Advanced Systems Security: Hardware Security

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Security Problems

- We have discussed lots of security problems
  - Malware on your computer
  - Attacks on memory errors
  - Return-oriented attacks
  - Compromised software
  - Compromised operating systems, etc.
- Is there any way new hardware features could prevent some attack vectors?

Hardware Features

- Trusted Platform Module (TPM)
  - Measure and attest the software running on a computer
- ARM TrustZone
  - Restrict execution of compromised operating systems
- Intel Software Guard Extensions (SGX)
  - Protect processing from compromised operating systems
- Intel Processor Trace (IPT)
  - Track control flow events
- Intel Memory Protection Extensions (MPX)
  - Check and enforce memory bounds

A Problem

- My computer is running a process
- It makes a request to your computer
  - Asks for some secret data to process
  - Provides an input you depend on
- How do you know it is executing correctly?
- Example
  - ATM machine is uploading a transaction to the bank
  - How does the bank know that this ATM is running correctly, so the transaction can be considered legal?

Question You Might Ask?

- Who owns the remote computer?
  - Does this tell you whether the computer has malware?
- Is the computer protected from ever running malware?
  - How would we know this?
- What is actually running on the computer?
  - How can get this information securely?
- Would any of these things enable you to determine whether to supply your personal information to the remote computer?

What would you do?

- Proof by authority (Certificates)
  - Validate the source of messages from the remote system
  - Tells you who (maybe), but not what/how
- Constrain the system (Secure Boot)
  - Remote system boots using only trusted software
  - Is only running if secure
- Inspect the runtime state (Authenticated Boot)
  - Remote system produces record of software run
  - You validate whether you trust the software

Secure Boot

- Check each stage in the boot process
  - Is code that you are going to load acceptable?
  - If not, terminate the boot process
- Must establish *a* ***Root-of-Trust***
  - A component trusted to speak for the correctness of others
  - Assumed to be correct because errors are **undetectable**

Authenticated Boot

- Secure boot enforces requirements and uses special hardware to ensure a specific system is booted
  - Implied verification (Good because it is)
- By contrast, we can **measure** each stage and have a **verifier** authenticate the correctness of the stage
  - Verifier must know how to verify correctness
  - Behavior is uncertain until verification
- What is root-of-trust for authenticated boot?

Secure v Authenticated Boot

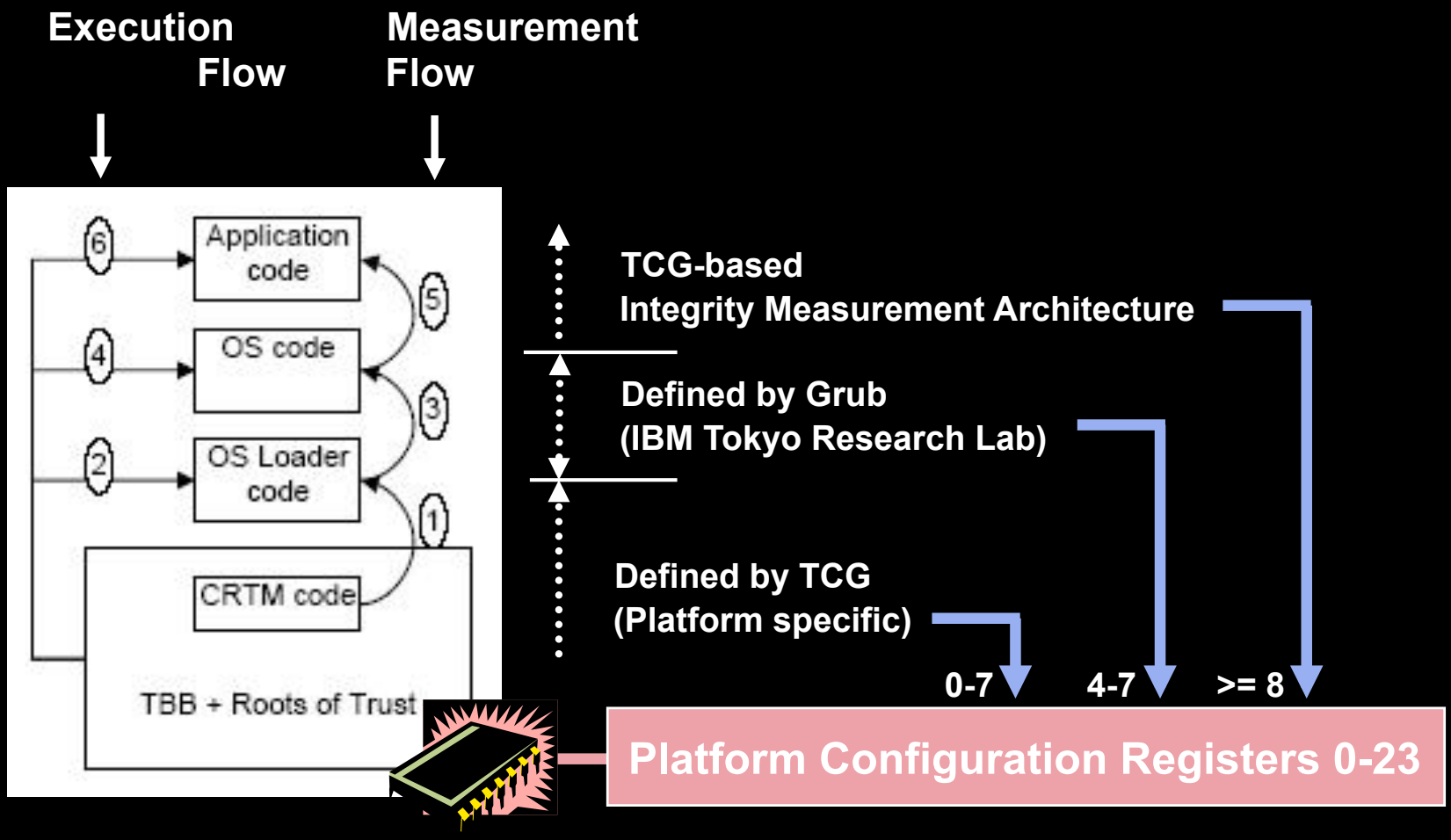
- Odd implications of each
- Secure boot enables you to tell if your machine is secure
  - But remote parties cannot tell
- Authenticated boot enables remote parties to tell if your machine is secure
  - But you cannot tell by using it yourself

Trusted Computing

PENNSTATE 1855

- The Trusted Platform Module (TPM) brought authenticated boot into the mainstream
- Essentially, the TPM offers few primitives
  - Measurement, cryptography, key generation, PRNG
  - Controlled by physical presence of the machine
  - BIOS is Core Root of Trust for Measurement (CRTM)
- Spec only discussed how to measure early boot phases and general userspace measurements

Trusted Computing Group Architecture

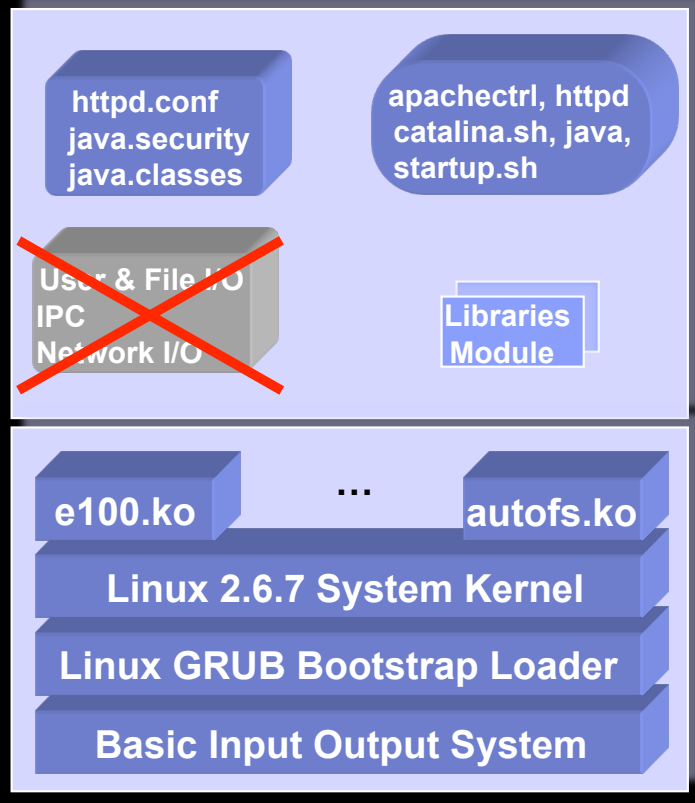


Trusted Computing

- What would you measure in the following system to prove it is running correctly to remote verifiers?

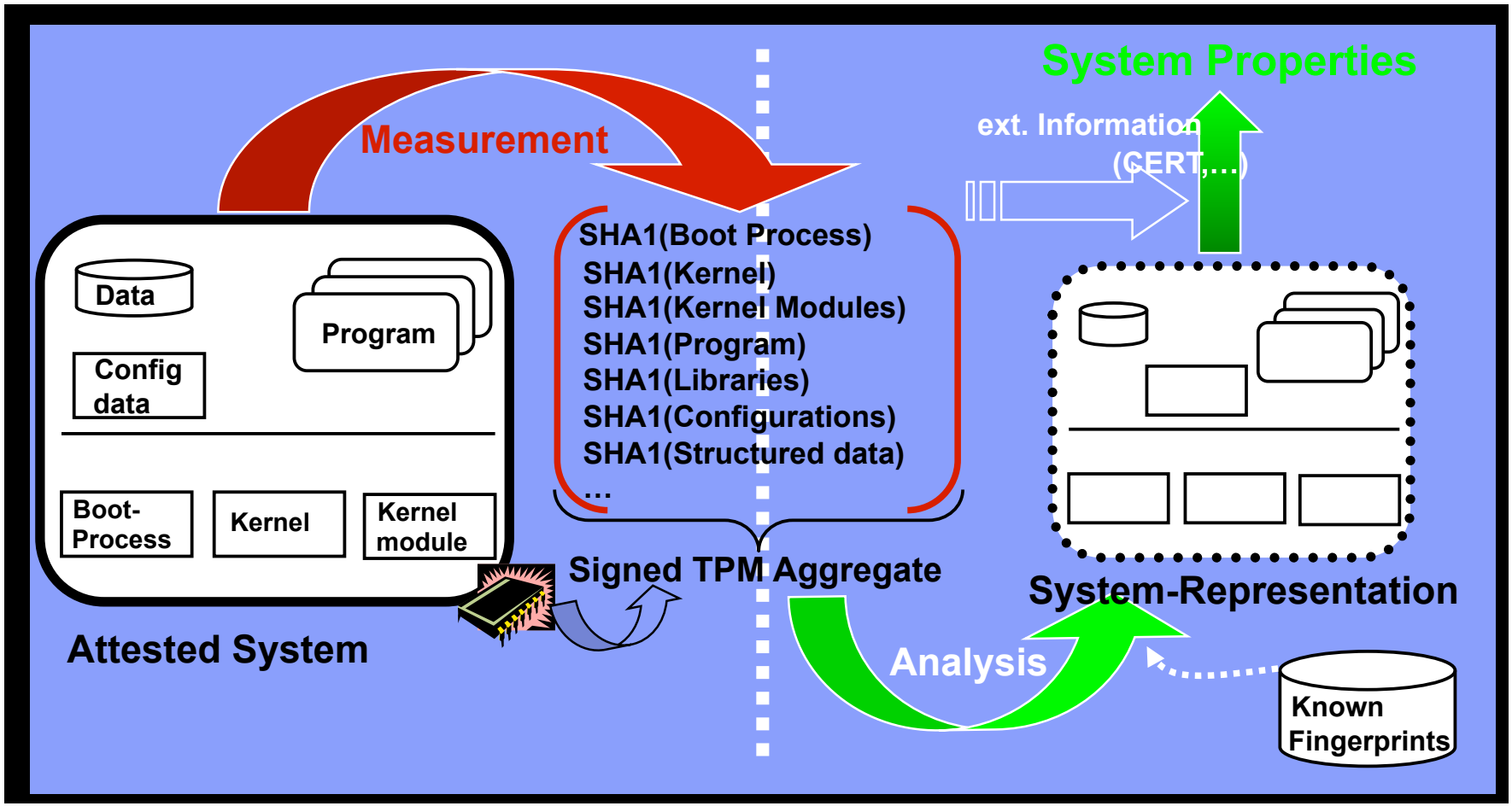
Example: Web Server

- **Executables**
  (Program & Libraries)
  - apachectrl, httpd, java, ..
  - mod_ssl.so, mod_auth.so, mod_cgi.so,..
  - libc-2.3.2.so libjvm.so, libjava.so, …
- **Configuration Files**
  - httpd.conf, html-pages,
  - httpd-startup, catalina.sh, servlet.jar
- **Unstructured Input**
  - HTTP-Requests
  - Management Data



Trusted Computing

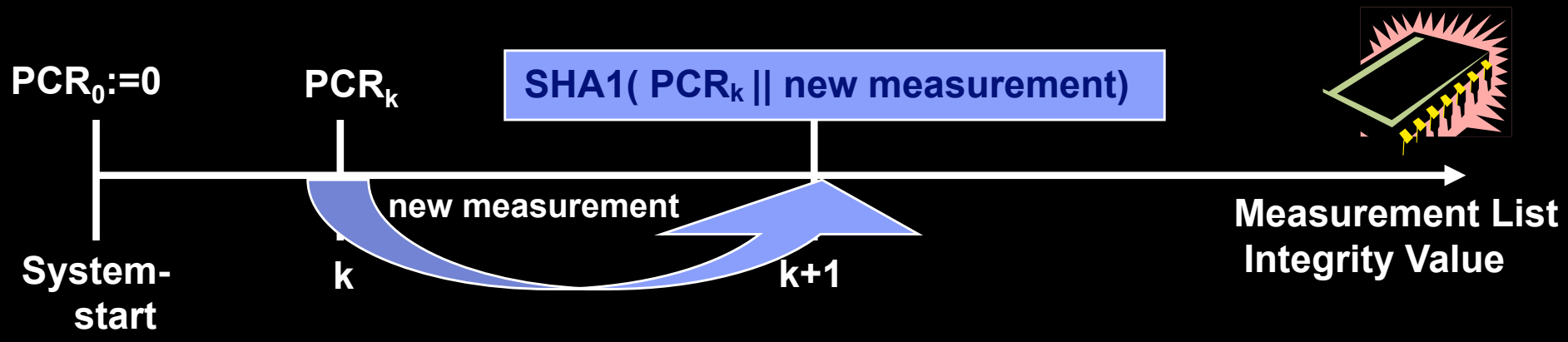
- How would you measure the following system to prove it is running correctly to remote verifiers?



Linux Integrity Measurement

Measurement list aggregation:

- **Compute** 160bit-SHA1 over the contents of the data (measurement)
- **Adjust** Protected hw Platform Configuration Register (PCR) to maintain measurement list integrity value
- **Add** measurement to ordered measurement list
  - → Executable content is recorded before it impacts the system
  - → That is, before it can corrupt the system

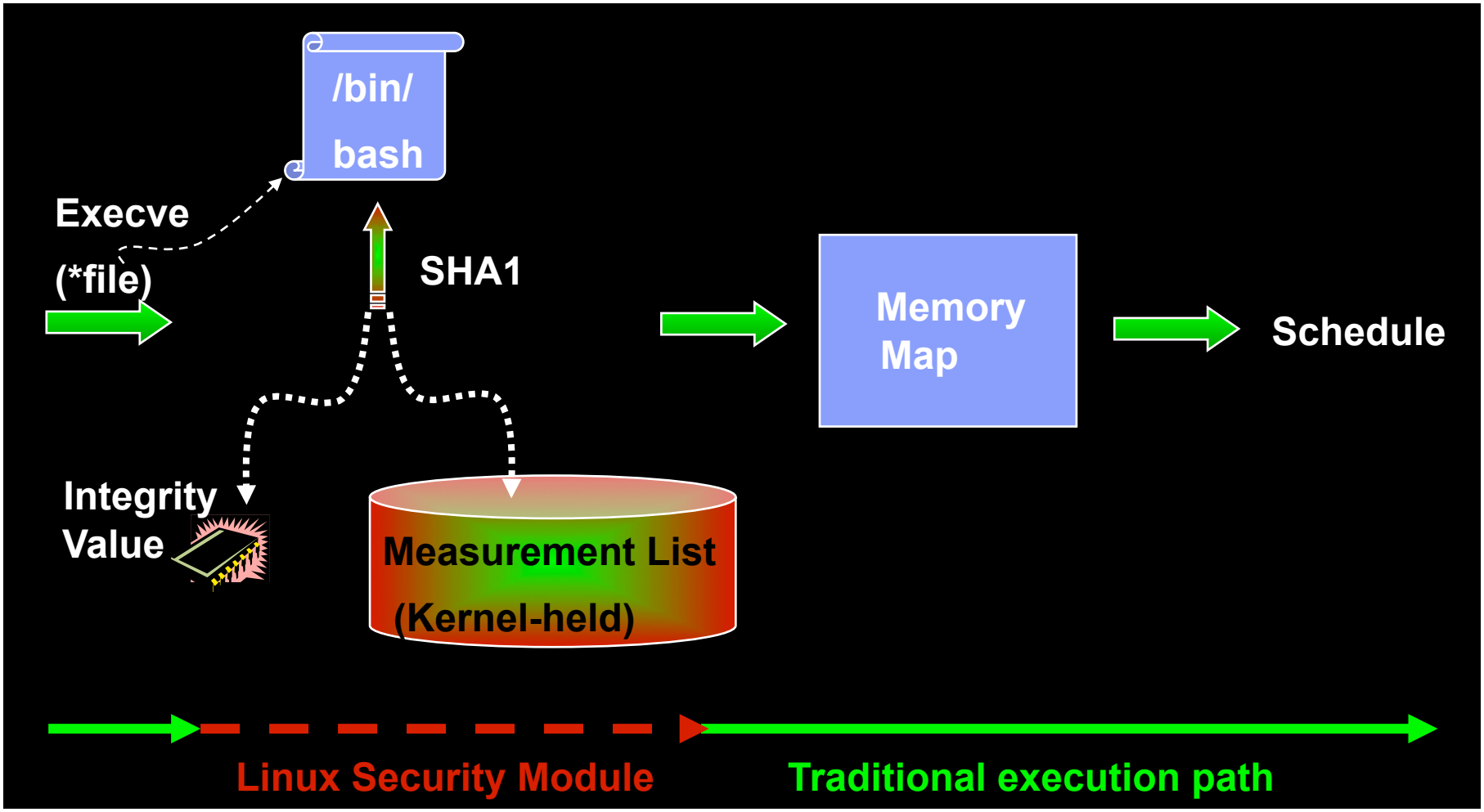


IMA Implementation

- Place hooks throughout Linux kernel
  - Later added more general LIM hooks
- Extend TPM PCR at file load-time
  - PCR = SHA1(File || PCR)
- Extend kernel-stored measurement list
  - List of SHA1 hashes taken by kernel
  - Including those requested by user space applications
- Generate attestation using TPM hardware
  - $S(K^{-}_{TPM}, PCR+nonce)$

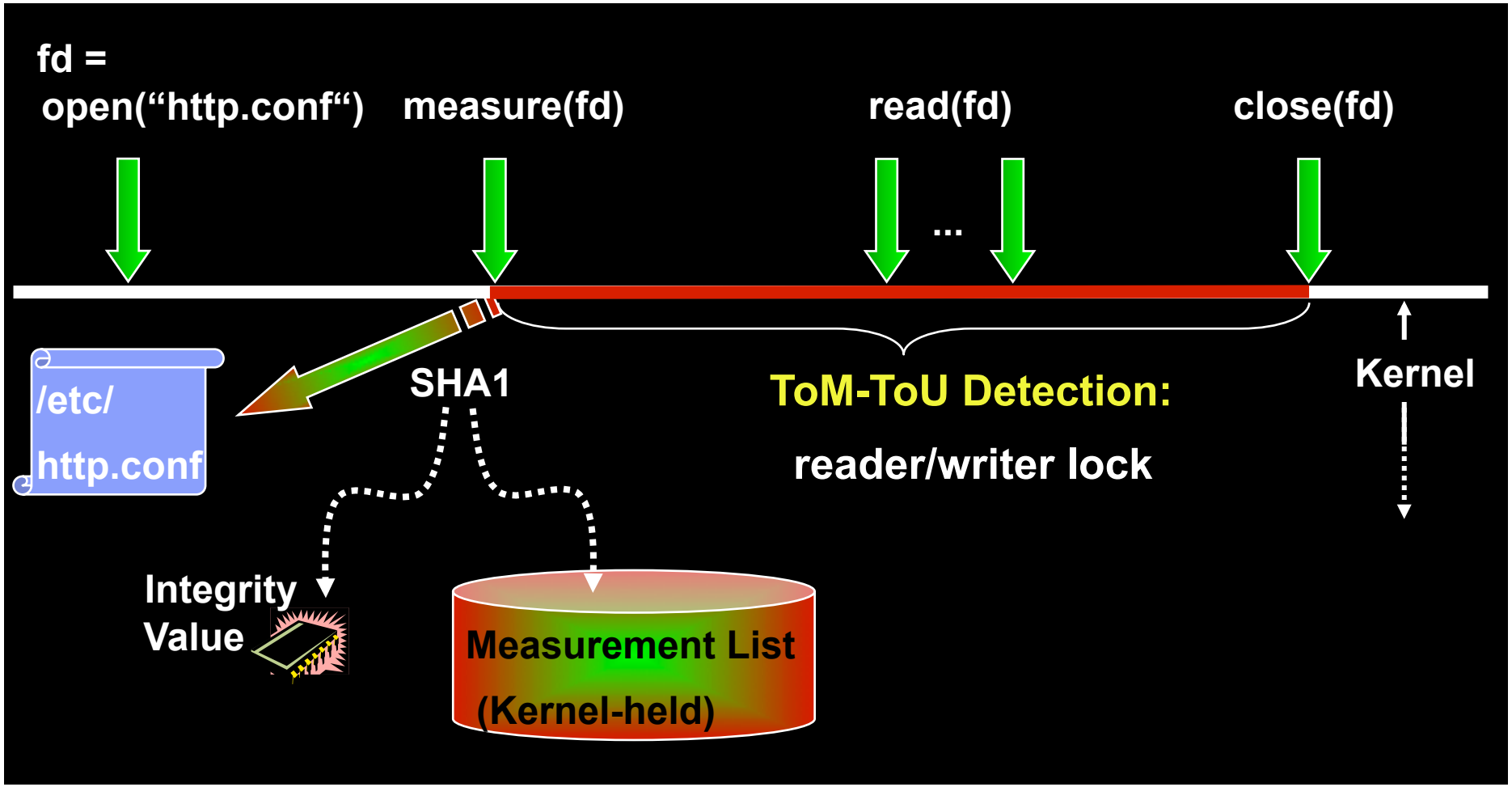
Linux Integrity Measurement

- Linux Kernel Measurements



Linux Integrity Measurement

- Application Measurements



TPM Concerns

- The TPM device was not embraced by all
- It could be useful for Digital Rights Management
  - Restrict the delivery of content to expected systems
- It could be used to lock out certain systems (non-Microsoft)
  - Restrict delivery of content to commercial systems (non-Linux)
- Screeds were written and produced
  - http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html
- And rebutted
  - http://www.linuxjournal.com/article/6633

Trust the OS?

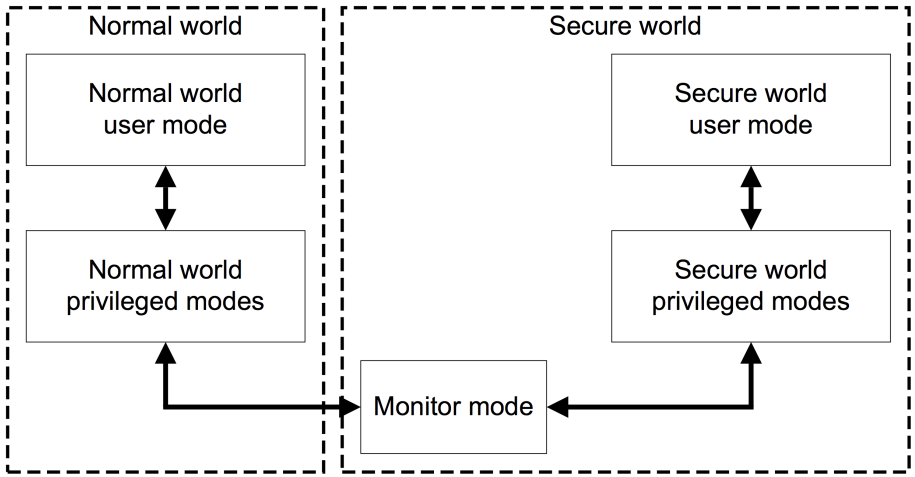
- Do we have to trust the operating system?
  - Kernel rootkits have been a serious threat for a while – now even for smartphone operating systems (e.g., Android)
  - CVE-2011-1823: an integer overflow bug in a daemon process on Android 3.0 enables an adversary to gain root privilege and install a kernel rootkit
- IMA trusts the OS
  - Remember Proxos and Overshadow?
- Can hardware help?

Goals

- Restrict kernel to only execute approved code
- Monitor kernel operations to enforce security
  - Even when the kernel has been compromised

Background: TrustZone

- Resources are partitioned into two distinct worlds
  - physical memory, interrupts, peripherals, etc.
- Each world has its autonomy over its own resources
- Secure world can access normal world resources, but *not* vice versa
- Run in time-sliced fashion



#1 Type Attacks

PENNSTATE 1855

- Goal: Enable write over code or execute over data
  - Disable the W⊕X protection
  - Change to a different set of page tables that are under attacker's control
  - Modify page table entries in place
  - Enable execution over code pages in the user space
  - Disable the MMU

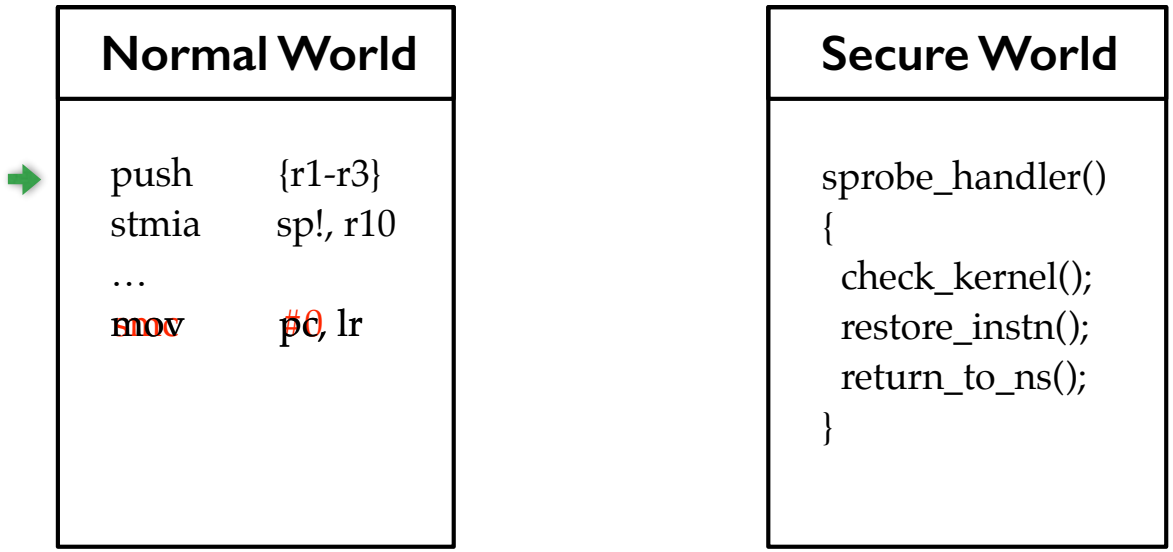
#2 Type Attacks

PENNSTATE 1855

- Goal: (1) Remap a code page to a physical frame containing data or (2) map a data page to a physical frame containing code
  - Change to a different set of page tables that are under attacker's control
  - Modify page table entries in place

SPROBE Mechanism

- We need an instrument mechanism that enables the secure world to be notified upon events of its choice in the normal world



SPROBE Placement

- Recall the specific attacks
  - Change to a different set of page tables that are under attacker's control
    - instrument all instructions that can be potentially used to switch the page table root
  - Modify page table entries in place
    - write-protect the whole page tables and instrument the first instruction in page fault handler

Evaluation

- Environment setup
  - Linux 2.6.38 in the normal world
  - Fast Models 8.1 for emulation
- Types of SPROBES
  - Type #1: 6 SPROBES for enforcing W⊕X protection
  - Type #2: 4 SPROBES for monitoring page table root
  - Type #3: 1 SPROBE for monitoring page table configuration
  - Type #4: 1 SPROBE for monitoring page table entries

Run One Trusted Process?

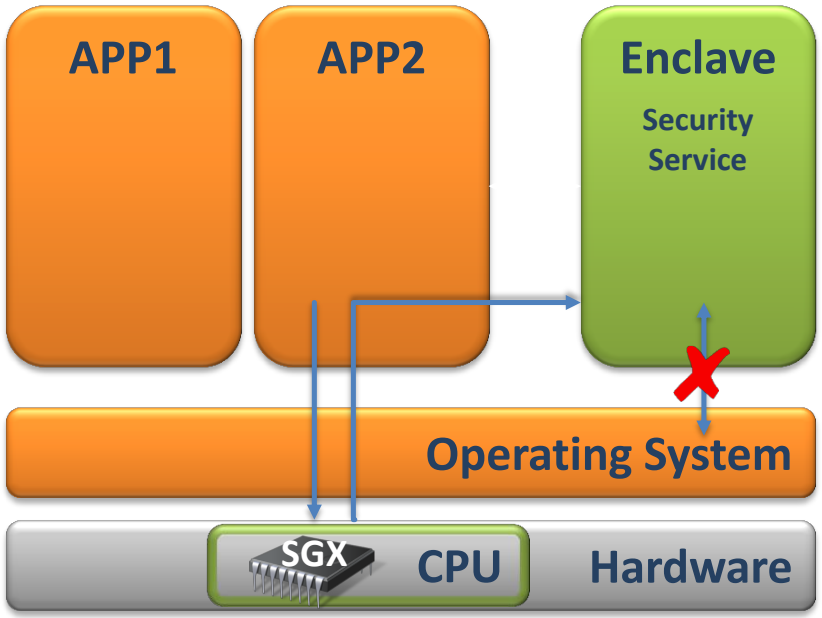
- What if we only want to run one high-integrity user-space process?
  - IMA won't help (still have to trust the OS)
  - TZ won't help (only manages OS)
- Interim solution: Dynamic Root-of-Trust Measurement (DTRM)
  - Reboot computer into new root of trust
  - Can run completely new environment
    - Flicker from CMU [EuroSys 2008]
  - But, reboot is expensive … what else can be done?

Intel Software Guard Ext

Intel® Software Guard Extensions (SGX)

[McKeen et al, Hoekstra et al., Anati et al., HASP'13]

- **Security critical code isolated in enclave**
- **Only CPU is trusted**
  - Transparent memory encryption
  - 18 new instructions
- **Enclaves cannot harm the system**
  - Only unprivileged code (CPU ring3)
  - Memory protection
- **Designed for Multi-Core systems**
  - Multi-threaded execution of enclaves
  - Parallel execution of enclaves and untrusted code
  - Enclaves are interruptible
- **Programming Reference available**



SYSTEM SECURITY LAB

PENNSTATE 1855

SGX Enclaves

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

RAM: Random Access Memory
OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)

SYSTEM SECURITY LAB

PENNSTATE 1855

SGX Memory Access Control

- **Access control in two direction**
  - From enclaves to “outside“
    - Isolating malicious enclaves
    - Enclaves needs some means to communicate with the outside world, e.g., their “host applications”
  - From “outside“ to enclaves
    - Enclave memory must be protected from
      - Applications
      - Privileged software (OS/VMM)
      - Other enclaves

OS: Operating System
VMM: Virtual Machine Monitor (also known as Hypervisor)

SGX MAC from enclaves to “outside“

- **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
  - Code fetches from inside an enclave to a linear address outside that enclave will results in a #GP(0) exception

MAC: Memory Access Control
#GP(0): General Protection Fault

PENNSTATE 1855

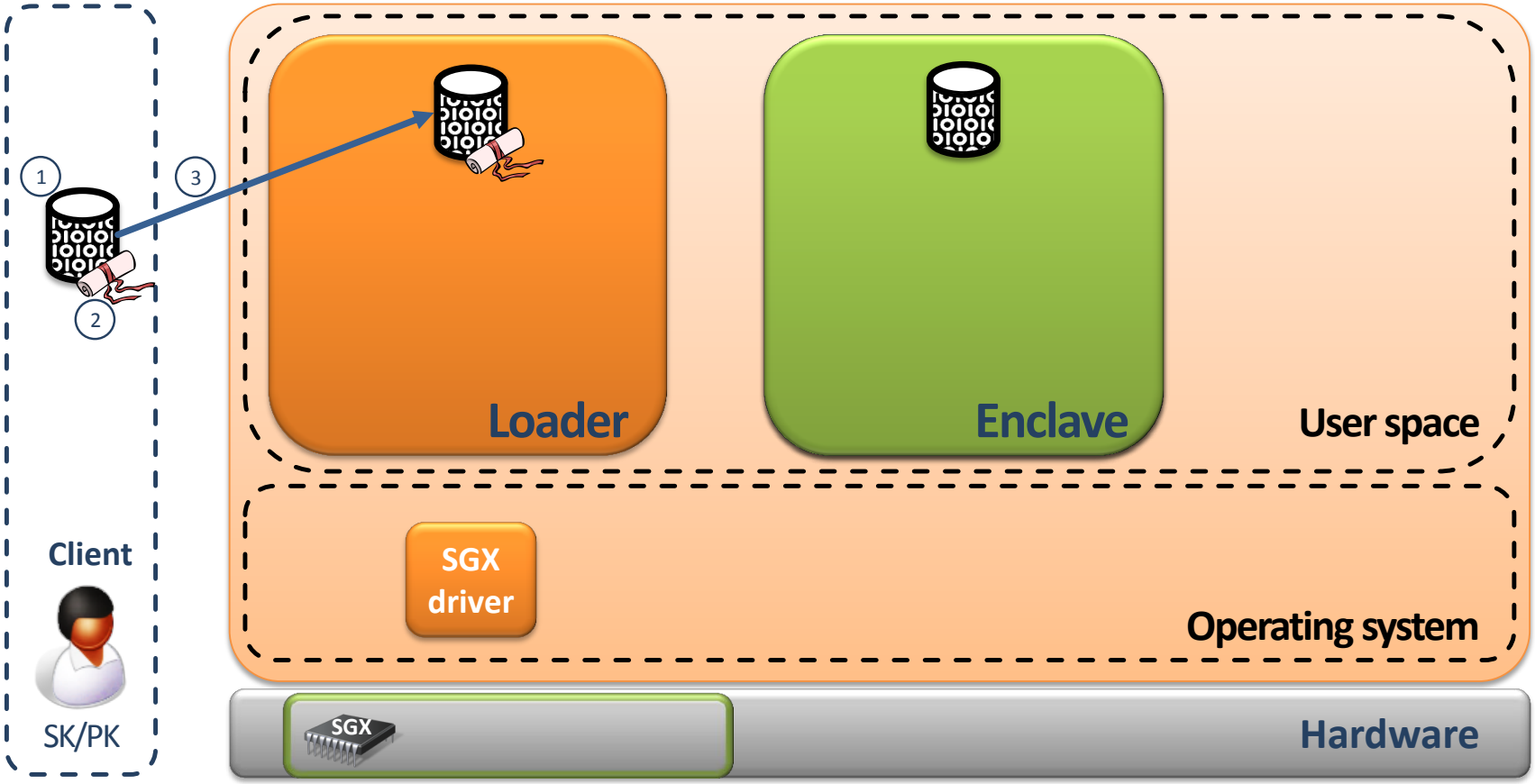
SGX MAC "outside" to enclaves

- **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 #GP(0)

MAC: Memory Access Control
EA: Enclave Access
#GP(0): General Protection Fault

PENNSTATE 1855

SGX – Create Enclave



1. Create App

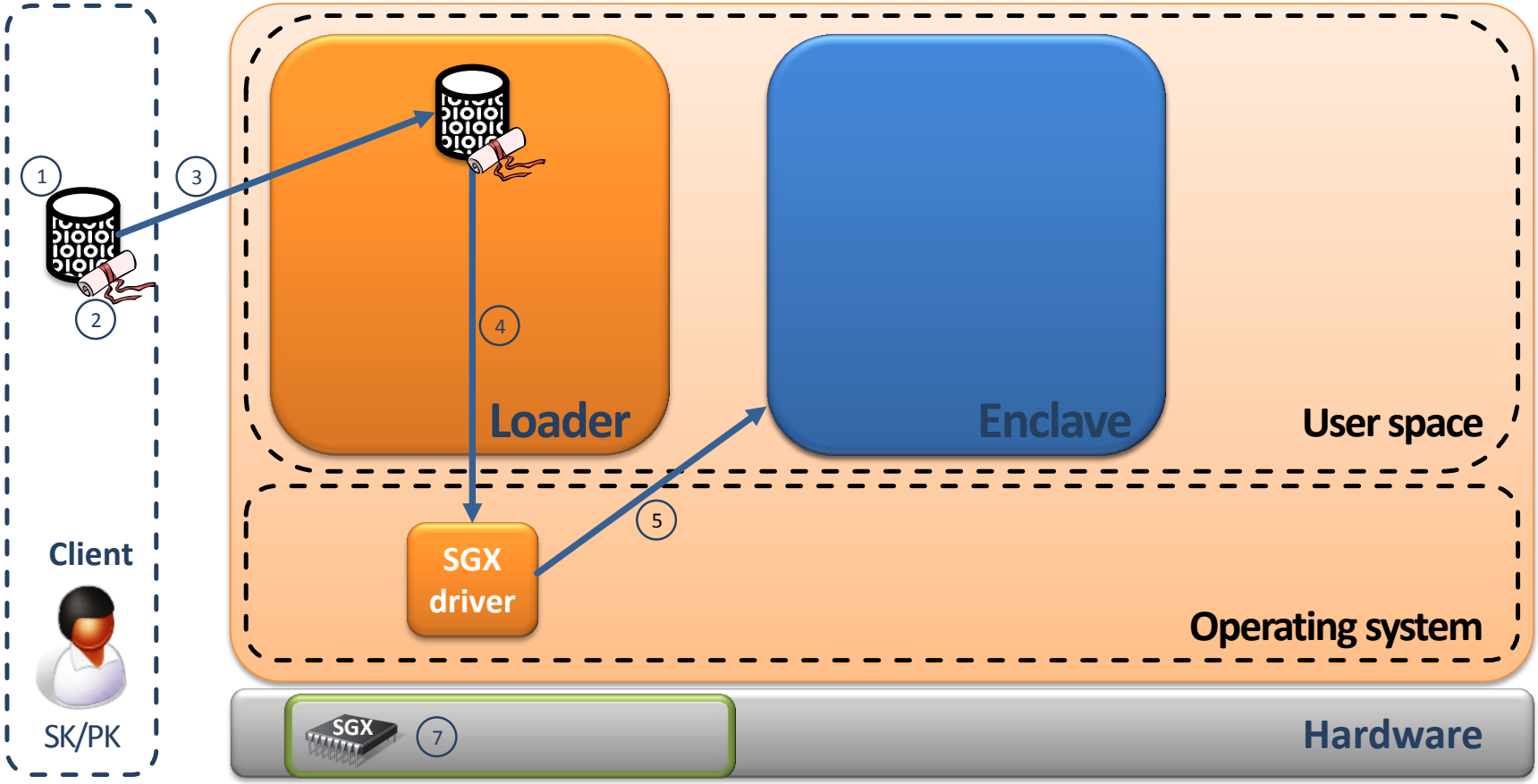
2. Create app certificate (includes HASH(App) and Client PK)

3. Upload App to Loader

Trusted Untrusted

PENNSTATE 1855

SGX – Create Enclave

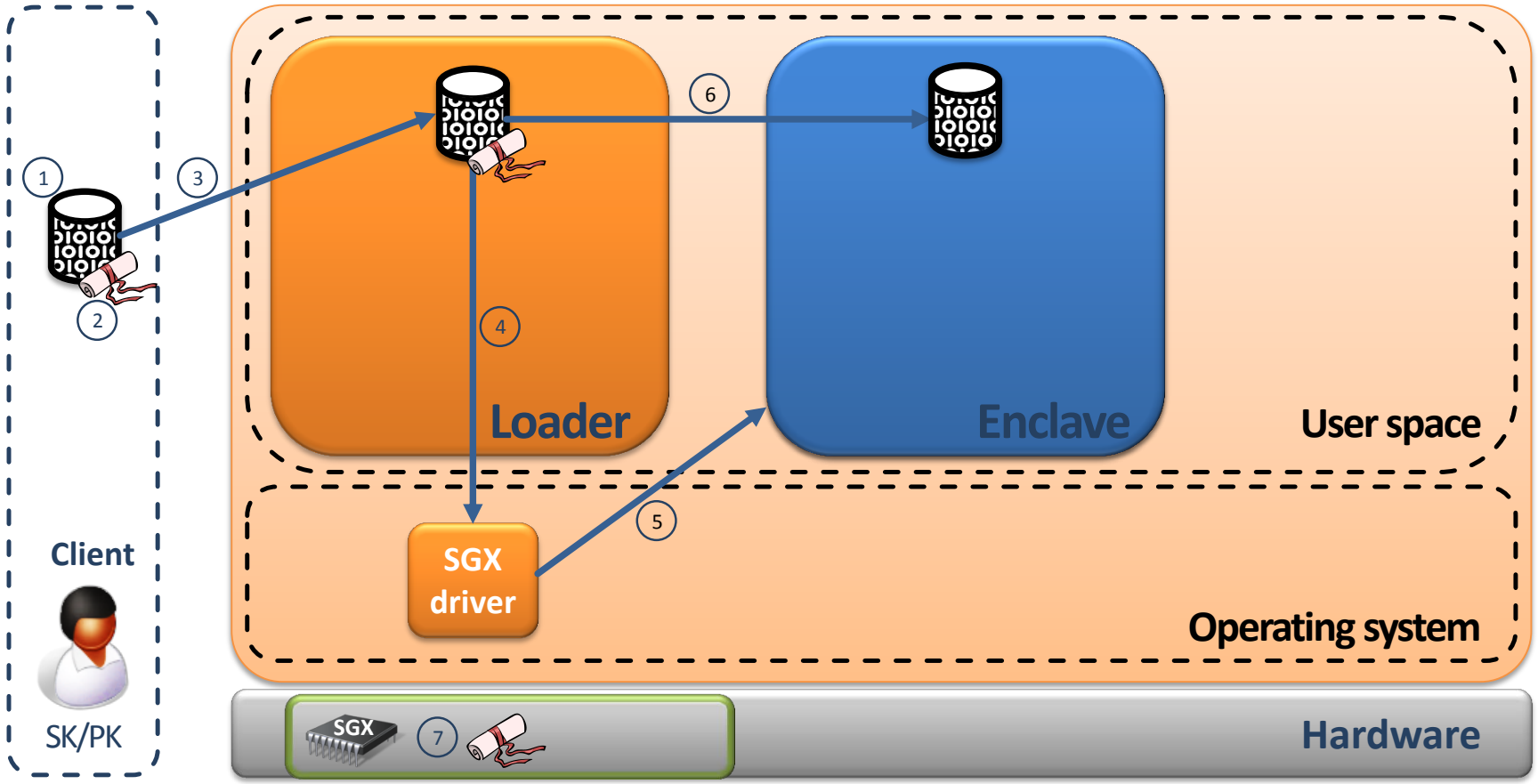


1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages

Trusted Untrusted

PENNSTATE 1855

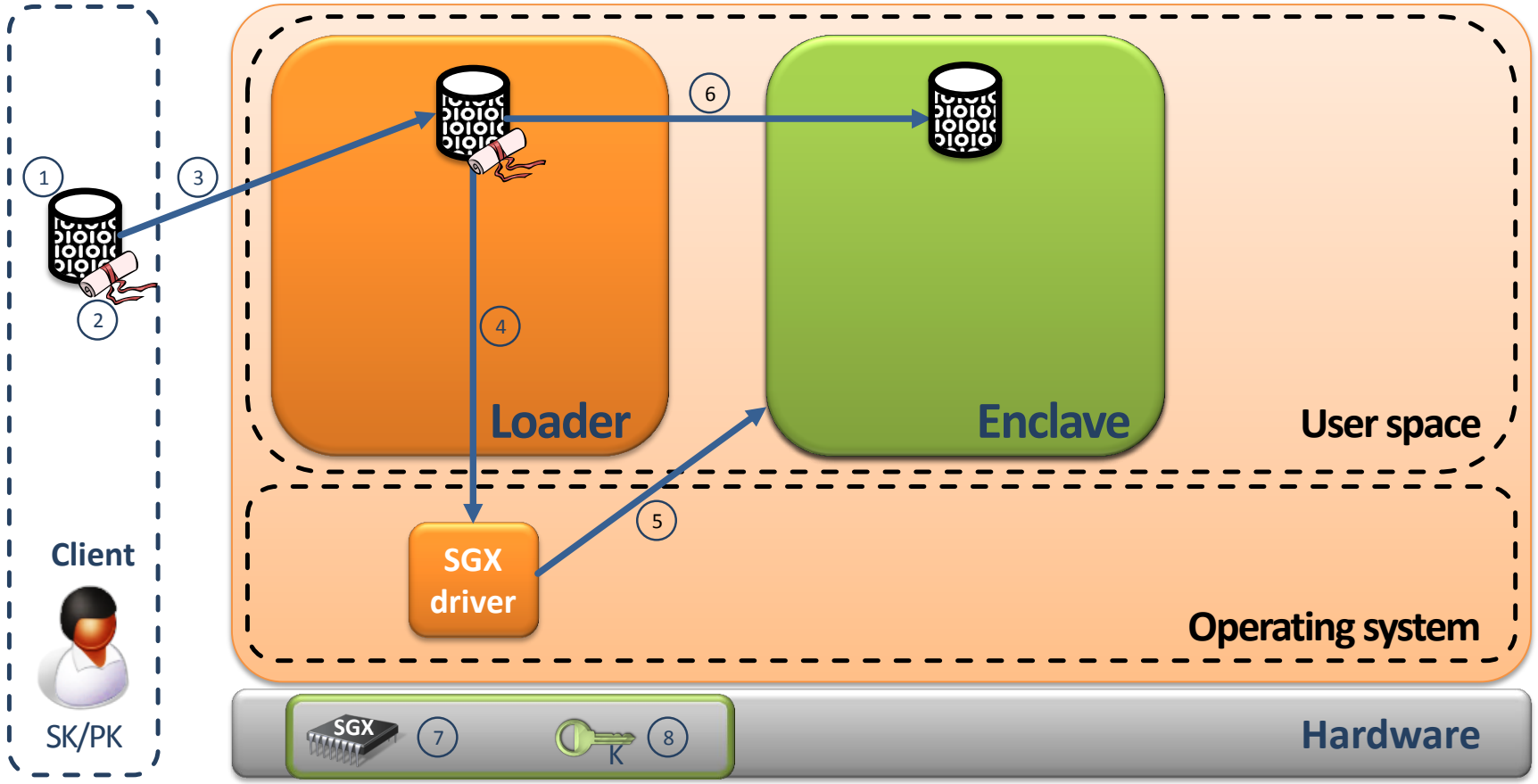
SGX – Create Enclave



1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity

Trusted | Untrusted

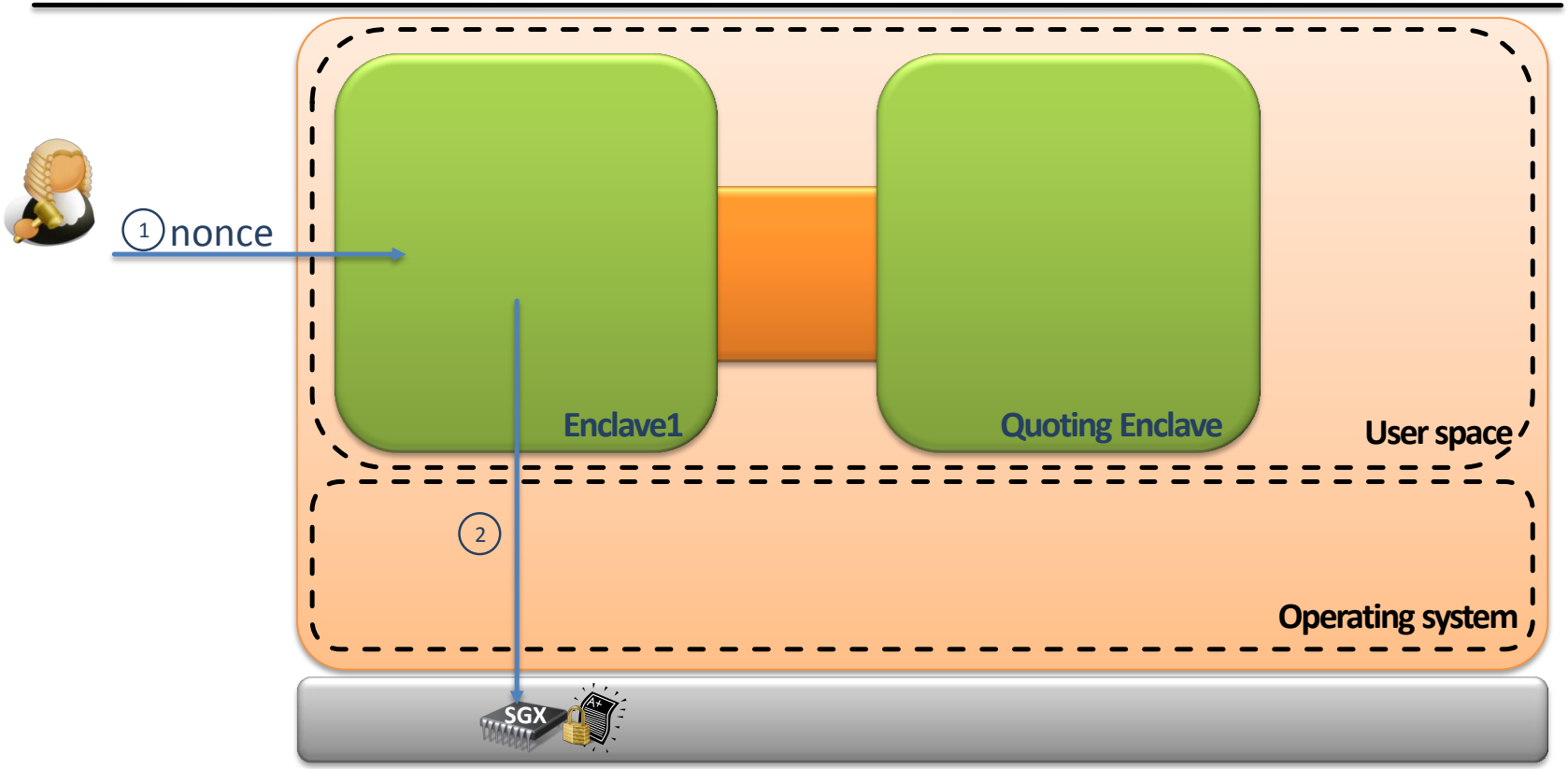
SGX – Create Enclave



1. Create App
2. Create app certificate (includes HASH(App) and Client PK)
3. Upload App to Loader
4. Create enclave
5. Allocate enclave pages
6. Load & Measure App
7. Validate certificate and enclave integrity
8. Generate enclave **K** key
9. Protect enclave

Trusted | Untrusted

SGX – Remote Attestation



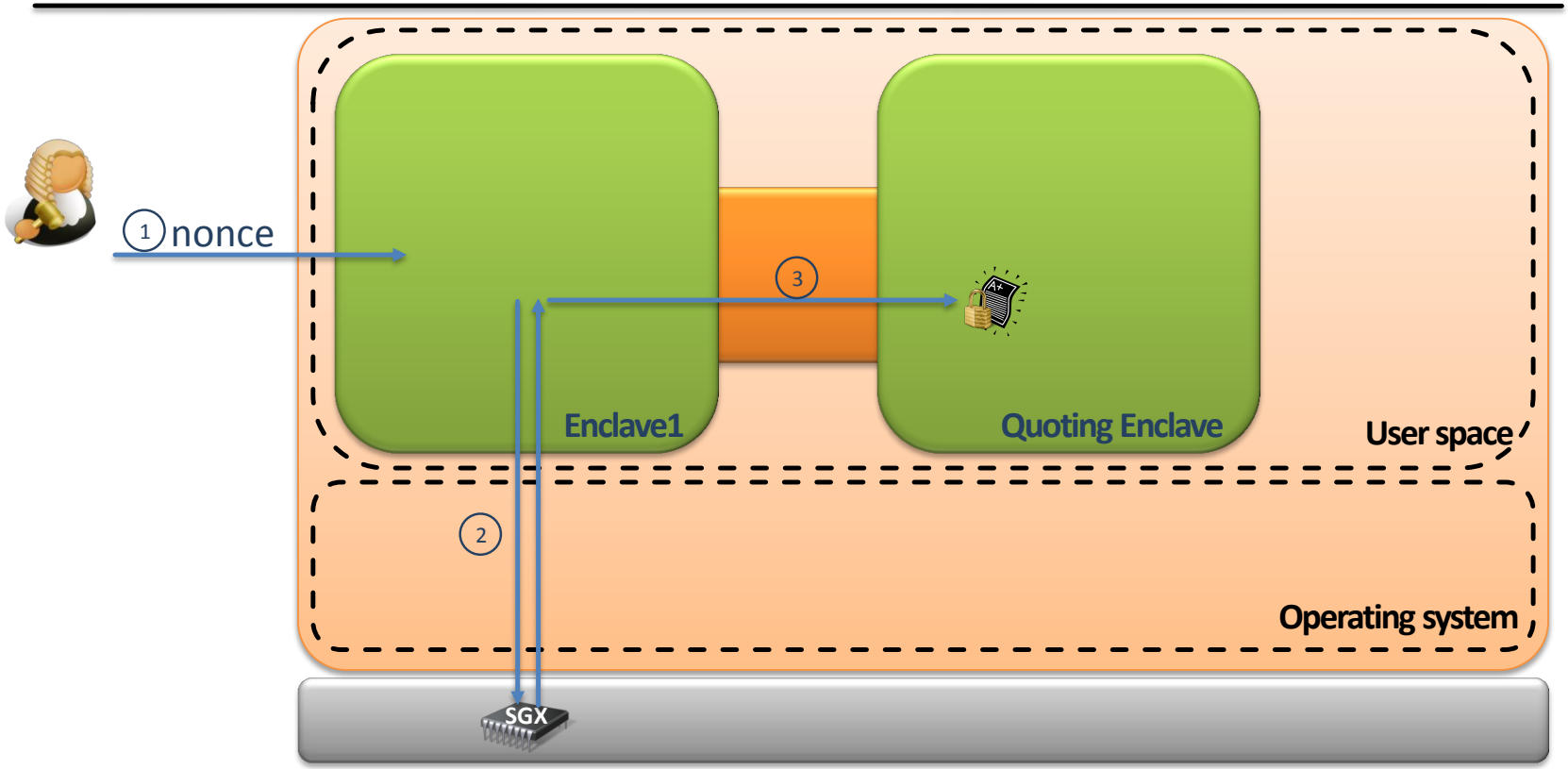
1. Verifier sends nonce

2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)

Trusted | Untrusted

PENNSTATE 1855

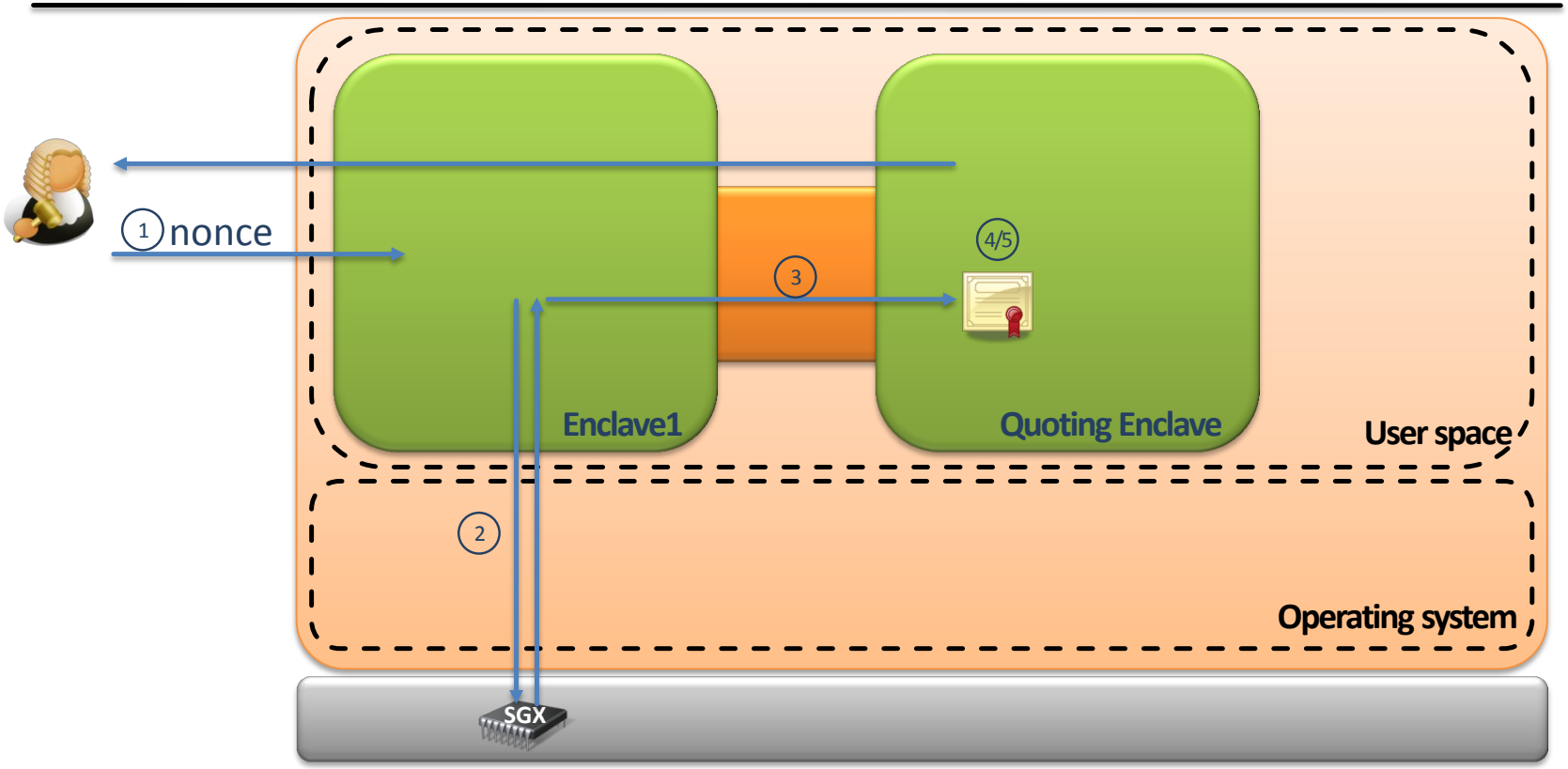
SGX – Remote Attestation



1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave

Trusted Untrusted

SGX – Remote Attestation

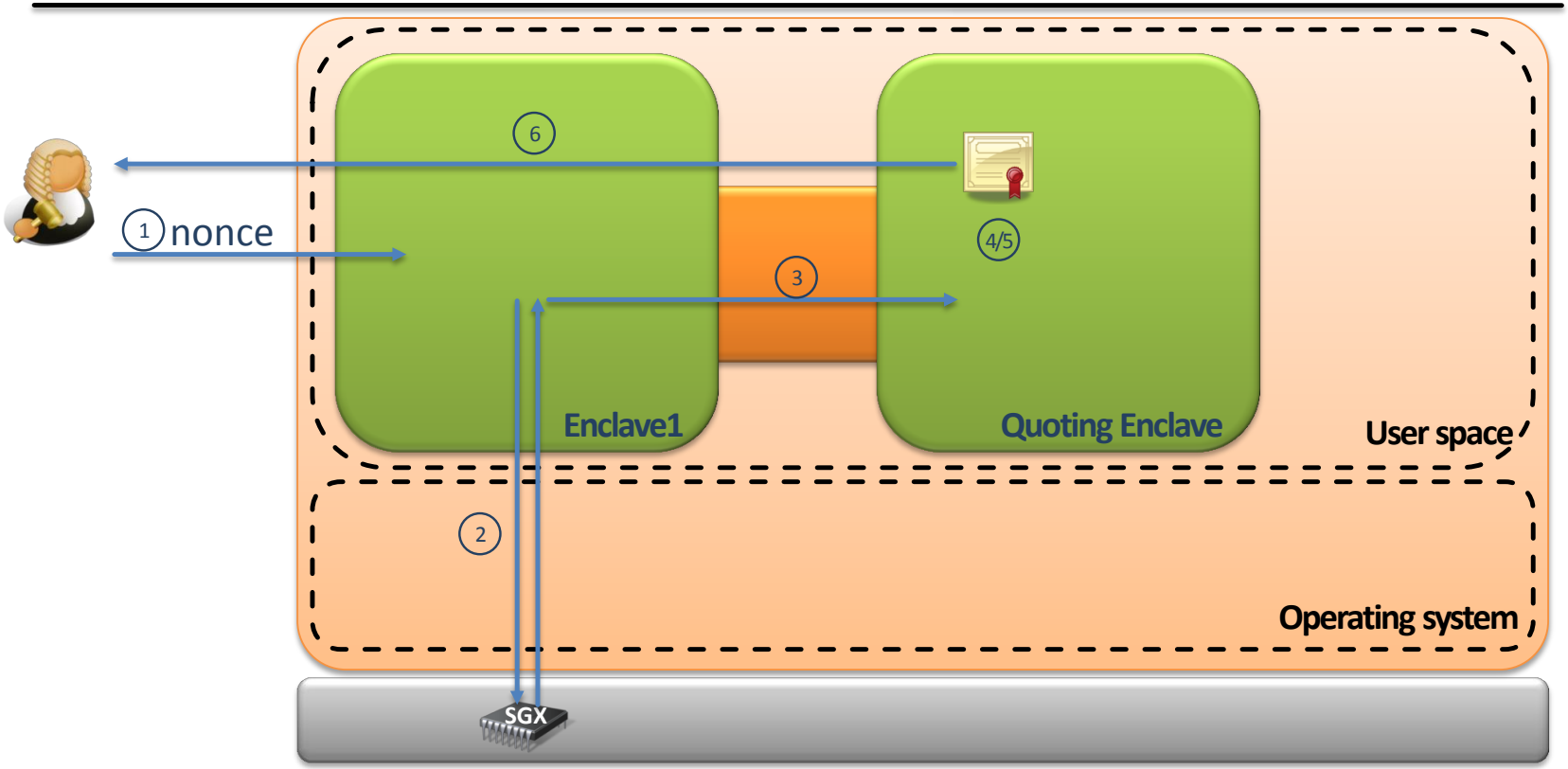


1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave
4. Quoting Enclave verifies Report
5. Signs Report with "Platform Key"
6. Signed Report is send to verifier

Trusted | Untrusted

PENNSTATE 1855

SGX – Remote Attestation



1. Verifier sends nonce
2. Generate Report = (HASH(Enclave1), ID-QuotingEnclave, nonce)
3. Pass Report to Quoting Enclave
4. Quoting Enclave verifies Report
5. Signs Report with "Platform Key"
6. Signed Report is send to verifier

Trusted | Untrusted

Another Problem

- Return-oriented attacks
  - Can hardware help detect those attacks?

Intel Processor Trace

- A new hardware feature that enables efficient recording of control-flow and timing information about software execution (3-5% overhead)
  - Initially available on the Broadwell processor
  - Fully implemented on the Skylake processor
- At each control choice, record a packet in memory
  - Conditional branches
  - Indirect call
  - Returns
- Enough to reconstruct the actual control flow

Intel Processor Trace

- The trace is recorded in data packets
  - Taken Not-Taken (TNT) tracks the direction of conditional branches
  - Target IP (TIP) records the target address of indirect branch, interrupts, and exceptions
  - Flow Update Packets (FUP) provide the source address of asynchronous events such as interrupts and exceptions
  - Time-Stamp Counter (TSC) packets aid in tracking wall-clock time
  - and many more (20 in total …)
- The decoding is complex
  - RET compression – only record if matches call
  - IP compression – only record part of the target address
  - Deferred TIP – move indirect call after some TNT bits