SCONE: Secure Container Technology & Secrets Management

Christof Fetzer, TU Dresden

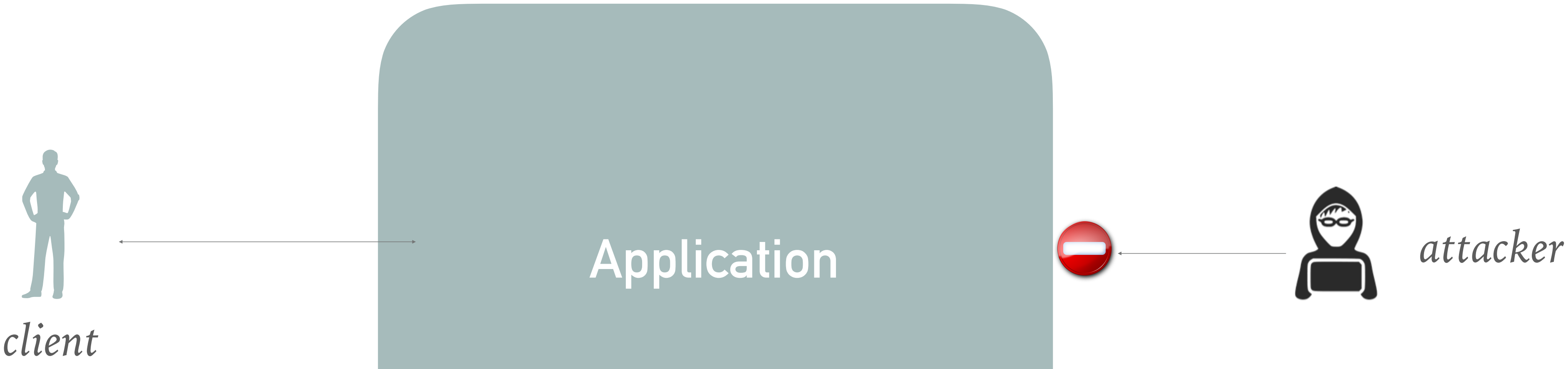
Sept 2018

https://sconedocs.github.io

TECHNISCHE UNIVERSITÄT DRESDEN

SecureCloud

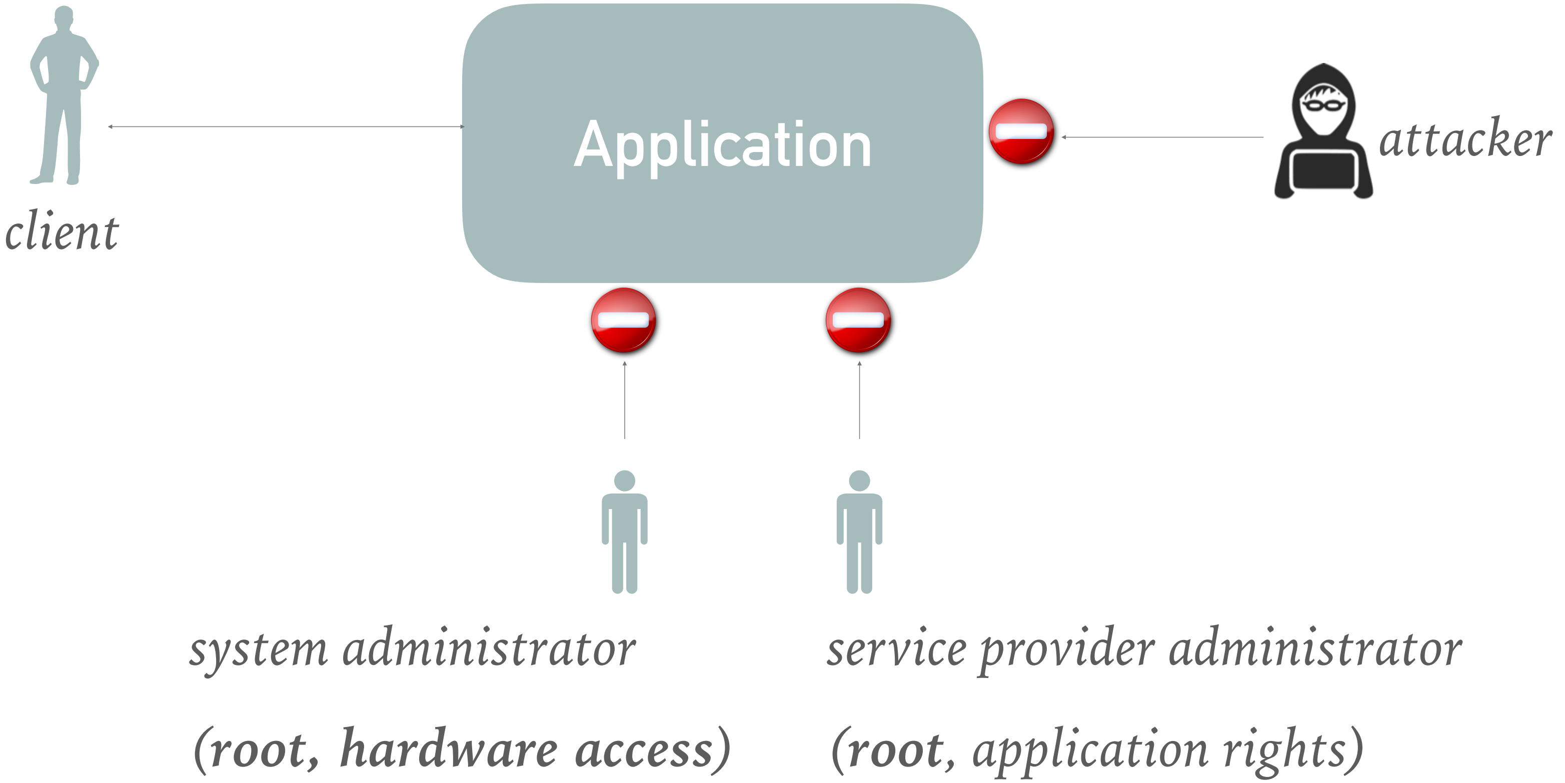
SCONE: Application-Oriented Security



Objective: *Ensure integrity and confidentiality of applications*

https://sconedocs.github.io

Threat Model



client

Application

attacker

system administrator

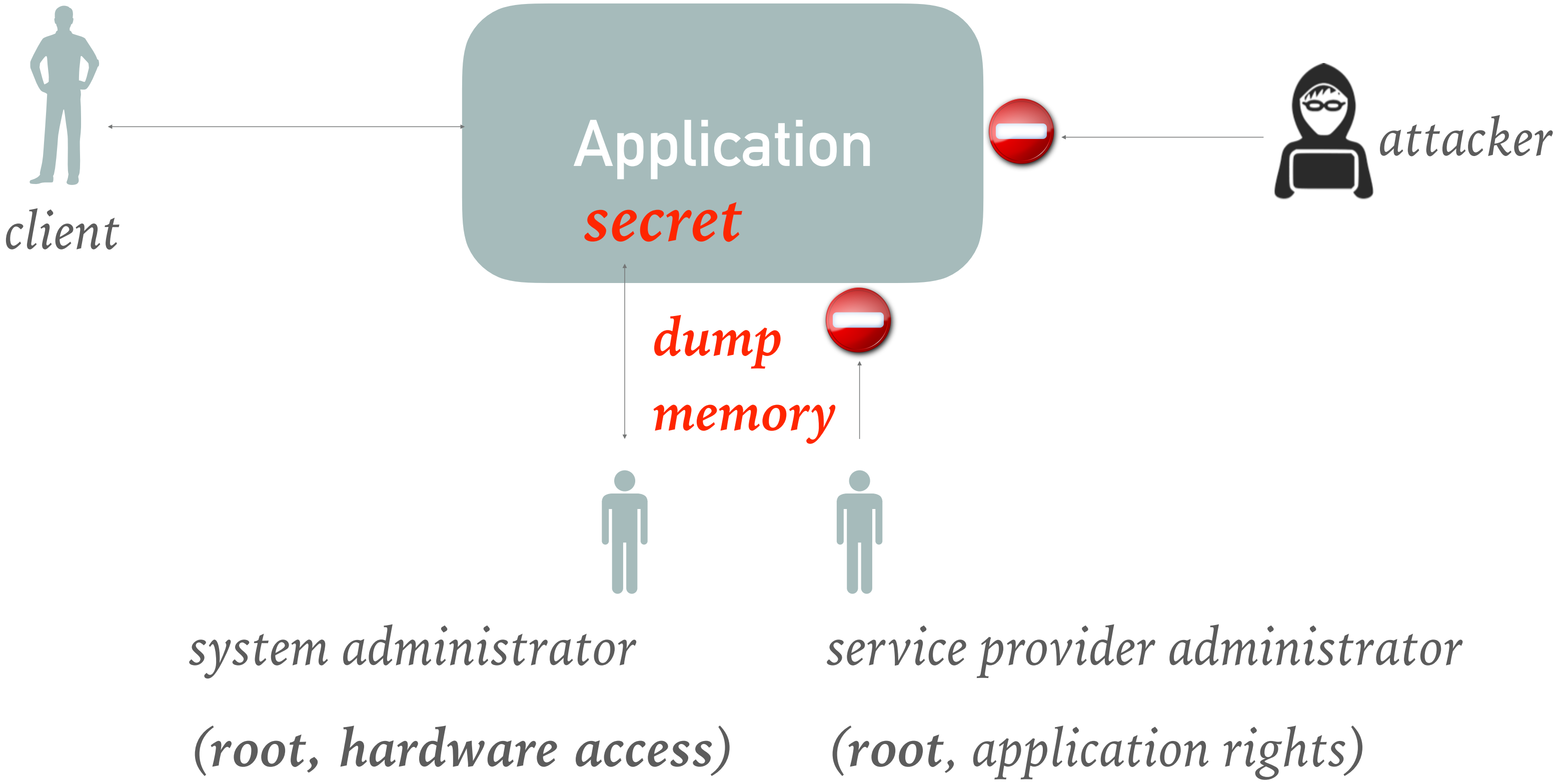
(root, hardware access)

service provider administrator

*(**root**, application rights)*

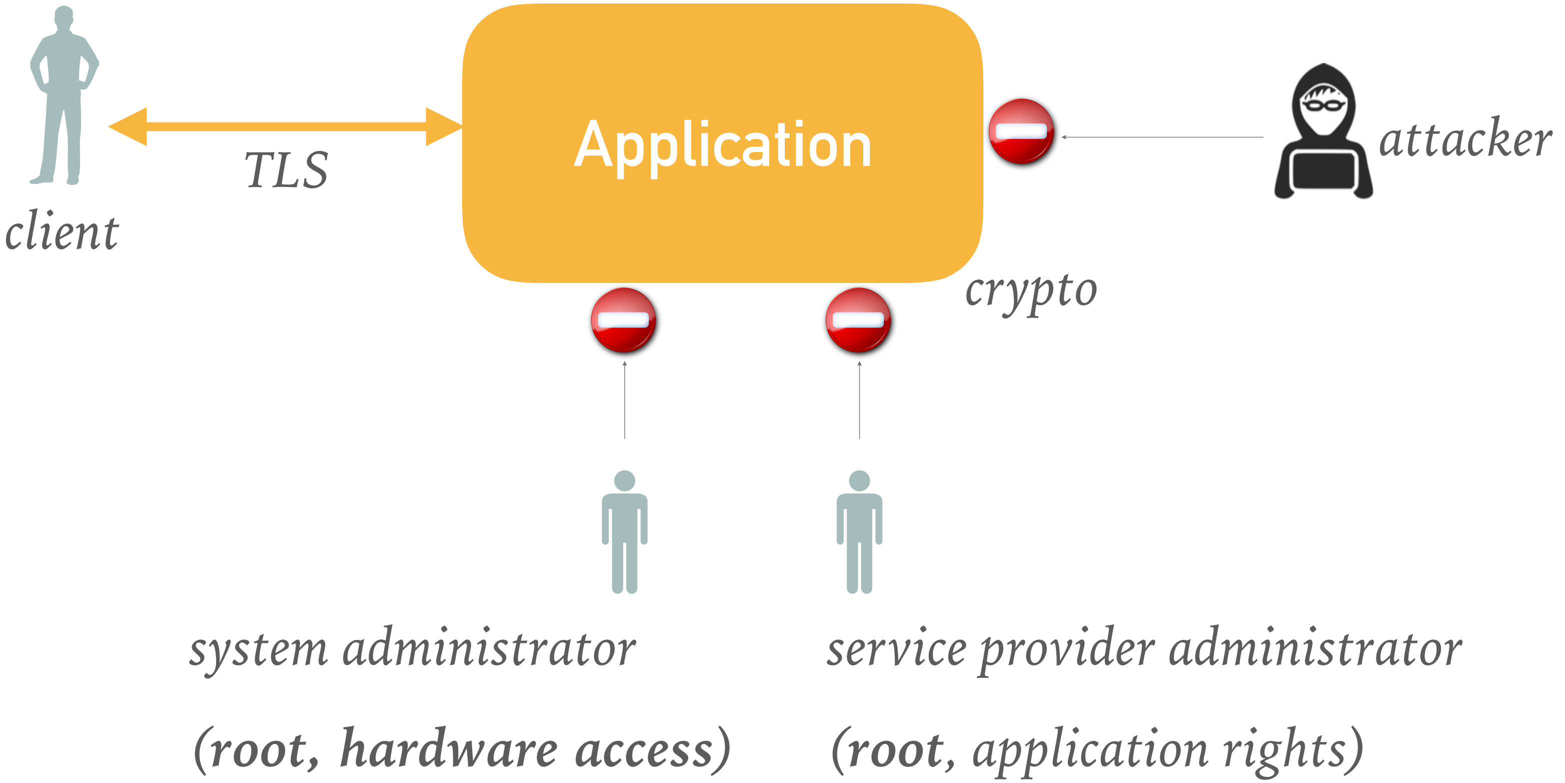
https://sconedocs.github.io

Implication: OS-based Access Control Insufficient



https://sconedocs.github.io

We need a cryptographic approach!



client

TLS

Application

attacker

crypto

system administrator

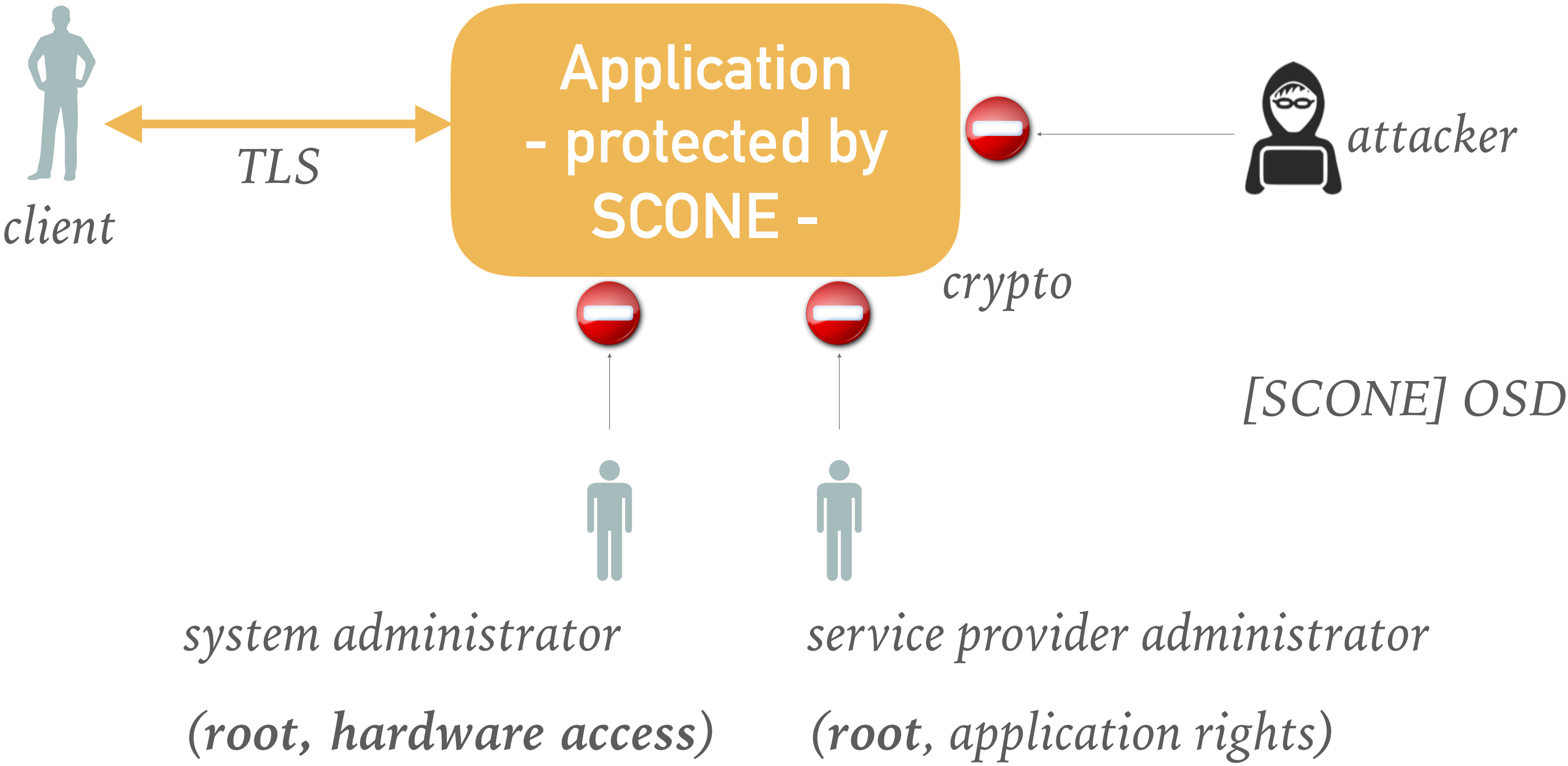
(root, hardware access)

service provider administrator

(root, application rights)

https://sconedocs.github.io

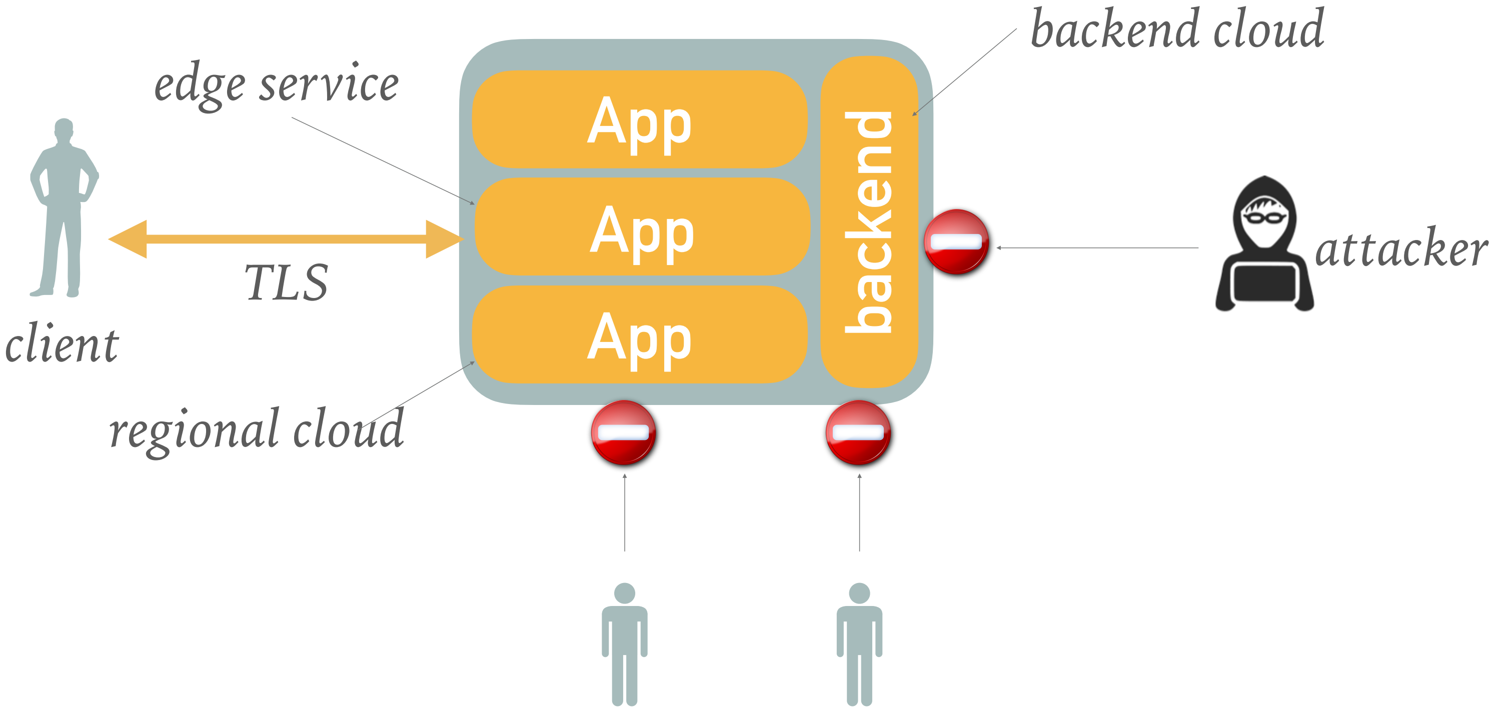
SCONE: E2E encryption **without source code changes**



[SCONE] OSDI2016

Languages: C, C++, Go, Rust, Java, Python, R, …

Distributed Applications - spread across clouds



system administrator

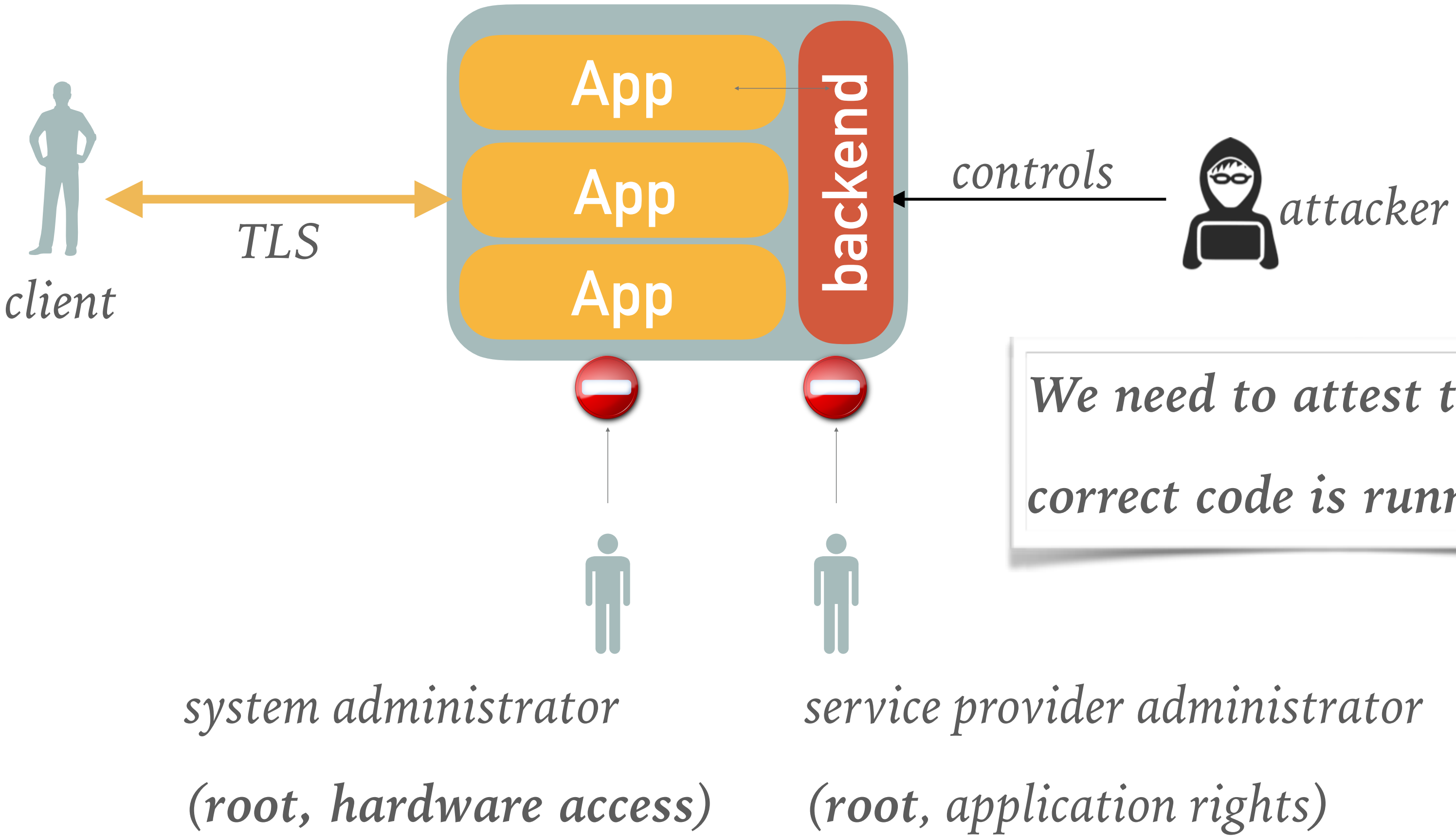
*(**root, hardware access**)*

service provider administrator

*(**root**, application rights)*

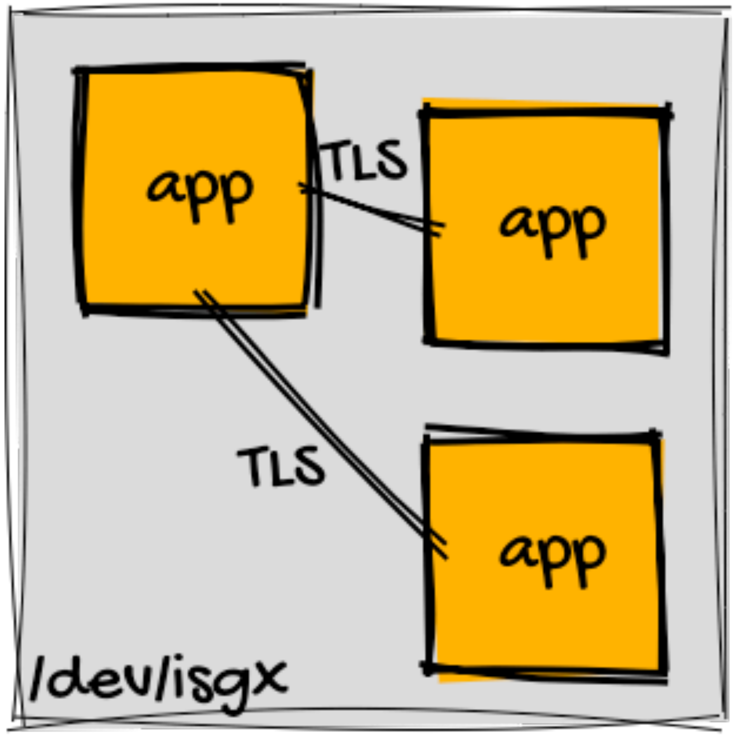
Initial Focus:
Cloud Native Applications

How do we know that correct code executes?



We need to attest that the correct code is running!

Approach: All communication is encrypted (TLS)

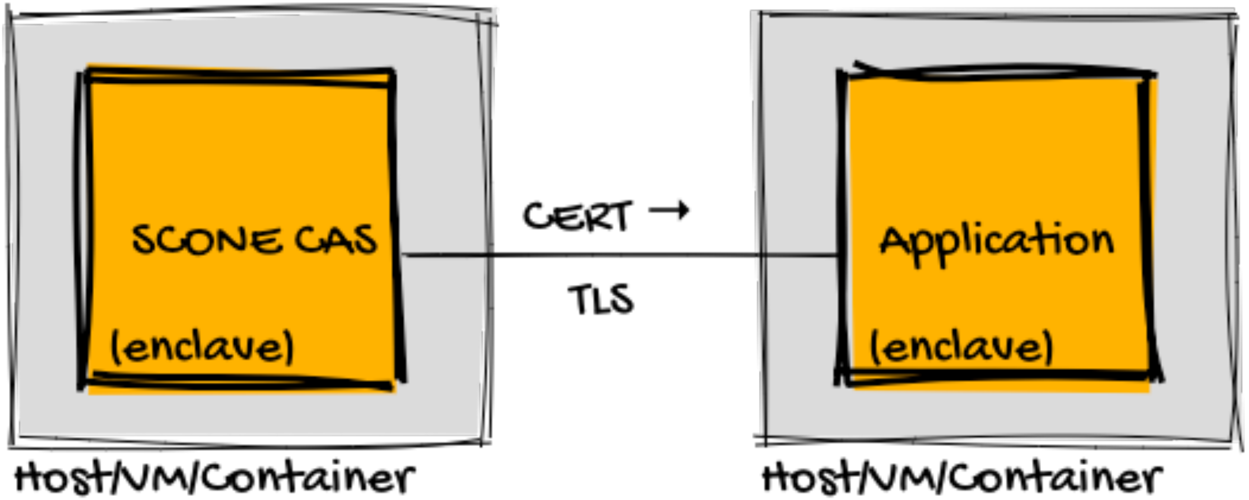


- Use TLS to authenticate
  - server app
  - client app
- We ensure that only app with
  - „correct code“ has access to TLS certificate

TLS: Transport Layer Security

Transparent Attestation during Startup

Configuration &
Attestation
Service



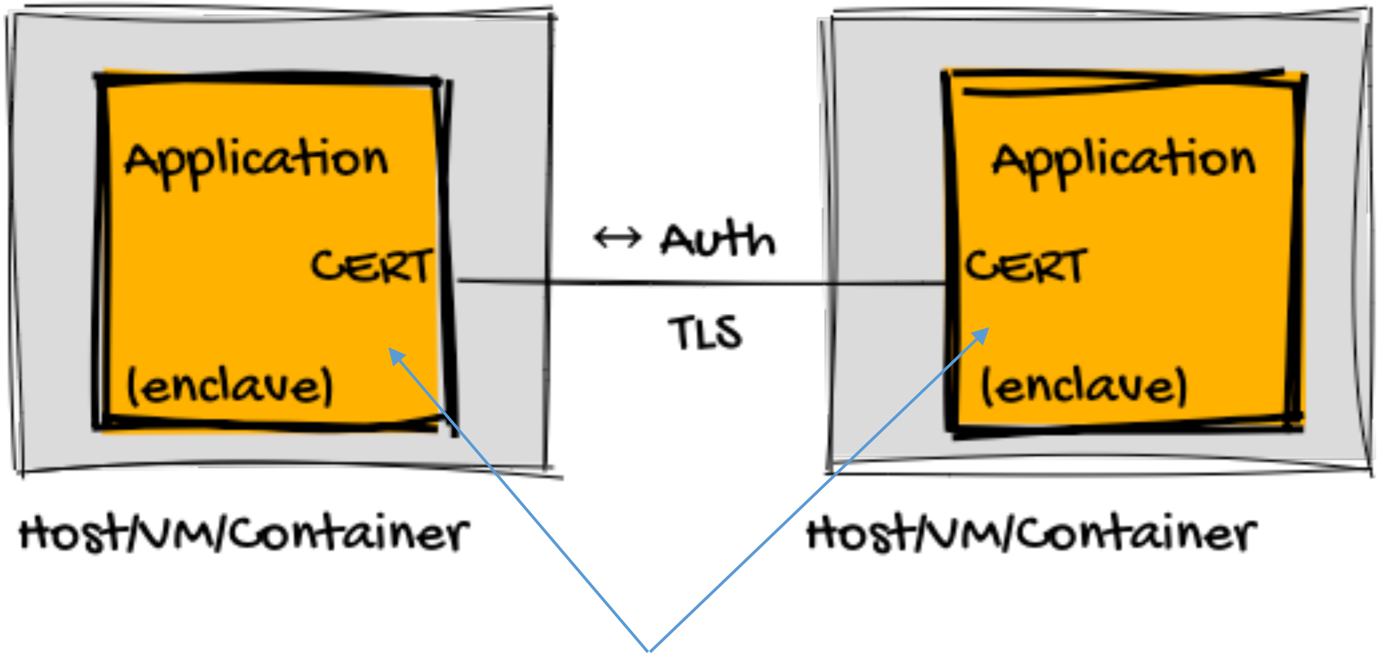
***certificate**: proves that application*

- *executes **correct code**,*
- *has the **correct file system state**, and*
- *in the **correct OS environment**, …*

https://sconedocs.github.io

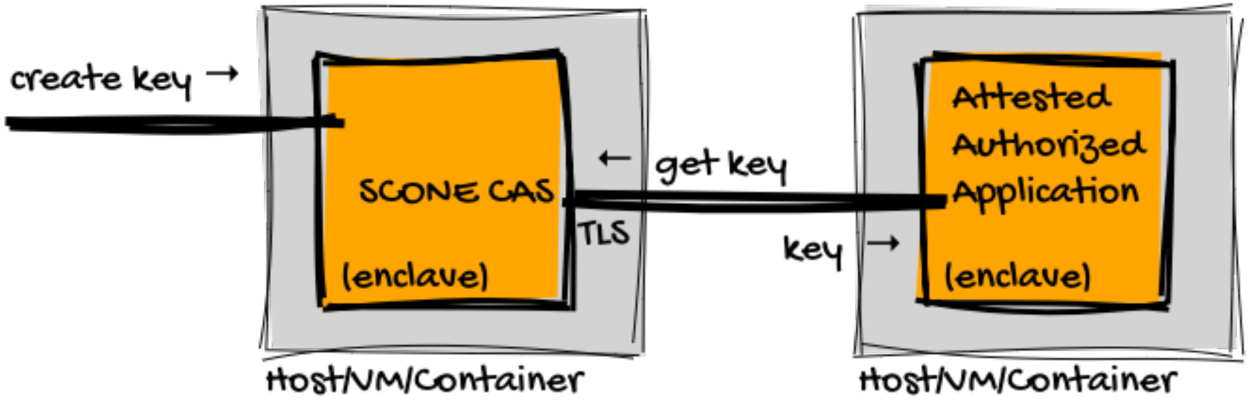
Transparent P2P Attestation via TLS

> *We run our internal CA and only components belonging to the same app can talk to each other …*



Certificate Authority (integrated in CAS)

Secrets Management

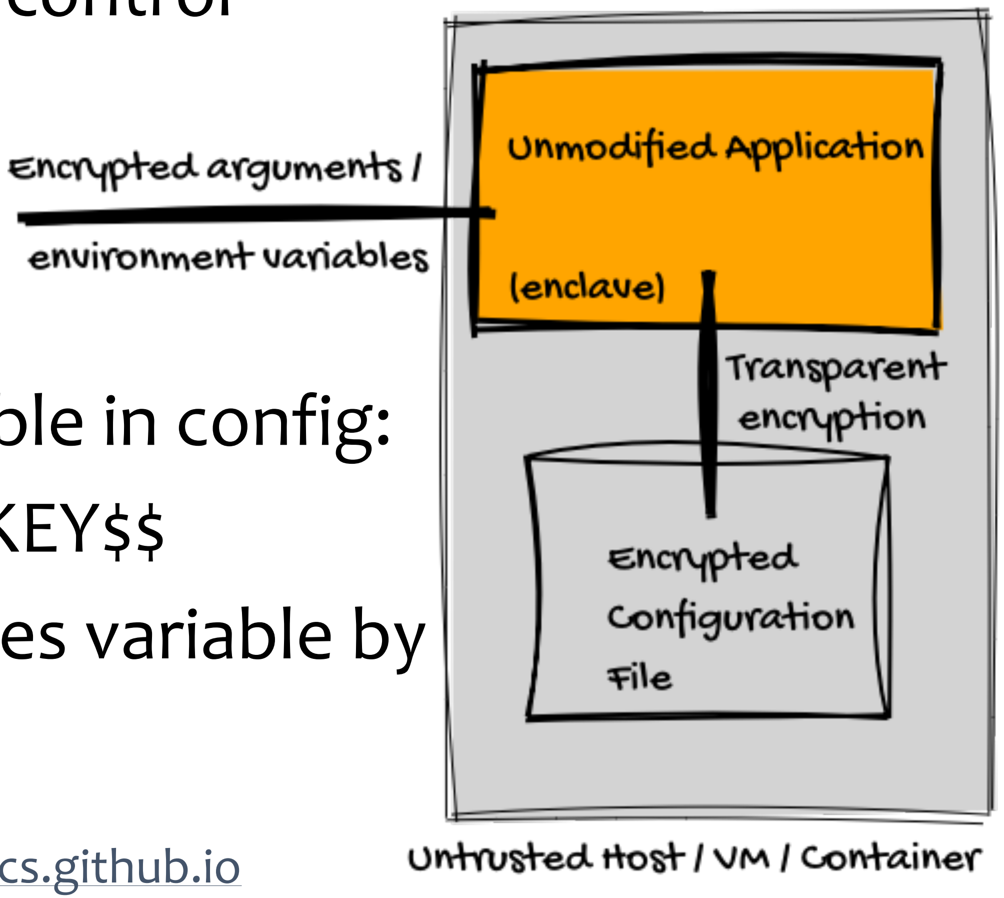


- SCONE has integrate secrets management
  - SCONE can inject secrets into
    - CLI arguments
    - environment variables
    - files (encrypted)

https://sconedocs.github.io

Example: MariaDB

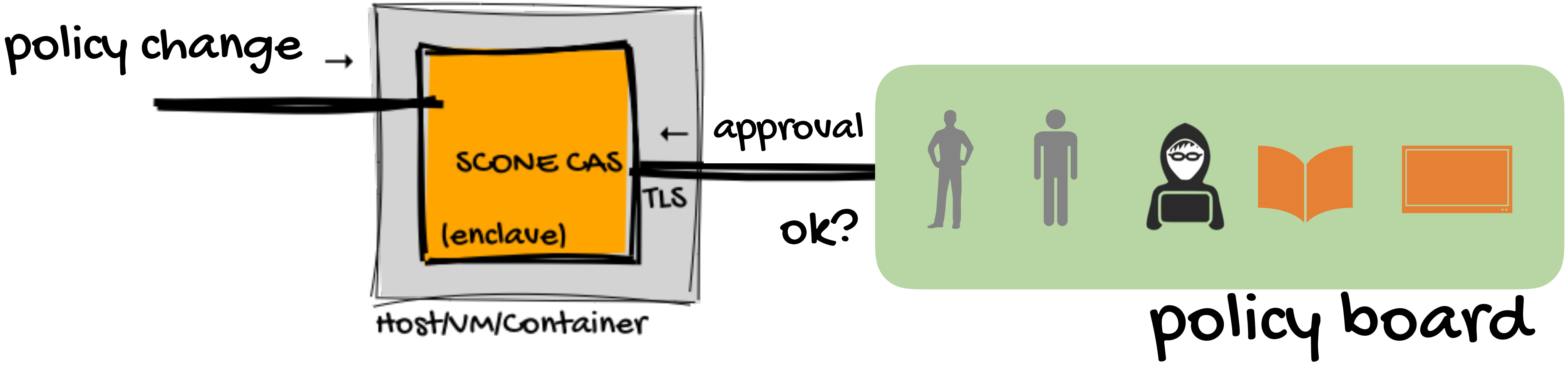
- Supports encryption of database
- Encryption key of database stored in config file
  - file protected via OS access control
  - file is not encrypted
- SCONE:
  - instead of key, store a variable in config:
    - $$SCONECAS:MARIADBKEY$$
  - SCONE transparently replaces variable by its value (i.e., the key)



https://sconedocs.github.io

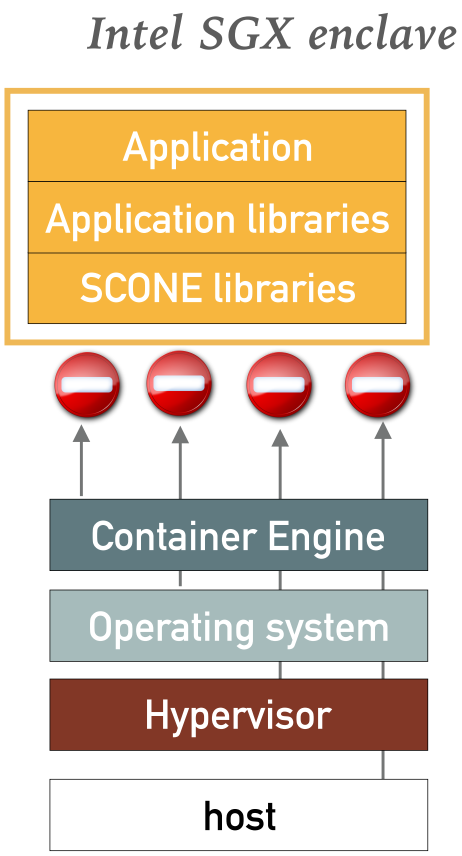
Management of Secrets

- Keys can be protected from any human access
  - only attested programs get access
- **To change security policy, approval by**
  - by a group of humans, and/or
  - a group of programs is required



https://sconedocs.github.io

Current Implementation

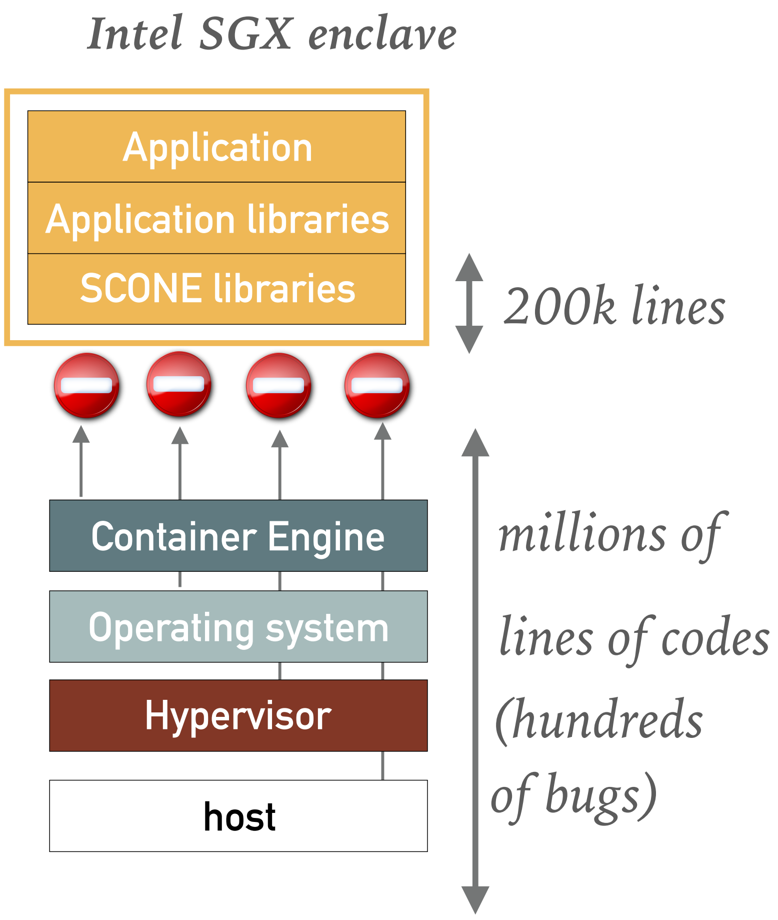


- Intel SGX protects application's
  - confidentiality
  - integrity
- by preventing accesses to
  - application state in cache and
  - encrypting main memory
- SGX is a TEE (Trusted Execution Environment)

SGX (Software Guard eXtensions) protects application from accesses by other software

https://sconedocs.github.io

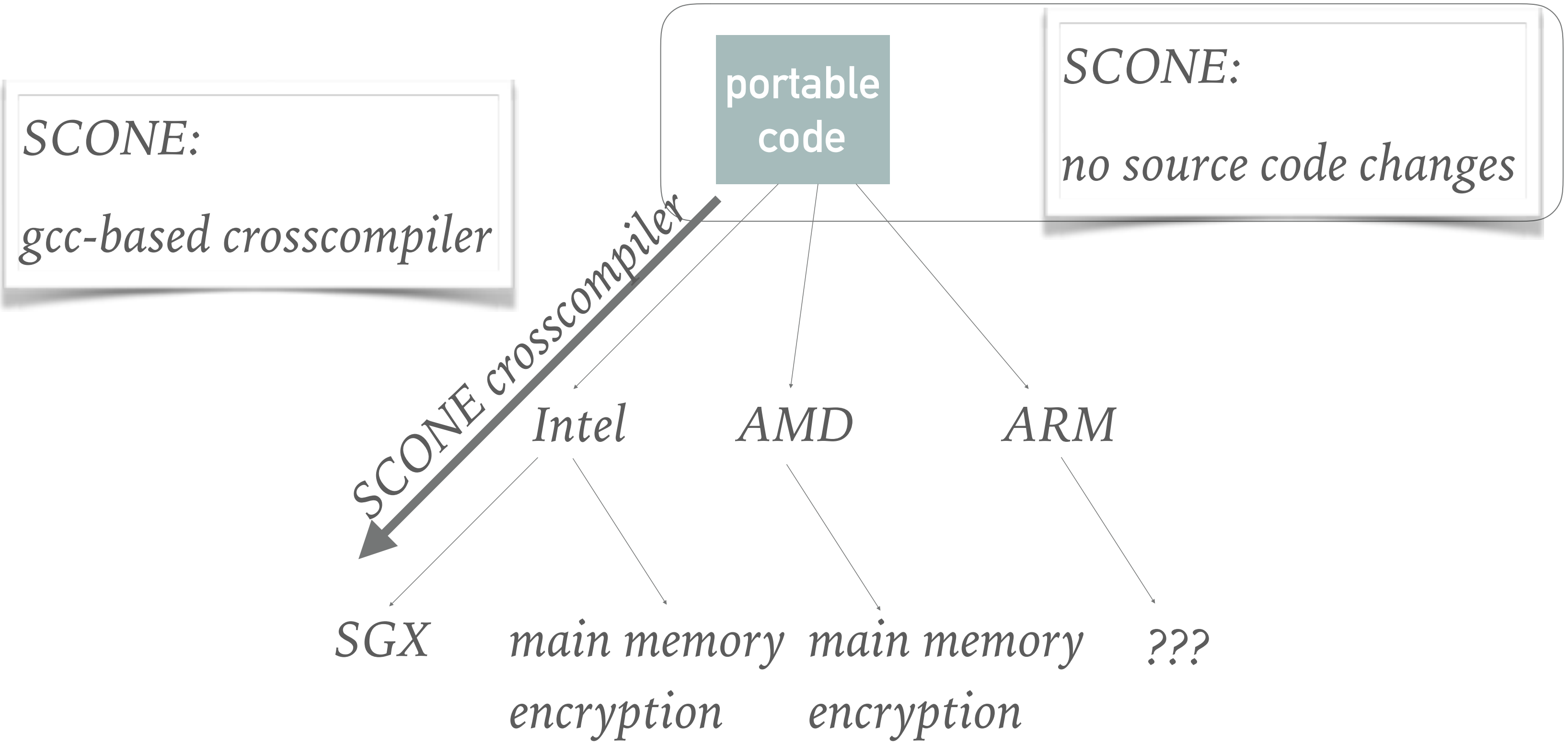
Defender’s Dilemma



- **Attackers**:
  - success by exploiting a single vulnerability
- **Defender**:
  - must protect against every vulnerability
    - **system software & application**
  - millions of lines of source code

https://sconedocs.github.io

SCONE platform: Designed for multiple Architectures



SCONE:

gcc-based crosscompiler

portable code

SCONE:

no source code changes

SCONE crosscompiler

Intel

AMD

ARM

SGX

main memory encryption

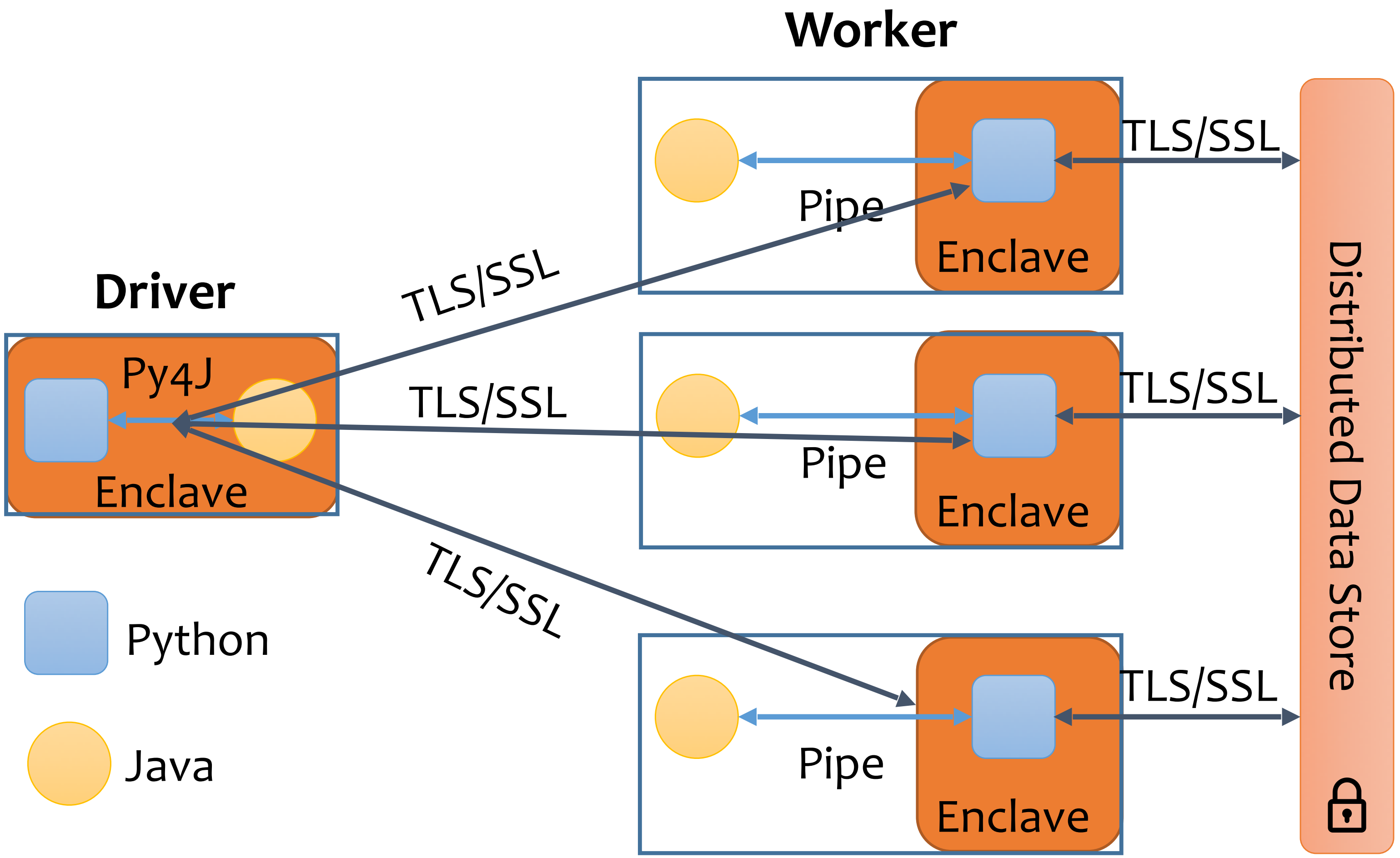
main memory encryption

???

Portability through cross-compilation

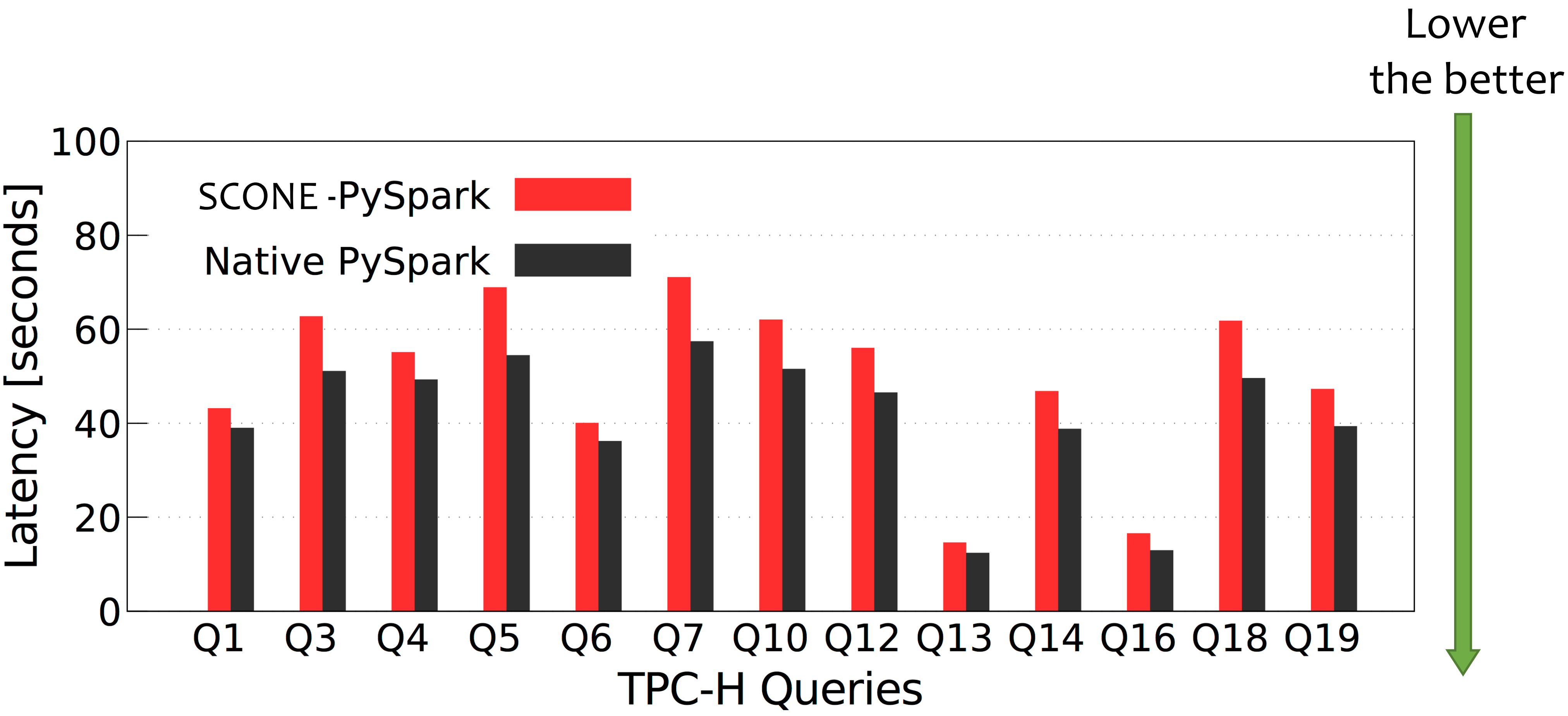
https://sconedocs.github.io

Use Case: SCONE-PySpark



https://sconedocs.github.io

Latency



< 22 % overhead compared to native execution

https://sconedocs.github.io

https://sconedocs.github.io