delivering security guarantees for large-scale data processing on untrusted hosts with a small TCB
security guarantees

trusted

HW based

data

integrity
delivering security guarantees for large-scale data processing on untrusted hosts with a small TCB

1 TB

large-scale data processing
delivering security guarantees for large-scale data processing on untrusted hosts with a small TCB without hardware devices, small code, small interface, and small TCB.
Some use cases

public cloud service provider
Some use cases

public cloud service provider

computational genomics

0.3TB per genome
...more generally...
Model

1. provide state authentication data

2. outsource large state

1. provide state authentication data
Model

1. provide state authentication data
2. outsource large state
3. send request
Model

1. provide state authentication data
2. outsource large state
3. send request
4. execute command

V

S
Model

1. provide state authentication data

2. outsource large state

3. send request

4. execute command

5. receive authenticated reply
Outline

• Goal
• Previous Work
• Our solution: key ideas and overview
• Evaluation
Outline

• Previous Work
Haven
(OSDI’14)

- designed for Intel SGX
- large TCB (due to libOS)
- 10s of new interface calls
+ works with unmodified applications

service

libOS

enclave

interface

picoprocess

host OS

VHD
VC$^3$
(IEEE S&P’15)

- designed for Intel SGX
- specific for Hadoop
+ small TCB
+ data confidentiality
+ can run unmodified Hadoop applications
## A Niche in the State of the Art

<table>
<thead>
<tr>
<th>Small TCB</th>
<th>Large State</th>
<th>Interface Calls</th>
<th>App Specific</th>
<th>Trusted Computing arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Haven</strong> (OSDI’14)</td>
<td>No</td>
<td>Yes</td>
<td>tens</td>
<td>No</td>
</tr>
<tr>
<td><strong>VC3</strong> (S&amp;P’15)</td>
<td>Yes</td>
<td>Yes</td>
<td>R,W</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>XMHF-TrustVisor</strong> (S&amp;P’13,’10)</td>
<td>Yes</td>
<td>No</td>
<td>none (but Minibox has tens)</td>
<td>No</td>
</tr>
<tr>
<td><strong>LaSt\textsuperscript{GT}</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>zero!</td>
<td>No</td>
</tr>
</tbody>
</table>

**MapReduce workloads**

**none** (but Minibox has tens)
Outline

• Our solution: key ideas and overview
Scenario: two execution environments
app’s execution flow
trusted env.
untrusted env.

the service code is running
the service code accesses data in memory
app’s execution flow

access data in block $b_i$

is $b_i$ in memory?

yes

keep going

trusted env.

untrusted env.

when data is available, there are no interruptions
app's execution flow

access data in block $b_i$

is $b_i$ in memory?

keep going

handle page fault

load data

otherwise, the service is interrupted and data memory pages are loaded
app’s execution flow

access data in block $b_i$

is $b_i$ in memory?

---

untrusted env.

handle page fault

load data

validate data

---

trusted env.

data is validated inside trusted environment, independently from service execution
app’s execution flow

access data in block $b_i$

is $b_i$ in memory? no

keep going

trusted env.

untrusted env.

resume

handle page fault

load data

validate data

service is resumed and only if data is valid, service can make progress
in practice
Architecture

Trusted address space
- state handler
- service code

Untrusted address space
- SMM (State map manager)
- other untrusted services

Supervisor

OS

SGX/TPM

Hardware
on TrustVisor, Supervisor is trusted
on SGX, Supervisor is untrusted
LaSt$^\text{GT}$ in 5 steps

• Offline data protection at the source
• State registration
• Data processing
• Lazy loading from memory & disk
• Execution verification
• Offline data protection at the source
Data protection

Hierarchical
- Incremental as data is created

Made for:
- Incremental validation as data is loaded
- Fast verification
- Single hash tree is unsuitable
State Hierarchy

root

directory

master chunks

chunks

blocks

files
- components are loaded separately
- unneeded components not loaded in memory
- state root (1 hash) allows state validation

**State Hierarchy**
• State registration
When the trusted execution environment is created, only the code is available inside.
trusted address space

state handler

SMM
(State map manager)

untrusted address space

service code

OS

Supervisor

grab root from disk

37
- registration is the first execution
- state handler installs root
- root is trusted
- state root is available before service code runs
• Data processing
- service code has view of entire state
- state not readily available: inefficient loading it upfront
- Service code execution begins
- Service accesses data in memory
- Data retrieval is fast if data is already available
page miss on access

- Service code may access data on missing pages

Supervisor

OS
A page fault is triggered when an attempt to access an untrusted address in the untrusted address space results in a page fault. Execution is interrupted, seamlessly waiting to continue. The supervisor then invokes the state handler to service the page fault, which involves transitioning to the trusted address space (SMM) to resolve the fault. This ensures that the execution state is not compromised and allows the system to continue safely.
• Lazy loading from memory & disk
Untrusted address space

Trusted address space

state handler service code state root data

SMM

page fault!

Supervisor OS
Let SMM handle missing data
- SMM loads data from disk
Untrusted address space

Trusted address space

- state handler
- service code
- state root
- data

Untrusted address space

SMM

data

page fault!

validate data

- in TrustVisor, validate in place
- in SGX, copy, validate, copy

Supervisor

OS
- If Supervisor is trusted, invalid data => no resume (e.g.: TrustVisor)
- If Supervisor is untrusted invalid data => no accept, so no access (e.g.: SGX)
Trusted address space

- state handler
- service code
- state root
- data

Untrusted address space

- SMM
- data

page hit on access

fault solved, data accessible on resume, continue…

resume

Supervisor

OS
- HW-based attestation of code identity, including input request, state root, output reply, nonce
- Client checks validity of attestation and intended identities/hashes
Outline

- Evaluation
## TCB size

<table>
<thead>
<tr>
<th></th>
<th>VC³</th>
<th>Haven</th>
<th>${\text{LaSt}}^{\text{GT}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KSLoC</strong></td>
<td>9.2</td>
<td>O(10³)</td>
<td>17</td>
</tr>
<tr>
<td>(lines of code x 1000)</td>
<td></td>
<td></td>
<td>7.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>92.6</td>
</tr>
</tbody>
</table>

- **SGX-based**
- **TPM/TXT based**

Library is small compared to real service.
Comparison

**XMHF-TrustVisor vs. LaSt\textsuperscript{GT}**

LaSt\textsuperscript{GT} is Incremental, Faster & Scalable
SQLite on LaSt\textsuperscript{GT}

- First large-scale experiment on hypervisor
- Data I/O can be optimized through state hierarchy
- SGX expected to improve substantially
Conclusions

• Security for large-scale data processing can be guaranteed with a small TCB

• Virtual memory-based data handling => zero interface

• No change to source code => easy integration

• One design can fit diverse HW & SW
ad maiora.

Secure Tera-scale Data Crunching with a Small TCB

Bruno Vavala¹,², Nuno Neves¹, Peter Steenkiste²
¹LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
²CSD, Carnegie Mellon University, U.S.

Abstract—Outsourcing services to third-party providers comes with a high security cost—to fully trust the providers. Using trusted hardware can help, but current trusted execution environments do not adequately support services that process very large scale datasets. We present LASTGT, a system that bridges this gap by supporting the execution of self-contained services over a large state, with a small and generic trusted computing base (TCB). LASTGT uses widely deployed trusted hardware to guarantee integrity and verifiability of the execution on a remote platform, and it securely supplies data to the service through simple techniques based on virtual memory. As a result, LASTGT is general and applicable to many scenarios such as computational genomics and databases, as we show in our experimental evaluation based on an implementation of LASTGT on a secure hypervisor. We also describe a possible implementation on Intel SGX.

support the execution of either small pieces of code and data [10], or large code bases [11], or specific software like database engines [12] or MapReduce applications [13]. Recent work [14] has shown how to support unmodified services. However, since “the interface between modern applications and operating systems is so complex” [30], it relies on a considerable TCB that includes a library OS. In addition, the above systems are specific for TPMs [10], [15], secure coprocessors [12], or Intel SGX [13]. Hence, porting them to alternative architectures (e.g., the upcoming AMD Secure Memory Encryption and Secure Encrypted Virtualization [36], [37]) requires significant effort. Clearly, it is desirable to design a generic system “not relying on idiosyncratic features of the hardware” [16].

We present LASTGT, a system that can handle a Large State on a Generic Trusted component with a small TCB.
“Never trust a computer you can’t throw out the window.”

Steve Wozniak
“No computer system can be absolutely secure.”

(excerpt from)

Intel’s Legal Desclaimer