Applying Runtime Verification to Group Key Establishment

Secure Communication in the Quantum Era (SPS G5448)
April 2019

Christian Colombo - University of Malta

Authenticated group key establishment (AGKE)

First step: Designing a protocol

Authenticated group key establishment (AGKE)

First step: Designing a protocol

Second step: Proving it is correct in principle

Authenticated group key establishment (AGKE)

First step: Designing a protocol

Second step: Proving it is correct in principle

Third step: What can go wrong at runtime?

What can go wrong at runtime?

(High level) Wrong protocol implementation	The protocol implementation might deviate from the verified (theoretical) design
Low level threats	Arithmetic overflows, undefined downcasts, and invalid pointer references
Hardware	Can hardware be trusted?

What can go wrong at runtime?

...but in practice is far from enough

(High level) Wrong protocol implementation

The protocol implementation might deviate from the verified (theoretical) design

Medium level threats: Malware, Data leaks, etc

Low level threats

and invalid pointer references

Hardware

Can hardware be trusted?

NATO OTAN

Unintended consequences

- Timing attacks
- Cache timing attacks
- Microarchitecture side-channel attack
- Power/EM/acoustic attacks
- Fault attacks
- Reaction attacks
- Data remanence attacks
- Attacks on random number generators

Timing attack

If (secret)

    Do something lengthy

Else

    Do something simple

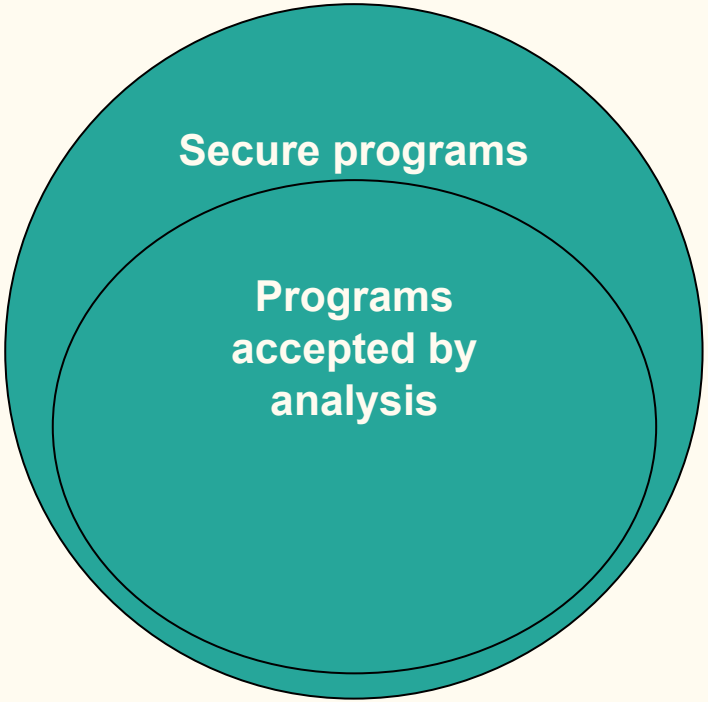
An external observer can learn the secret by observing the duration of the execution.

(or the power used or any other side effect)

What can we do?

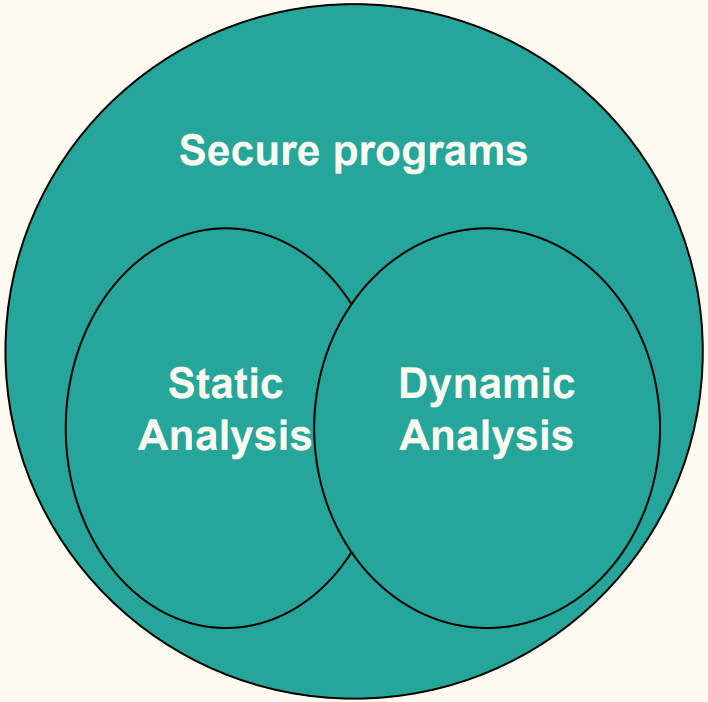
Analyse code to make sure secrets can’t be leaked!

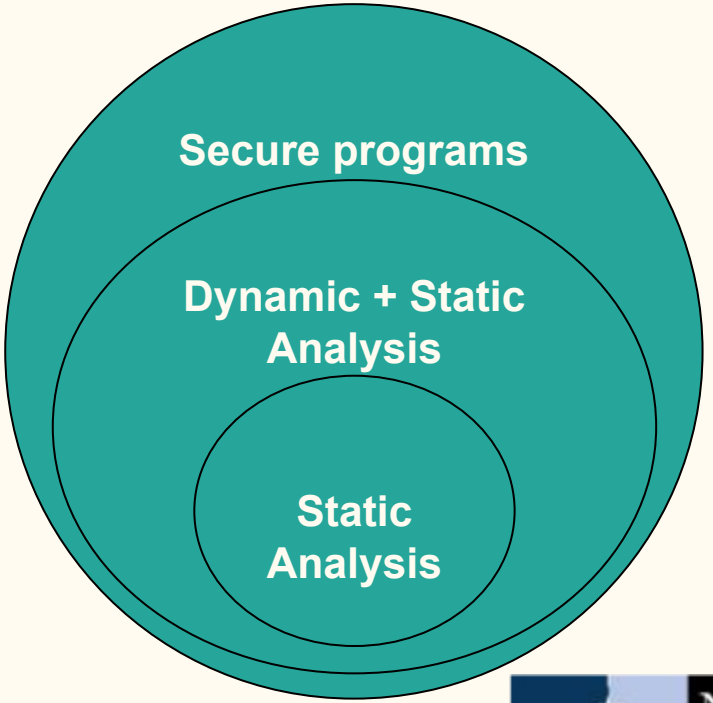
Identifying secure programs



NATO OTAN

Soundness/Completeness of dynamic analysis

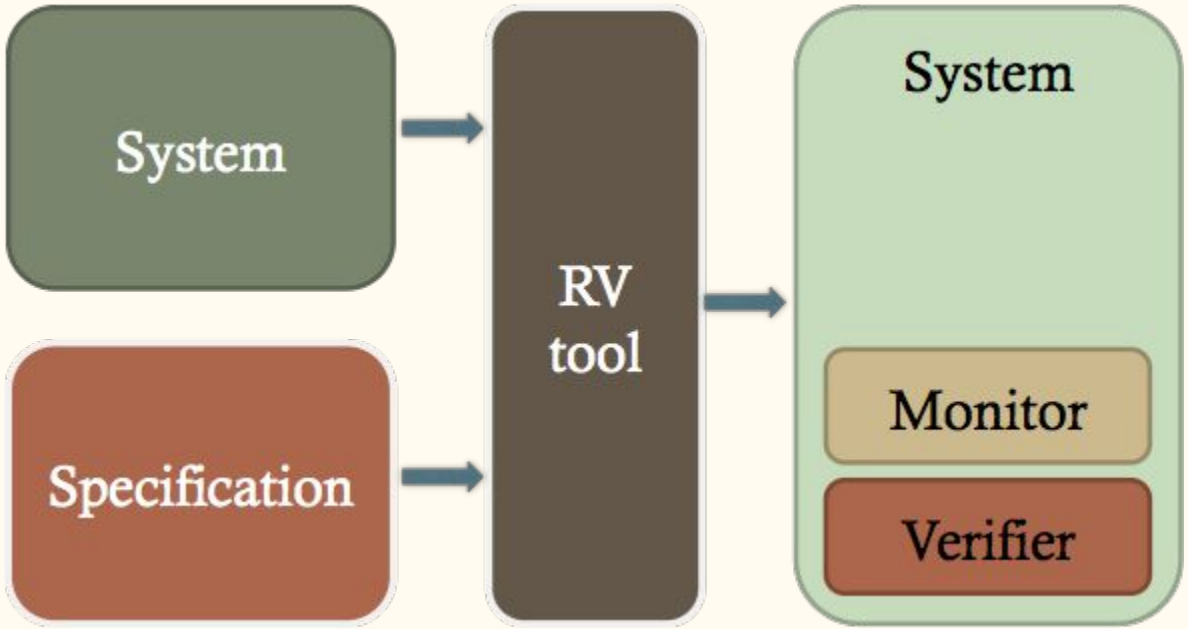




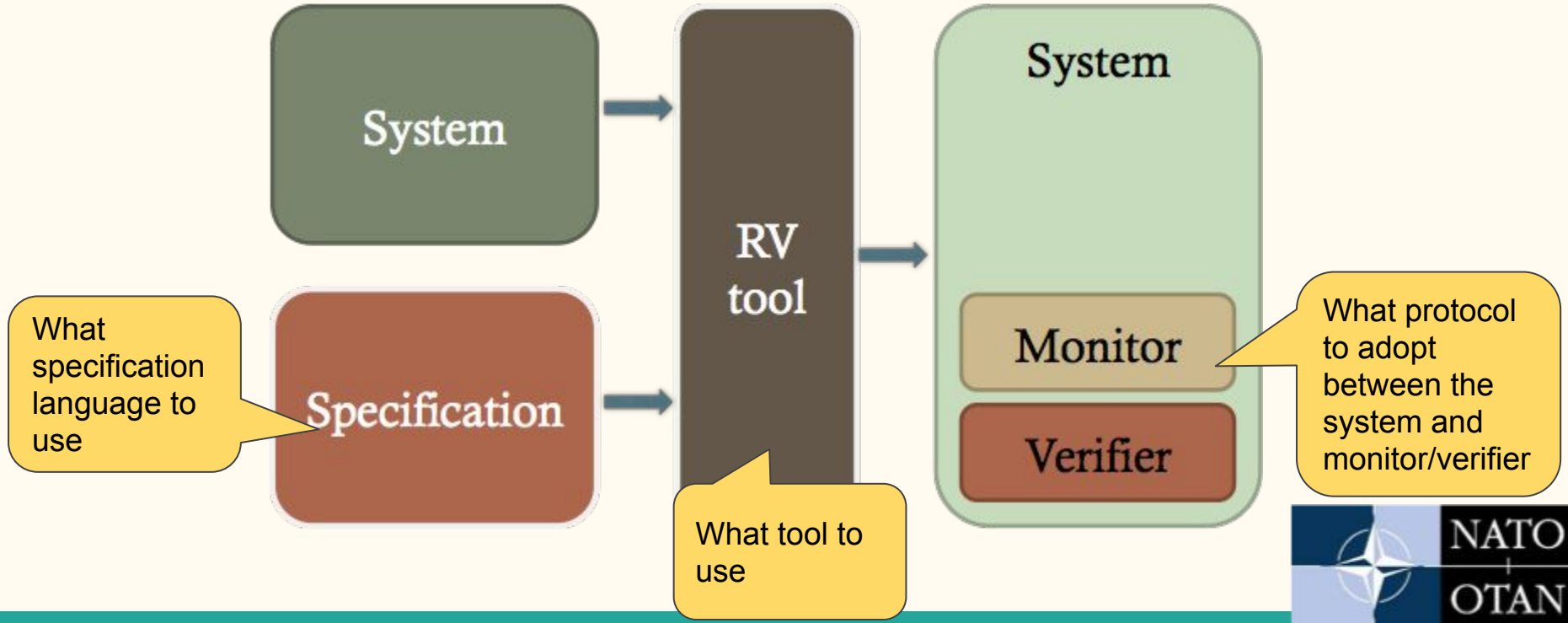
NATO OTAN

How do we use these techniques in practice?

Runtime Verification



Runtime Verification



High level logic

- Before any data is sent by the client,

  the server hash is verified to match the client's version

- If the operation is of type “Send”,

  then the message receiver ID must be in the set of approved receiver IDs

Low level considerations

General considerations for any code

- Arithmetic overflows
- Undefined downcasts
- Invalid pointer references

Mid-level

Applicable to any crypto protocol

Data flow monitoring

E.g. Ensuring no control is decided on secret data

(which affects the timing of the program)

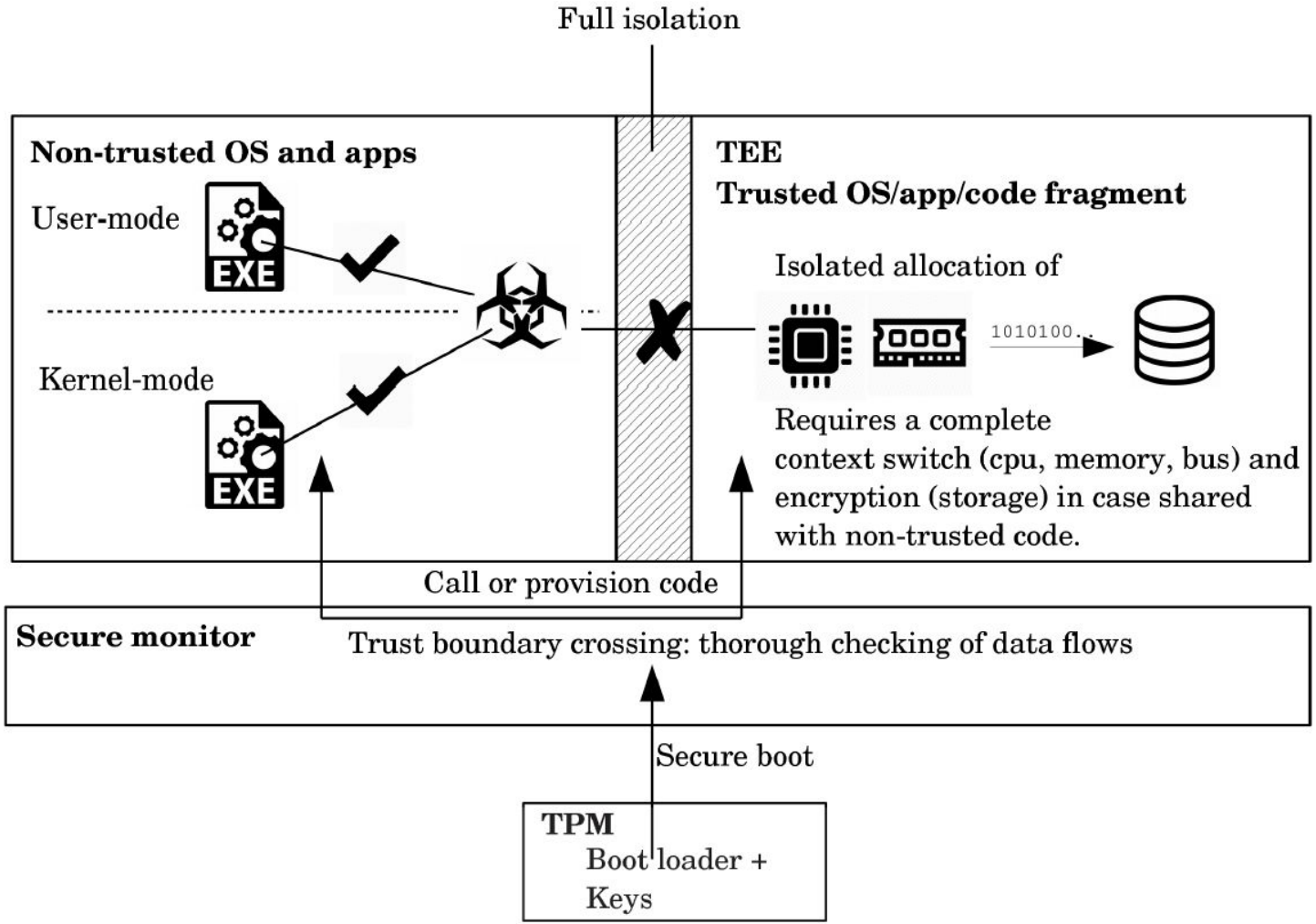
Challenges for RV

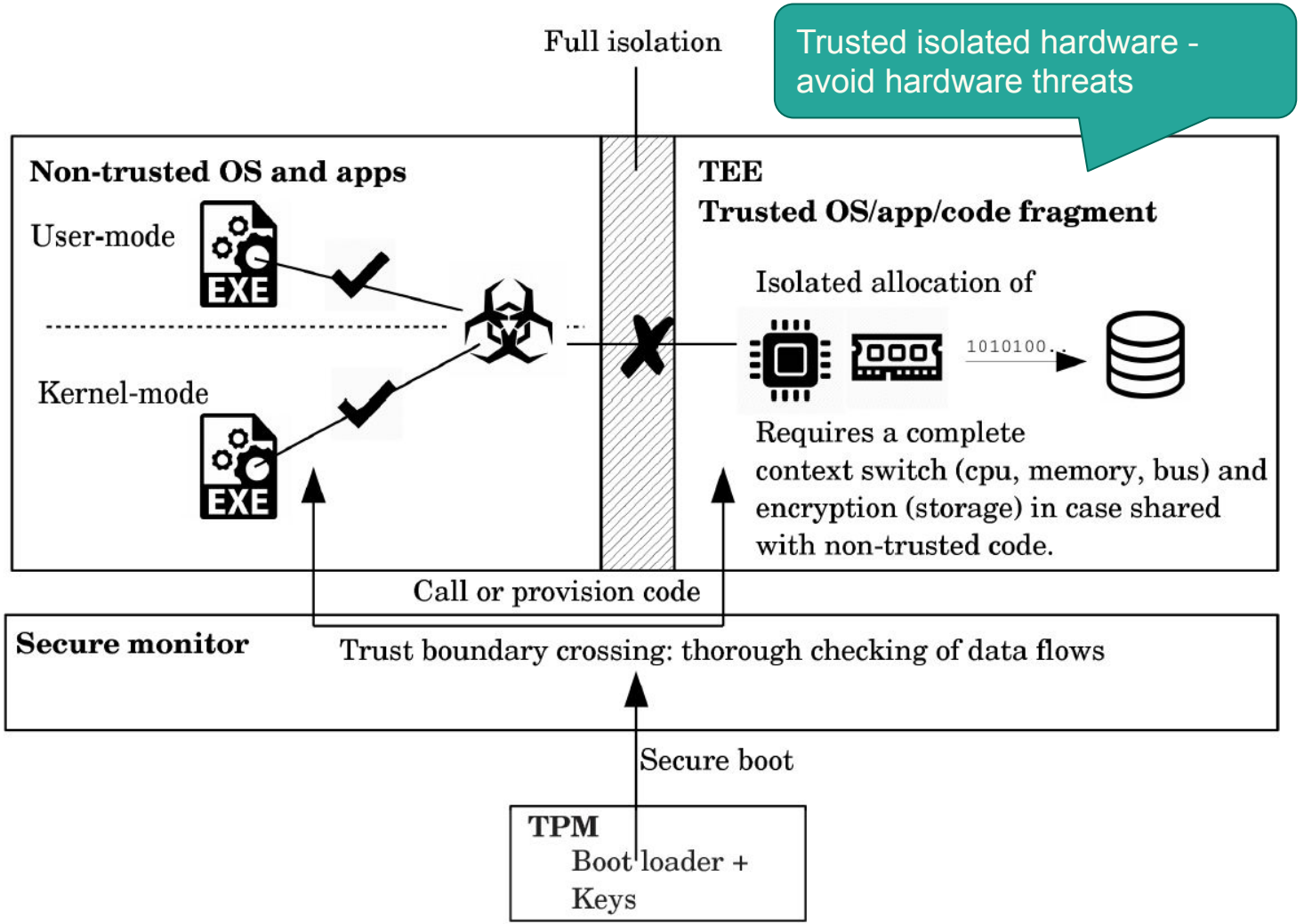
Over and above the usual correctness and **overheads** concerns

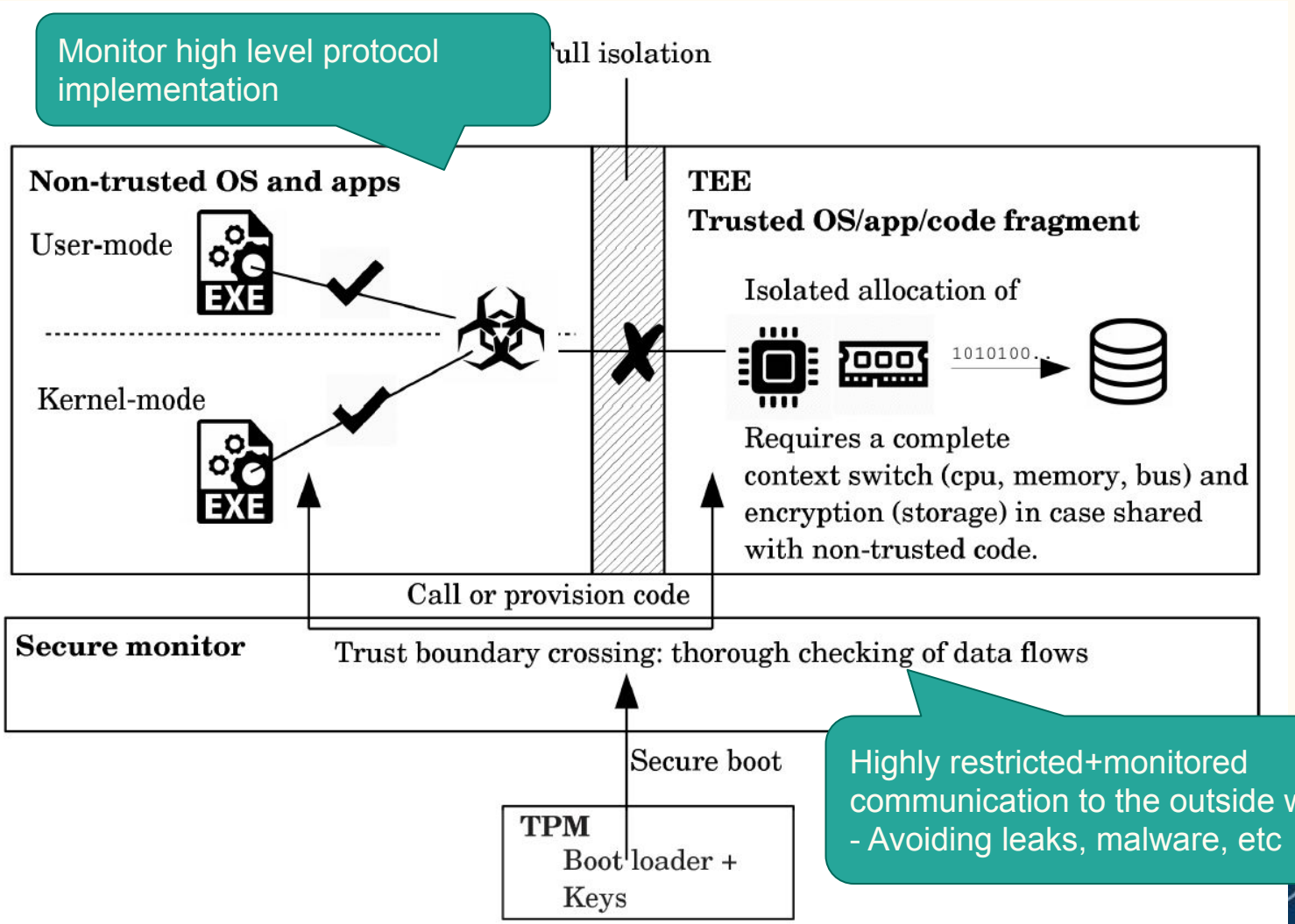
The monitor can present an additional security vulnerability

- As a piece of code
- As a reaction-triggering device

Our plan of comprehensive approach: Trusted Execution Environment (TEE)


Full isolation
Non-trusted OS and apps
User-mode
EXE
Kernel-mode
EXE
TEE
Trusted OS/app/code fragment
Isolated allocation of
1010100..
Requires a complete
context switch (cpu, memory, bus) and
encryption (storage) in case shared
with non-trusted code.
Call or provision code
Secure monitor
Trust boundary crossing: thorough checking of data flows
Secure boot
TPM
Boot loader +
Keys


Full isolation
Trusted isolated hardware - avoid hardware threats
Non-trusted OS and apps
User-mode
EXE
Kernel-mode
EXE
TEE
Trusted OS/app/code fragment
Isolated allocation of
1010100..
Requires a complete context switch (cpu, memory, bus) and encryption (storage) in case shared with non-trusted code.
Call or provision code
Secure monitor
Trust boundary crossing: thorough checking of data flows
Secure boot
TPM
Boot loader + Keys


Monitor high level protocol implementation
ull isolation
Non-trusted OS and apps
User-mode
EXE
Kernel-mode
EXE
TEE
Trusted OS/app/code fragment
Isolated allocation of
1010100..
Requires a complete context switch (cpu, memory, bus) and encryption (storage) in case shared with non-trusted code.
Call or provision code
Secure monitor
Trust boundary crossing: thorough checking of data flows
Secure boot
TPM
Boot loader + Keys
Highly restricted+monitored communication to the outside world - Avoiding leaks, malware, etc

What can go wrong at runtime?

...but in practice is far from enough