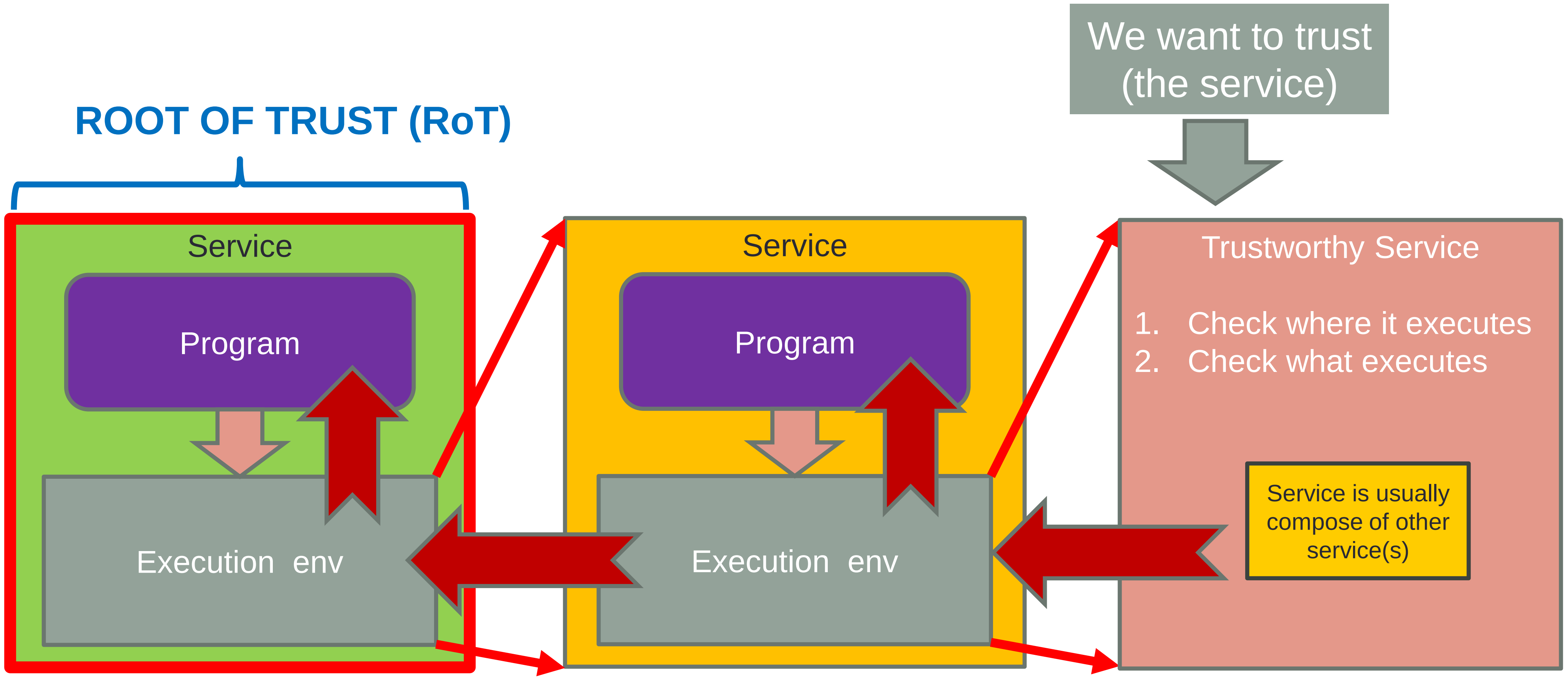


For example: How & why trust HW?

- Trust by reputation (e.g., by a company that I trust)
- Trust by relying on a third party (e.g., recommendation)
- Assurance of design
  - Review (e.g., common criteria use)
  - Proofs (by modeling of HW)
- Assurance of production
  - HW is produced according to design



Trust chain – its start is the Root of Trust (RoT)



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

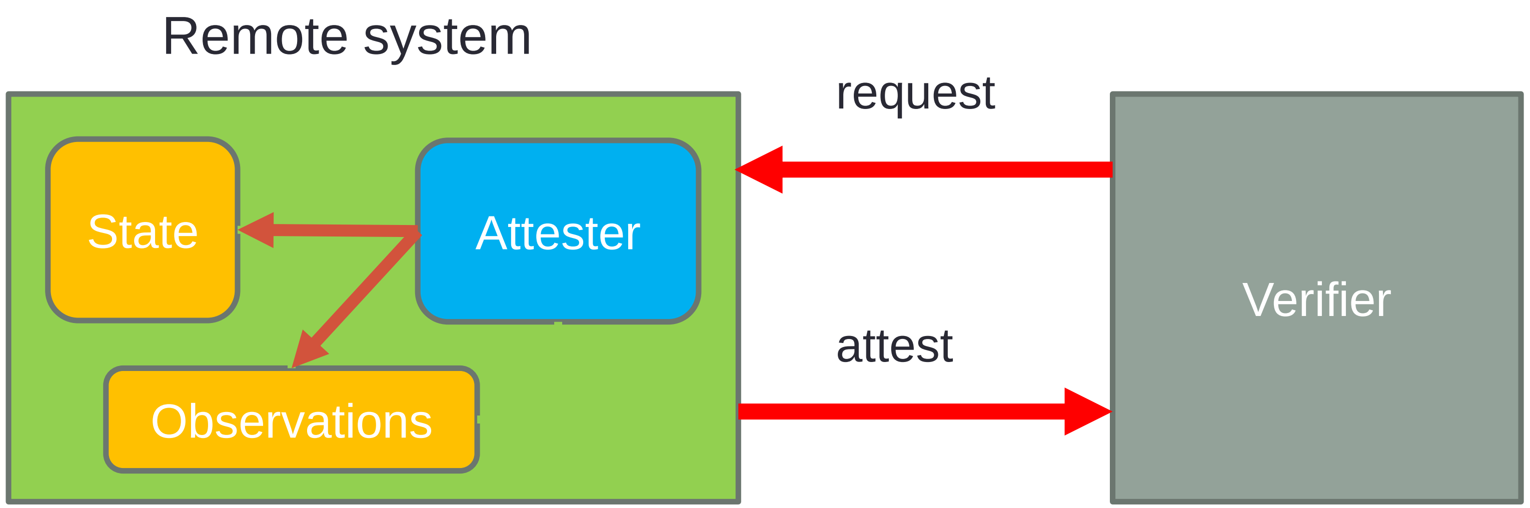
Note: RoT is not only data (e.g. keys) but also logic, therefore we say that a RoT is an engine.

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

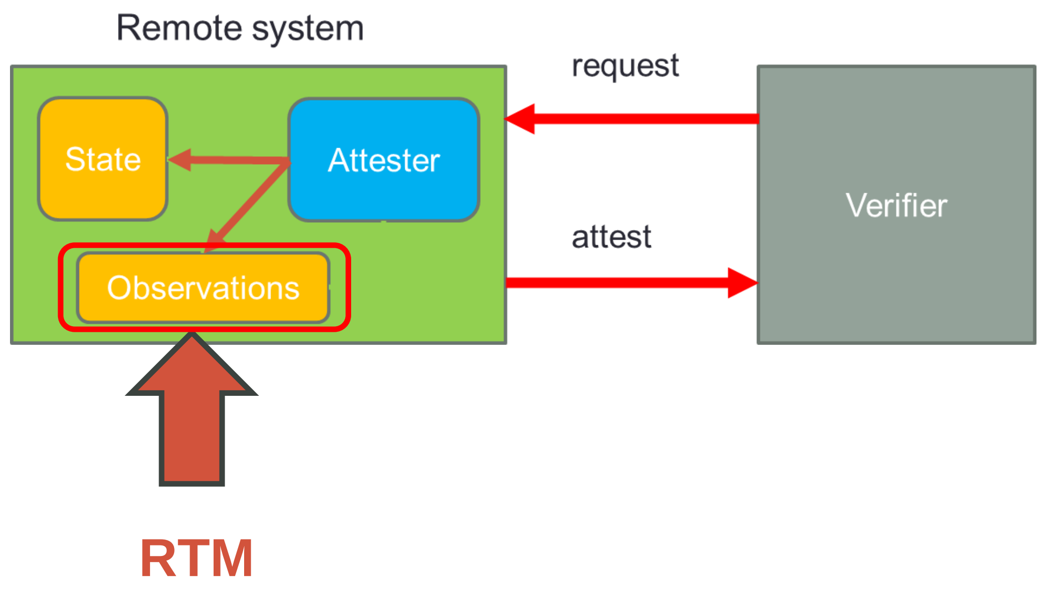


Note: similarity to a challenge-response based authentication

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 (RTM)

- The RTM can be used to create attestation reports of the measurements that a verifier can use to make decisions on (e.g., to establish a secure connection to use the remote system)



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 realizations are 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 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

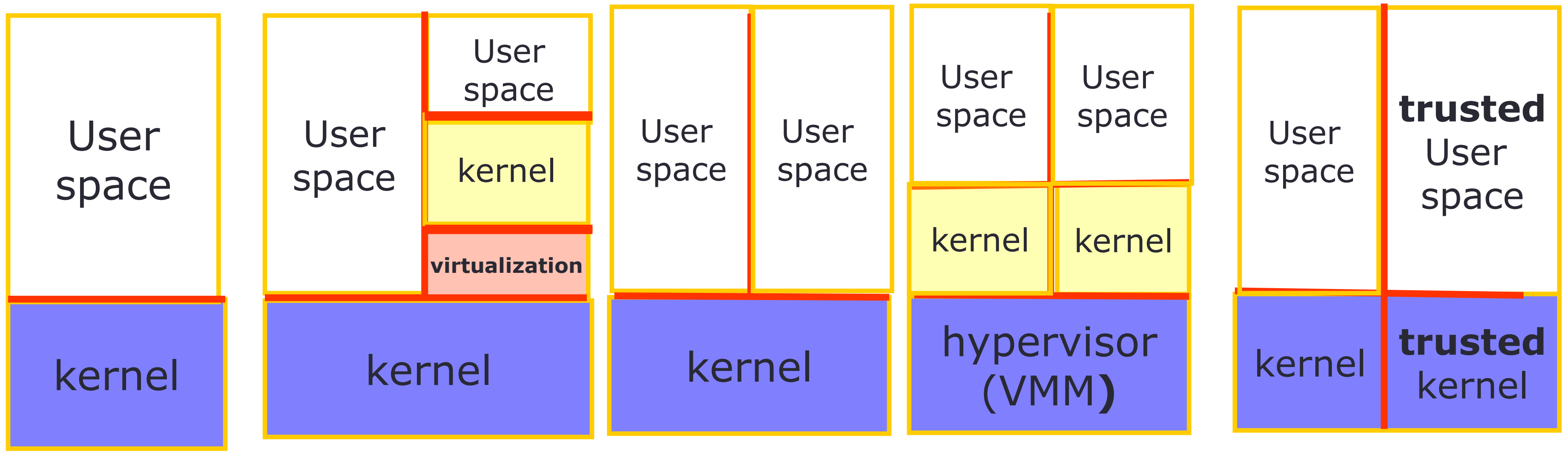
  - Modify OS
    - try to create isolation (VMs, Containers or OS features)
    - Dockers, SystemD, SE Linux

Our focus

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

Some

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

Partly based on slide material from Dries Schellekens

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

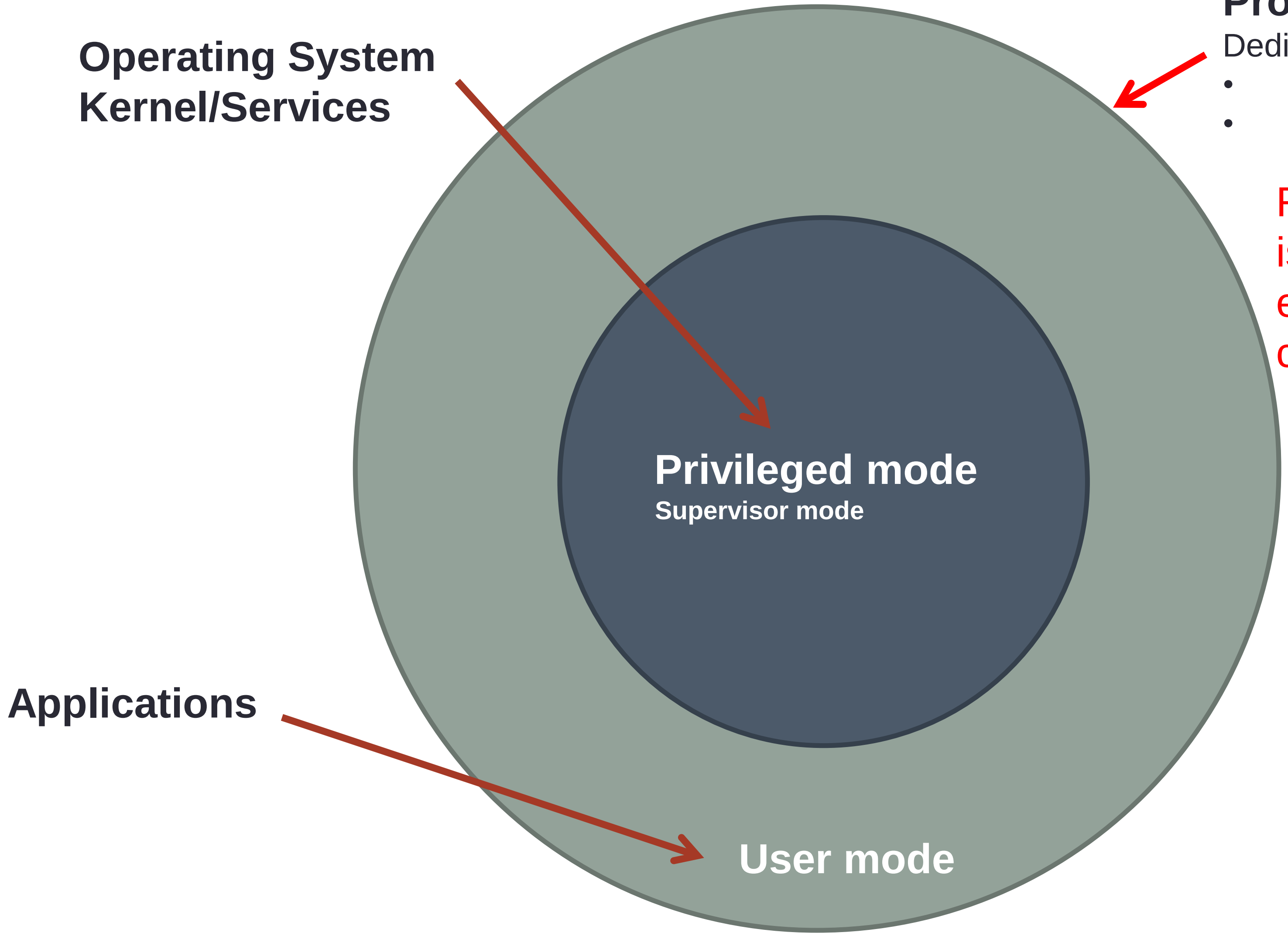
Also

- Intel TDX and AMD SEV

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.

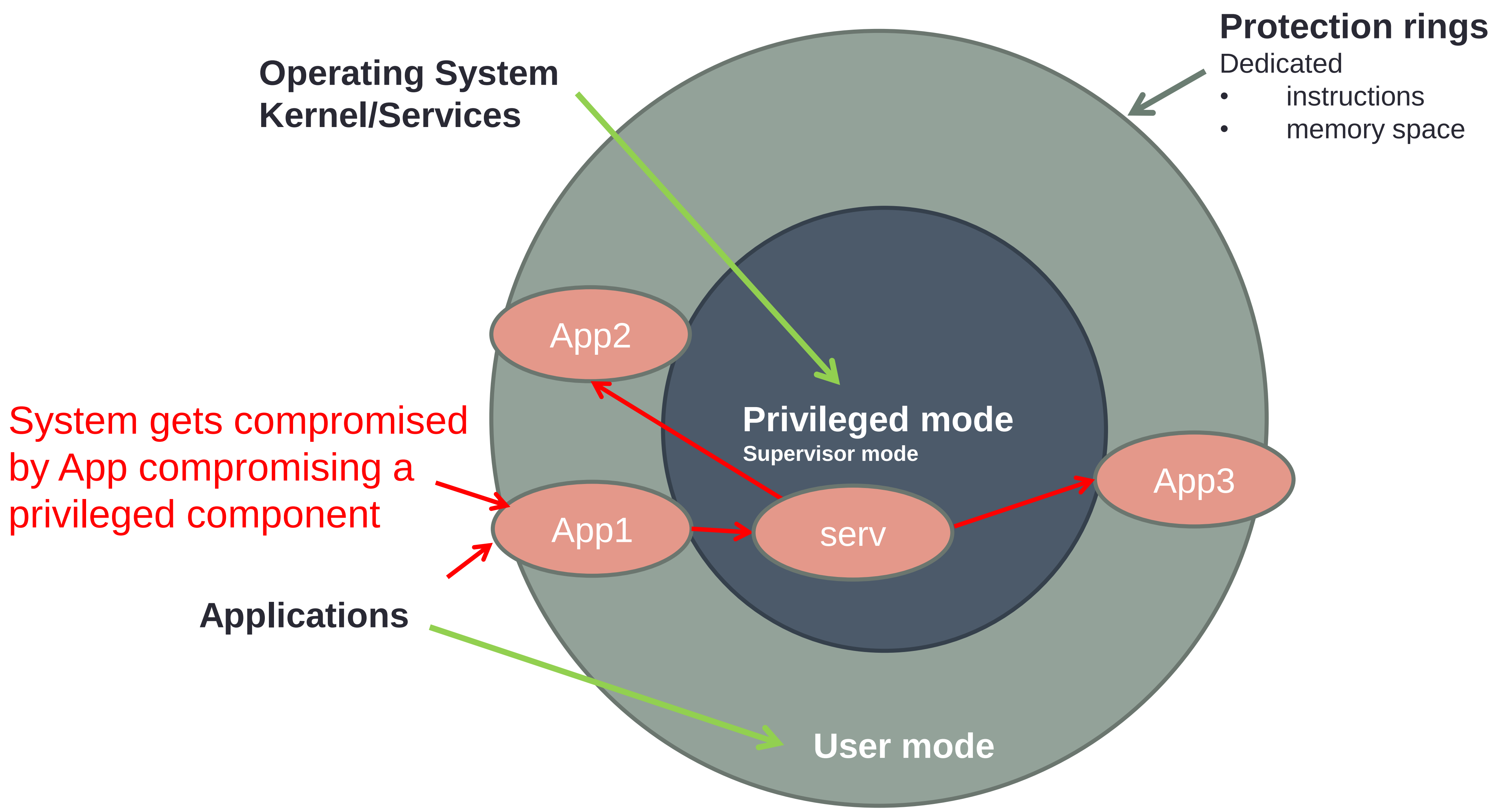
ARM standard approach



Protection rings
Dedicated
- instructions
- memory space

Rings create isolation via hw enforced access control

Security problem for applications

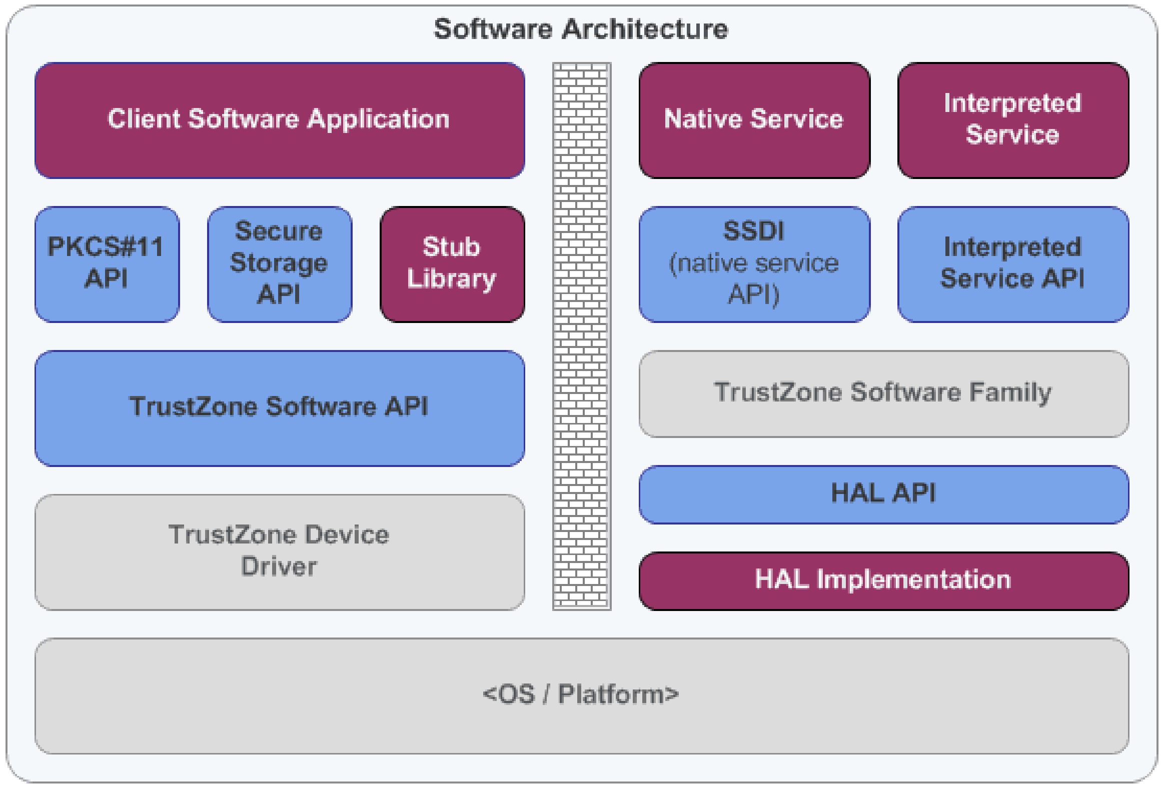


ARM TrustZone

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

Normal world

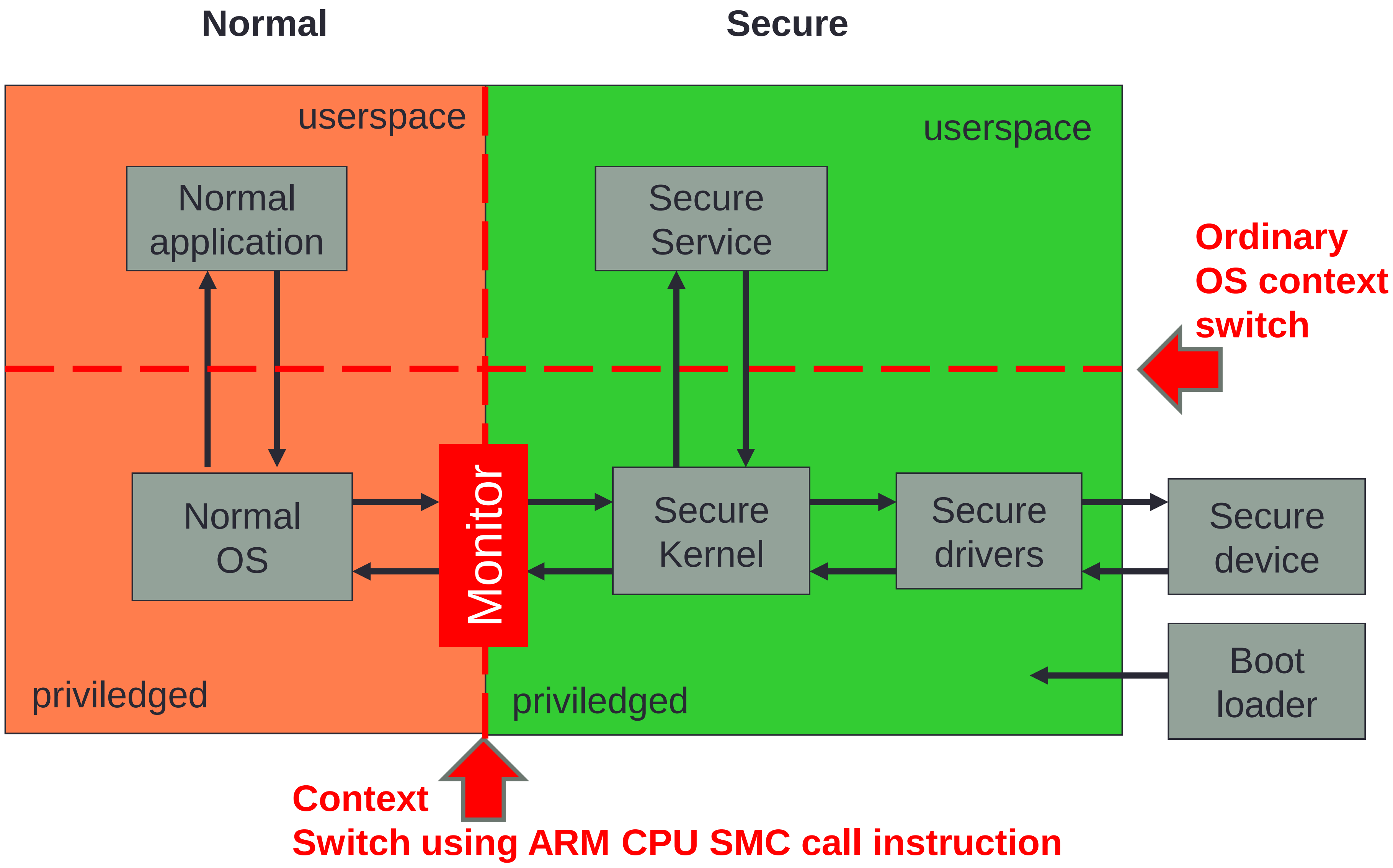
Secure world



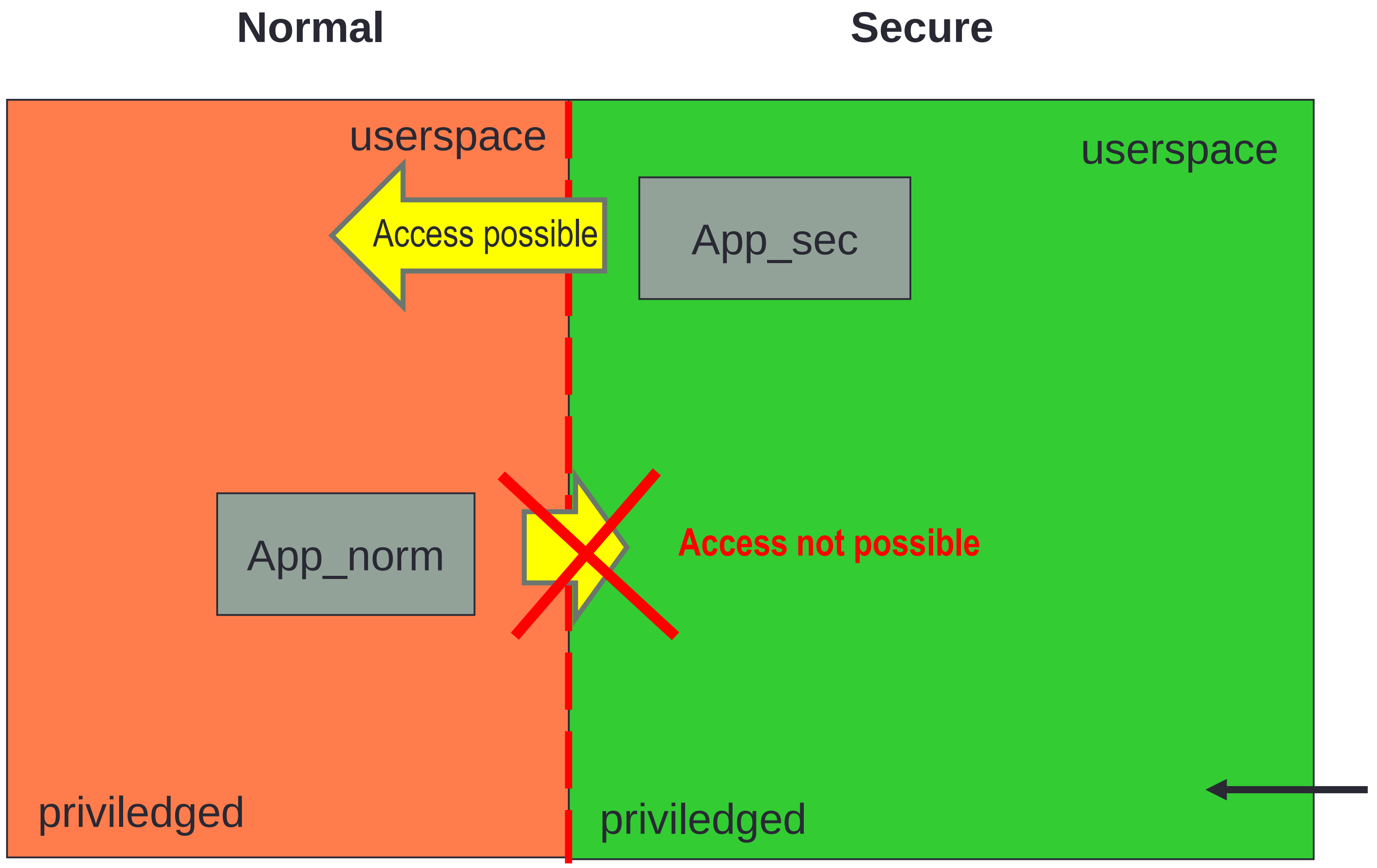
Basic idea

- **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 - asymmetric

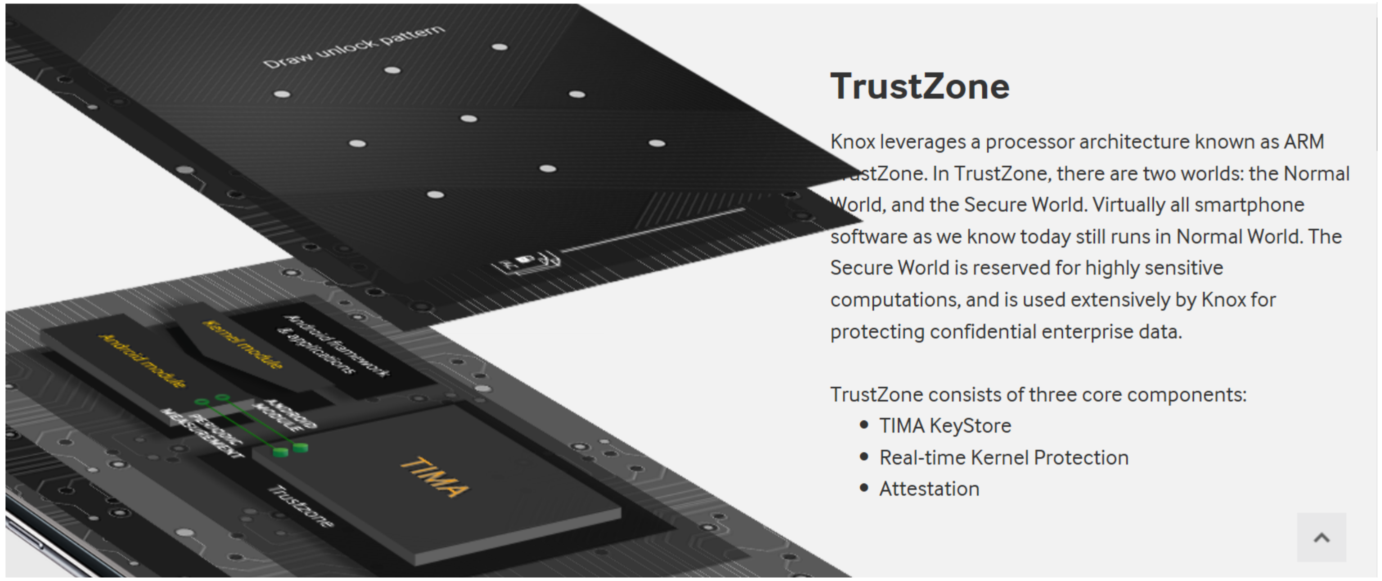


TrustZone use

Widespread in use in smartphones using Qualcomm and Samsung chipsets

Forms a core of Samsung’s KNOX solution

- https://www.samsungknox.com/en



TrustZone

Knox leverages a processor architecture known as ARM ustZone. In TrustZone, there are two worlds: the Normal World, and the Secure World. Virtually all smartphone software as we know today still runs in Normal World. The Secure World is reserved for highly sensitive computations, and is used extensively by Knox for protecting confidential enterprise data.

TrustZone consists of three core components:

- TIMA KeyStore
- Real-time Kernel Protection
- Attestation

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 in 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 specific pros and cons but we use it here as a generic example of a TEE technology

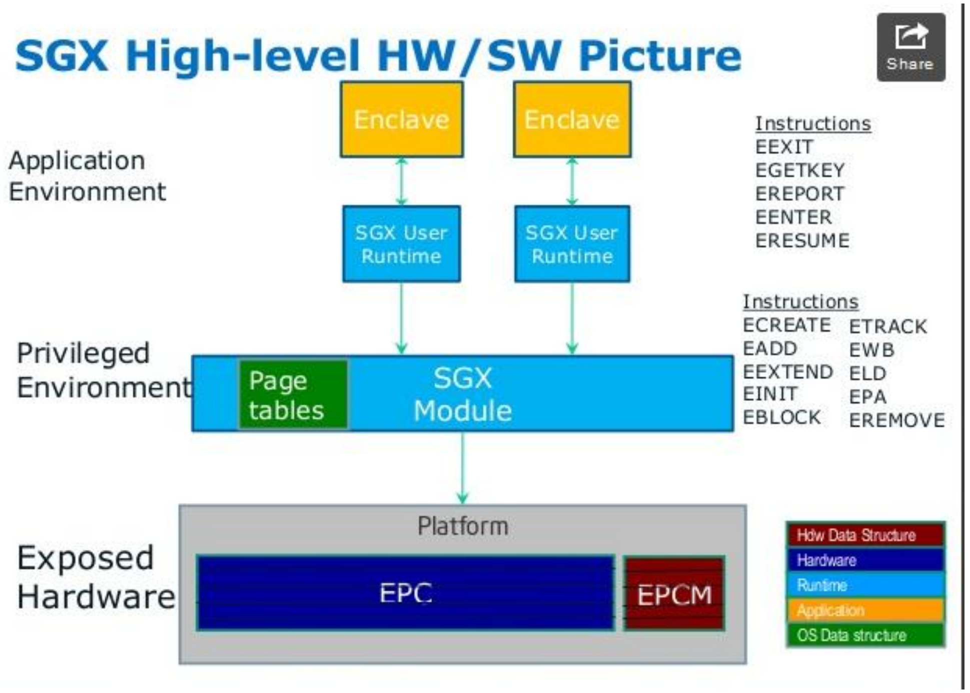
Other technologies are

- AMD SEV (secure encrypted virtualization) of which there are several versions with their own differences
- Intel TDX (trust domain extension)

These technologies are not technically equivalent but can be viewed as equivalent when viewed as generic TEEs.

Enclaves

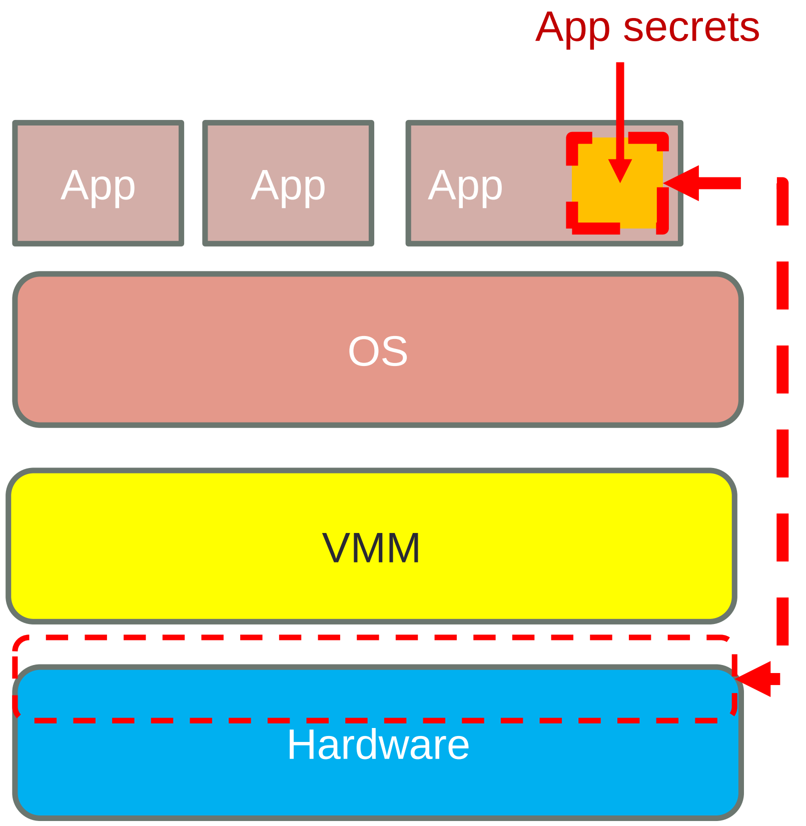
- Enclaves are isolated memory regions of code and data
- One part of physical memory (RAM) is reserved for enclaves and is called Enclave Page Cache (EPC)
- EPC memory is encrypted in the main memory (RAM)
- EPC is managed by OS or VMM
- Trusted hardware consists of the CPU Die only



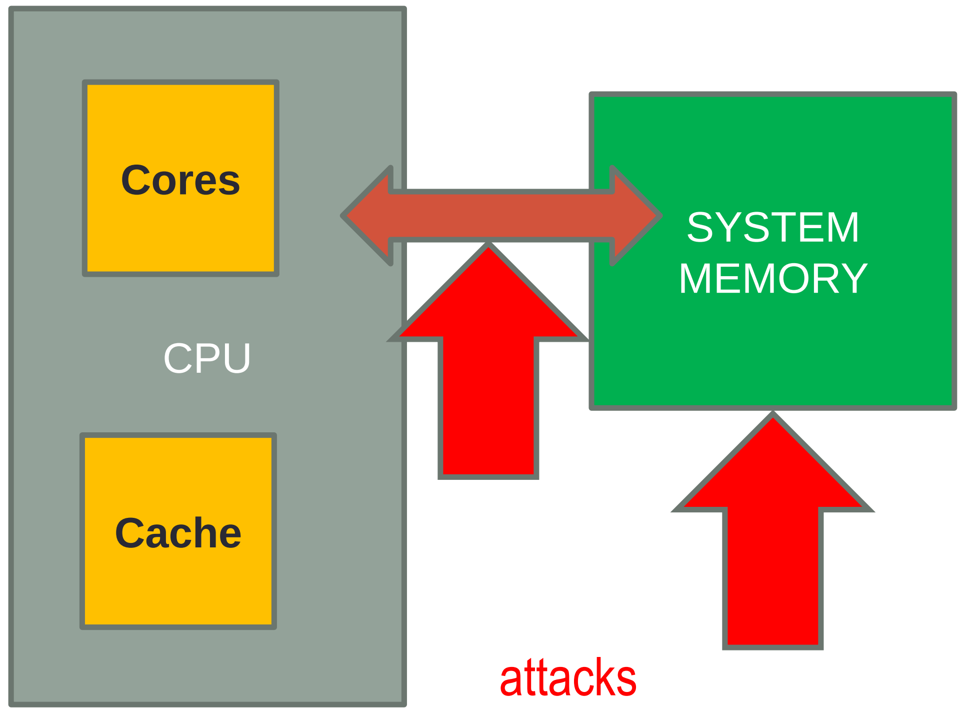
More info see this good overview paper:
Victor Costan and Srinivas Devadas, SGX explained:
https://eprint.iacr.org/2016/086.pdf

Reduced attack surface with SGX

- Application gains ability to defend is own secrets
  - Smaller attack surface (App enclave+processor)
  - Malware that subverts OS or VMM, BIOS, drivers cannot steal app secrets

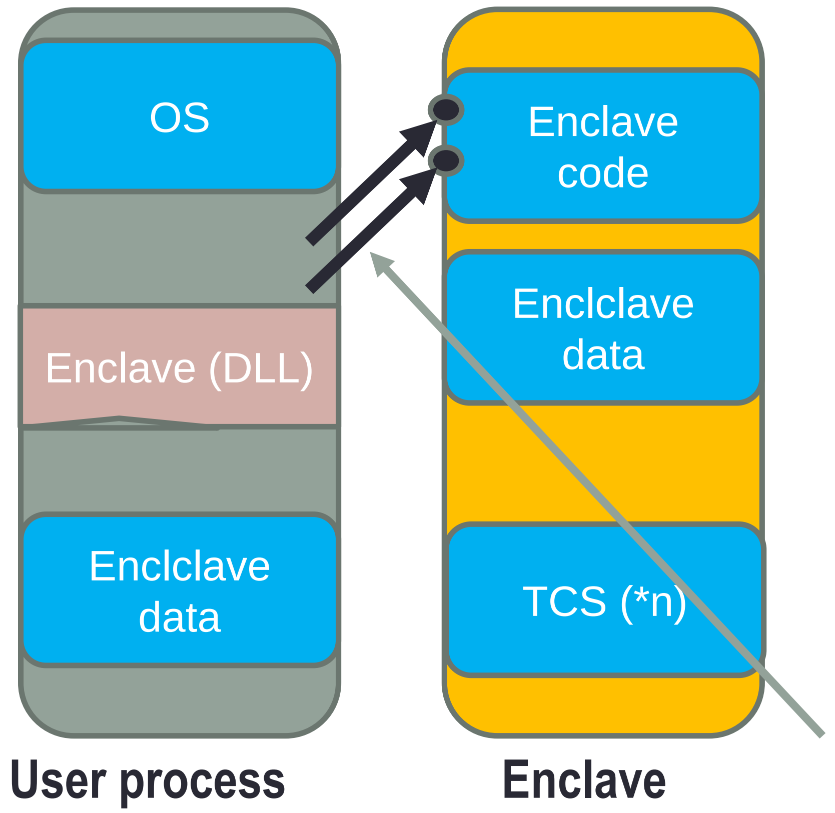


Protection against Memory Snooping



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 – “going in and out of enclaves”

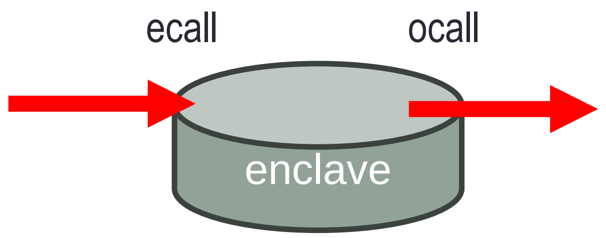
Interactions with enclaves goes via what Intel defined as ECALLs and OCALLs:

- **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 (e.g. Ericsson could be an ISV but also Telia could act as ISV)

- IMPORTANT NOTIONS
  - Launch Authority
  - MRENCLAVE
  - MRSIGNER
  - SGX Keys
  - Attestation

"Vendor" should not be taken very strictly here

Launch Authority

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

- New SGX servers have various ways to realize this (and use BIOS to select mode)
  - By Intel
  - By infrastructure owner (e.g., Microsoft Azure cloud)
  - By actor running a dedicated 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.

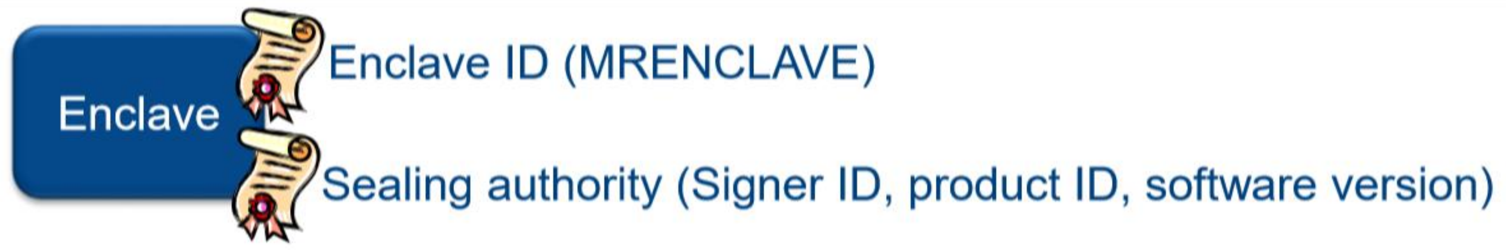
  - The MRENCLAVE typically tells about
    - HW version
    - ASIC/SGX Firmware version
    - Enclave configuration parameters

Measurement is basically a recorded cryptographic hash

MRSIGNER

- **MRSIGNER is a notion introduced by SGX that reflects enclave's sealing authority**. The sealing authority signs the enclave

  - This value is represented by a hash over sealing authority's public key and is part of 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

Unique for each Enclave instance

(based on a process that combines
- Enclave parameters and
- System parameters

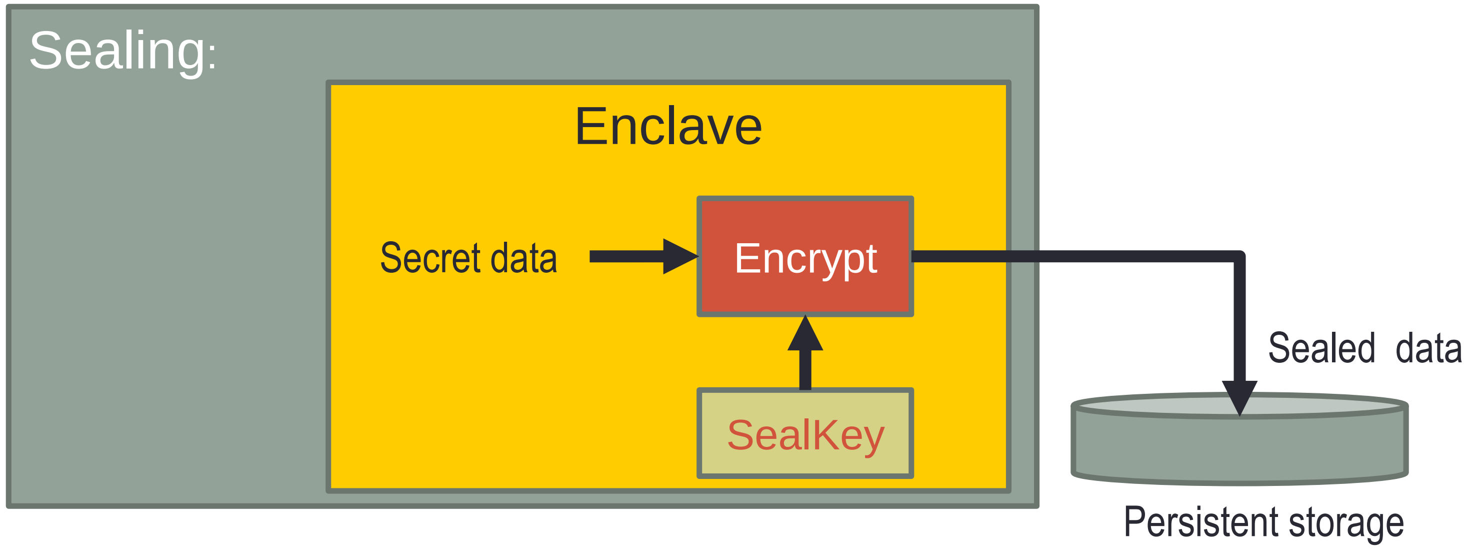
Not so relevant anymore

~~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)

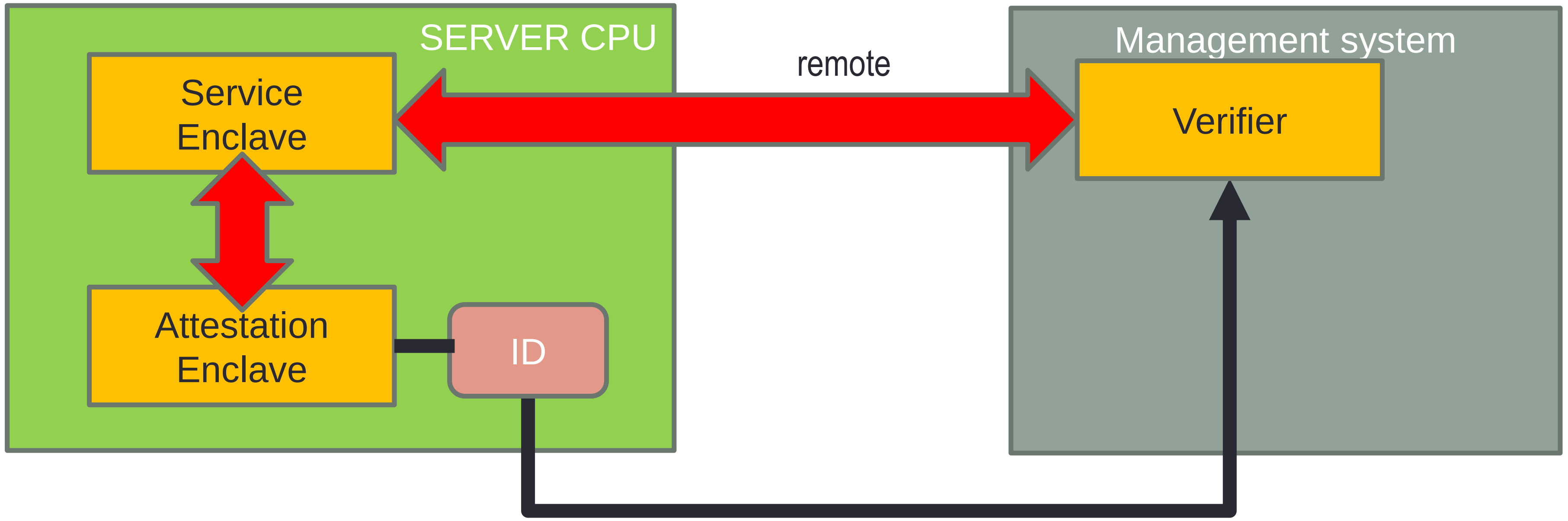
- Sealing is the process of encrypting enclave secrets for persistent storage to disk. Encryption is performed using a **private Seal Key that is unique to that particular platform and enclave**, and is unknown to any other entity



Sealkey is derived via EGETKEY HW operation that derives the key from the hw key

Attestation

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

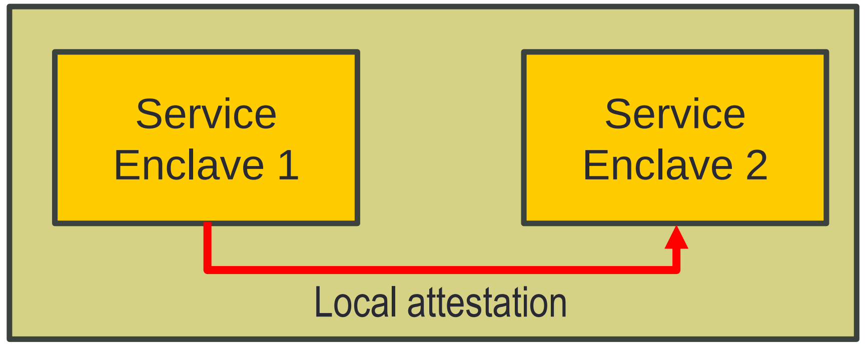


RoT anchor (e.g. certificate link to ID credentials in server HW)
Trust via Out-of-Band setup (e.g. using a root certificate)

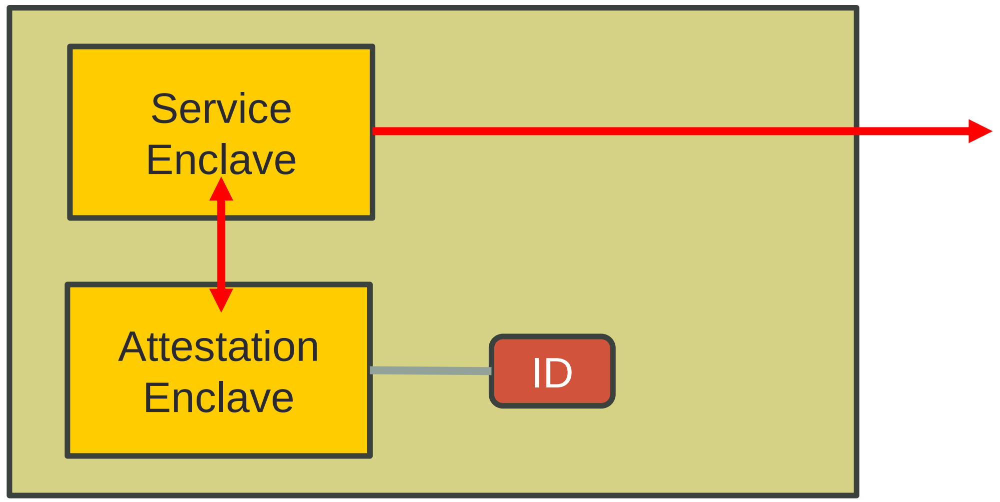
Two variants of attestation in SGX

SGX has two kinds of attestation

- Local attestation
  (on same CPU)



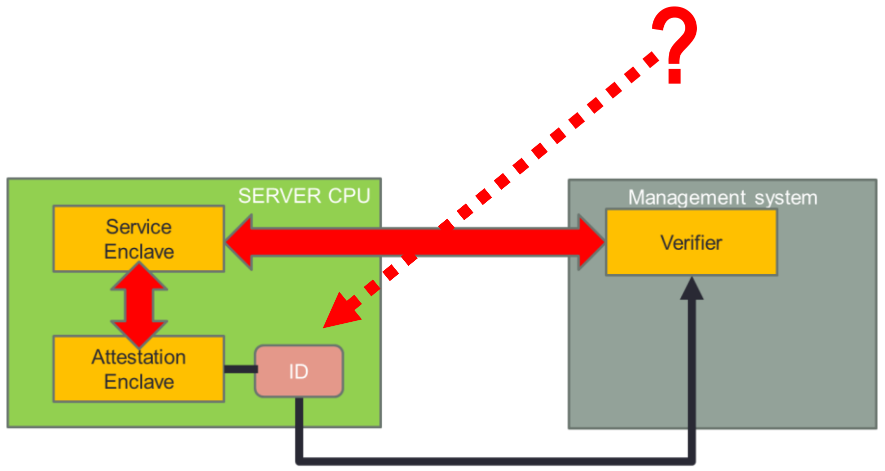
- Remote attestation
  (using an attestation enclave)



Attestation: which identity?

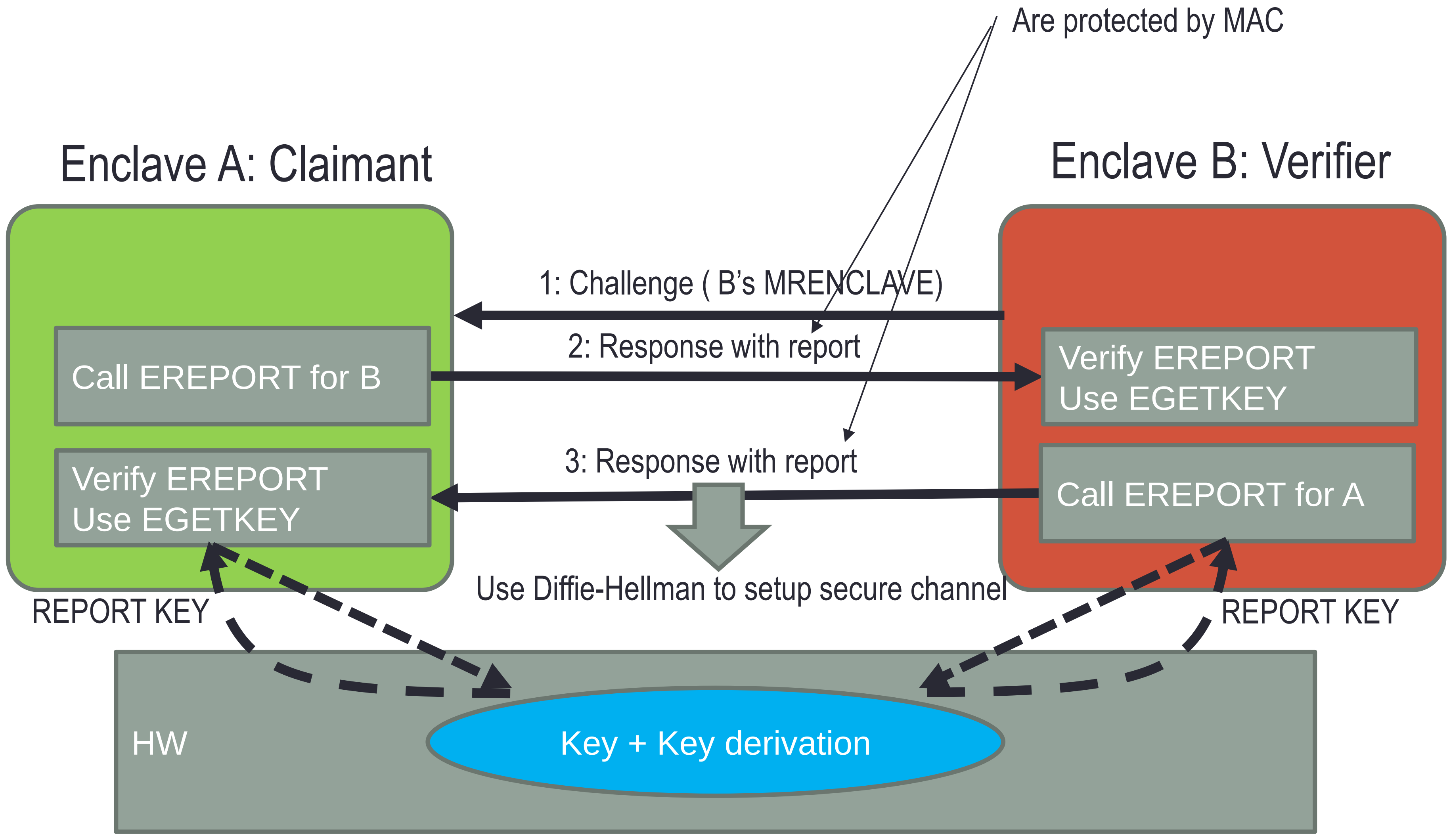
For Intel SGX there are in essence two solutions

- 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 crypto scheme)



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

Local attestation



Remote attestation = QUOTING

- SETUP
  - The HW platform has an identity key (EPID type key) that is used for signing and for which an certificate exists that can be used to verify signatures that have created by signing with this key.
  - Intel maintains a server the *Intel Attestation Server (IAS)* where the certificate obained and can be checked for validity.

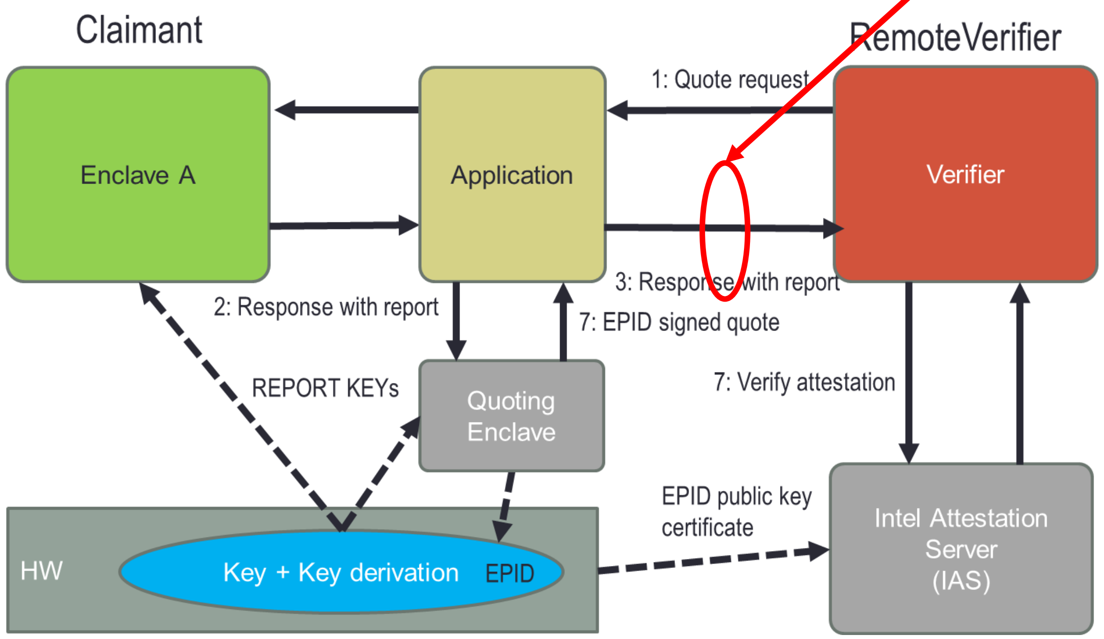
- QUOTING process
  - The attestation is performed indirectly using a quoting enclave that signs the quote
  - The validity of quote is verified using the IAS.

The term ’quoting’ is also used in the Trusted Computing Group specifications when performing 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 the enclave A



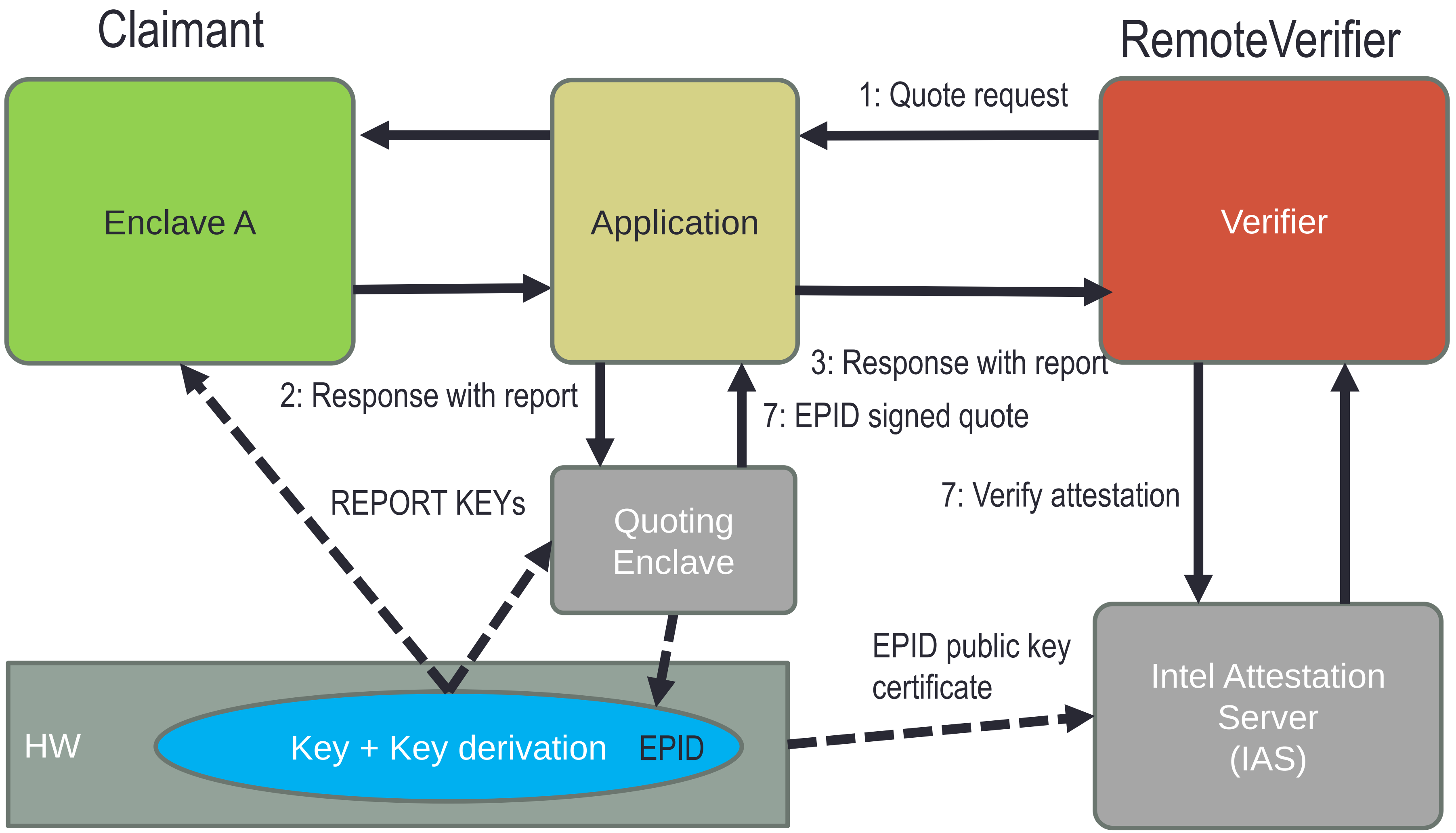
The verifier compares report data using

1. Quote reference values from ASIC supplies

2. Quote reference values from ISV (code developer)

Remote attestation – the verifier

Are protected by MAC



Two use cases of SGX

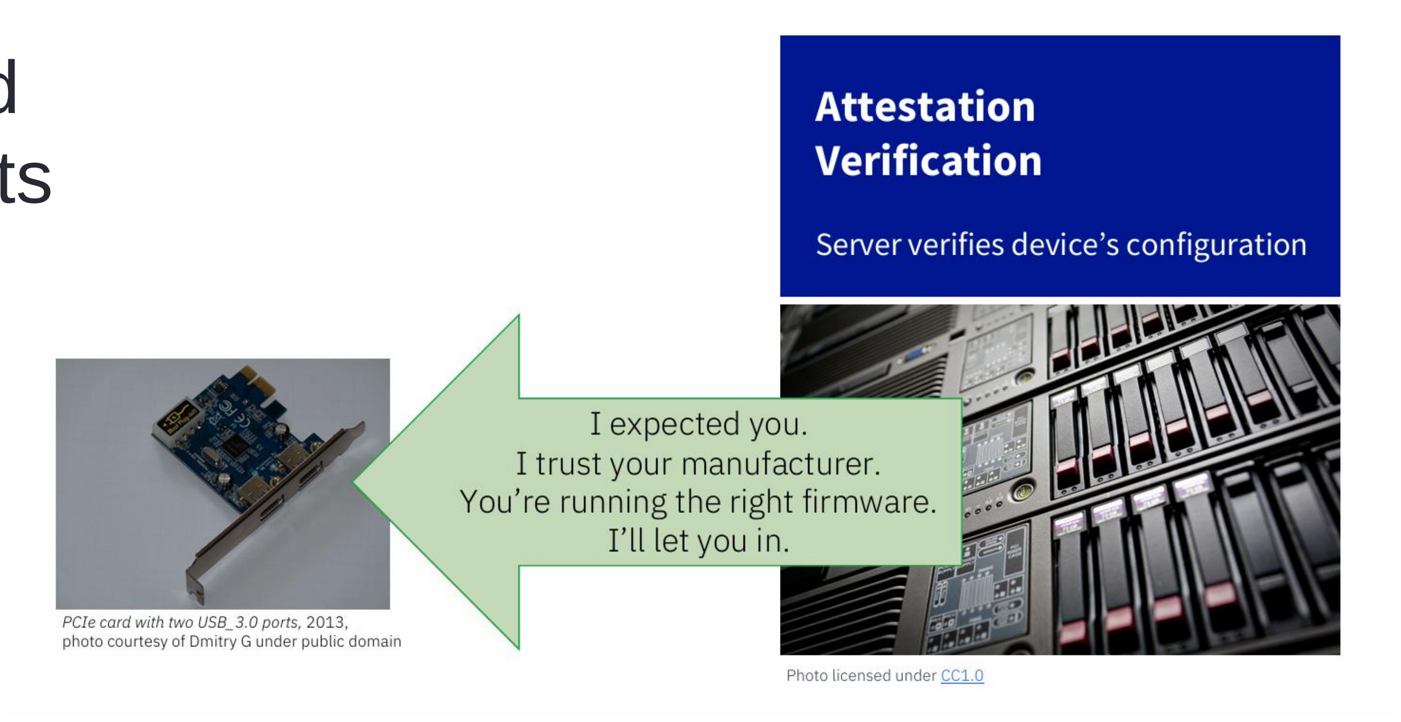
- **Protecting Machine-Learning models:**
  - ML models are trained on valuable data and as such one often wants to keep the model confidential. SGX can be used to perform ML based computation in cloud without “loosing” the model.

- **Application (e.g., TLS) key protection:**
  - Use attestation
    - to check if we indeed have trustworthy enclave
    - Establish trust relation between enclave and verifier
  - Then load or create key into enclave
    - Use trust relation to make this key to get a trusted identity (e.g. a certificate)

Board attestation

- Attestation is today also used to secure that the components in a (server) system are trustworthy

- Link CPU packages
- Link perihpherals


Attestation Verification
Server verifies device's configuration
I expected you.
I trust your manufacturer.
You're running the right firmware.
I'll let you in.
PCIe card with two USB_3.0 ports, 2013, photo courtesy of Dmitry G under public domain
Photo licensed under CC1.0

DMTF (former Distributed Management Task Force): [All DMTF Standard Publications | DMTF]

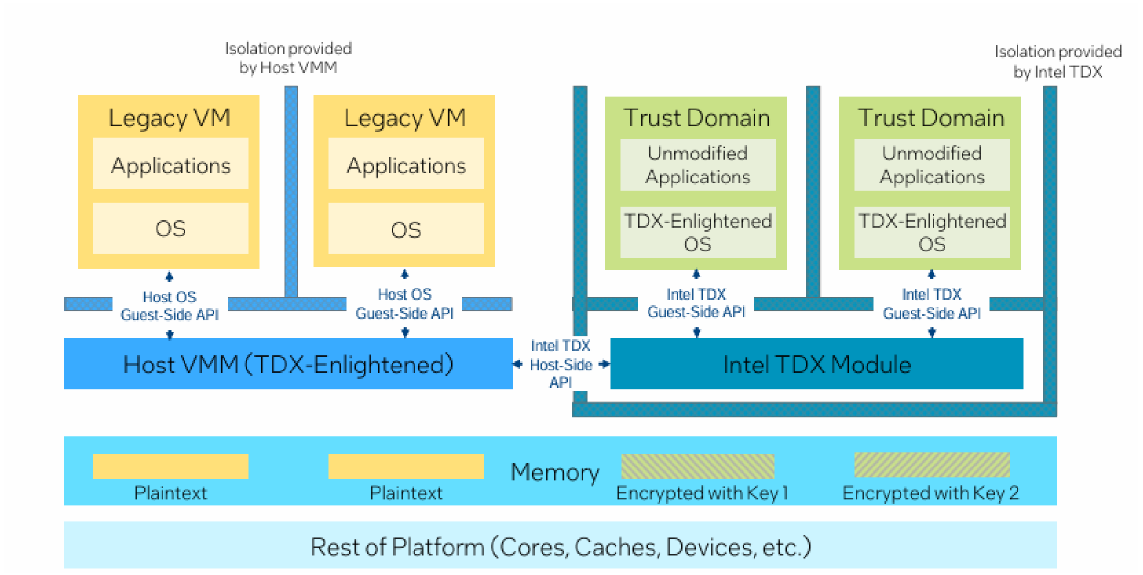
[https://crypto.orange-labs.fr/acg/workshop/pdf/8-IBM-Attestation_in_IT_v2.pdf]

Other confidential computing technologies

Name reference	Description
Intel TDX	- enforces cryptographic VM isolation - Supports memory encryption and integrity protection - Supports attestation, Leveraging SGX attestation - SW debug with loss of confidentiality
AMD SEV	- enforces cryptographic VM isolation via AMD PSP - supports memory encryption (SEV), CPU state encryption (SEV-ES), integrity protection (SEV-SNP) - provides hardware isolated layers within VMs through VMPL
IBM Secure Execution	- protects storage virtual machines (SVMs) on IBM Z and LinuxONE. - leverages a trusted firmware, Ultravisor, to bootstrap and run SVMs - provides end-to-end protection from the boot image to memory and throughout execution
IBM PEF	- protects SVMs on Power ISA - leverages the Protected Execution Ultravisor to manage SVM execution - utilizes TPM, secure boot, and trusted boot for integrity check and bootstrap SVMs
Arm CCA	- introduces Realm world for running confidential VMs - introduces Root world to enforce address space isolation through GPT - support attestation for Realm environment
RISC-V CoVE	- introduces the Trusted Security Manager (TSM) to manage Trusted Virtual Machine (TVM) life cycles - uses a Memory Tracking Table (MTT) to track memory page assignment - adopts a layered attestation architecture

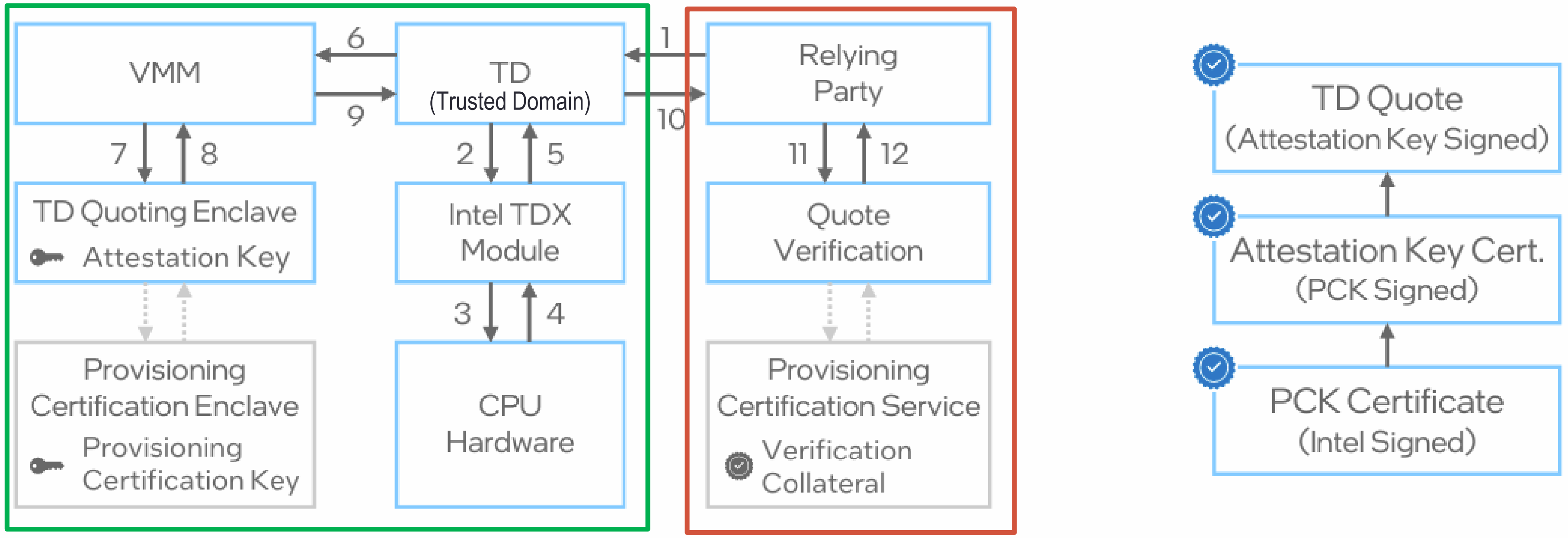
Intel TDX (trusted domain extension)

- 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 the runs in the normal OS world.
  - This complicates things when using existing software – some solutions exist for this. but the solutions are not ideal (for example not covering all languages)
- TDX creates a TEE where a whole VM can run



Intel TDX remote attestation

- Process:



- TDX leverages Intel SGX attestation:
  - One set of PCK certificates, distribution, caching services to support SGX & TDX
  - Requires SGX be enabled in host for TDX attestation

From Intel TDX Deep dive, Benny Fuhry, Intel, 2024-02-04

AMD SEV-(SNP)

- SEV = Secure Encrypted Virtualization
- SEV-SNP = + Secure Nested Paging

Two variants/versions 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

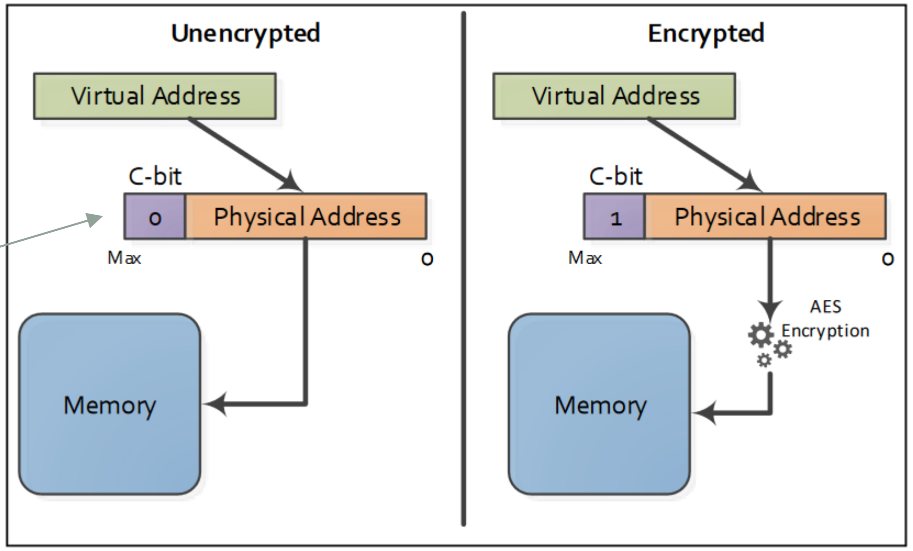
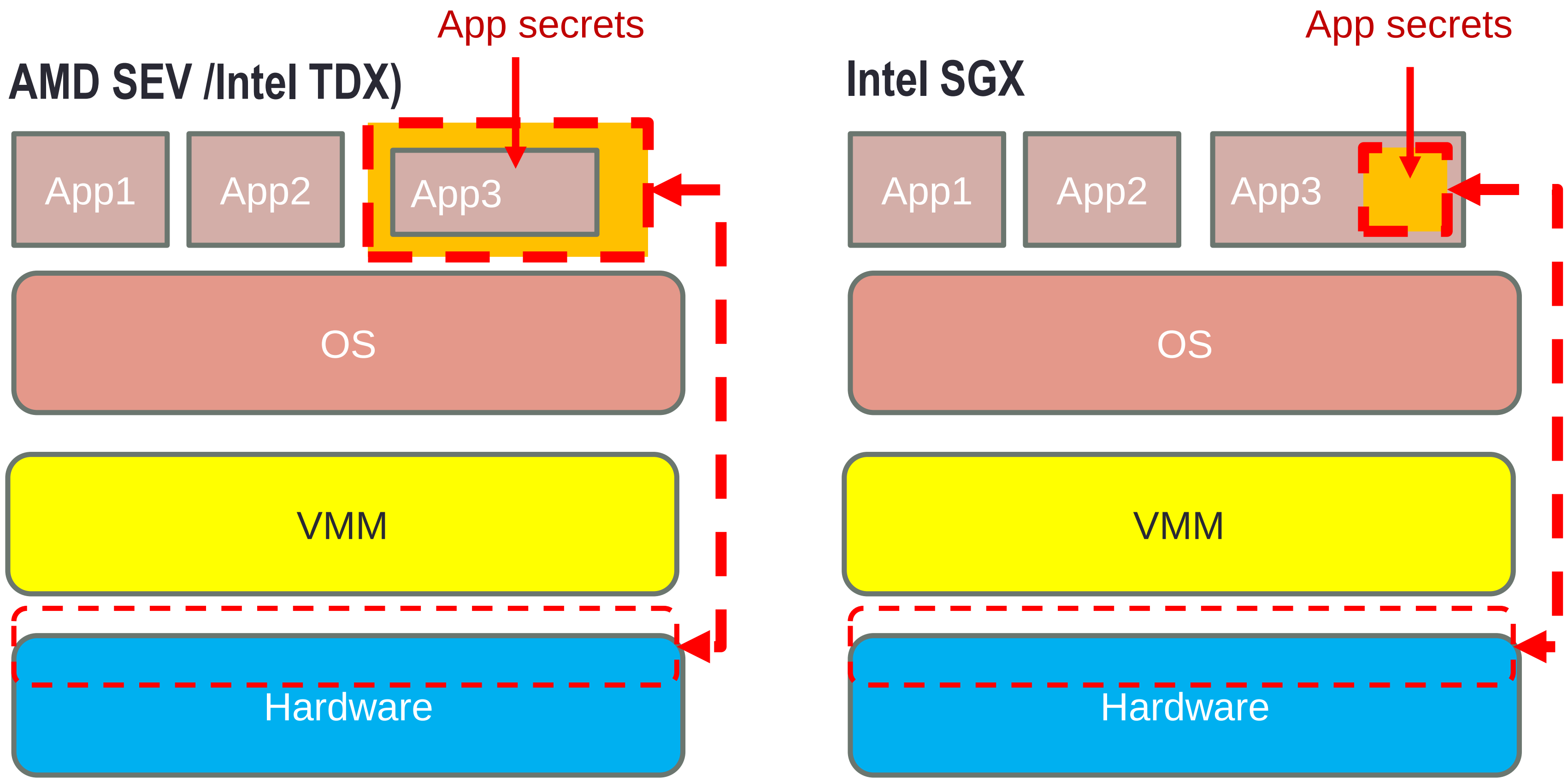


FIGURE 1: ENCRYPTION CONTROL

AMD SEV vs SGX

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



AMD SEV

Compared to Intel SGX, AMD SEV

- requires no changes of application to run it in encrypted space
- can already by 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

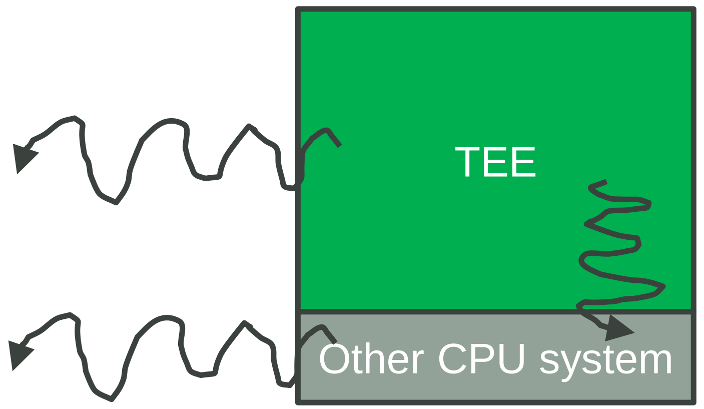
- There are industrial activities ongoing to establish frameworks that are technology neutral (i.e. cover SGX as well as SEV) that allow the
  - Execution of code in an HW protected environment
  - Code and data is encrypted
  - Environment can be verified through remote attestation

E.g.

- Fortanix: Fortanix products
- Redhat: Enarx: https://enarx.dev/
- Microsoft: Azure Confidential Computing
- Google: Google Cloud

General problems of TEEs

- Side-channel attacks
- Side-channel attacks
- And mode side-channel attacks



- ASICS are very complex and often have not been designed from the starts to deal with the threat of side-channels that leak information. So, it is very hard to be sure that – also in TEE solutions – side-channels do not exist.

- And frankly HW vendors are rather clear that 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 the “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 remote attestation wrt to trustworthiness?
- What is the main principle of Zero Trust
- What is a RoT and 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 relevance?
- 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

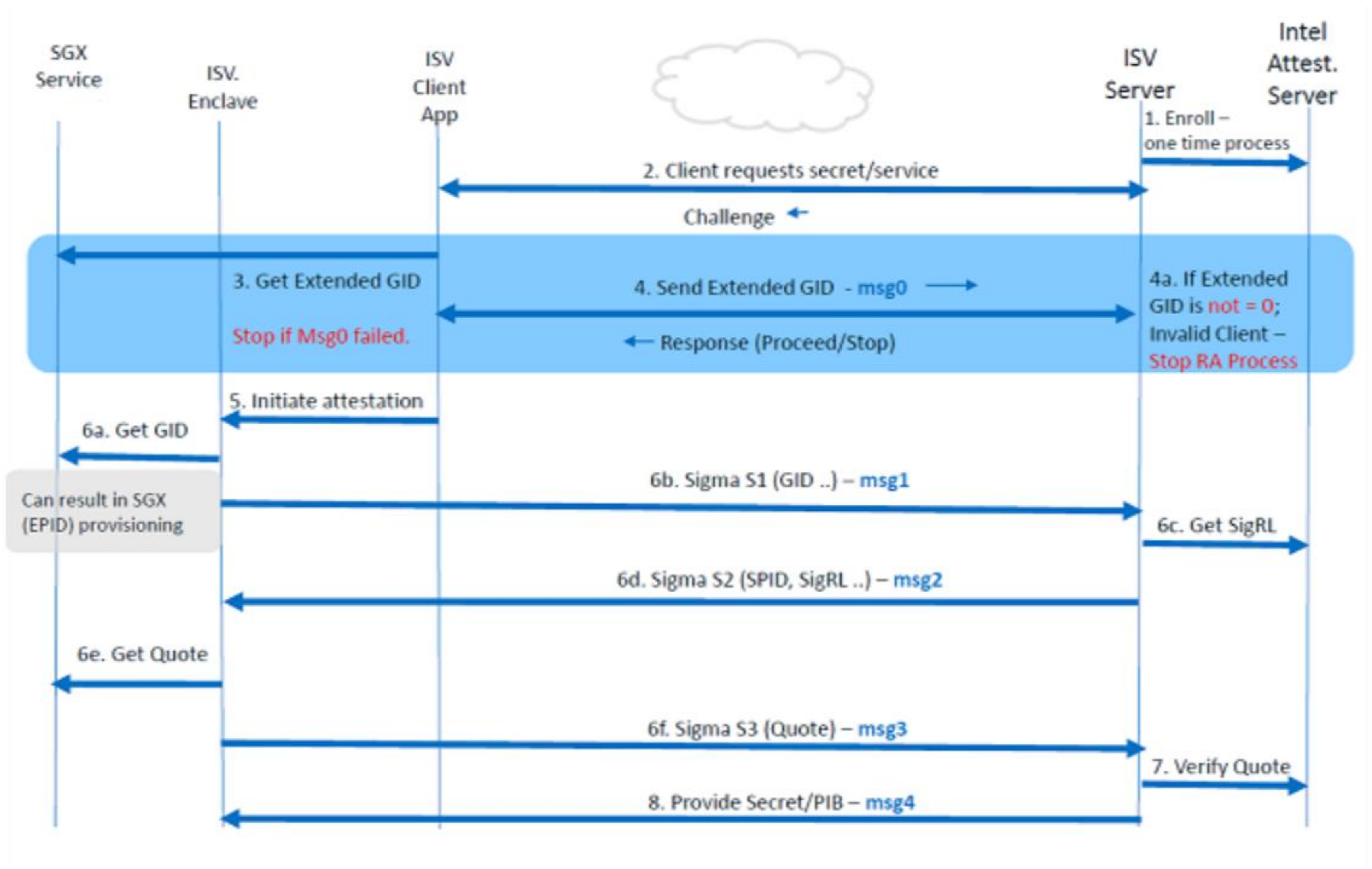
- Why is there a need to have ecalls and ocalls in SGX?
- SGX lacks a secure interrupt what does that imply wrt to starving an enclave by the OS?
- Explain the role of MRENCLAVE and MRSIGNER
- Does SealKey always depend on MRENCLAVE / MRSIGNER?
- Can I just program an application with an enclave and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

BONUS MATERIAL

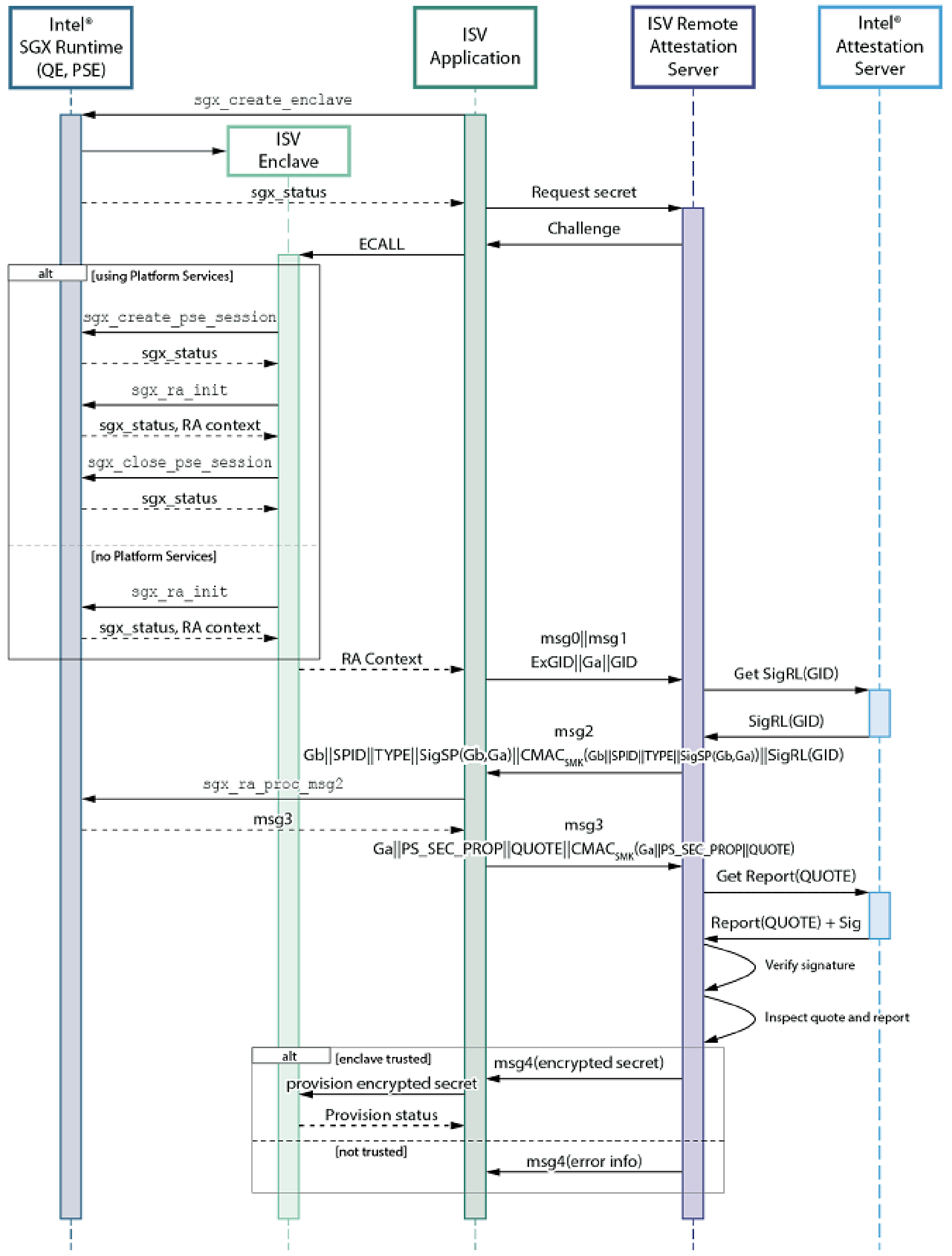
Enclave Lifecycle

- To initialize an enclave, four of the new CPU instructions provided by SGX architecture are used, each will be provided with a wrapper available for developers' usage (will be explained in sample code):
- **ECREATE**
  - An enclave is born when the system software issues the ECREATE instruction, which turns a free EPC page into the SECS for the new enclave.
  - ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software. Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE, and ATTRIBUTES.
  - ECREATE validates the information used to initialize the SECS, and results in a page fault or general protection fault if the information is not valid. ECREATE will also fault if SECS target page is in use; already valid; outside the EPC; addresses are not aligned; unused PAGEINFO fields are not zero.
- **EADD**
  - The system software can use EADD instructions to load the initial code and data into the enclave. EADD is used to create both TCS pages and regular pages. This function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. EADD reads its input data from a Page Information (PAGEINFO) structure.
  - The PAGEINFO structure contains:
  - The virtual address of the EPC page (LINADDR)
  - The virtual address of the non-EPC page whose contents will be copied into the newly allocated EPC page (SRCPGE)
  - A virtual address that resolves to the SECS of the enclave that will own the page (SECS).
  - The virtual address, pointing to a Security Information (SECINFO) structure, which contains the newly allocated EPC page's access permissions (R, W, X) and its EPCM page type (RT_REG or PT_TCS).
  - EADD validates its inputs, and modifies the newly allocated EPC page and its EPCM entry.
  - EADD ensures that the EPC page is not allocated to another enclave, the page's virtual address falls within the enclave's ELRANGE and all the reserved fields in SECINFO are set to zero.
- **EEXTEND**
  - While loading an enclave, the system software will also use the EEXTEND instruction, which updates the enclave's measurement used in the software attestation process. It updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string compromising of "EEXTEND" || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. RCX register contains the effective address of the 256 byte region of an EPC page to be measured.
- **EINIT**
  - This function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using EENTER instruction.
  - When EINIT completes successfully, it sets the enclave's INIT attribute to true. This opens the way for ring 3 application software to execute the enclave's code, using the SGX instructions. On the other hand, once INIT is set to true, EADD cannot be invoked on that enclave anymore, so the system software must load all the pages that make up the enclave's initial state before executing the EINIT instruction.
- **END OF ENCLAVE INITIALIZATION**

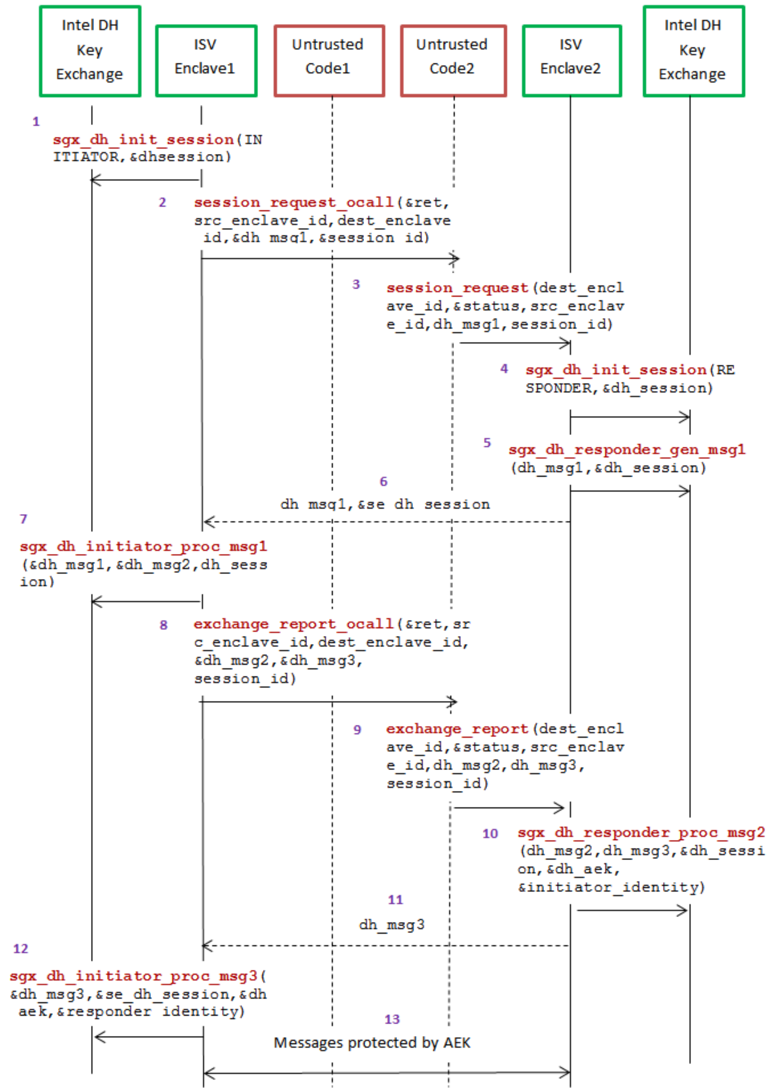
Remote attestation protocol


SGX Service
ISV. Enclave
ISV Client App
ISV Server
Intel Attest. Server
1. Enroll – one time process
2. Client requests secret/service
Challenge
3. Get Extended GID
Stop if Msg0 failed.
4. Send Extended GID - msg0
Response (Proceed/Stop)
4a. If Extended GID is not = 0; Invalid Client – Stop RA Process
5. Initiate attestation
6a. Get GID
Can result in SGX (EPID) provisioning
6b. Sigma S1 (GID ..) – msg1
6c. Get SigRL
6d. Sigma S2 (SPID, SigRL ..) – msg2
6e. Get Quote
6f. Sigma S3 (Quote) – msg3
7. Verify Quote
8. Provide Secret/PIB – msg4

Official flow



Local attestation in Intel-sdk


Intel DH Key Exchange
ISV Enclave1
Untrusted Code1
Untrusted Code2
ISV Enclave2
Intel DH Key Exchange
1
sgx_dh_init_session(INITIATOR,&dhsession)
2
session_request_ocall(&ret, src_enclave_id,dest_enclave id,&dh_msg1,&session_id)
3
session_request(dest_enclave_id,&status,src_enclave_id,dh_msg1,session_id)
4
sgx_dh_init_session(RESPONDER,&dh_session)
5
sgx_dh_responder_gen_msg1 (dh_msg1,&dh_session)
6
dh_msg1,&se_dh_session
7
sgx_dh_initiator_proc_msg1 (&dh_msg1,&dh_msg2,dh_session)
8
exchange_report_ocall(&ret,src_enclave_id,dest_enclave_id, &dh_msg2,&dh_msg3, session_id)
9
exchange_report(dest_enclave_id,&status,src_enclave_id,dh_msg2,dh_msg3, session_id)
10
sgx_dh_responder_proc_msg2 (dh_msg2,dh_msg3,&dh_session,&dh_aek, &initiator_identity)
11
dh_msg3
12
sgx_dh_initiator_proc_msg3(&dh_msg3,&se_dh_session,&dh_aek,&responder_identity)
13
Messages protected by AEK

Owner EPOCH

- Owner Epoch should be configured by the user during boot time by inserting a password. The value is then stored persistently between power cycles on platform’s nonvolatile flash memory.
- After the platform has booted, Owner Epoch value is stored in a dedicated register and is accessible only by enclave’s trusted security interface as an additional parameter for deriving enclave-only products.

- Note: The Owner Epoch value must remain the same for an enclave to obtain the same keys used by previous runs. However, it may be desired to change the Owner Epoch value when platform changes owner. This denies access of a new platform user to personal information previously sealed by other users until the correct Owner Epoch password is restored.

Use of the IAS

- In order to utilize the Intel IAS for Remote Attestation, Intel requires that the independent software vendor (ISV) take the following steps:

1. Obtain a signed certificate from a recognized certificate authority. Refer to Certificate Requirements for Intel Attestation Services for more information. For testing purposes, the ISV may use a self-signed certificate instead of one signed by an authority. Follow this guide on How to create Self-Signed Certificates for use with Intel SGX Remote Attestation using OpenSSL.
2. Register that certificate with Intel® Development Services by completing the Development Services Access Request form. Note that an Intel® Developer Zone (Intel® DZ) account is required for this step. Join the Intel Developer Zone to register for an account.

There is a way, if needed, that does allow the ISV to determine whether multiple requests originated from the same platform.

Privacy properties for EPID

- EPID key issuing can be blinded
  - Issuer does not need to know Member Private Key • EPID signatures are anonymous •
- EPID signatures are untraceable
  - Nobody including the issuer can open an EPID signature and identify the member

Facts

Details of EPID are complicated see the DrDobbs article on EPID for an overview if you are interested

EPID construction is based on

admissible bilinear map functions, that is mappings

$$e: G1 \times G2 \rightarrow GT$$

that satisfy the following

A construction of e: Tate pairing using elliptic curves over GF(q)

For all $u \in G1$, $v \in G2$, and for all integers a, b, the equation

$$e(u^a, v^b) = e(u, v)^{ab}$$

holds.

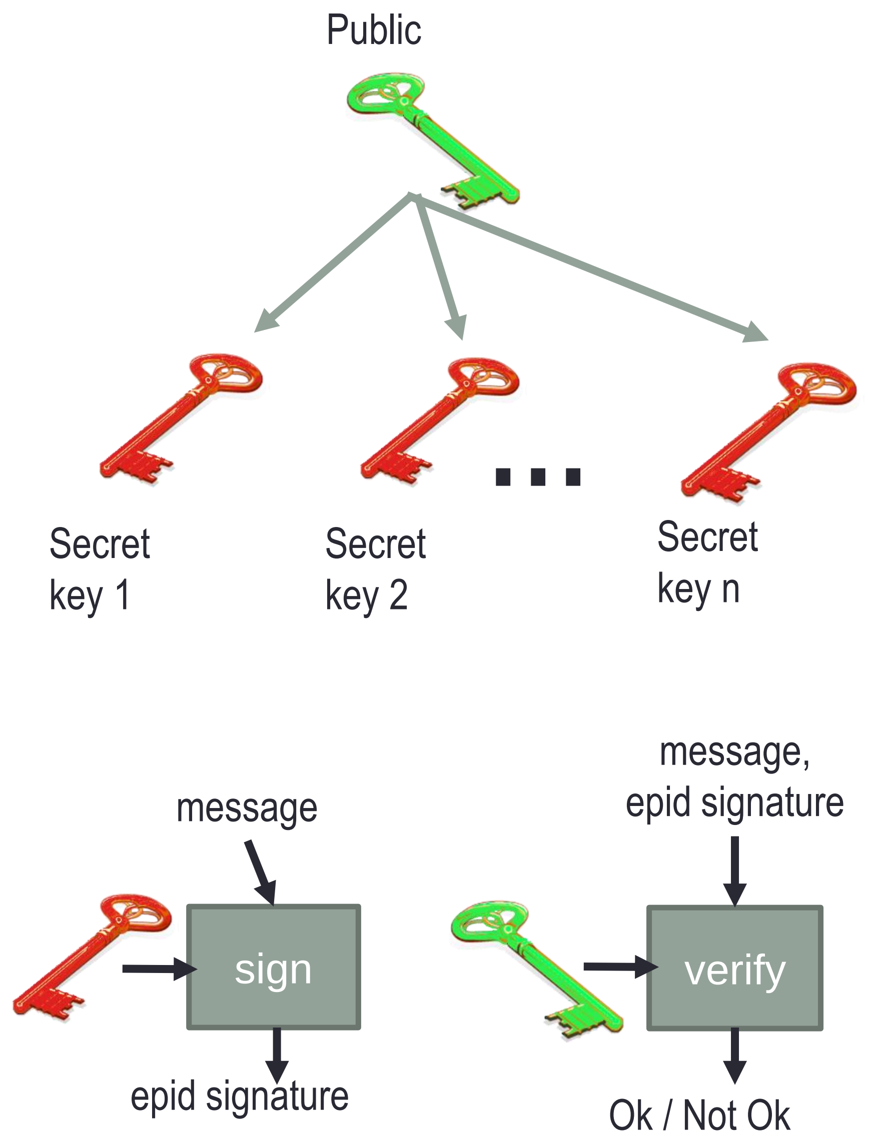
EPID (identity for SGX 1.0)

- EPID keys are keys that are programmed into most of Intel chipsets and play an important role in SGX 1.0.
- The use of EPID has received criticism and likely newer SGX version will provider alternatives to EPID keys.
- EPID keys are group keys that to some degree provide unlikability (anonymity)

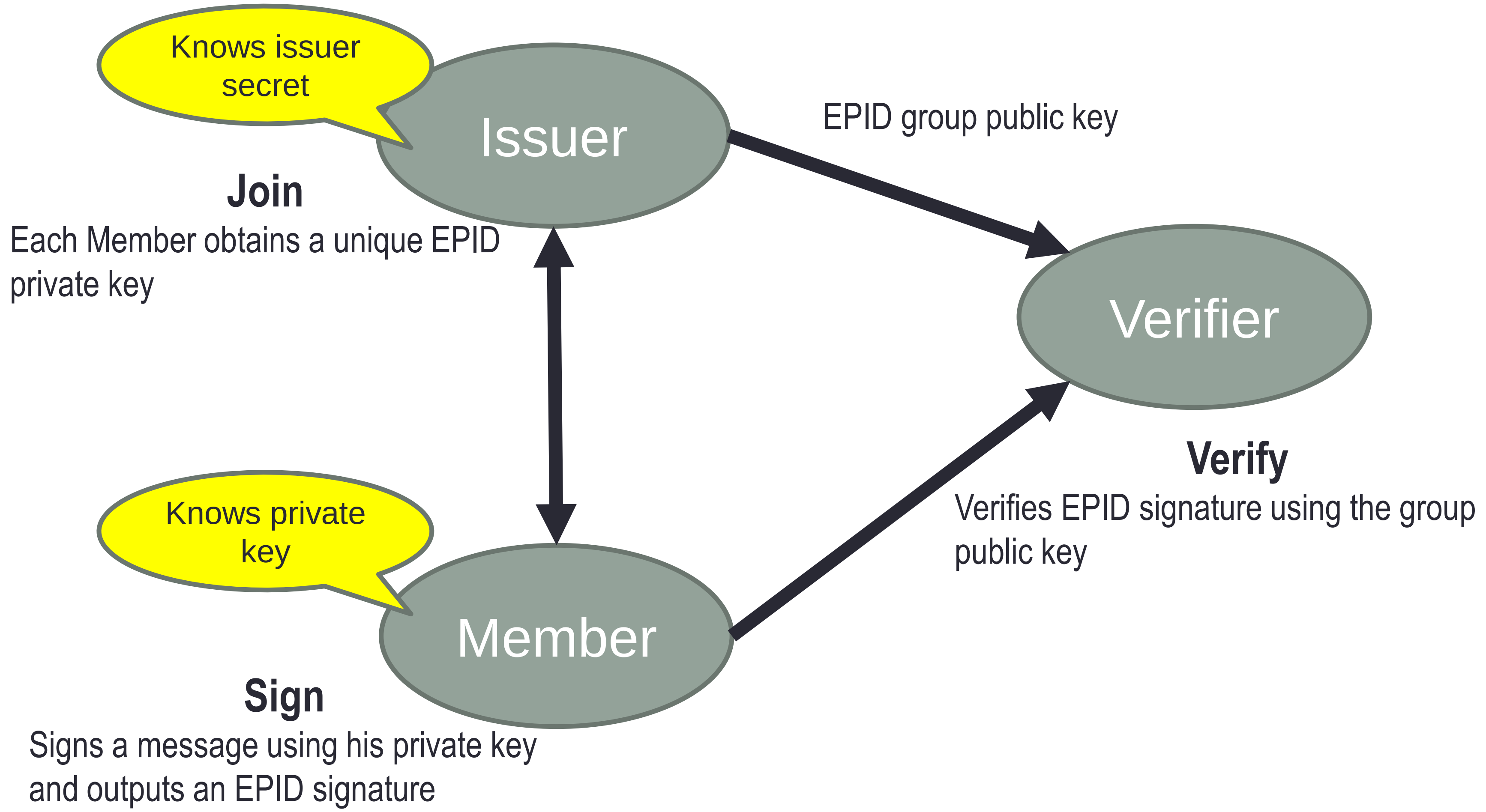
See information on EPID in reading material

EPID identities in SGX

- To support attestation SGX can use EPID identities
- **One group public key** corresponds to multiple private keys
  - Each unique private key can be used to generate a signature
  - Signature can be verified using the group public key
  - But: needs a rather involved revocation solution



EPID setup


Knows issuer secret
Issuer
EPID group public key
Join
Each Member obtains a unique EPID private key
Verifier
Verify
Verifies EPID signature using the group public key
Knows private key
Member
Sign
Signs a message using his private key and outputs an EPID signature

http://csrc.nist.gov/groups/ST/PEC2011/presentations2011/b

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.