SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

ST Engineering Electronics

Careful-Packing: A Practical and Scalable Anti-Tampering Software Protection enforced by Trusted Computing

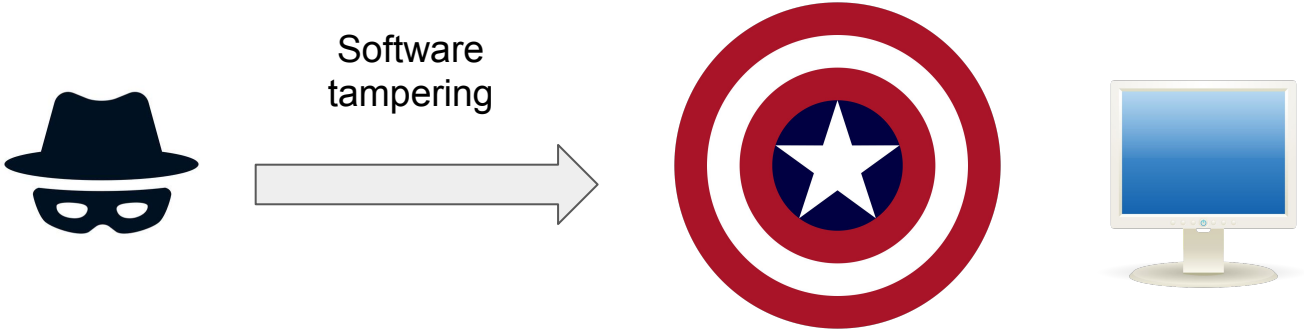
Flavio Toffalini[1,2], Martín Ochoa[2,3], Jun Sun[1,2], and Jianying Zhou[1]
[1]Singapore University of Technology and Design
[2]ST Electronics-SUTD Cyber Security Laboratory
[3]Cyxtera Technologies

Agenda

- Introduction
- Attacker’s goal and assumptions
- Careful-Packing’s approach
- Challenges
- Implementation
- Evaluation

The Problem

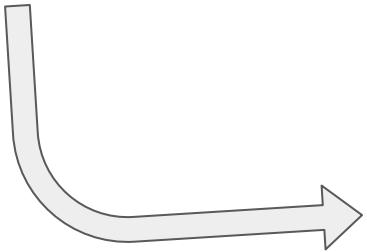
Secure Monitor:
- Antivirus
- Intrusion Detector
- Insider Threat Detector



Careful-Packing's goal

Avoiding an attacker to tamper with the software binary code.

Software

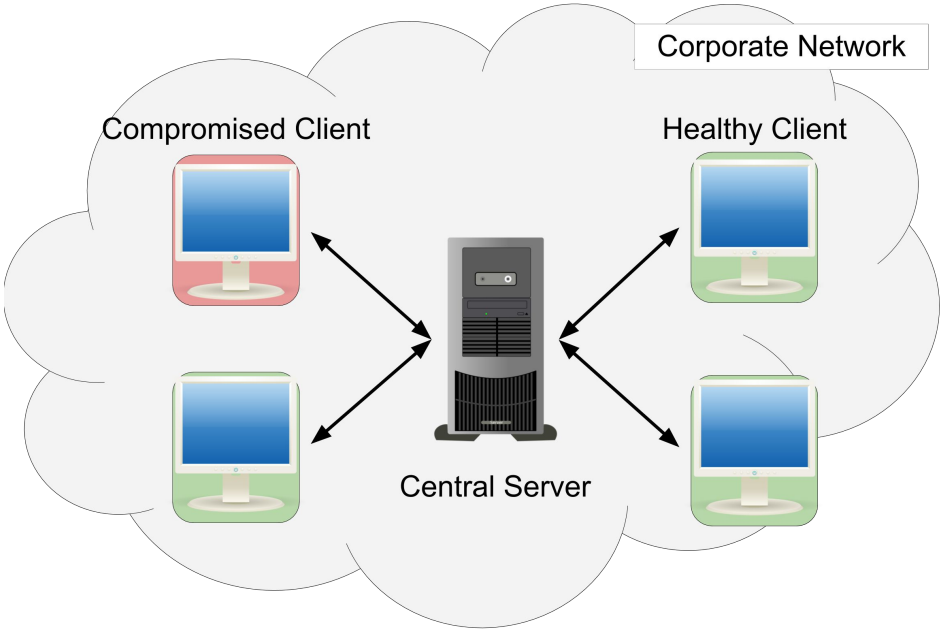


```
void makeALotOfMoney() {
  int x = 0;
  for (int i = 0; i < maxIteraction; i++) {
    int dollars = initialBudget();
    If (dollars > 10)
       // use a strategy
    else
      // use the super strategy..
    x += dollars;
  }
}
```

Important piece of code to protect (**critical section**)

Attacker’s goal

Install a **running compromised** client in a **corporate network**



Attacker model

- Tamper code **offline**: patching
- Tamper code **online**: patching & repair
- Debug/Emulate

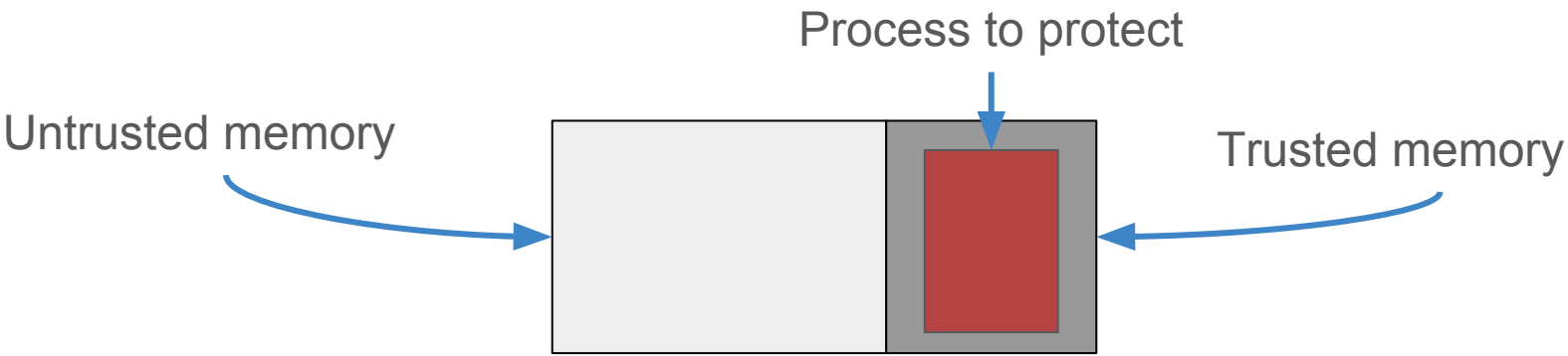
Assumption: OS Task scheduler not compromised

Previous Approaches - Fully Trusted Computing (SGX)

How: memory regions physically isolated by hardware

Idea: move the entire process (or a piece of) in these areas

Limitation: (i) difficult interaction with the OS, (ii) limited available space

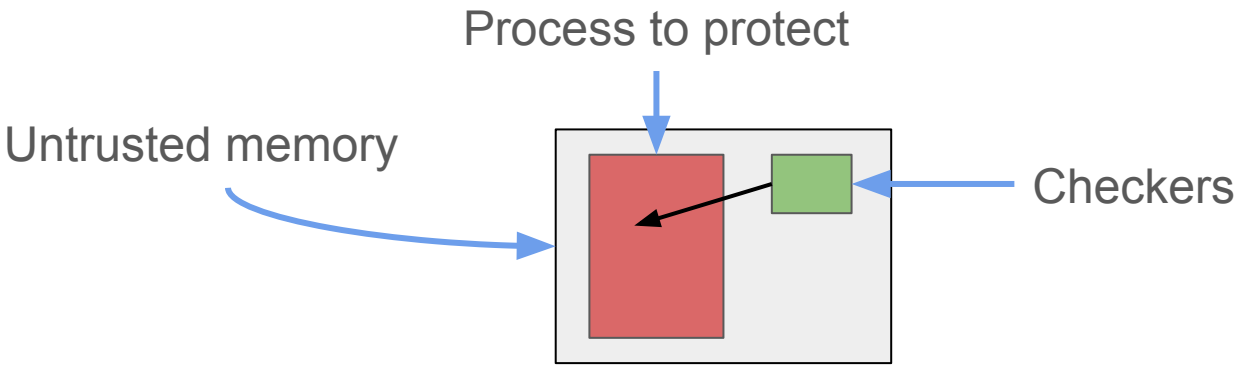


Previous Approaches - Purely Software Anti-Tampering (AT)

How: the software checks itself

Idea: using special functions (**checkers**) that inspect the memory

Limitation: these approaches rise the bar but not resolve the problem

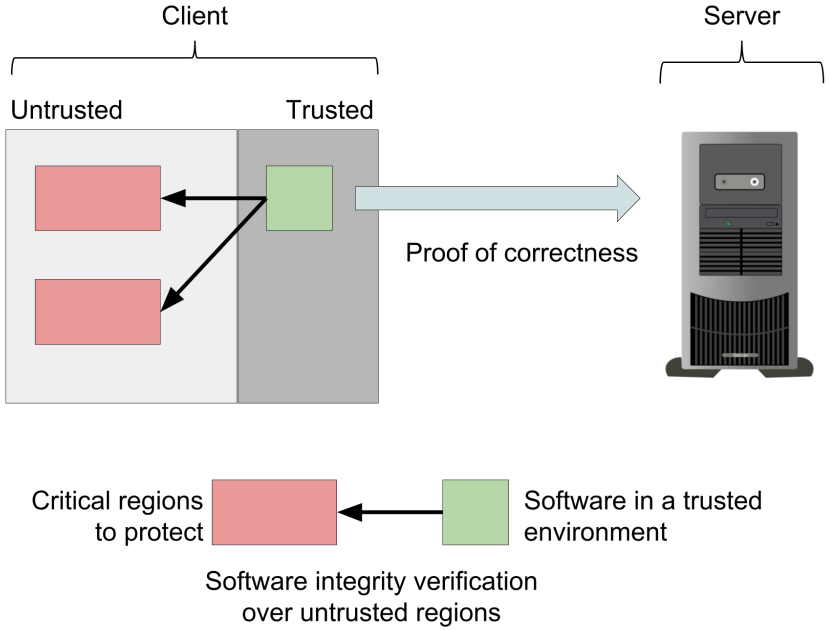


Careful-Packing's Approach

Checkers inside a trusted region

Critical Sections outside

From **trusted region** monitors the outside code!



Challenges

Denial of service (packing, heartbeat)

Concurrency

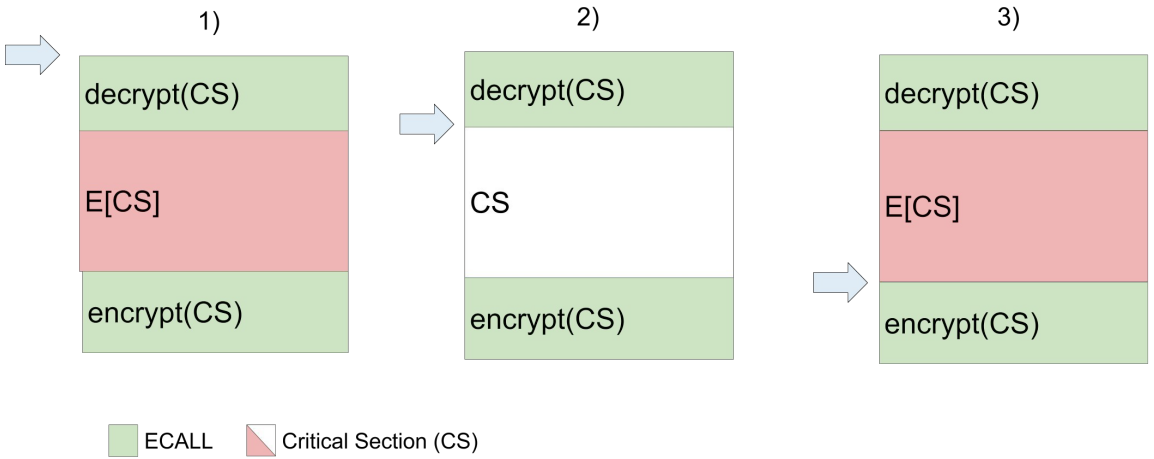
Installation phase

Booting phase

Packing

Critical Sections (CS) encrypted most of the time

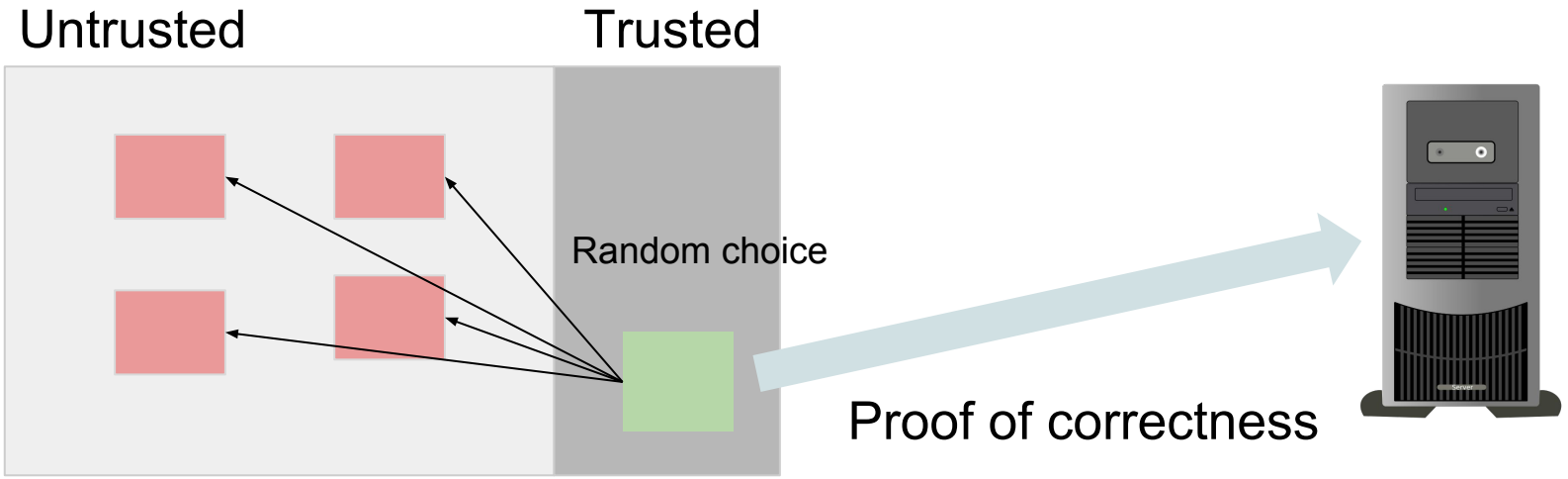
CS plain only when the process needs to execution them



Heartbeat

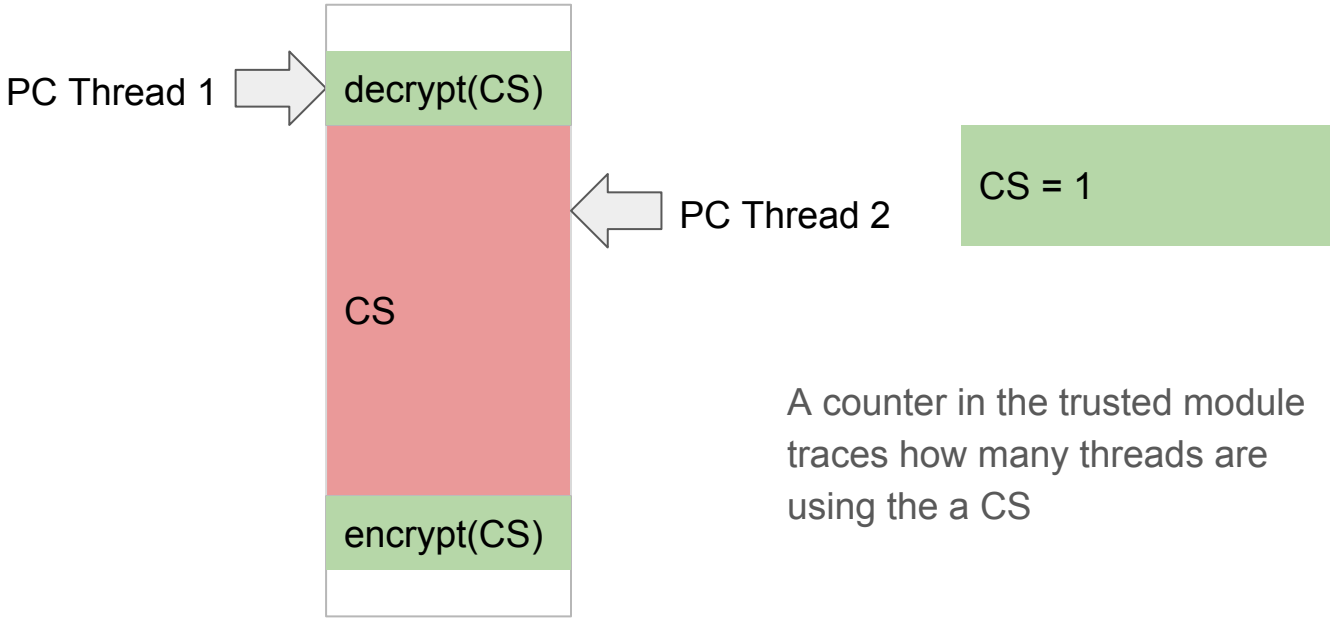
Insert proof of correctness inside a normal communication between client and server.

If the server does not receive a proof and/or the proof is wrong -> the attack is caught



Concurrency

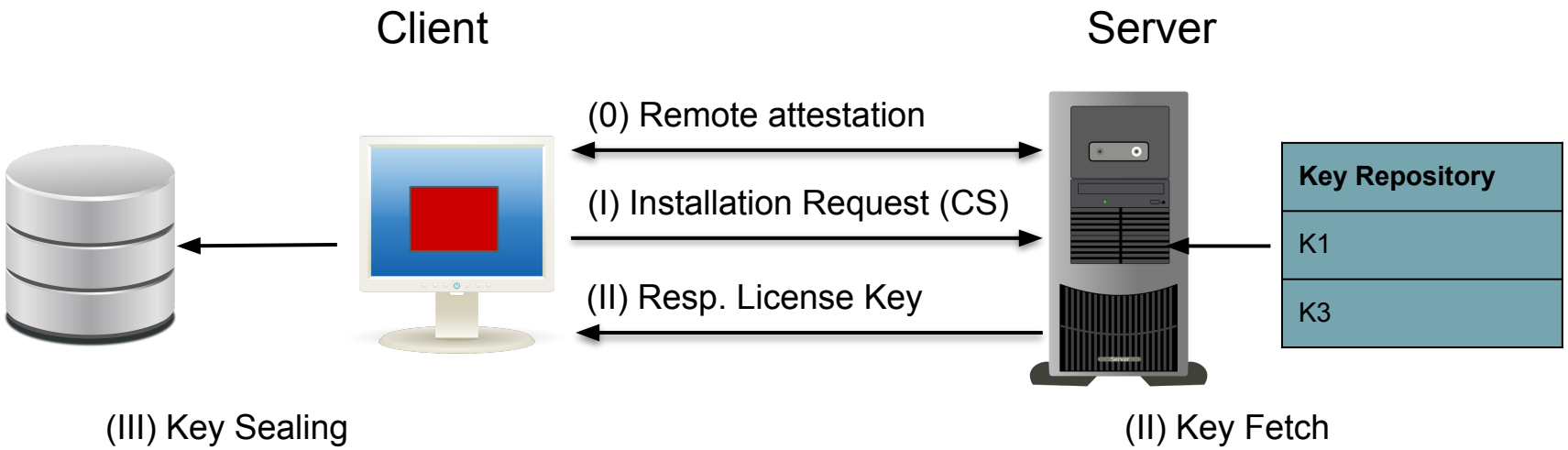
The CS can be invoked by concurrency threads.



A counter in the trusted module traces how many threads are using the a CS

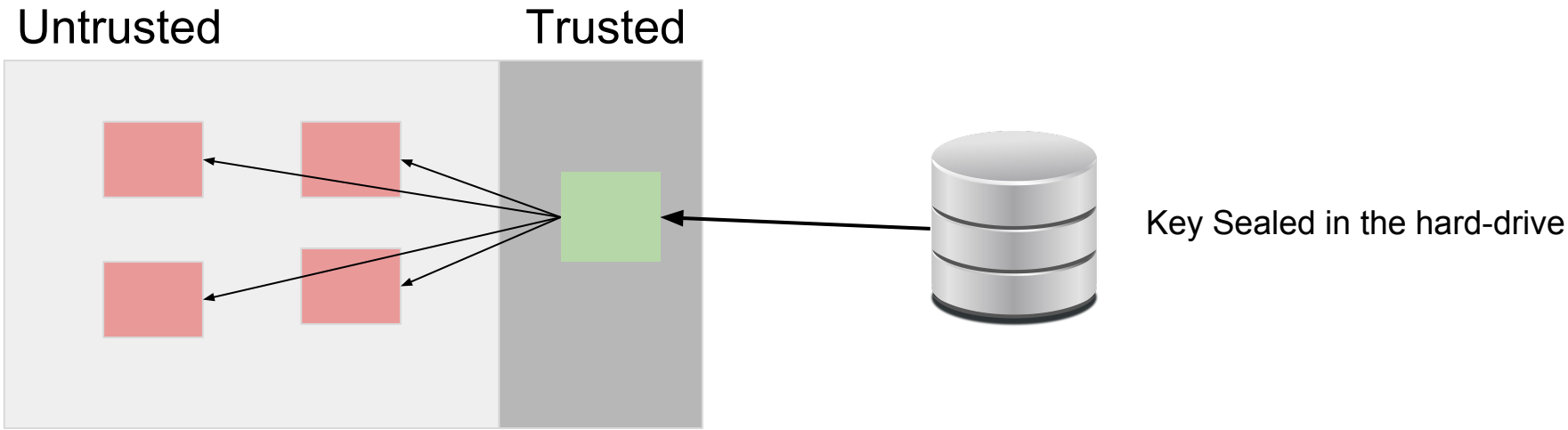
Installation phase

Server: has a list of valid **license keys** and the **critical sections** associated
Client: identifies itself through one of its **critical section**



Boot phase

Client loads the **sealed key** to execute the packing mechanism



Proof of Concept Implementation

Monitoring agent (simple keystrokes and mouse tracing).

A client is executed on a client, collects data, and ships them to a central server.

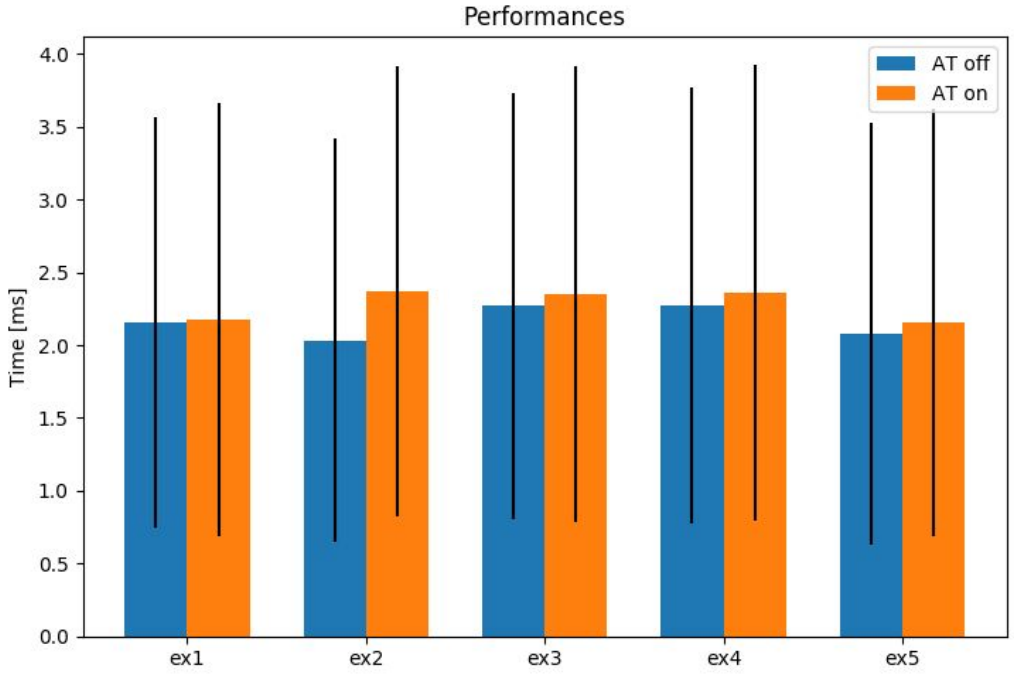
Based on SGX for Windows platforms.

-**10 LoC** on top of the original program (only function call to **checkers**).

- Enclave size: **~300Kb**

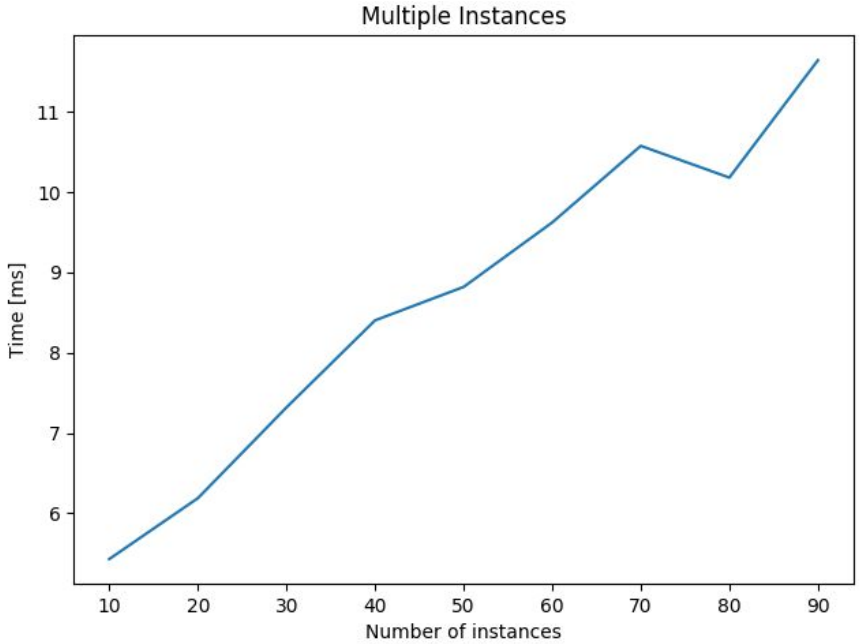
Evaluation - single instance

Around **5%** with related to the original program



Evaluation - multiple parallel instances

Linear overhead with respect to the number of instances (**no bottleneck**)



Conclusion & Take away

- To some extent, it is possible to combine trusted computing technologies and anti-tampering techniques to increase the security guarantees

- Extend careful-packing to protect variables (now we limit to the code)

- Fully untrusted environment: work on techniques that do not require a trusted scheduler

The End!

Thanks for your attention