Confidential Computing

Peter Pietzuch
Massachusetts Institute of Technology
Designing Systems Support for Trusted Execution using Intel SGX

Peter Pietzuch

Imperial College London

http://lsds.doc.ic.ac.uk
<prp@imperial.ac.uk>

Microsoft Research Faculty Summit – Redmond, WA, USA – August 2018
Cloud Tenants Must Trust Cloud Providers

Tenants expose applications & data to cloud providers
Tenants View Clouds as Untrusted Black Boxes

Cloud providers operate complex cloud stacks

Threats from privileged code attacks
– Security vulnerabilities exist:
  - Xen hypervisor: 184 (2012-16)
  - Linux kernel: 721 (2012-16)
– Many attacks exploit vulnerabilities
  - Control-flow hijacking, code injection attacks, return-oriented programming

Threats from insider attacks
– Administrators, staff with physical access

OS kernel
Hypervisor
Firmware
Cloud platform
Staff
…

Linux: ~20M LOC
KVM: ~13M LOC
OpenStack: ~2M LOC
Existing Cloud Security Model Doesn’t Help…

Cloud providers do not trust cloud tenants

Cloud security mechanisms focus on protecting privileged system software (OS, hypervisor)
  – e.g. tenant isolation using VMs

trusted execution gives control over security to cloud tenants
Trusted Execution with Intel SGX

Introduces concept of **userspace enclaves**
- Isolated memory regions for code and data

Enclave memory **encrypted & integrity-protected**
- Automatically performed by the hardware

Enclave memory only accessible by enclave code
- Protected from privileged code (OS, hypervisor)
Promise: Protect Cloud Tenants using Enclaves

Enclave retains flexibility to run arbitrary cloud applications
- Unlike approaches based on software encryption, homomorphic encryption, secure multi-party computation, …

如何支持云应用在SGX enclave中运行？
Design Space: Systems Support for SGX Enclaves

(a) Library OS: SGX-LKL, Haven

(b) System call interface: SCONE [OSDI'16]

(c) Partitioned application: Glamdring [USENIX ATC'17], Intel SGX SDK

Application
Standard C library

Library OS
Runtime support

Hypercall interface
Loader/Starter
Host OS (Linux)

in-app function calls
Trusted application functions
Shim layer

Untrusted application functions
Host OS (Linux)

Partitioned application:
Glamdring [USENIX ATC'17], Intel SGX SDK

Policy exit interface
Loader/Starter
Host OS (Linux)

in-app function calls
Trusted application functions
Shim layer

Untrusted application functions
Host OS (Linux)
Challenge: Enclave Transitions are Expensive

- Entering/exiting an enclave comes with performance cost
  - CPU performs checks and transparently saves/restores state

- Must exit enclave to perform system calls
  - System calls invoke OS kernel, which is untrusted
Enclaves: Performance Cost of System Calls

10x reduction in system call rate
Idea: Reduce Number of Enclave Transitions

1. Provide user-level threading inside enclaves
   – Enclave threads can remain inside enclave
   – Thread scheduler switches between user-level threads

2. Provide OS functionality inside enclaves to avoid transitions for systems calls
   – Thread synchronisation
   – Memory management
   – File systems
   – Networking
   – Signal handling

User process

Enclave

Application code

OS kernel
SGX-LKL: System Support for Enclaves

SGX-LKL runs **unmodified Linux applications** in SGX enclaves
- Applications and dependencies provided via encrypted disk image

Linux kernel functionality available inside enclaves
- Based on **Linux Kernel Library (LKL):** Architecture-specific port of Linux kernel
  ([github.com/lkl](https://github.com/lkl))
  - Trusted file system and network stacks

1. **User-level threading**
   - In-enclave synchronisation primitives (**futex** implementation)

2. **Asynchronous system calls**
   - Similar to **SCONE** [OSDI'16]

3. **Custom memory allocator**
   - Integration between kernel and enclave memory allocator
SGX-LKL Architecture

**Untrusted host**
- SGX-LKL loader
  - Host system call handler
  - Signal handler
  - Initialisation/loading

**LKL**
- Native operations
- Network/block device operations
- System call Interface
- Standard C library
- C wrappers for non-LKL syscalls (memory, threading, synchronisation, time)
- Memory management
- User-level lthread scheduler

**Libs**
- C wrappers for LKL syscalls
- In-enclave signal handler
- Linker/loader

**Enclave**
- libsgxlkl.so
- Mounts disk image/loads application & dependencies
- Creates enclave

**Unmodified application**
- Runtime/interpreter
- Library

**Ext4 disk image**
- Mounts disk image/loads application & dependencies

**SGX-LKL Architecture**

- System call Interface
- Network/block device operations
- Native operations
- Standard C library
- Asynchronous system call request/response queues

Peter Pietzuch - Imperial College London
SGX-LKL provides trusted Linux file system support
  – Encryption/integrity protection performed at disk block level

Uses standard Linux device mapper API for disks
  – dm-crypt for encryption
  – dm-verity for integrity protection

SGX-LKL provides trusted network stack
  – Enclave applications can use arbitrary network protocols (TCP, UDP, ...) securely

Uses TUN/TAP interface to send/receive packets via host OS kernel
  – Performs layer-2/3 encryption inside enclave (e.g. IPSec)
Workload-independent host calls

- `fcntl`
- `ioctl`
- `lseek`
- `close`
- `mmap`
- `mremap`
- `munmap`
- `exit`
- `gettid`
- `pipe`

Workload-dependent host calls

- `read`
- `readv`
- `pread64`
- `preadv`
- `write`
- `writev`
- `pwrite64`
- `pwritev`
- `fdatasync`
- `mprotect`
- `msync`
- `sigaction`
- `sigpending`
- `sigprocmask`
- `sigsuspend`
- `sigtimedwait`
- `tkill`
- `clock_gettime`
- `clock_getres`
- `gettid`

Host interface is **side channel**

- Workload-dependent host calls may leak sensitive data

» **Ongoing work:** Can we make the SGX-LKL host interface oblivious?
SGX-LKL: Supported Applications

Launches **Linux binaries** from **Alpine Linux** inside enclaves

– Nginx
– Redis
– Memcached
– Python, Perl
– …

Support for **managed language runtimes**

– Oracle Hotspot **JVM** (Java/Scala): OpenJDK
– V8 **JavaScript** Engine

Try it on GitHub:
www.github.com/lsds/sgx-lkl
Design Space: Systems Support for SGX Enclaves

(a) Library OS: SGX-LKL, Haven

(b) System call interface: SCONE [OSDI’16]

(c) Partitioned application: Glamdring [USENIX ATC’17], Intel SGX SDK
How to Protect Large Cloud Applications?

Consider deploying **Apache Spark** inside an SGX enclave

```
def main(args: Array[String]) {
  new SparkContext(new SparkConf().
    .textFile(args(0))
    .flatMap(line => {line.split(" ")})
    .map(word => {(word, 1)})
    .reduceByKey{case (x, y) => x + y}
    .saveAsTextFile(args(1))
}
```

Attacker's can exploit vulnerabilities inside enclave code

- Only data and processing code is sensitive
Partition Cloud Applications to Minimise TCB

Many examples of manual partitioning of applications by developers

Can we automatically determine the minimum functionality to run inside an enclave?
Glamdring: SGX Partitioning Framework

[USENIX ATC'17]

1. Annotations, Application code

2. Static analysis
   - Forward analysis
   - Backward analysis

3. Enclave boundary relocation
   - Partition specification
   - Source-source transformation
   - Invariants
   - Runtime invariant checks

4. Enclave code
   - Untrusted application code

Compiler-based framework for partitioning C applications
#pragma glamdring sensitive source(data)

```c
void read(char* data) {
  ...
}
```

1. Developers Annotate Security-Sensitive Data
2. Static Analysis to Identify Sensitive Code

Program dependence graph

To ensure **data confidentiality**: forward dataflow analysis
To ensure **data integrity**: backward dataflow analysis
3. Producing Partitioned SGX Application

Static analysis

- Forward analysis
- Backward analysis

Partition specification

Enclave boundary relocation

Source-source transformation

Source-to-source compiler based on LLVM
Add runtime checks that enforce invariants on program state used by static analysis

4. Upholding Static Analysis Invariants

- Annotations
- Application code

Static analysis
- Forward analysis
- Backward analysis

Partition specification

Enclave boundary relocation

Source-source transformation
- Invariants
- Runtime invariant checks

Enclave code

Untrusted application code
**Evaluation: How Much Code Inside Enclave?**

<table>
<thead>
<tr>
<th>Application</th>
<th>Total code size (LOCs)</th>
<th>Enclave size (LOCs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>31,000</td>
<td>12,000 (40%)</td>
</tr>
<tr>
<td>DigitalBitbox</td>
<td>23,000</td>
<td>8,000 (38%)</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>176,000</td>
<td>38,000 (22%)</td>
</tr>
</tbody>
</table>

- Enclave contains less than 40% of application code.
Summary: Securing Cloud Apps using Intel SGX

**Trusted execution** promises to enhance security for cloud tenants
- But requires new systems support and developer tools

Tenants want to run **unmodified existing applications** with SGX
- **SGX-LKL** provides user-level threading, file system and networking support

Developers require **automated tooling** when using enclaves
- **Glamdring** semi-automatically partitions applications for Intel SGX

Thank You — Any Questions?

Peter Pietzuch
https://lsds.doc.ic.ac.uk — prp@imperial.ac.uk
Backup Slides
What Can Cloud Tenants Do Today?

Use encrypted communication channels (TLS)?
- Protects data in transit but not once in cloud environment

Encrypt data before sending to cloud environment?
- Only works for some cloud services (e.g. storage)
- Limits functionality of cloud services

Use homomorphic encryption?
- Large performance overhead and limited applicability

What about integrity?
- Challenging to ensure that computation was executed faithfully
Linux Kernel Library (LKL)

Architecture-specific port of mainline Linux (github.com/lkl)
- Good maintainability
- Mature implementation

LKL Architecture
- Follows Linux no MMU architecture
- Full filesystem support
- Full network stack available
Memory Layout in SGX-LKL

Need correct initialisation of LKL & libc
- Relocation/linking/loading

Support for position dependent code
- Leaves gap for application load address (0x400000+)
- Map enclave from 0x0 page due to SGX restrictions
Support for Dynamic Linking

Dynamic Linker-Enabled Control Flow

**Starter**
(initialise SGX, allocate memory, load enclave code and start system call threads)

**Application code**
(dynamically linked against musl-libc)

**stage 1 & 2 dynamic linker**
(perform linker-internal relocations, to allow function calls to work correctly)

**LKL boot process**
(initialises device drivers and internal process table, system calls possible after this point)

**stage 3 dynamic linker**
(load application binary & necessary libraries from disk image and perform relocations)

**musl-libc starter**
(initialise enclave memory & libc, and host-ocall buffer)
Support for Disk Encryption

Initial enclave code and data measured by CPU
But must ensure confidentiality/integrity of disk image
  – Loaded binary and dependent libraries must be trustworthy

Idea: Support encryption/integrity protection at block level

Uses standard Linux device mapper API
  – \texttt{dm-crypt} for encryption
  – \texttt{dm-verity} for integrity protection
    – Merkle tree for disk block verification
    – Leaf nodes contain hashes of disk blocks

Peter Pietzuch - Imperial College London
Support for Linux Networking

Use in-enclave trusted Linux network stack

TUN/TAP interface to send/receive packets via host kernel

- Layer-2/3 encryption possible in-enclave 
  (e.g. **IPSec**, **VPNs**)
- Support arbitrary network protocols with encryption

Used by Google in production for app-level proxies:

- **Application**: calls recv
- **Libc**
- **LKL**: invokes lkl_syscall
- **LKL virtio device driver**
- **Enclave ocall code**
- **Syscall thread**: puts syscall on queue
  - syscall thread pops syscall off queue and invokes read
  - **Host kernel**: reads packet from TAP interface buffer

Peter Pietzuch - Imperial College London
SGX-LKL: Debugging Support

GDB plugin
- Breakpoints, watchpoints, stack traces
- Dynamically loads required symbols
- Supports software simulation and hardware SGX mode

Perf support
- Passes required enclave symbols to perf

Peter Pietzuch - Imperial College London
## Comparison: SGX-LKL vs Graphene-SGX

<table>
<thead>
<tr>
<th>Feature</th>
<th>SGX-LKL</th>
<th>Graphene-SGX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Library OS implementation</strong></td>
<td>Linux Kernel Library (LKL) (github.com/lkl)</td>
<td>Custom implementation</td>
</tr>
<tr>
<td></td>
<td>Arch-specific fork of Linux kernel</td>
<td></td>
</tr>
<tr>
<td><strong>Process model</strong></td>
<td>Single process</td>
<td>Multi-process (fork(), IPC support)</td>
</tr>
<tr>
<td><strong>Standard C library support</strong></td>
<td>musl libc (<a href="http://www.musl-libc.org">www.musl-libc.org</a>)</td>
<td>glibc</td>
</tr>
<tr>
<td><strong>Support for unmodified binaries</strong></td>
<td>✓ (from Alpine Linux)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Application packaging</strong></td>
<td>Encrypted block device image</td>
<td>Encrypted files on host FS + manifest file</td>
</tr>
<tr>
<td><strong>File I/O support</strong></td>
<td>Complete Linux VFS impl. in enclave</td>
<td>Relies on host OS FS impl.</td>
</tr>
<tr>
<td></td>
<td>Support for arbitrary Linux FSs (Ext4, btrfs, xfs)</td>
<td>Support for host FS only</td>
</tr>
<tr>
<td><strong>Networking I/O support</strong></td>
<td>Complete Linux network stack in enclave</td>
<td>Relies on host OS network stack</td>
</tr>
<tr>
<td></td>
<td>Support for arbitrary network protocols</td>
<td>UNIX socket support only</td>
</tr>
<tr>
<td></td>
<td>Layer 2/3 encryption</td>
<td>Layer 7/4 encryption</td>
</tr>
<tr>
<td><strong>Threading model</strong></td>
<td>User-level (N:M) threading</td>
<td>Kernel-level (1:1) threading</td>
</tr>
<tr>
<td><strong>Synchronisation support</strong></td>
<td>Enclave futex implementation</td>
<td>Relies on host OS futex impl.</td>
</tr>
<tr>
<td><strong>System call support</strong></td>
<td>Asynchronous system call invocations</td>
<td>Synchronous system call invocations</td>
</tr>
<tr>
<td></td>
<td>(No enclave transitions)</td>
<td>(Requires enclave transitions)</td>
</tr>
<tr>
<td><strong>Enclave shielding</strong></td>
<td>Relies on Linux kernel impl. for shielding</td>
<td>Custom shield implementation</td>
</tr>
<tr>
<td></td>
<td>(e.g. block device encryption, IPSec etc)</td>
<td></td>
</tr>
<tr>
<td><strong>Support for enclave signal handling</strong></td>
<td>✓ (partial)</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Other enclave system support</strong></td>
<td>Anything provided by Linux kernel (!)</td>
<td>X</td>
</tr>
</tbody>
</table>

Peter Pietzuch - Imperial College London
**SGX-LKL Performance: Java**

**DaCapo** benchmark results for JVM with SGX-LKL vs. non-SGX execution

- Intel Xeon E3-1280 v5 at 3.70 GHz with 64 GB RAM

Performance overhead for large enclaves due to SGX memory paging
Few enclave transitions due to asynchronous system call interface
SGX-LKL Performance: JavaScript

Octane benchmark results for node.js with SGX-LKL vs. non-SGX execution

Competitive overhead for JavaScript
Workloads mostly compute-heavy
Security Threats in Data Science

External attacker

VM

OS

Data science job

Malicious insider

Hypervisor

Hardware

Other VM

Malicious tenant
Secure Machine Learning

Secure **machine learning (ML)** killer application
- Resource-intensive thus good use case for cloud usage
- Raw training data comes with security implications

Complex implementations of ML algorithms cannot be adapted for SGX
- Consider Spark MLlib with 100s of algorithms

**Challenges**
- Extremely **data-intensive** domain
- Must support **existing frameworks** (Spark, TensorFlow, MXNet, CNTK, …)
- ML requires **accelerators** support (GPUs, TPUs, …)
- Prevention of **side-channel** attacks
State of the Art

Protect confidentiality and integrity of tasks and input/output data

VC3 [Schuster, S&P 2015]
- Protects MapReduce Hadoop jobs
- Confidentiality/integrity of code/data; correctness/completeness of results
- No support for existing jobs → Re-implement for VC3

Opaque [Zheng, NSDI 2017]
- Hide access patterns of distributed data analytics (Spark SQL)
- Introduces new oblivious relational operators
- Does not support arbitrary/existing Scala Spark jobs
Minimising Enclave Code for Spark

- Reduces trusted computing base (TCB)

How should developers identify the sensitive code?
SGX-Spark Design

- Protects data processing from cloud provider
- Ensures confidentiality & integrity of existing big data jobs
- No modifications for end users
  - Different from Microsoft's VC3
- Low performance overhead

Code outside of enclave only accesses encrypted data

Only **SparkExecutor** inside SGX enclave

Requires two collaborating JVMs
SGX-Spark Architecture

 RDD transformations (map, filter, union, join, …)
 RDD actions (reduce, collect, count, …)
 Task description (including control flow)
 Data en-/decryption

Spark interface
Task placement
Scheduling
Coordination
Input / Output

Peter Pietzuch - Imperial College London
Spark Executor for SGX

Outside JVM

SGX JVM

Iterators used for data access

Dataflow graph inside SGX enclave

Peter Pietzuch - Imperial College London
Example: Smart Grid Data Processing

Pre-Processing MySQL
- Measurements every 15 min
- MySQL prefilters invalid data
- Generation of 3 CSV files:
  - Faults
  - Customer
  - DSM (Service and Maintenance Department)

Scala/Spark
- Reads 3 CSV files
- Filters invalid locations
- Joins Customers with DSMs
- Joins (Customer, DSM) with Faults
- Computes total number of faults and total fault duration for each customer

Output
- Number of faults per consumer with respective service interruption times

<table>
<thead>
<tr>
<th>Customer_id</th>
<th>Customer Name</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Contract number</th>
<th>Medidor Serial</th>
<th>Service area abbreviation</th>
<th>Service area name</th>
<th>Number of faults</th>
<th>Total fault duration in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>5467</td>
<td>USICAP</td>
<td>-23.27197</td>
<td>-51.05277</td>
<td>39633896</td>
<td>31606197</td>
<td>LNA</td>
<td>Londrina</td>
<td>2</td>
<td>1800</td>
</tr>
</tbody>
</table>
Example: Smart Grid Data Processing

[Diagram showing a process flow with input files (in1, in2, in3) being encrypted and sent to an enclave. The encrypted data is then decrypted and processed within the enclave. Outputs (out) are sent back to the outside environment.]
Summary: SGX-LKL and SGX-Spark

**SGX-LKL:** Library OS for complex complex Linux applications
- Based on the standard Linux Kernel
- Trusted file system and network stack
- User-level threading and asynchronous system calls
- Lean host OS interface

**SGX-Spark:** Transparent SGX enclave support for Spark
- Uses SGX-LKL to run Oracle HotSpot JVM
- Designed around SGXSparkExecutor
- Transparent encryption for RDDs
Enclave Transitions

Untrusted code

Create enclave
Call trusted function

... 

Call gate

Trusted code

Trusted function

Execute

Return

Enclave
Intel Software Guard Extensions (SGX)

SGX adds **new enclave execution mode**
- New CPU instructions to manipulate enclaves

**Memory encryption engine (MEE)** protects enclave memory
- Current enclave sizes restricted to 128 MB

Support for **remote attestation**
- Permits clients to verify that they are interacting with genuine SGX enclave

Intel SGX SDK for Windows & Linux

SGX support available in **recent Intel CPUs**
- Skylake (2015), Kaby lake (2016)

SGX will become widely available on commodity CPUs
Trade-Offs When Using Trusted Execution

Untrusted component

Attack surface

Performance overhead

Sensitive code and data

Secure enclave

TCB size
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
Enclave Page Cache (EPC)

Physical memory region protected by MEE
- EPC holds enclave contents

Shared resource between all enclaves running on platform
- Currently only 128 MB
- ~96 MB available to user, rest for metadata

Content encrypted while in DRAM, decrypted when brought to CPU
- Plaintext in CPU caches
SGX Multithreading Support

SGX allows multiple threads to enter same enclave simultaneously
- One thread control structure (TCS) per thread
- Part of enclave, reflected in measurement

TCS limits number of enclave threads
- Upon thread entry TCS is blocked and cannot be used by another thread

Each TCS contains address of entry point
- Prevents jumps into random locations inside of enclave
SGX Paging

SGX provides mechanism to evict EPC page to unprotected memory
- EPC limited in size

Paging performed by OS
- Validated by HW to prevent attacks on address translations
- Metadata (MAC, version) kept within EPC

Accessing evicted page results in page fault
- Page is brought back into EPC by OS
- Hardware verifies integrity of page
- Another page might be evicted if EPC is full
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 calculates enclave measurement hash during enclave construction
Different measurement if enclave modified
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
Local Attestation

Prove identity of A to local enclave B

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 I

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
Remote Attestation II

PE communicates with Intel attestation service
- 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
Summary of SGX Architecture

- Enclave
  - Application runtime
    - Application
  - Runtime

- SGX user runtime
  - Page tables
    - OS structure
  - Hardware
    - Platform

- Instructions
  - EEXIT
  - EGETKEY
  - EEREPORT
  - ENTER
  - ERESUME
  - ECREATE
  - EADD
  - EEXTEND
  - EINIT
  - EBLOCK
  - ETRACK
  - EWB
  - ELD
  - EPA
  - EREMOVE
Thank you!