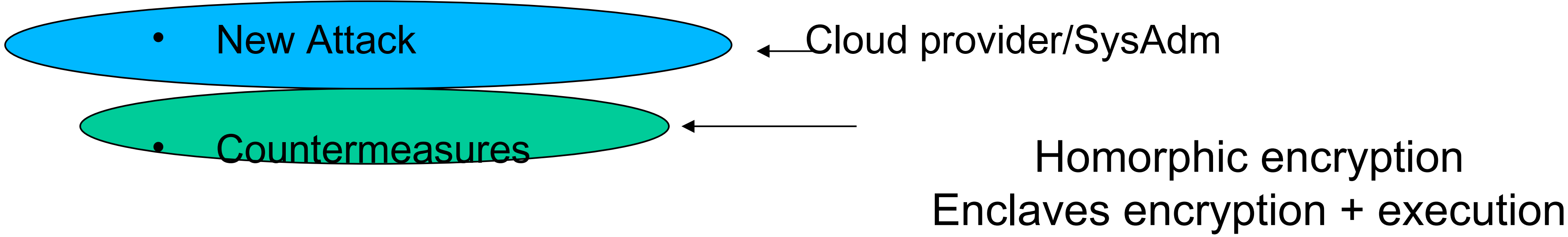
ICT Risk Assessment

Fabrizio Baiardi
f.baiardi@unipi.it

Syllabus

- Security
  - New Threat Model



  - New Attack
  - Countermeasures

Cloud provider/SysAdm

Homorphic encryption
Enclaves encryption + execution

Working with encrypted data (soft solution)

- A client stores its data on a cloud system
- The client wants to implement some computations on the data without leaking any information about
  - the data
  - the data and which data is used by the computation
- Examples
  - Store your personal information on the cloud and compute your tax declaration
  - Store some information on the cloud and search this information
- Requires some proper encryption scheme because only a few schemes satisfies the constrains

Homorphic encryption = Holy gray of encryption

Let

- *R* and *S* be sets
- E an encryption function $R \to S$

E is

- **Additively homomorphic if** $E(a+b)=\mathrm{PLUS}(E(a), E(b))$
- **Multiplicatively homomorphic if** $E(a \times b)=\mathrm{MULT}(E(a), E(b))$
- **Mixed-multiplicatively homomorphic** $E(xy)=\text{Mixed-mult}(E(x),E(y))$

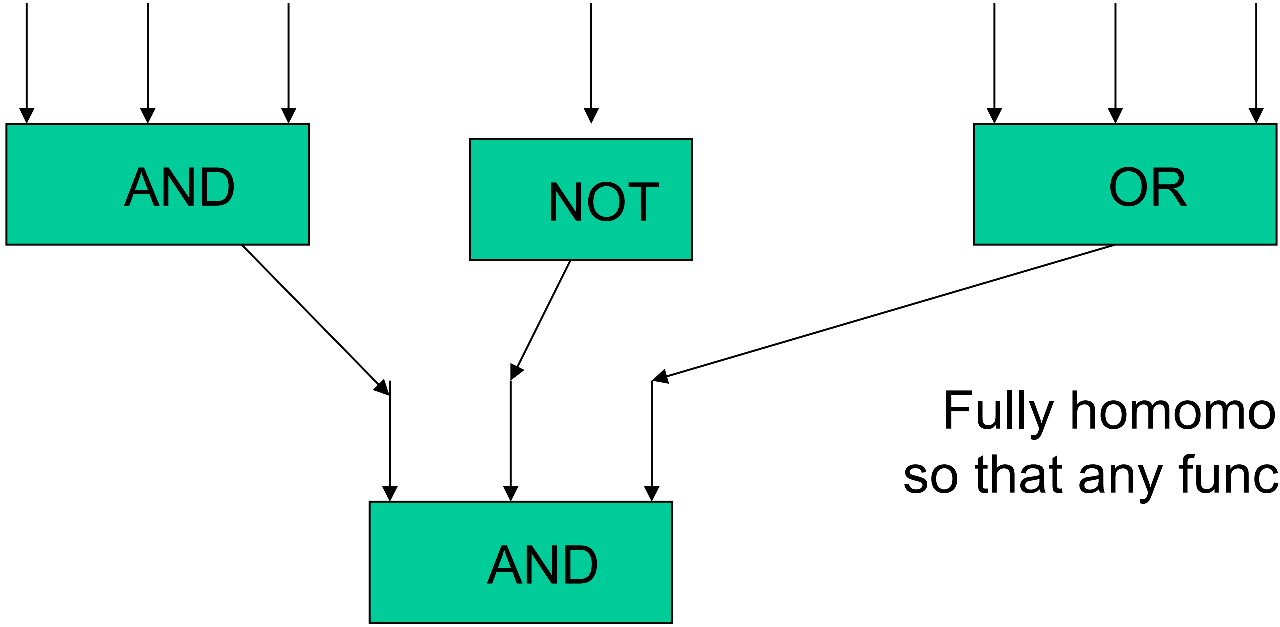
E is fully homomorphic if there are no limitations on the manipulations that can be performed.

Homomorphic encryption

- Data is stored at the provider
- Computation is implemented at the provider
- Inputs are encrypted by the client
- The output is transmitted to the client that decrypts it
- No trivial solution is accepted = almost all the computation has to be executed by the provider to prevent cases where
  - the data is transmitted to the client,
  - the client decrypts the data
  - the client computes the results
  - the results are encrypted
  - the results are transmitted to the provider

Fully homomorphic

In the following the manipulation will be represented as a circuit that implements some boolean operations on the data of interest and where the operators are gates



Fully homomorphic = NAND gates
so that any function can be computed

Meaning

- Any computation can be expressed as a Boolean circuit: a series of additions and multiplications
- Using such a scheme, any circuit (consisting of AND and XOR) could be homomorphically evaluated, effectively allowing the construction of programs which may be run on the encryptions of their inputs to produce an encryption of their output
- Since such a program never decrypts its inputs, it could be run by an untrusted party without revealing its inputs and internal state.

But our case introduce further constrains-1

- No optimization of the computation is possible
  - Circuit minimization may not be applied because it leaks information about data that is accessed
  - A random access machine cannot be used because it leaks the information of which data has been accessed by the computation (use an Oblivious RAM is possible)
  - This efficiency can be recovered only if information about data that has been used can leak

- The size of the output must be fixed in advance = the number of output wires in the circuit must be fixed in advance.
  - If I request all of my files that contain a combination of keywords, I should also specify how much data I want to be retrieved (e.g. 1MB).
  - From my request, the cloud will generate a circuit for a function that outputs the first megabyte of the correct files,
  - The output is truncated or padded with zeros prevent leaking something a priori about the relationship between the function (that is known) and my data.

But our case introduce further constrains-2

- ***semantic security*** against chosen-plaintext attacks (CPA) : given a ciphertext *c* that encrypts either *m*0 or *m*1, it is hard for an adversary A to decide which of the two values c encrypts, even if it is allowed to choose both *m*0 and *m*1.

  “hard” = if A runs in polynomial time and guesses correctly with probability 1/2 + *OE*, then *OE* = A’s *advantage*, must be negligible.

  Otherwise, A *breaks* the semantic security of the encryption scheme.

- If an encryption scheme is *deterministic* (= there is only one ciphertext that encrypts a given message) then it cannot be semantically secure.

  An attacker can easily tell whether *c* encrypts *m*0 by encrypting *m*0 and by checking if the results is equal to *c*.

- A semantically secure encryption scheme must be *probabilistic*
  - several ciphertexts that encrypt a given message
  - encryption chooses one randomly according to some distribution

Encryption scheme e

- Four algorithms
  - KeyGen_e , Encrypt_e , Decrypt_e (must be efficient)
  - Evaluate_e
- *Efficient =* runs in time poly(L) where L = bit-length of the keys.
- KeyGen_e uses L to generate
  - a single key sk in a *symmetric* scheme,
  - two keys an *asymmetric* scheme, a public key pk and secret key sk.
- Evaluate_e is associated to a set F_e of *permitted functions* such that
  - f in F_e
  - if $c1, \ldots, ct$ are such that $ci = \text{Encrypt}_e(\text{pk}, mi)$ then
    - $\text{Evaluate}_e,(\text{pk}, f, c1, \ldots, ct) = c$
    - $f(m1, \ldots, mt) = \text{Decrypt}_e(\text{sk}, c)$ (sk if symmetric)

e is fully homomorphic if any function belongs to F_e

Constrains

- decrypting *c,* the output of Evaluate$_e$ takes the *same amount of computation* as decrypting *c*1, a ciphertext output by Encrypt$_e$
- *c* is the same size as *c*1 (*compact ciphertexts* requirement)

Informally,

- the size of *c* and the time needed to decrypt it do not grow with the complexity of *f*; rather, they are *completely independent* of *f*
- the complexity of Decrypt$_e$, as well as those of KeyGen$_e$ and Encrypt$_e$, must remain polynomial in L

A first approximation - 1

Assume L= N, P = L^2 and Q = L^5 *A (symmetric) Encryption Scheme*

KeyGen$_e$(L): The key is a random P-bit odd integer p.

Encrypt$_e$(p, m): To encrypt a bit m in {0, 1},

1) choose a random N-bit number m' such that $m' = m \bmod 2$.
2) output $c = m' + pq$, where q is a random Q-bit number.

Decrypt$_e$(p, c): Output (c mod p) mod 2 where

1) (c mod p) = c' in $(-p/2, p/2)$
2) p divides $c - c'$

m' and m have the same parity

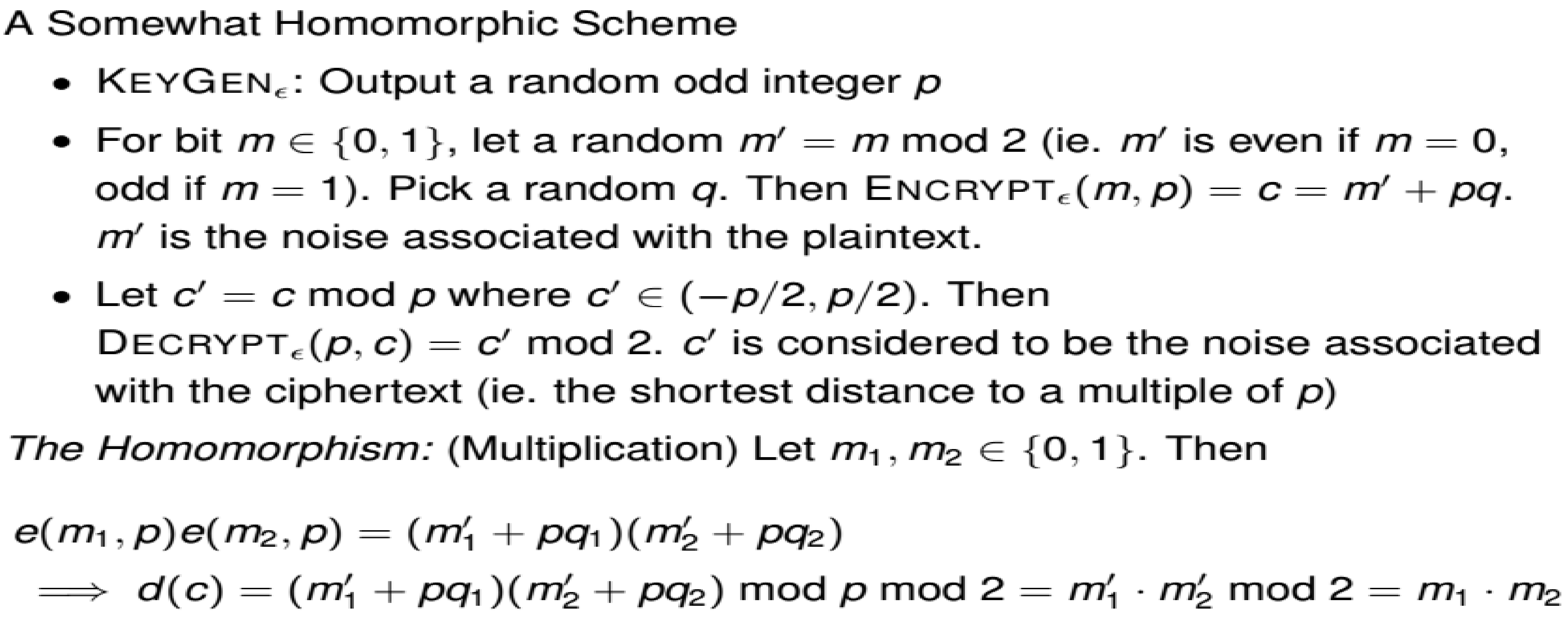
we recover q by finding the multiple of p closest to c and the noise parity is the encrypted bit

(c mod p) = *noise* associated to the ciphertext c
= distance to the nearest multiple of p

Decryption works because the noise m' has the same parity as the message m.

A ciphertext output by Encrypt is a *fresh* ciphertext, since it has small (N-bit) noise.

A first approximation – 1 bis



A Somewhat Homomorphic Scheme

- $\textsc{KeyGen}_\epsilon$: Output a random odd integer p
- For bit $m \in \{0, 1\}$, let a random $m' = m \bmod 2$ (ie. m' is even if $m = 0$, odd if $m = 1$). Pick a random q. Then $\textsc{Encrypt}_\epsilon(m, p) = c = m' + pq$. m' is the noise associated with the plaintext.
- Let $c' = c \bmod p$ where $c' \in (-p/2, p/2)$. Then $\textsc{Decrypt}_\epsilon(p, c) = c' \bmod 2$. c' is considered to be the noise associated with the ciphertext (ie. the shortest distance to a multiple of p)

The Homomorphism: (Multiplication) Let $m_1, m_2 \in \{0, 1\}$. Then

$$e(m_1, p)e(m_2, p) = (m_1' + pq_1)(m_2' + pq_2)$$

$$\implies d(c) = (m_1' + pq_1)(m_2' + pq_2) \bmod p \bmod 2 = m_1' \cdot m_2' \bmod 2 = m_1 \cdot m_2$$

A first approximation - 2

$\text{Add}_e(c1, c2) = c1 + c2$

$\text{Sub}_e(c1, c2) = c1 - c2$

$\text{Mult}_e(c1, c2) = c1 \bullet c2.$

$\text{Evaluate}_e(f, c1, \ldots, ct) =$

1) Express f as a circuit C with XOR and AND gates
2) Let C' be the same circuit as C, but with XOR and AND gates replaced by addition and multiplication gates over the integers.
3) Output $c = f'(c1, \ldots, ct)$ where f' is the multivariate polynomial that corresponds to C'.

If this work, we can deduce a pubblic encryption scheme

A full homomorphic scheme

Suppose that

a) *e* can handle *De* augmented by some gate, e.g., Add; call this augmented circuit *D*Add.
b) *c*1 and *c*2 encrypt *m*1 and *m*2 respectively, under pk1,

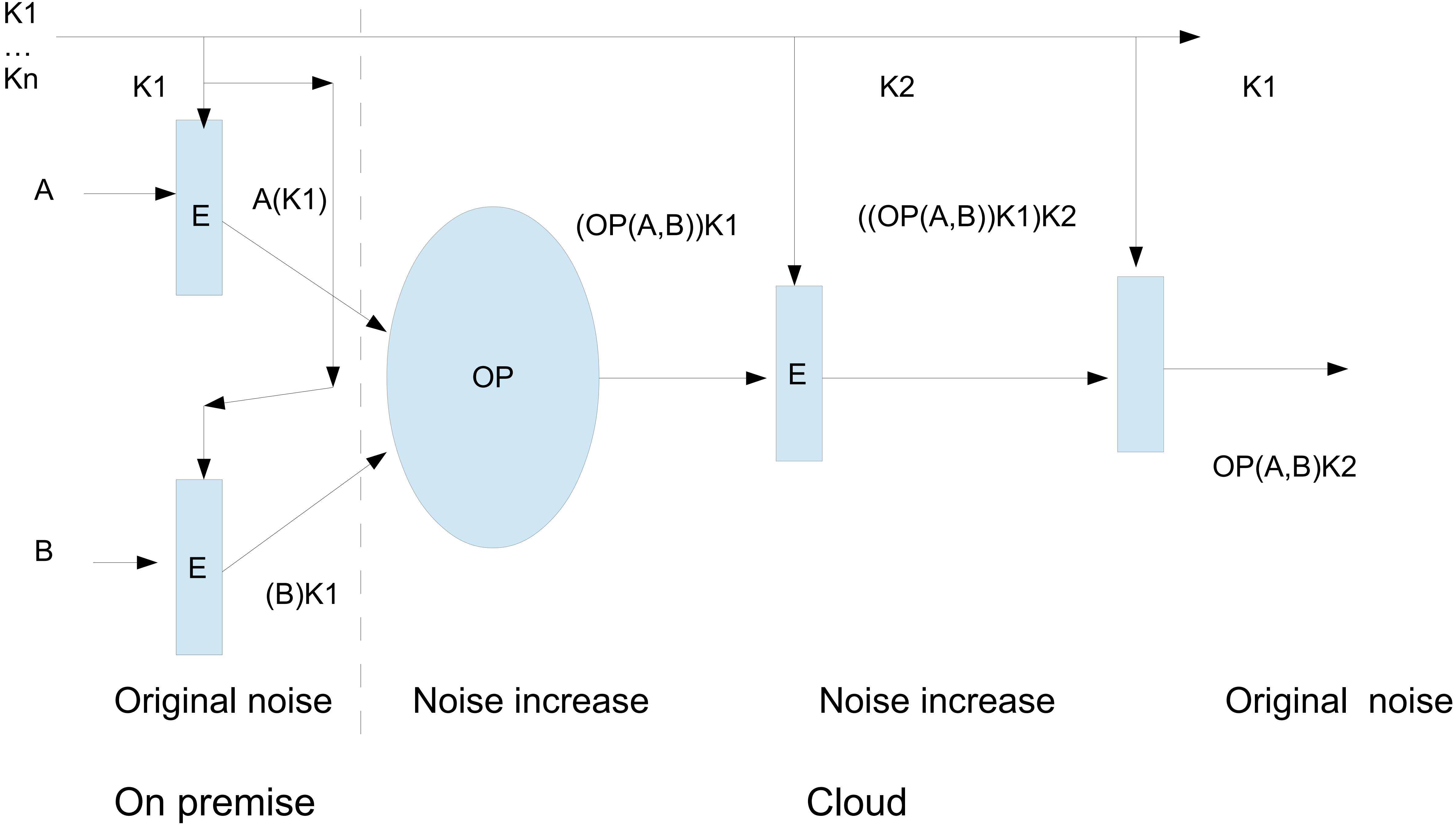
if *c′* 1 and *c′* 2 encrypt the bits of the ciphertexts under pk2 then

c =Evaluate$_e$(pk2, *D*Add, sk1, *c* ‘1 , *c* ‘2)

encrypts $m1 \oplus m2$ under pk2 .

We get a fully homomorphic encryption scheme *e*’ by recursing this process where the key in *e*’ is

a) a sequence of public keys (pk1, …, pka+1)
b) a chain of encrypted secret keys sk1, ..., ska, where sk*i* is encrypted under pk*i*+1.


K1
…
Kn
K1
K2
K1
A
E
A(K1)
(OP(A,B))K1
((OP(A,B))K1)K2
OP
E
OP(A,B)K2
B
E
(B)K1
Original noise
Noise increase
Noise increase
Original noise
On premise
Cloud

A full homomorphic scheme

To evaluate a function *f* in *e*’,

1. we express *f* as a circuit, topologically arrange its gates into levels,
2. scan sequentially the levels and for a gate at level $i + 1$(e.g., an Add gate)
   1. take as input the encrypted secret key sk*i* and a couple of ciphertexts associated to output wires at level *i* that are under pk*i*,
   2. homomorphically evaluate *D*Add to get a ciphertext under pk*i*+1 associated to a wire at level $i + 1$.
3. output the ciphertext associated to the output wire of *f*.

Putting the encrypted secret key bits sk1, ..., ska in the public key of *e*’ is not a problem for security because these bits are indistinguishable from encryptions of 0 as long as *e* is semantically secure

Last step: reduce the complexity of the key, instead of several pubblic keys we have the same key for all the level (no information is leaked by revealing the encyption of a secret key under a pubblic key, circular security)

Breakthrough(2009)

IBM Researcher Solves Longstanding Cryptographic Challenge

Discovers Method to Fully Process Encrypted Data Without Knowing its Content; Could Greatly Further Data Privacy and Strengthen Cloud Computing Security

ARMONK, N.Y. - 25 Jun 2009: An IBM Researcher has solved a thorny mathematical problem that has confounded scientists since the invention of public-key encryption several decades ago. The breakthrough, called "privacy homomorphism," or "fully homomorphic encryption," makes possible the deep and unlimited analysis of encrypted information -- data that has been intentionally scrambled -- without sacrificing confidentiality.

IBM's solution, formulated by IBM Researcher Craig Gentry, uses a mathematical object called an "ideal lattice," and allows people to fully interact with encrypted data in ways previously thought impossible. With the breakthrough, computer vendors storing the confidential, electronic data of others will be able to fully analyze data on their clients' behalf without expensive interaction with the client, and without seeing any of the private data. With Gentry's technique, the analysis of encrypted information can yield the same detailed results as if the original data was fully visible to all.

Using the solution could help strengthen the business model of "cloud computing,"

Practical … ☺ ?

According to an article on Forbes.com, Gentry's solution has a catch: It requires immense computational effort. In the case of a Google search, for instance, performing the process with encrypted keywords would multiply the necessary computing time by around 1 trillion, Gentry estimates.

1 trilion = 10^{12}

If we exploit Moore’s law , after 40 years an homomorphic search would be as efficient as a search today

An hardware solution

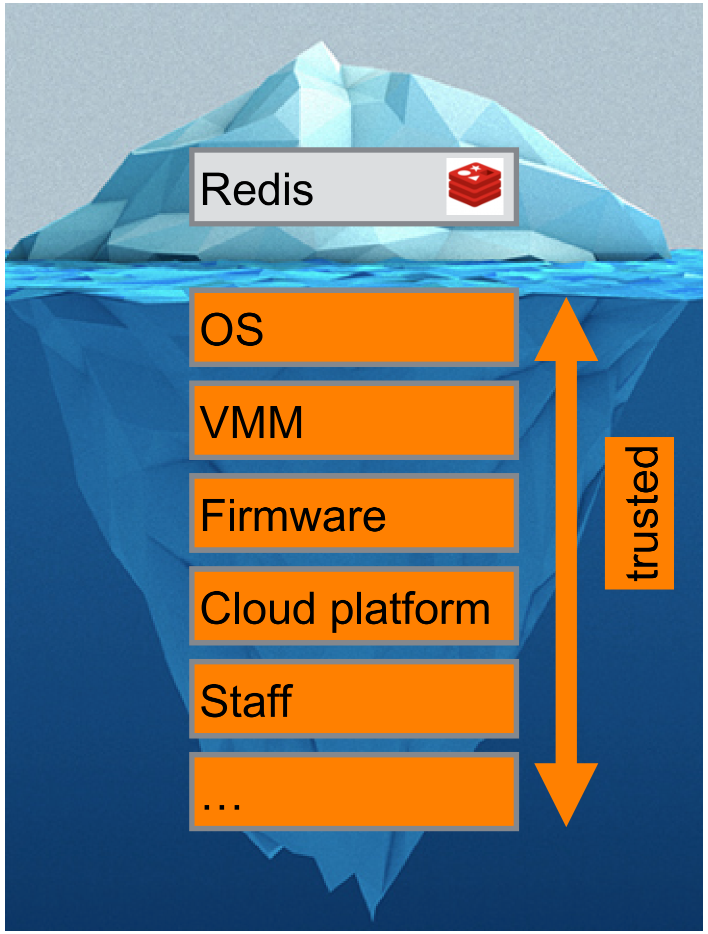
1. Trustworthy data processing in untrusted clouds
2. Overview of Intel SGX
3. Description of SGX-LKL Design
4. Description of preliminary SGX-Spark Design
5. Source code release of Java support on GitHub

Trust Issues: Provider Perspective

Cloud provider does not trust users

Use virtual machines to isolate users from each other and the host

VMs only provide one way protection

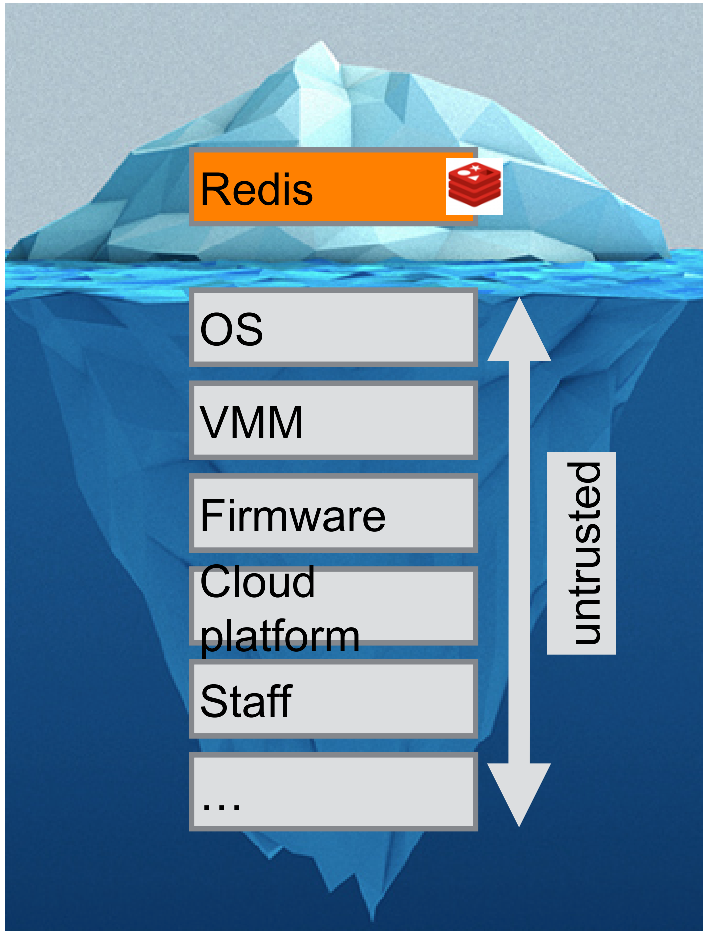


Trust Issues: User Perspective

Users trust their applications

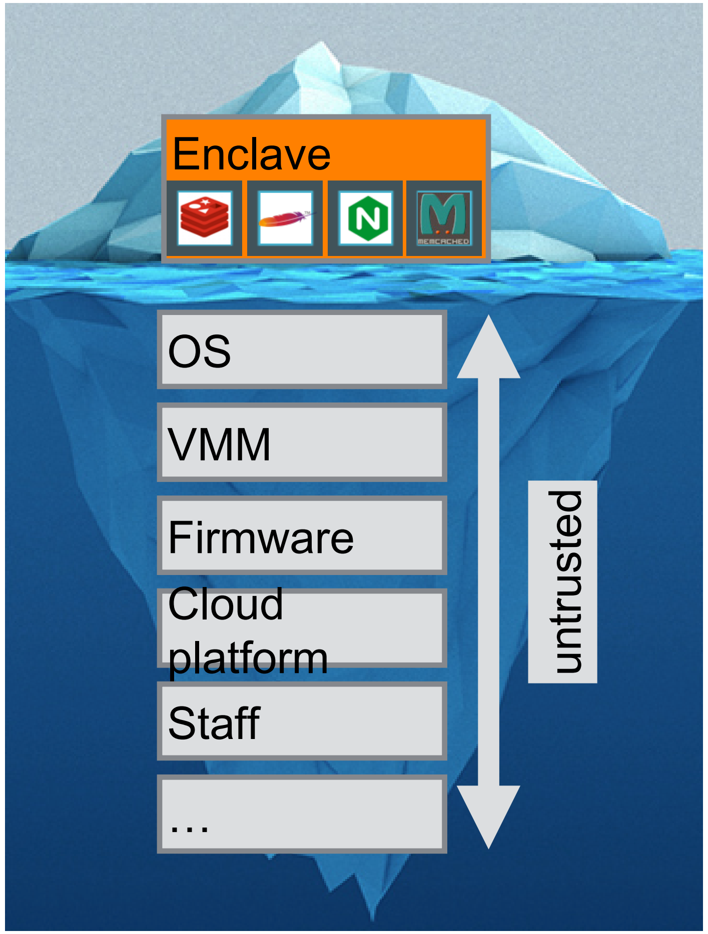
Users must implicitly trust cloud provider

Existing applications implicitly assume trusted operating system system admin



or

Trusted Execution Support with Intel SGX



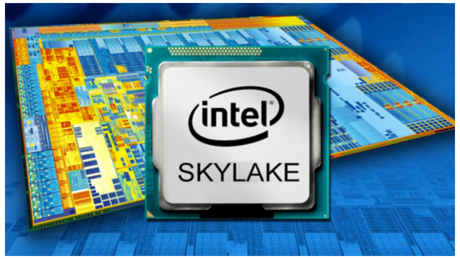
Users create HW-enforced trusted environment (enclave)

Supports unprivileged user code

Protects against strong attacker model

Remote attestation

Available on commodity CPUs



Trusted Execution Support with Intel SGX

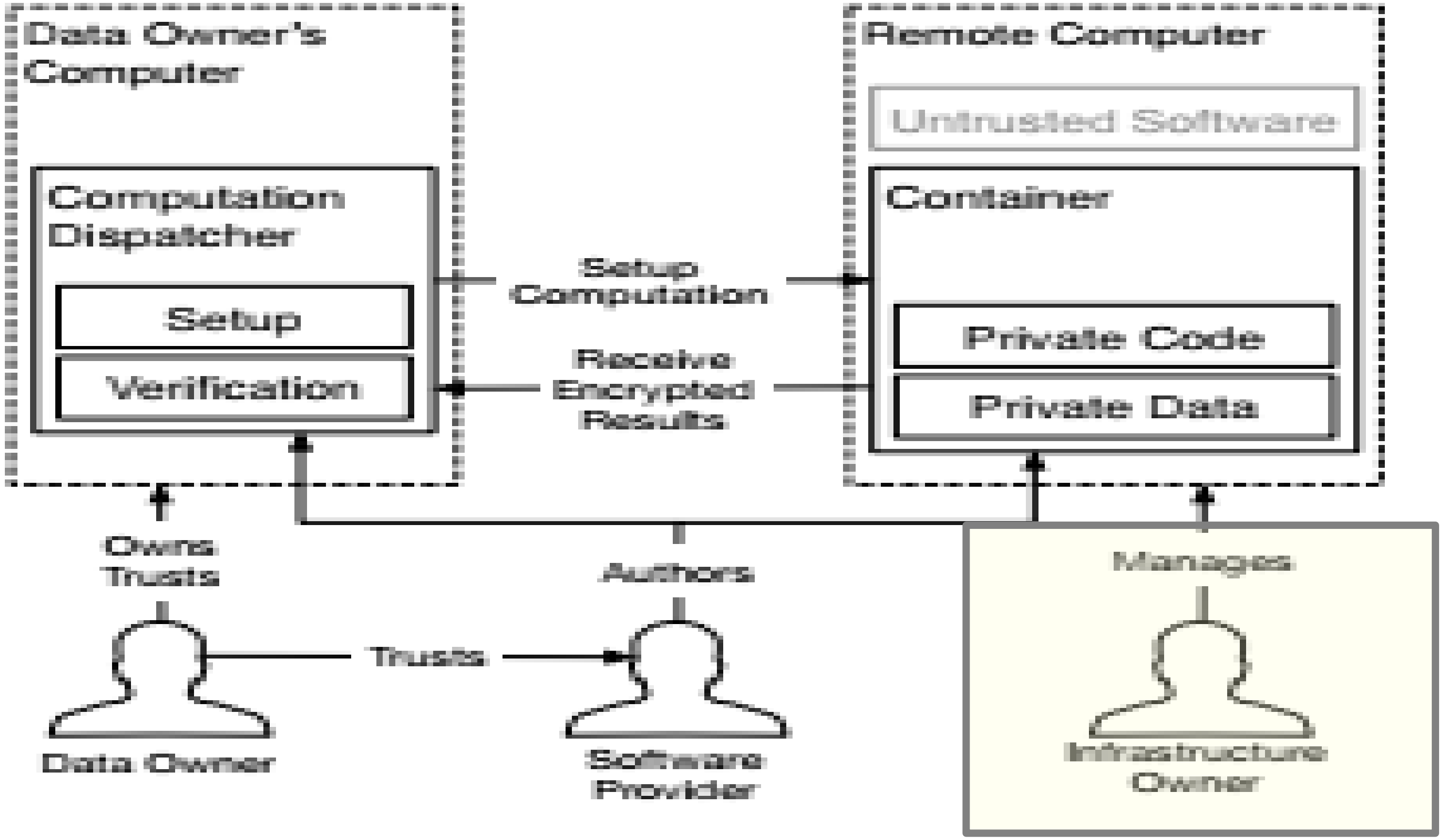


Figure 1: Secure remote computation. A user relies on a remote computer, owned by an untrusted party, to perform some computation on her data. The user has some assurance of the computation's integrity and confidentiality.

Trusted/ Untrusted?

2. Overview of Intel SGX

Trusted Execution Environments

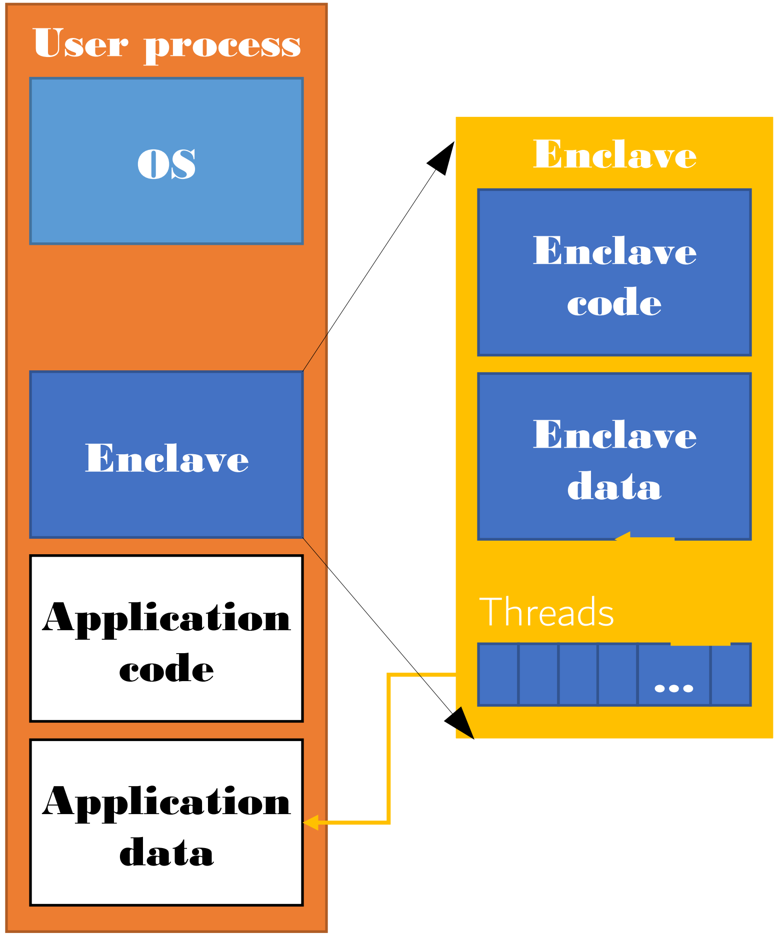
Trusted execution environment (TEE) in process

- Own code & data
- Controlled entry points
- Provides confidentiality & integrity
- Supports multiple threads
- Full access to application memory

TEE = Intel

Enclave = Microsoft

Confidential Computing Jan 2021



Intel Software Guard Extensions (SGX)

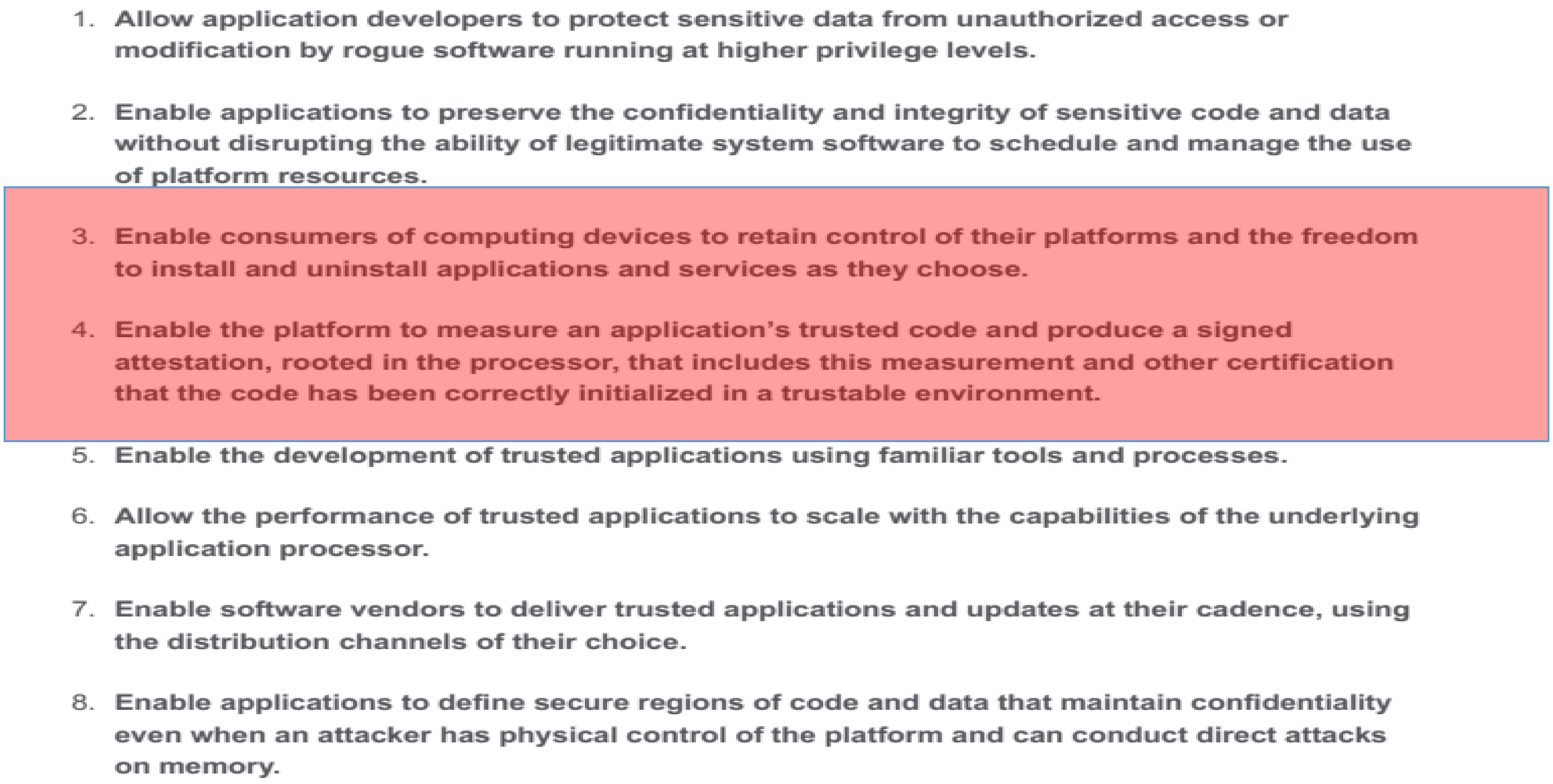
Extension of Instruction Set Architecture (ISA) in recent Intel CPUs

- Skylake (2015), Kaby lake (2016)

Protects confidentiality and integrity of code & data in untrusted environments

- Platform owner considered malicious
- Only CPU chip and isolated region trusted

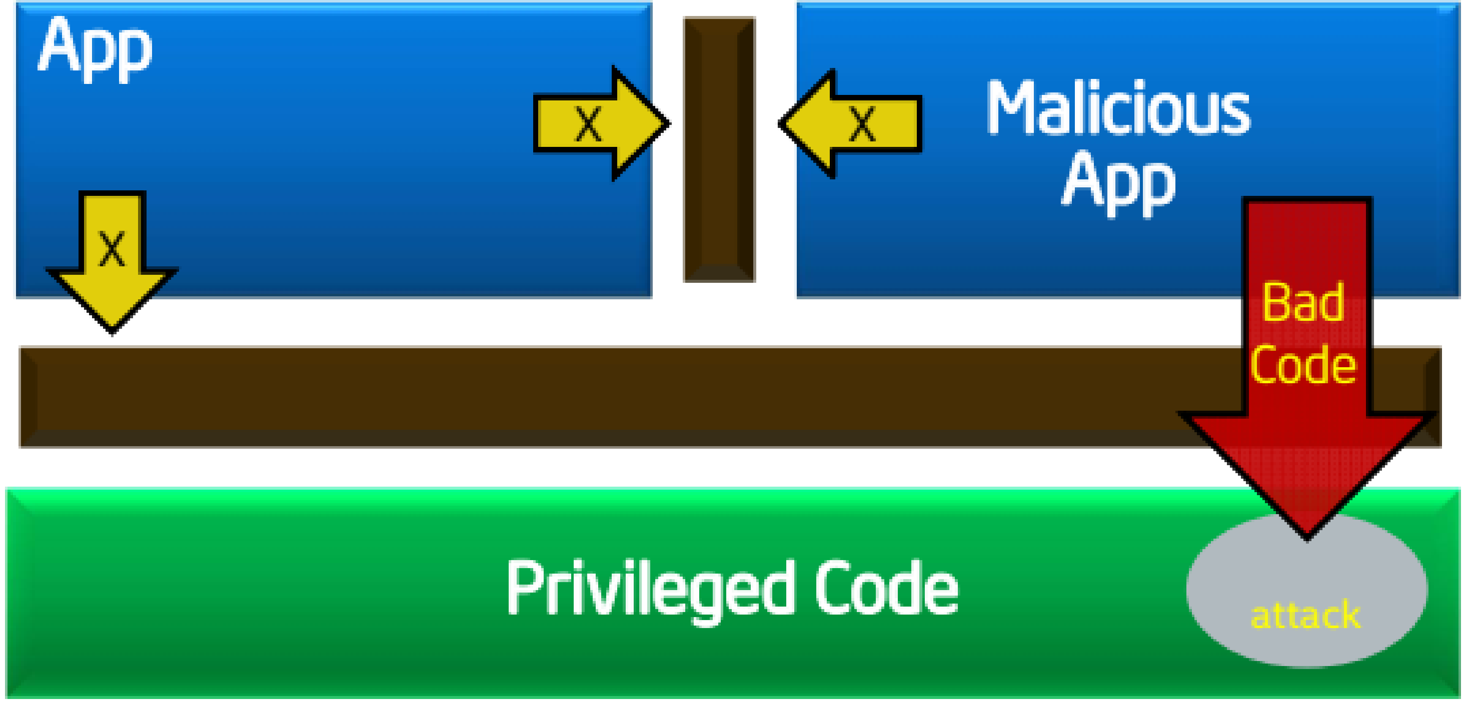
In a few words



1. Allow application developers to protect sensitive data from unauthorized access or modification by rogue software running at higher privilege levels.
2. Enable applications to preserve the confidentiality and integrity of sensitive code and data without disrupting the ability of legitimate system software to schedule and manage the use of platform resources.
3. Enable consumers of computing devices to retain control of their platforms and the freedom to install and uninstall applications and services as they choose.
4. Enable the platform to measure an application's trusted code and produce a signed attestation, rooted in the processor, that includes this measurement and other certification that the code has been correctly initialized in a trustable environment.
5. Enable the development of trusted applications using familiar tools and processes.
6. Allow the performance of trusted applications to scale with the capabilities of the underlying application processor.
7. Enable software vendors to deliver trusted applications and updates at their cadence, using the distribution channels of their choice.
8. Enable applications to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory.

The Basic Issue: Why Aren’t Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps …



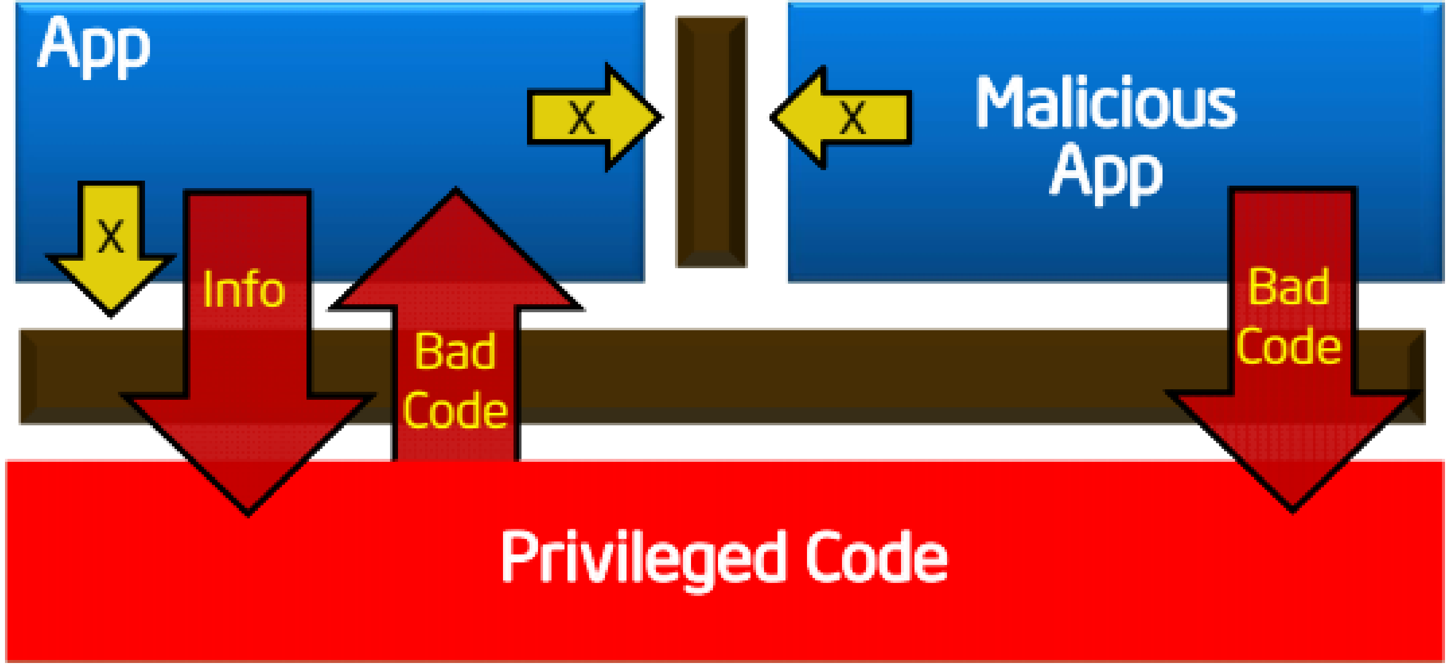
… and apps from each other …

… UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

Apps not protected from privileged code attacks

The Basic Issue: Why Aren't Compute Devices Trustworthy?

Protected Mode (rings) protects OS from apps …



… and apps from each other …

… UNTIL a malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps

Apps not protected from privileged code attacks

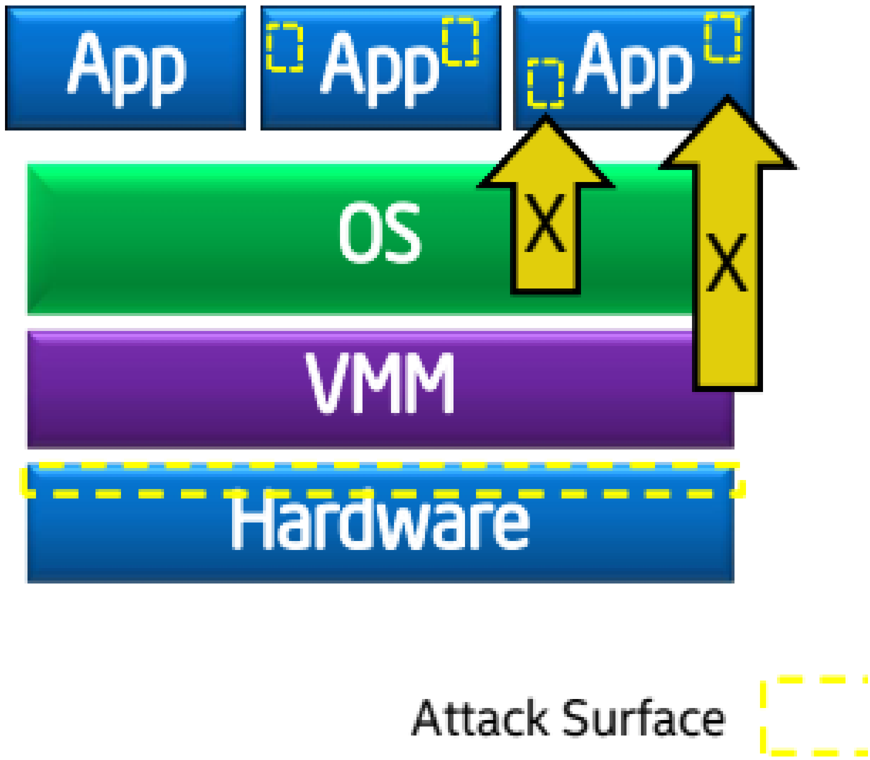
Application gains ability to defend its own secrets

- Smallest attack surface (App + processor)
- Malware that subverts OS/VMM, BIOS, Drivers etc. cannot steal app secrets

Familiar development/debug

- Single application environment
- Build on existing ecosystem expertise

Attack surface with Enclaves



SGX Enclaves

SGX introduces notion of **enclave**

- Isolated memory region for code & data
- New CPU instructions to manipulate enclaves and new enclave execution mode

Enclave memory **encrypted** and **integrity-protected** by hardware

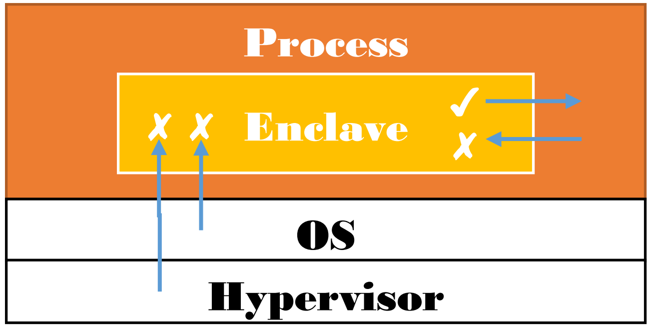
- Memory encryption engine (MEE)
- No plaintext secrets in main memory

Enclave memory can be accessed only by enclave code

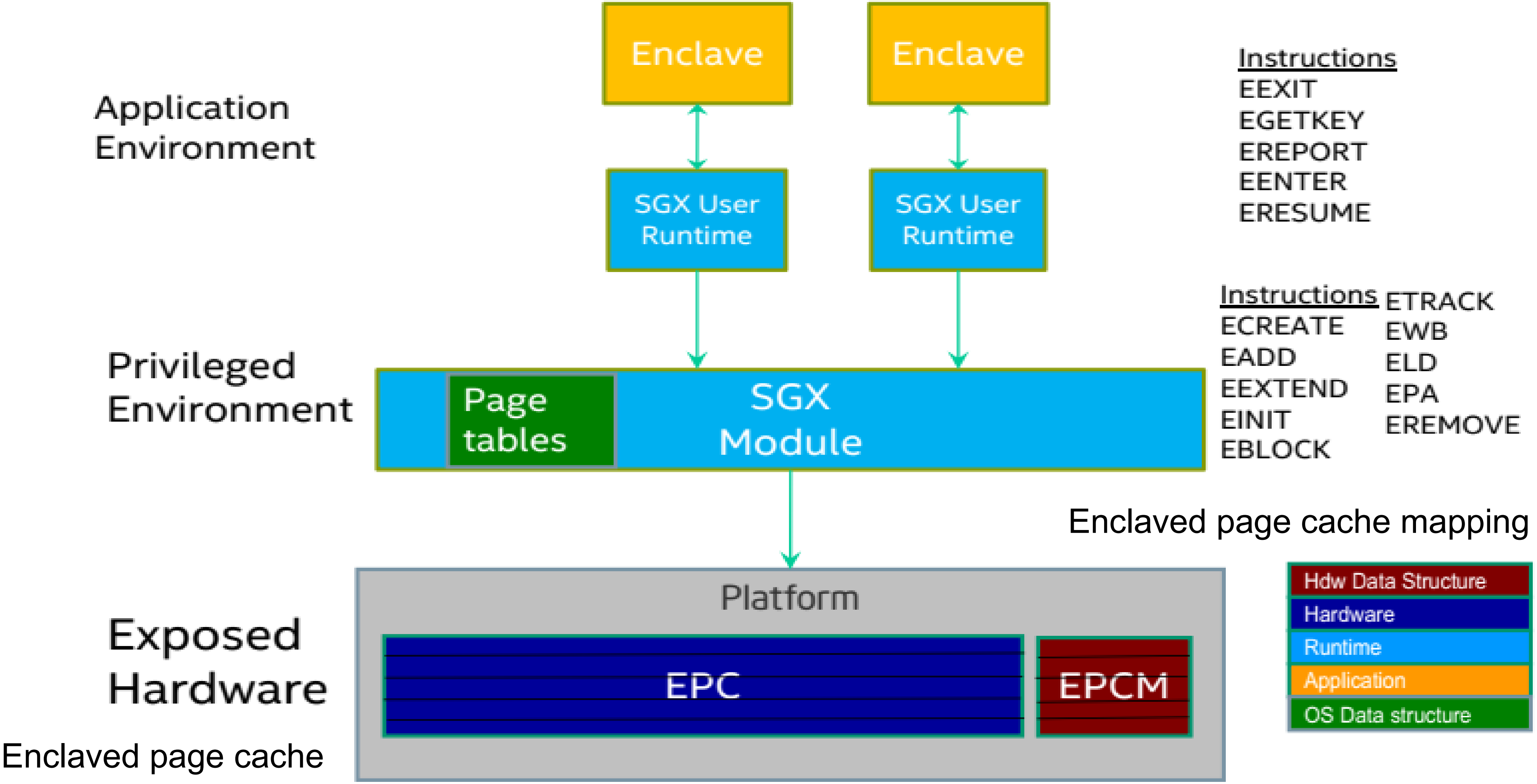
- Protection from privileged code (OS, hypervisor)

Application has ability to defend secrets

- Attack surface reduced to just enclaves and CPU
- Compromised software cannot steal application secrets



SGX High-level HW/SW Picture



Application Environment

Enclave

Enclave

SGX User Runtime

SGX User Runtime

Instructions
EEXIT
EGETKEY
EREPORT
EENTER
ERESUME

Privileged Environment

Page tables

SGX Module

Instructions
ECREATE
EADD
EEXTEND
EINIT
EBLOCK
ETRACK
EWB
ELD
EPA
EREMOVE

Enclaved page cache mapping

Exposed Hardware

Platform

EPC

EPCM

Enclaved page cache

Hdw Data Structure
Hardware
Runtime
Application
OS Data structure

Memory Access Control

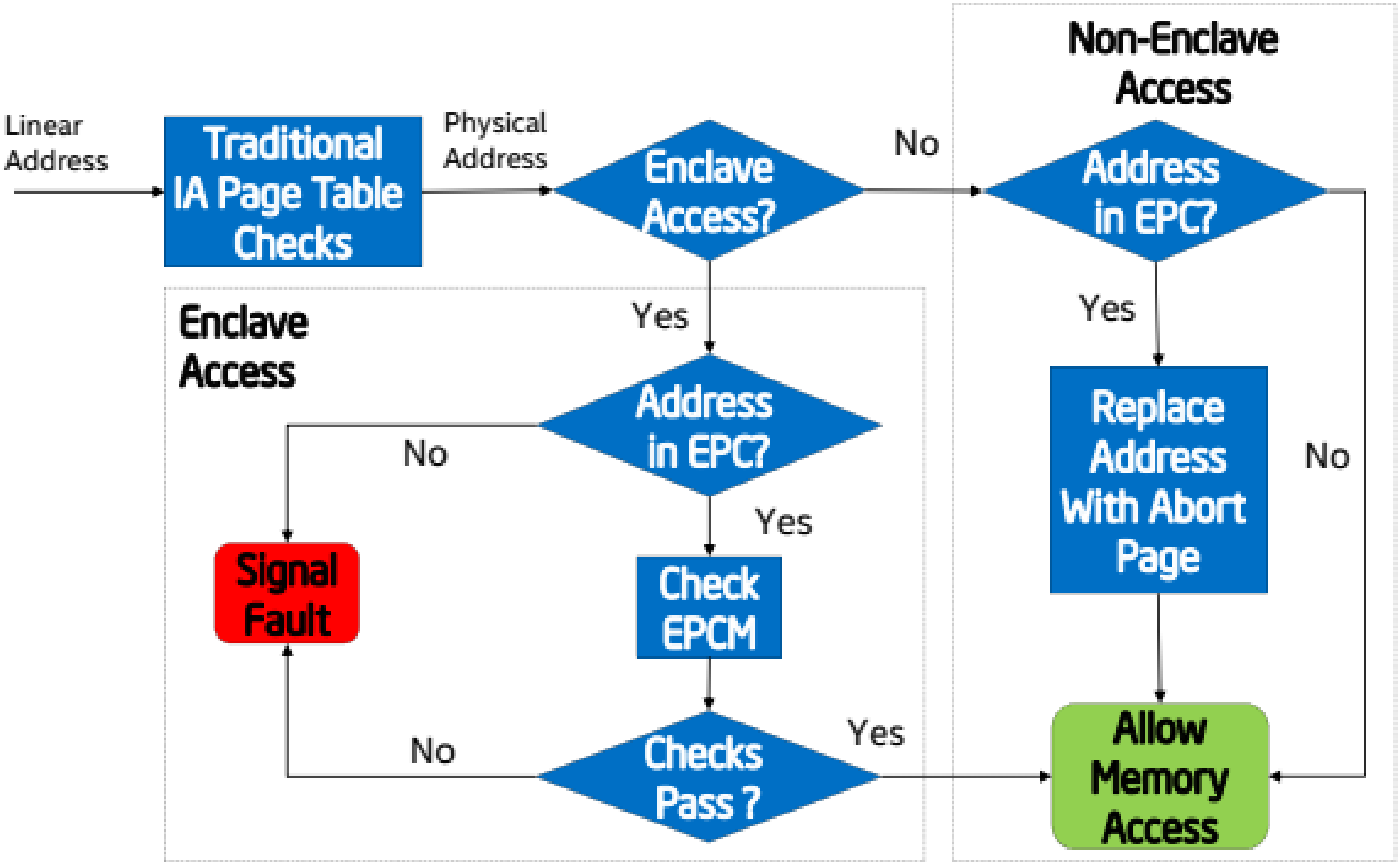
MAC from enclaves to “outside”:

- All memory access has to conform to segmentation and paging policies by the OS/VMM.
- Enclaves cannot manipulate those policies, only unprivileged instructions inside an enclave (enclaves cannot change enclaves).
- Code fetches from inside an enclave to a linear address outside that enclave will results in a General Protection Fault (0)exception.

From “outside” to enclaves

- Non-enclave accesses to EPC memory results in abort page semantics.
- Direct jumps from outside to any linear address that maps to an enclave do not enable enclave mode and result in a about page semantics and undefined behavior.
- Hardware detects and prevents enclave accesses using logical-to-linear address translations which are different than the original direct EA used to allocate the page.
- Detection of modified translation results in General Protection Fault (0).

SGX Access Control



SGX Instructions and Data Structures:

18 Instruction

- 13 Supervisor Instructions.
- 5 User Instructions.

13 Data Structures

- 8 data structures associated to a certain enclave.
- 3 data structures associated to certain memory page(s).
- 2 data structures associated to overall resource management.

SGX Supervisor Instructions:

ENCLS[EADD]	Add a page
ENCLS[EBLOCK]	Block an EPC page
ENCLS[ECREATE]	**Create an enclave**
ENCLS[EDBGRD] ENCLS[EDBGWR]	Read/Write data by debugger
ENCLS[EEXTEND]	Extend EPC page measurement
ENCLS[EINIT]	Initialize an enclave
ENCLS[ELDB]	Load an EPC page as blocked
ENCLS[ELDU]	Load an EPC page as unblocked
ENCLS[EPA]	Add version array
ENCLS[EREMOVE]	Remove a page from EPC
ENCLS[ETRACK]	Activate EBLOCK checks
ENCLS[EWB]	Write back/invalidate an EPC page

SGX User Instructions:

User Instruction	Description
ENCLU[EENTER]	Enter an Enclave
ENCLU[EEXIT]	Exit an Enclave
ENCLU[EGETKEY]	Create a cryptographic key
ENCLU[EREPORT]	Create a cryptographic report
ENCLU[ERESUME]	Re-enter an Enclave

SGX Data Structures in Details:

SGX Enclave Control Structure (SECS)	Represents one enclave and it store Hash, ID, size .
Thread Control Structure (TCS)	one for each thread in the enclave. It stores Entry point,pointer to SSA.
State Save Area (SSA)	It save the state of the running threat when an AEX occurs
Page Information (PAGEINFO)	data structure used as a parameter to the EPC-management instruction Linear Address, Effective address of the page (aka virtual address SECINFO + SECS
Security Information (SECINFO)	Meta-data about an enclave pag R/W/X,Page type (SECS, TCS, normal page or VA)
Paging Crypto MetaData (PCMD):	Crypto meta-data of a paged-out page. With PAGEINFO it used to verify, decrypt, and reload a paged-out page EWB writes out (the reserved field and) MAC values. ELDB/U reads the fields and checks the MAC Contains Enclave ID, SECINFO and MAC

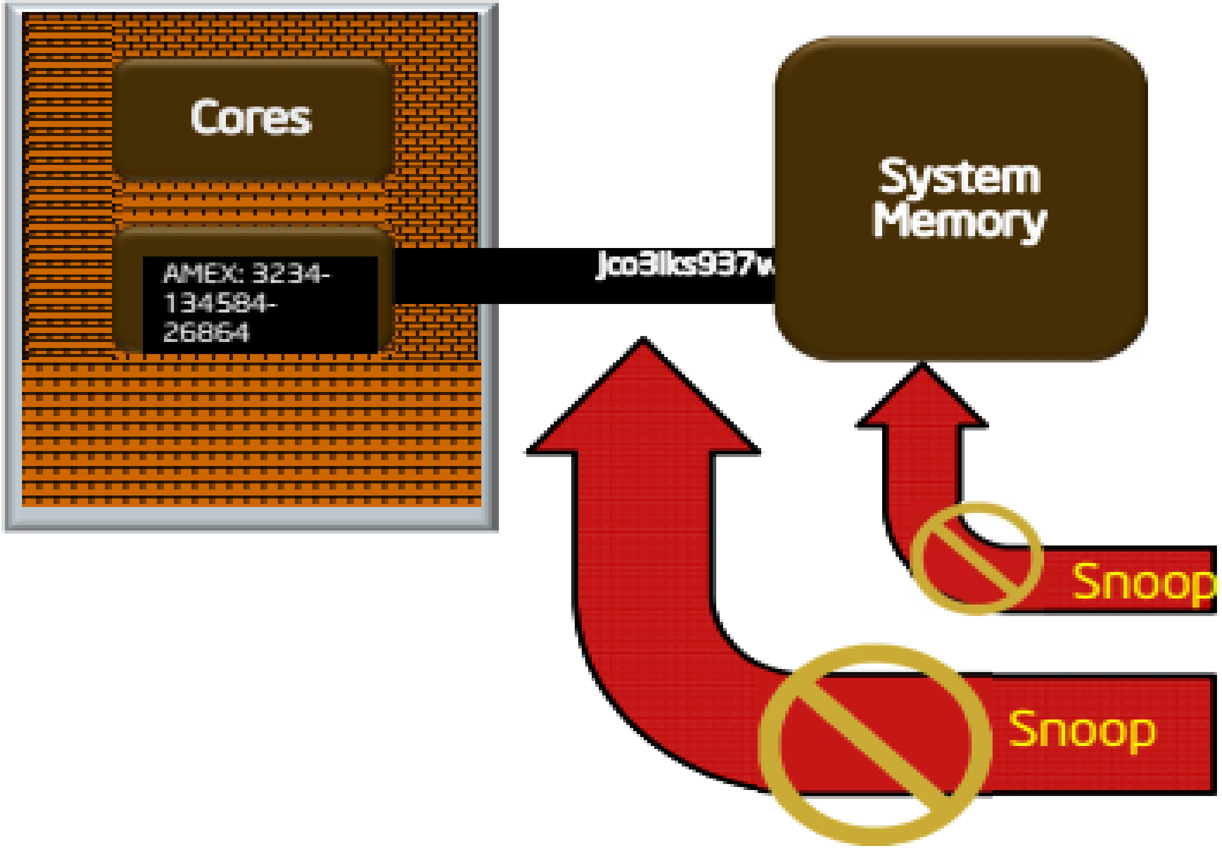
SGX Data Structures in Details:

Version Array (VA)

Each VA page is an EPC page to securely store the versions of evicted EPC pages with 512 slots, each with an 8-byte version number for a page evicted from the EPC.

- When an EPC page is evicted, an empty slot in a VA page receives the unique version number of the evicted page
- When the EPC page is reloaded, a VA slot must hold the page version when the VA slot is cleared.
- When evicting a VA page, a version slot in some other VA page must be used to receive the version for the VA being evicted
- Version number = nonce to prevent reply attack

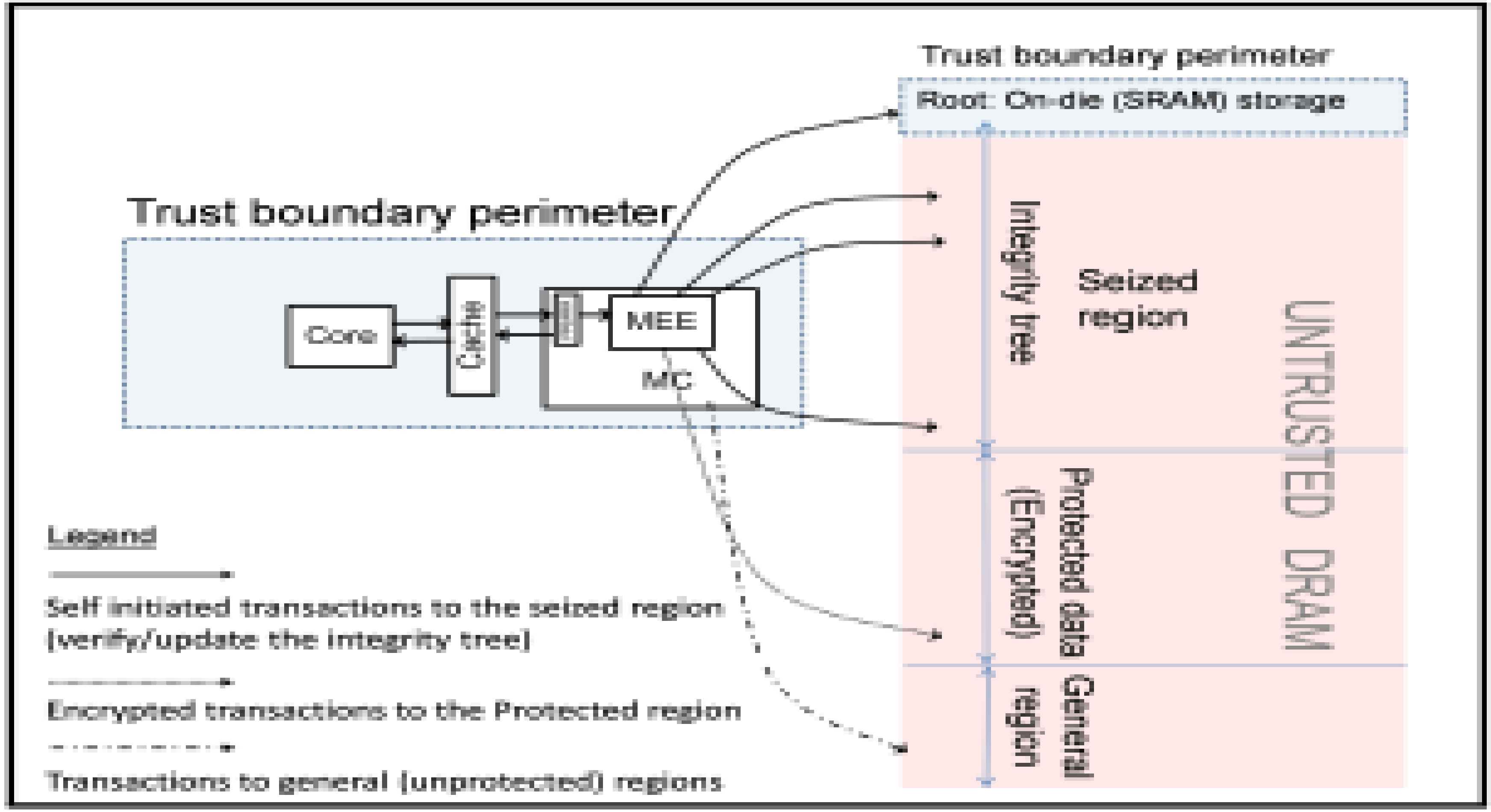
Protection vs. Memory Snooping Attacks



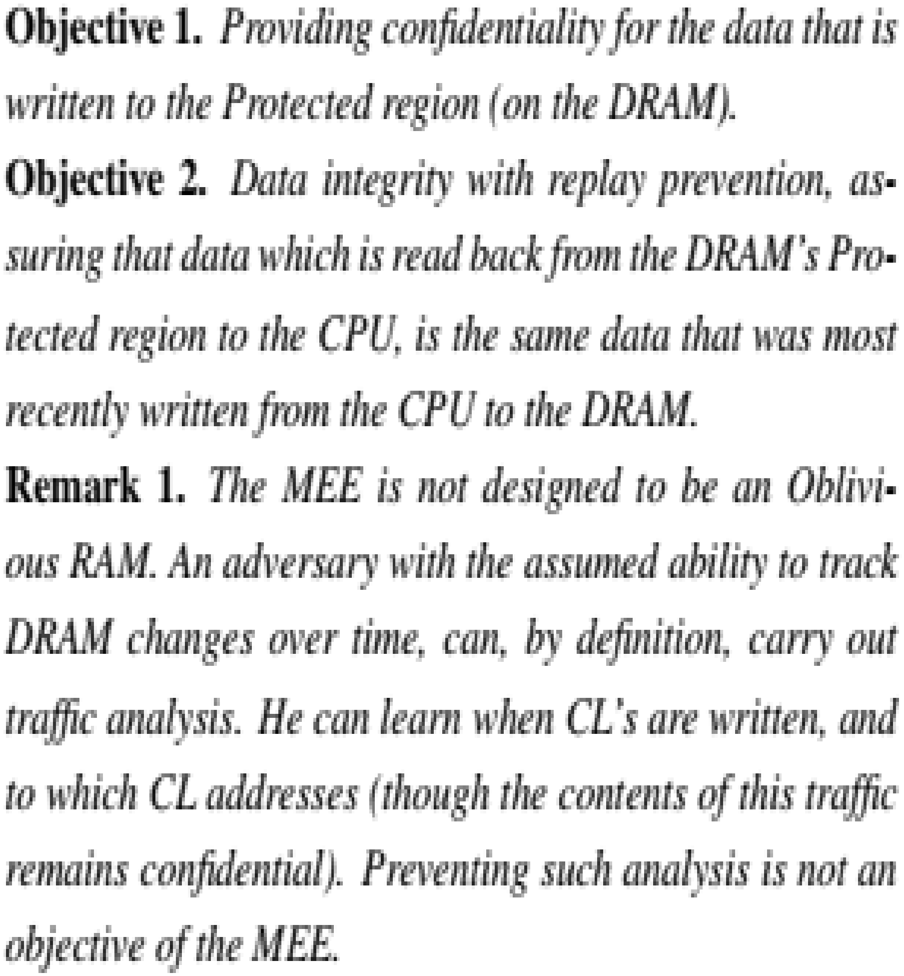
Non-Enclave Access

- Security perimeter is the CPU package boundary
- Data and code unencrypted inside CPU package
- Data and code outside CPU package is encrypted and/or integrity checked
- External memory reads and bus snoops see only encrypted data

Memory Encryption Engine



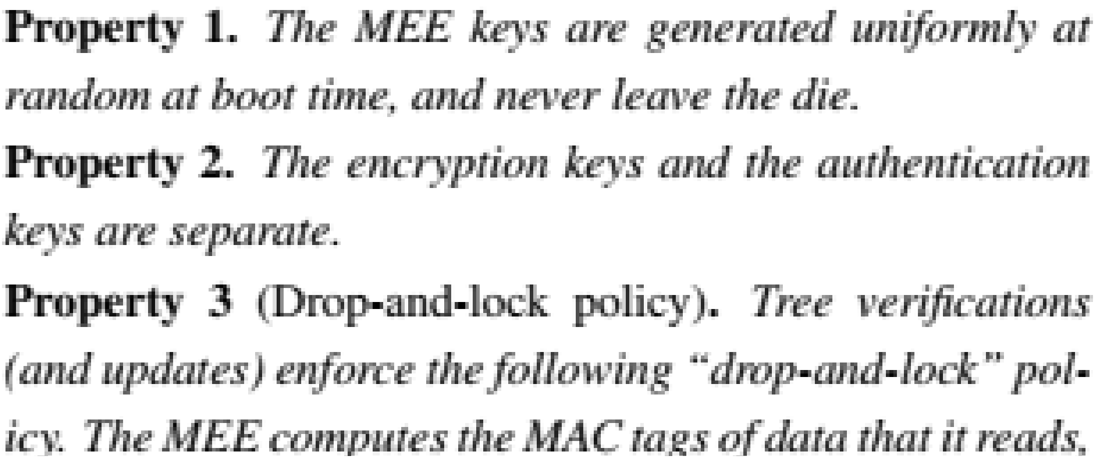
Memory Encryption Engine

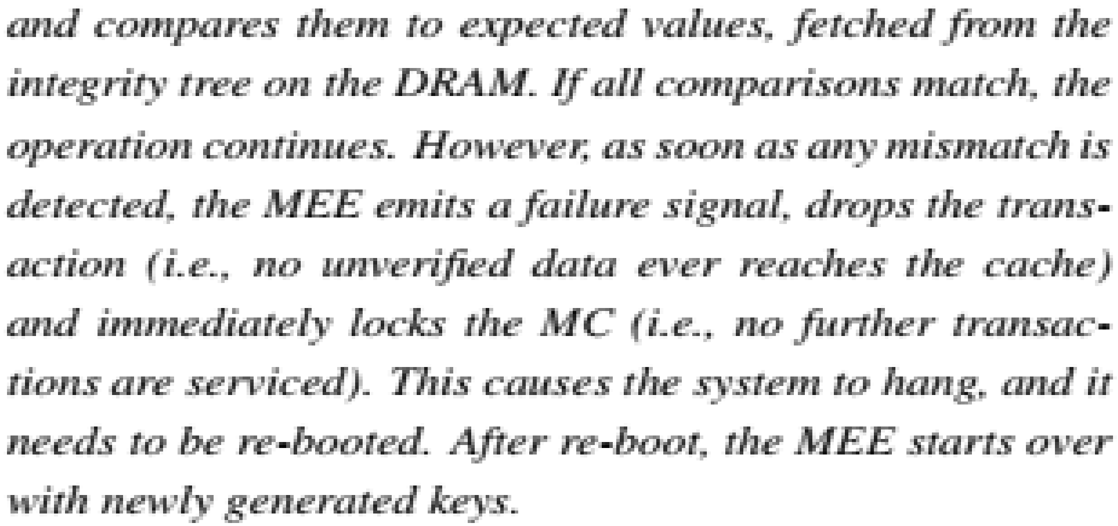


Objective 1. *Providing confidentiality for the data that is written to the Protected region (on the DRAM).*

Objective 2. *Data integrity with replay prevention, assuring that data which is read back from the DRAM's Protected region to the CPU, is the same data that was most recently written from the CPU to the DRAM.*

Remark 1. *The MEE is not designed to be an Oblivious RAM. An adversary with the assumed ability to track DRAM changes over time, can, by definition, carry out traffic analysis. He can learn when CL's are written, and to which CL addresses (though the contents of this traffic remains confidential). Preventing such analysis is not an objective of the MEE.*



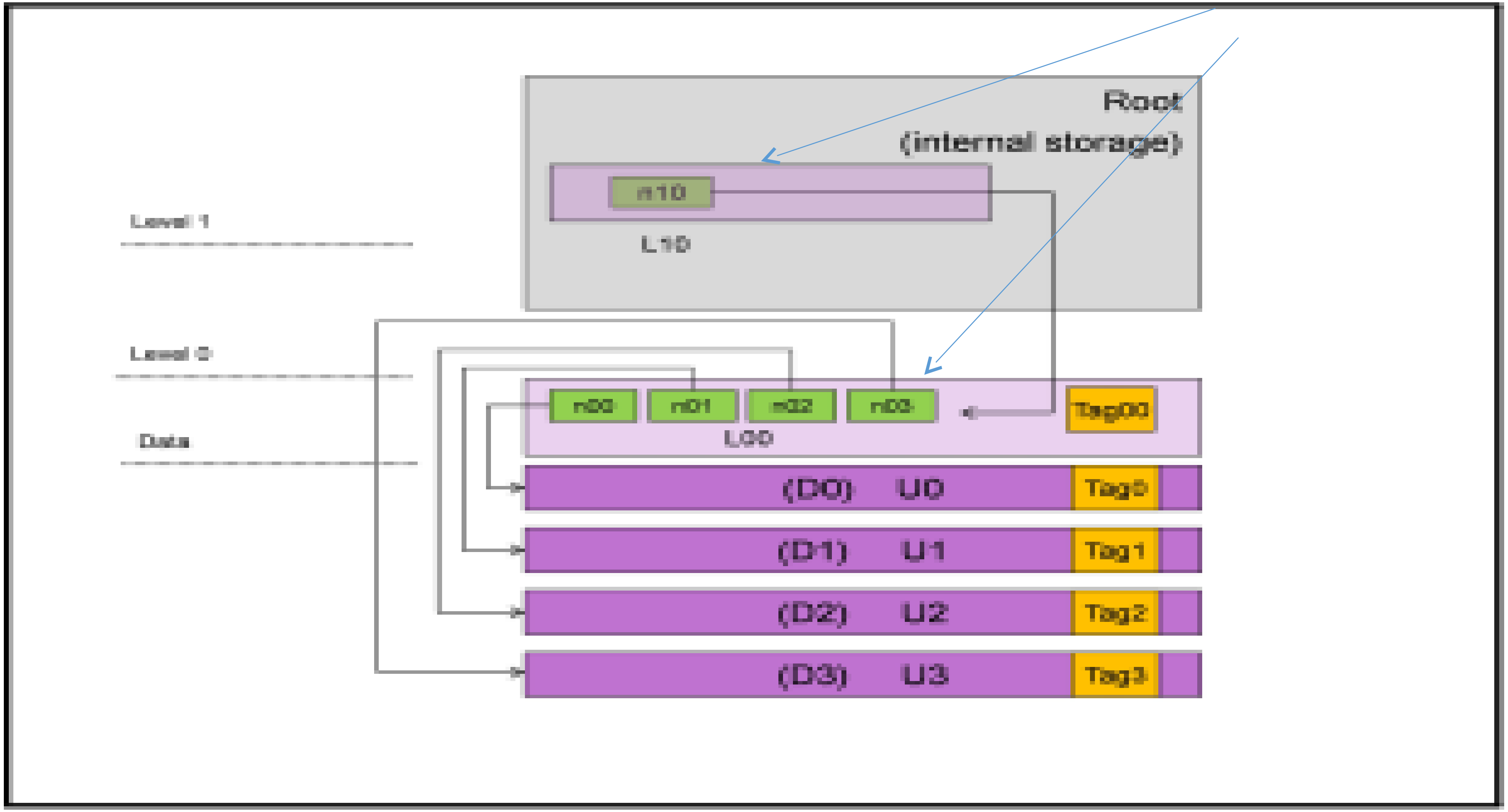


Property 1. *The MEE keys are generated uniformly at random at boot time, and never leave the die.*

Property 2. *The encryption keys and the authentication keys are separate.*

Property 3 (Drop-and-lock policy). *Tree verifications (and updates) enforce the following "drop-and-lock" policy. The MEE computes the MAC tags of data that it reads, and compares them to expected values, fetched from the integrity tree on the DRAM. If all comparisons match, the operation continues. However, as soon as any mismatch is detected, the MEE emits a failure signal, drops the transaction (i.e., no unverified data ever reaches the cache) and immediately locks the MC (i.e., no further transactions are serviced). This causes the system to hang, and it needs to be re-booted. After re-boot, the MEE starts over with newly generated keys.*

Memory Encryption Engine: Integrity Tree



nonces

Memory Encryption Engine: Performance

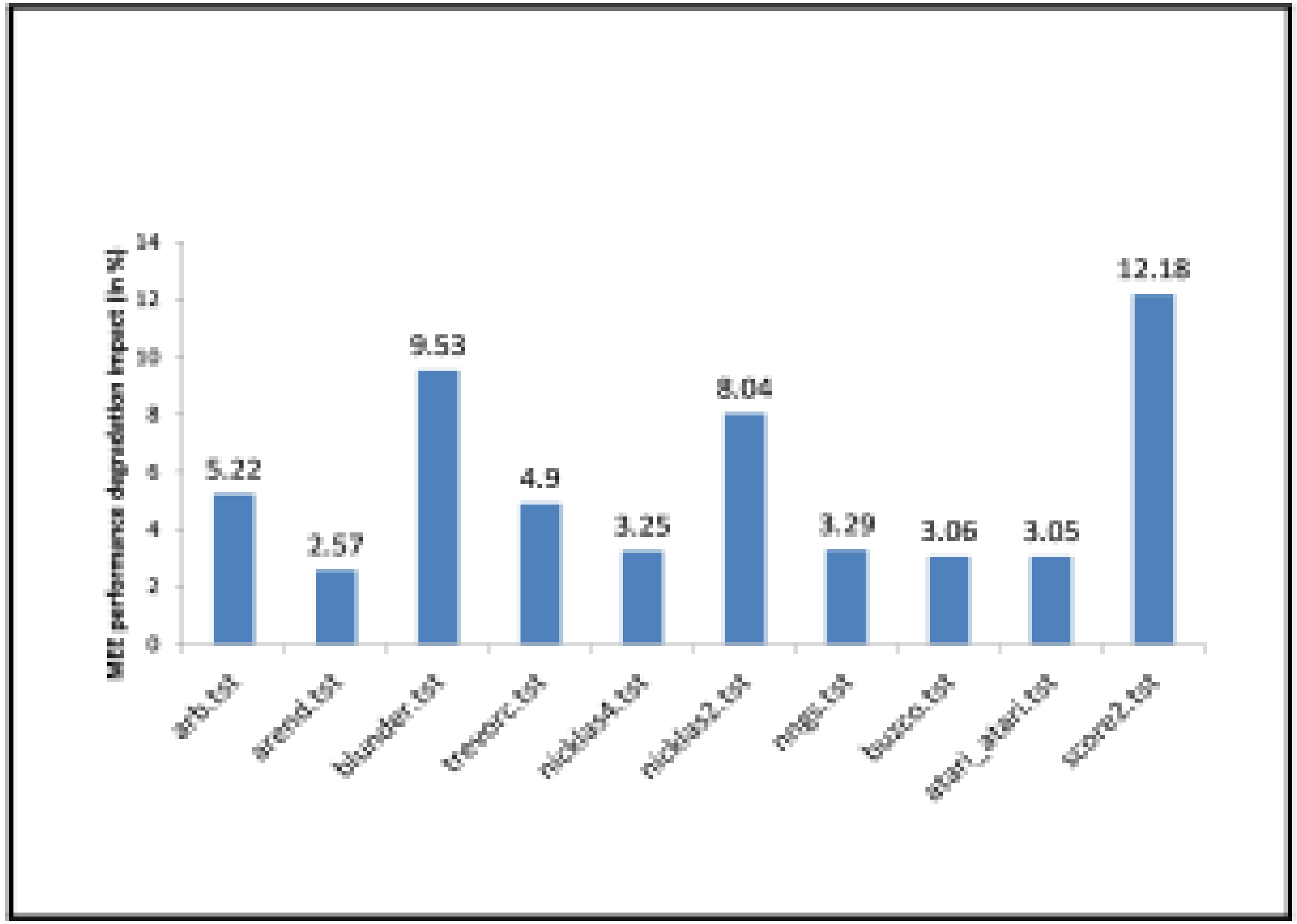


Figure 5: Performance comparison of the 445.gobmk component of SPECINT 2006, with 10 input files (see explanations in the text). The bars show that the performance degradation (in %) incurred by enabling the MEE, varies from 2.2% to 14%, with an average of 5.5%.

SGX SDK Code Sample

SGX application: untrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
  ...
  while(1)
  {
    receive(request_buf);
    ret = EENTER(request_buf, response_buf);
    if (ret < 0)
      fprintf(stderr, "Corrupted message\n");
    else
      send(response_buf);
  }
  ...
}
```

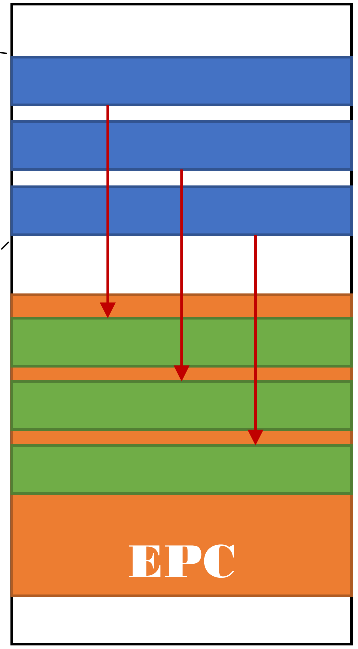
Enclave: trusted code

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

- Receives encrypted requests
- Processes them in enclave
- Sends encrypted responses

SGX Enclave Construction

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```



Enclave populated using special instruction (EADD)

- Contents initially in untrusted memory
- Copied into EPC in 4KB pages

Both data & code copied before execution commences in enclave

SGX Enclave Construction

Enclave contents distributed in plaintext

- Must not contain any (plaintext) confidential data

Secrets provisioned after enclave constructed and integrity verified

Problem: what if someone tampers with enclave? Supply chain attack

- Contents initially in untrusted memory

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    copy_msg(output_buf, external_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Write unencrypted response to

SGX Enclave Attestation

Is my code running on remote machine intact?

Is code really running inside an SGX enclave?

- Local attestation
  - Prove enclave’s identity (= measurement) to another enclave on same CPU
- Remote attestation
  - Prove enclave’s identity to remote party

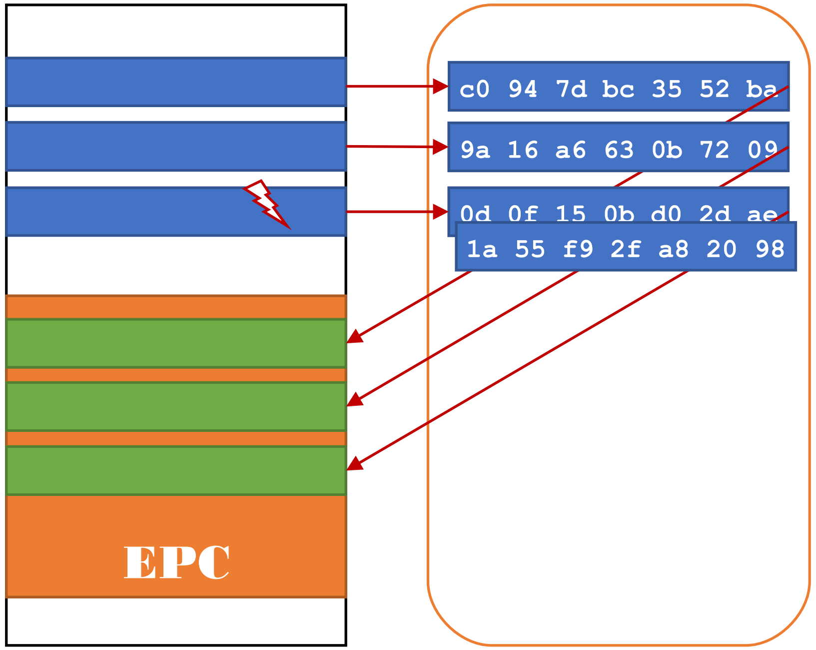
Once attested, enclave can be trusted with secrets

SGX Enclave Measurement

CPU calculates enclave measurement hash during enclave construction

- Each new page extends hash with page content and attributes (read/write/execute)
- Hash computed with SHA-256

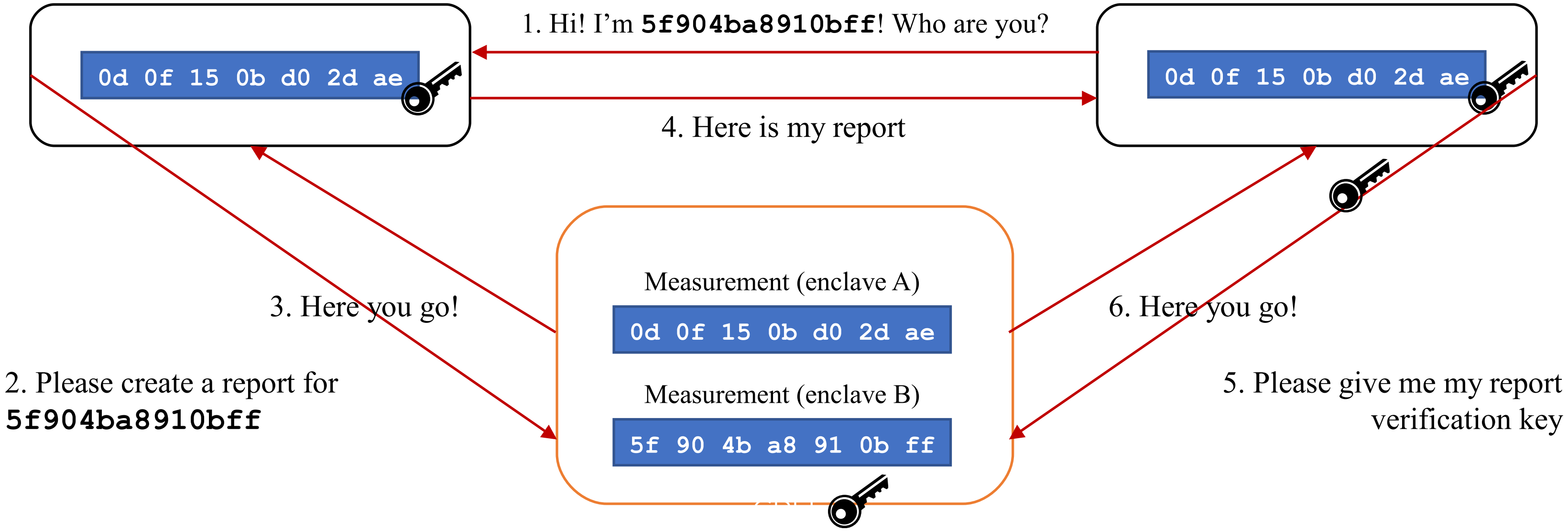
Measurement can be used to attest enclave to local or remote entity



CPU computes enclave measurement hash during enclave construction
Different measurement if enclave modified

Local Attestation

Prove identity of A to local enclave B



1. Hi! I'm **5f904ba8910bff**! Who are you?

4. Here is my report

0d 0f 15 0b d0 2d ae

0d 0f 15 0b d0 2d ae

3. Here you go!

2. Please create a report for
5f904ba8910bff

Measurement (enclave A)
0d 0f 15 0b d0 2d ae

Measurement (enclave B)
5f 90 4b a8 91 0b ff

6. Here you go!

5. Please give me my report verification key

1. Target enclave B measurement required for key generation
2. Report contains information about target enclave B, including its measurement
3. CPU fills in report and creates MAC using report key, which depends on random CPU fuses and target enclave B measurement
4. Report sent back to target enclave B
5. Verify report by CPU to check that generated on same platform, i.e. MAC created with same report key (available only on same CPU)
6. Check MAC received with report and do not trust A upon mismatch

Remote Attestation

Transform local report to remotely verifiable “quote”

Based on provisioning enclave (PE) and quoting enclave (QE)

- Architectural enclaves provided by Intel
- Execute locally on user platform

Each SGX-enabled CPU has unique key fused during manufacturing

- Intel maintains database of keys
- Similar to TPM assumptions

Remote Attestation

PE communicates with Intel attestation service (acting as CA)

- Proves it has key installed by Intel
- Receives asymmetric attestation key

QE performs local attestation for enclave

- QE verifies report and signs it using attestation key
- Creates quote that can be verified outside platform

Quote and signature sent to remote attester, which communicates with Intel attestation service to verify quote validity

SGX Limitations & Research Challenges

Amount of memory enclave can use needs to be known in advance

- Dynamic memory support in SGX v2

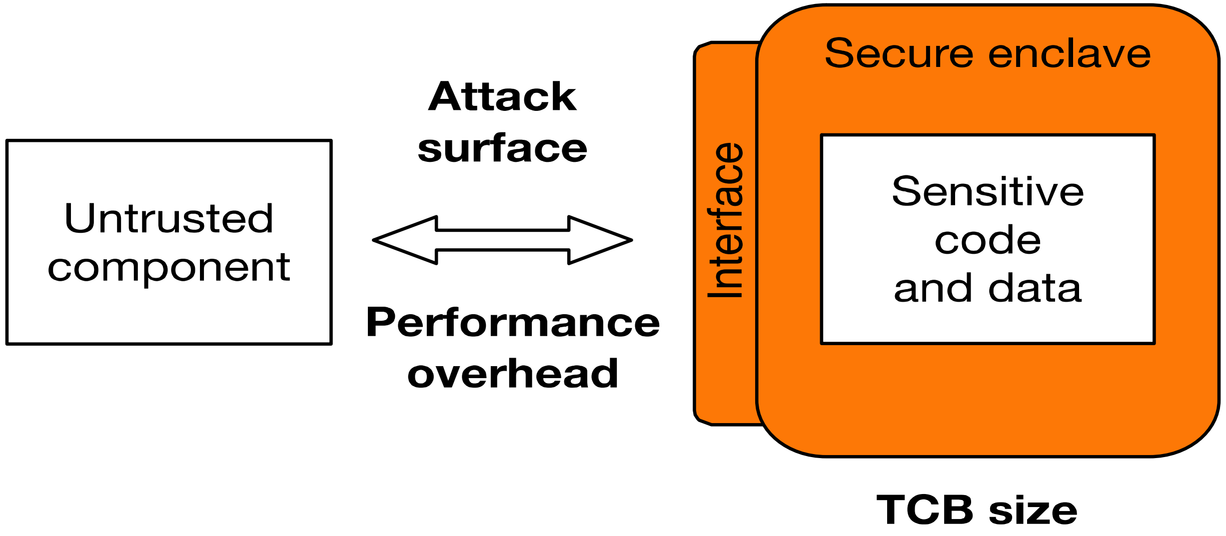
Security guarantees not perfect

- Vulnerabilities within enclave can still be exploited
- Side-channel attacks possible

Performance overhead

- Enclave entry/exit costly
- Paging very expensive

Application partitioning? Legacy code? …



Untrusted component

Attack surface

Performance overhead

Interface

Secure enclave

Sensitive code and data

TCB size

COMPARISON

	HW TEE	Homomorphic Encryption	Secure element e.g., TPM
Data integrity	Y	Y (subject to code integrity)	Keys only
Data confidentiality	Y	Y	Keys only
Code integrity	Y	No	Y
Code confidentiality	Y (may require work)	No	Y
Authenticated Launch	Varies	No	No
Programmability	Y	Partial (“circuits”)	No
Attestability	Y	No	Y
Recoverability	Y	No	Y

Table 1 - comparison of security properties of Confidential Computing vs. HE and TPMs

3. Description of SGX-LKL

SGX-LKL: Supporting Managed Runtimes in SGX

Many applications need runtime support

- JVM
- .NET
- JavaScript/V8/Node.js

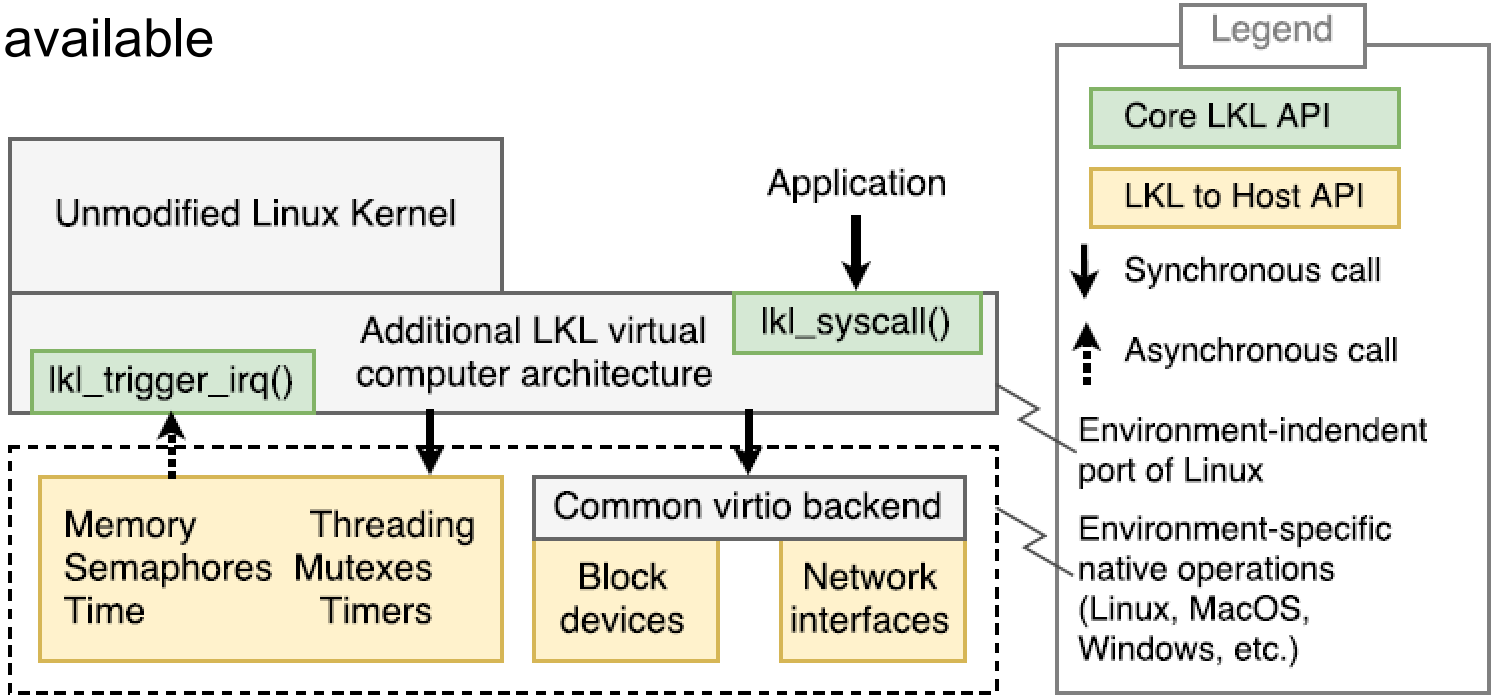
Requires complex system support

- Dynamic library loading
- Filesystem support
- Signal handling
- Complete networking stack

SGX-LKL: Linux Kernel Library in SGX Enclaves

Based on **Linux Kernel Library (LKL)**

- Implemented as architecture-specific port of mainline Linux (`github.com/lkl`)
- Follows Linux no MMU architecture
- Full filesystem support
- Full network stack available



Unmodified Linux Kernel

Application

lkl_syscall()

Additional LKL virtual computer architecture

lkl_trigger_irq()

Memory
Semaphores
Time

Threading
Mutexes
Timers

Common virtio backend

Block devices

Network interfaces

Legend

🟩 Core LKL API

🟨 LKL to Host API

↓ Synchronous call

↑ Asynchronous call

Environment-indendent port of Linux

Environment-specific native operations (Linux, MacOS, Windows, etc.)

SGX-LKL Architecture

Runs **unmodified Linux applications** in SGX enclaves

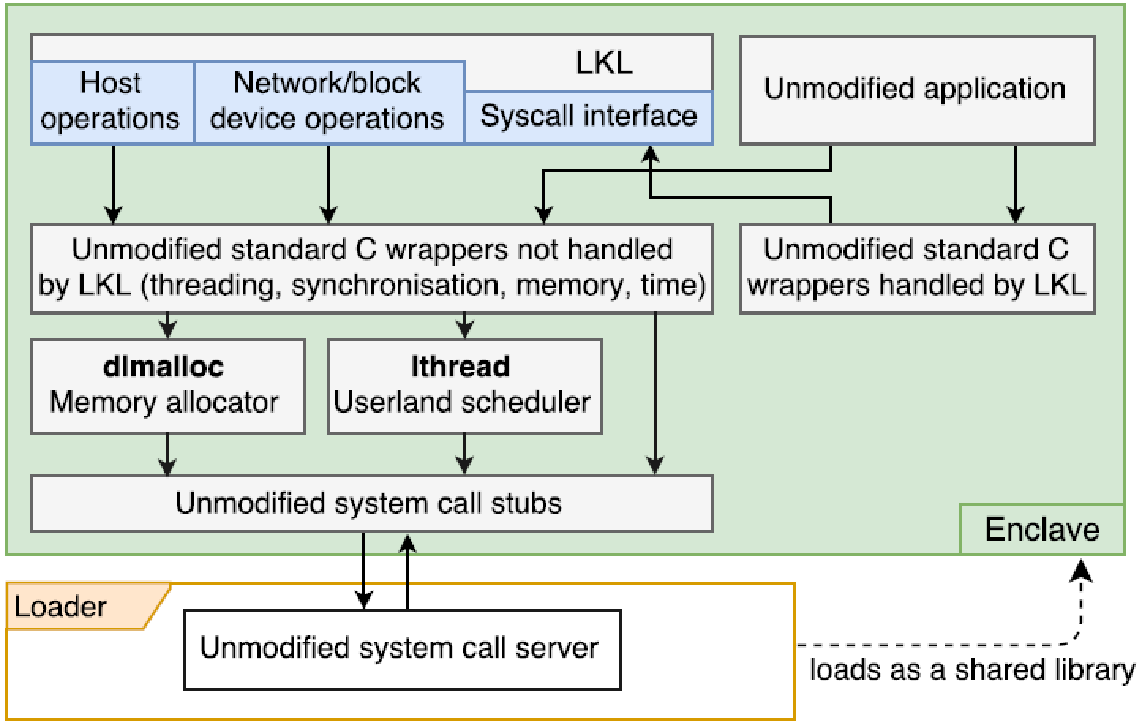
Applications and dependencies provided via **disk image**

Full Linux kernel functionality available

Custom memory allocator

User-level threading

- In-enclave synchronisation primitives



SGX-LKL Architecture: How many instructions in enclaves

Application	Total code size (LOCs)	Enclave size (LOCs)
Memcached	**31,000**	**12,000 (40%)**
DigitalBitbox	**23,000**	**8,000 (38%)**
LibreSSL	**176,000**	**38,000 (22%)**

Memcached is an in-memory key-value store for small chunks of arbitrary data

Digital Bitbox DBB1707 Hardware Wallet Criptovalute Per Btc/eth