Experiences in Intel SGX research

Dongsu Han and Seongmin Kim
KAIST

Joint work with:
Changho Choi, Soham Desai*, Juhyeng Han, Jaehyung Ha,
Prerit Jain*, JaeHyuk Lee, Youjung Shin,
Brent Byounghoon Kang and Taesoo Kim*

APNet 2017
Trend 1: Security and Privacy
Critical Factors in Technology Adoption

• Demands for “security” and “privacy” are increasing
  – Widespread use of Transport Layer Security (TLS)
  – Popularity of anonymity networks (e.g., Tor)
  – Use of strong authentication/encryption in WiFi

• Expectation on security and privacy impacts design decisions:
  – Operating system (iOS, Android)
  – Apps/services (e.g., messenger, adblocker)
  – Network infrastructure (inter-domain SDN)
Trend 1: Security and Privacy

- Demands for “security” and “privacy” are increasing
  - Widespread use of Transport Layer Security (TLS)
  - Popularity of anonymity networks (e.g., Tor)
  - Use of strong authentication/encryption in WiFi

- Expectation on security and privacy impacts design decisions:
  - Operating system (iOS, Android)
  - Apps/services (e.g., messenger, adblocker)
  - Network infrastructure (inter-domain SDN)
Trend 2: Commoditization of Trusted Execution Environment

- Trusted Execution Environment (TEE)
  - Isolated execution: integrity of code, confidentiality
  - Remote attestation

- Commoditization of TEE

The commoditization of TEE brings new opportunities for networking.
SGX : Isolated Execution

• Application keeps its data/code inside the “enclave”
  – Smallest attack surface by reducing TCB (App + processor)
  – Protect app’s secret from untrusted privilege software (e.g., OS, VMM)
SGX : Remote Attestation

- Attest an application on remote platform
- Check the identity of enclave (hash of code/data pages)
- Can establish a “secure channel” between enclaves
SGX Research: Current Status and Challenges

• SGX specification released in 2013.
  – SGX available in Intel’s Skylake microarchitecture (2015)
  – Hardware and software implementations for SGX lag behind their specifications.

SGX CPU and SDK is now available! But..

• Specification for SGX [revision 1 & 2] is not fully available on the SGX hardware (only functionalities in revision 1)

• SGX technology has a complex license model
  – Hard to obtain full license.

Barriers to SGX research
Our work

1. Open-source emulator platform for SGX research
   - OpenSGX [NDSS16]

2. What impact does SGX have on networking?
   - A first Step Towards Leveraging Commodity Trusted execution Environments for Network Applications [HotNets15]
   - Enhancing Security and Privacy of Tor’s Ecosystem by using Trusted Execution Environment [NSDI17]
   - SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module [APNet17]
Network Applications + TEE = ?

- What impact does TEE have on networking? [HotNets15]

- Previous efforts: Adopting TEE to cloud platform
  - Haven [OSDI’14]: Protects applications from an untrusted cloud
  - VC3 [S&P’15]: Trustworthy data analytics in the cloud
1. Network infrastructure: Software-defined inter-domain routing [HotNets2015]
2. Peer-to-peer systems: Tor anonymity network [NSDI2017]
Our work

1. OpenSGX [NDSS16]: Open-source emulator platform for SGX research

2. What impact does SGX have on networking?
   - A first Step Towards Leveraging Commodity Trusted execution Environments for Network Applications [HotNets15]
   - Enhancing Security and Privacy of Tor’s Ecosystem by using Trusted Execution Environment [NSDI17]
   - SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module [APNet17]
Background: Enclave Life Cycle

Virtual Address Space

Physical Address Space

1. ECREATE
   Create an enclave

2. EADD
   Add pages

3. EINIT
   Finalize the enclave

4. EENTER
   Enters the enclave

5. EEXIT
   Leaves the enclave
OpenSGX: Design Goal

• Offer a complete platform for SGX research
  – To explore software and hardware design space of SGX
  – To develop and evaluate SGX-enabled applications

• Solve non-trivial issues on SGX software components
  – Support for system software and user-level APIs
  – Familiar programming model and interface
  – Secure design to defend against potential attack vectors (e.g., Iago attacks)
OpenSGX: Component Overview

- Emulated SGX hardware
- OS emulation layer
- OpenSGX user library
- OpenSGX toolchain

- Enclave loader
- Performance monitor
- Enclave debugger

Enclave Program

Enclave loader
Runtime library
SGX Libraries
- Trampoline
- Stub

OpenSGX toolchain
SGX OS Emulation
SGX QEMU (HW emulation)
OpenSGX: Approach

- Using **user-space** emulation of QEMU
  - Binary translation to support SGX instructions
  - QEMU helper routine to implement complex instructions
Hardware Emulation

- Emulates SGX data structures and SGX processor key
- Enclave page cache (EPC) memory management
  - Direct mapping on virtual memory
  - Access protection: Instrument memory access

```
QEMU's translation rou

Case (Load | Store) {
  1. Prohibit access from host to EPC
  2. Prohibit others enclaves’ EPC to current enclave’s EPC
}
```

Virtual address space
Instruction Support

• OpenSGX supports most instructions in the specification
  – Except for debugging related instructions (e.g., EDBGRD)
  – Instead, it offers rich environment for debugging since it is a “software emulator” (e.g., GDB stub)

• Provides simple C APIs which wraps assembly code
  – User-level instructions (ENCLU) : accessible to user-level APIs
  – Super-level instructions (ENCLS) : Requires system support
OS Emulation Layer

• Emulate OS to execute the privileged SGX instructions
• OS-neutral interface for:
  – Bootstrapping (EPC allocation)
  – Enclave initialization & page translation
  – Dynamic EPC page allocation

<table>
<thead>
<tr>
<th>System call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_sgx_init()</td>
<td>Allocate EPC memory region</td>
</tr>
<tr>
<td>sys_init_enclave()</td>
<td>Create an enclave, Add and measure EPC pages</td>
</tr>
<tr>
<td>sys_add_epc()</td>
<td>Allocates a new EPC page to the running enclave</td>
</tr>
<tr>
<td>sys_stat_enclave()</td>
<td>Obtains the enclave performance statistics</td>
</tr>
</tbody>
</table>
Narrow interface for SGX user lib: Trampoline and Stub

“A strict and narrow interface to handle enclave-host communication using shared data/code”

Trampoline: Shared code to call user-level APIs in the wrapper

Stub: Shared data to specify the function code and arguments

Enclave

Lib

<Specification>
fcode : FUNC_MALLOC
mcode : EAUG
size: 100

Code
...
malloc(100);
...
(Shared)

Heap

Wrapper

Emulated OS

FULL!
Trampoline and Stub Interface

“A strict and narrow interface to handle enclave-host communication using shared data/code”

Enclave

Lib
malloc()
... sgx_exit(tram);
...

Code
... malloc(100);
...

Heap

(Shared)

Trampoline

EEXIT
... if (fcode ==
FUNC_MALLOC)
alloc_tramp();
...

Stub
heap_end()
...

System Call

Emulated OS

Call EAUG
int sys_add_epc()
{ encls(EAUG, ...);
...

User-level APIs to request system calls

alloc_tramp()
{ ...
sys_add_epc();
...
}

Wrapper

User-level APIs to request system calls

alloc_tramp()
{ ...
sys_add_epc();
...
}
OpenSGX: Usage Example

• Similar to building a C program
  – Compile (Similar to gcc)
  – Sign (Using programmer’s key)
  – Execution (Compiled enclave binary is loaded and executed)

```c
void enclave_main(){
    char *hello = "hello sgx!\n";
    sgx_enclave_write(hello, strlen(hello));
    sgx_exit(NULL);
}
```

```bash
$ opensgx -c hello.c
$ opensgx -s hello.sgx -key sign.key
$ opensgx hello.sgx hello.conf hello sgx!
```
OpenSGX: Current Status

• Available at github, released in May 2015
  – 14k LoC
  – https://github.com/sslab-gatech/opensgx
  – 11 Contributors (Gatech, KAIST, Two sigma, MITRE, ...)
  – 31 unique cloners, 1,645 Views (January, 2016)
  – Used in academia: S-NFV [SDN-NFV Security16], Edge functions [SEC16], SGX-enabled VM migration [IEEE SERVICES16], System-level OpenSGX [Computers & Security17], ...

• Our current community

2σ TWO SIGMA
Our work

1. OpenSGX [NDSS16]: Open-source emulator platform for SGX research

2. What impact does SGX have on networking?
   - A first Step Towards Leveraging Commodity Trusted execution Environments for Network Applications [HotNets15]
   - Enhancing Security and Privacy of Tor’s Ecosystem by using Trusted Execution Environment [NSDI17]
   - SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module [APNet17]
Tor anonymity network

- Tor: the most popular anonymity network for Internet users
  - Helps users to defend against traffic analysis and keep user’s privacy (e.g., what sites you visit, IP address)
  - Freely available as an open source
  - 1.8 million users on a daily basis

"One cell is enough to break Tor's anonymity"

Not anonymous: attack reveals BitTorrent users on Tor network

The geographic location of Tor relays *

* from Onionview, https://onionview.codeplex.com/
Tor network: Threat model

• 3-hop onion routing: a single Tor entity cannot know both client and server

• Tor’s Threat model
  – Tor is a volunteer-based network: Tor relays are not trusted

Can run a Tor relays of his own

Can compromise some fraction of Tor relays

Can observe some fraction of network traffic
Limitations of Tor

Tor clients

Processing Unit: Cell (512 Bytes)

Information visible to attackers

Attacker's can modify the behavior

Bandwidth

Modify or inject the cell
Give false information to others

Demultiplex and identify a circuit
SGX-Tor: Leveraging Intel SGX on Tor

Intel SGX + Tor network = SGX-Tor

- Improved trust model
- Operational privacy
- Practicality

Improved trust model
- Spells out what users trust in practice
- Provides ultimate privacy

Operational privacy
- Protects sensitive data and Tor operations
- Prevents modifications on Tor relays

Practicality
- The chance of having more hardware resources donated
- Incrementally deployable
- Compatibility
SGX-Tor: Design and Implementation

User process (Tor application)

Enclave memory

Tor code/data (Core)
- Circuit Establishment
- Hidden service
- Encryption/Decryption
- Cell/Consensus creation

SSL Library
- Integrity check

Attestation Module
- Seals/unseals private data

Sealing Module

Crypto/TLS operations

Securely obtains the entropy and time value

Validates the enclave hash of the Tor program

Encrypts and stores the sensitive data outside the enclave

APNet 2017
SGX-Tor: Design and Implementation

User process (Tor application)

**Enclave memory**

- **Tor code/data (Core)**
  - Circuit Establishment
  - Hidden service
  - Voting
  - Encryption/Decryption
  - Cell/Consensus creation

- **SSL Library**
  - Crypto/TLS operations
  - Integrity check

- **Attestation Module**
  - Seals/unseals private data

- **Sealing Module**

**Application**

- **SGX Runtime Library**
- **OCALL/ECALL Wrapper**

- **Tor code/data (Untrusted)**

- **Standard Library (glibc)**

**System Call**

**Enclave Creation**

**Request system services**

**Enclave initialization**

**Enclave Call**

**Trusted**

**Untrusted**
SGX-Tor: Design and Implementation

User process (Tor application)

Enclave memory

- Tor code/data (Core)
  - Circuit Establishment
  - Hidden service
  - Voting
  - Encryption/Decryption
  - Cell/Consensus creation

SSL Library

Integrity check

Attestation Module

Seals/unseals private data

Sealing Module

Crypto/TLS operations

Enclave initialization

ECALL

OCALL

Narrow interface

Application

SGX Runtime Library

OCALL/ECALL Wrapper

Tor code/data (Untrusted)

Standard Library (glibc)

Sanity checking
1. Argument length
2. Address range

Trusted - Untrusted
Attacks defeated by using SGX-Tor

1. Tagging attack

2. Bandwidth inflation
   1. BW scanning
   2. Detect scanning
   3. Report fake BW

Malicious relay (modified Tor)

Inflated!

Advertised BW

Directory authorities

4. Create consensus document

Replay Cell counting

Tor clients

TLS channel

Entry

Middle

Exit

DesTnation

Plain-text

APNet 2017
Attacks defeated by using SGX-Tor

1. Tagging attack
2. Bandwidth inflation

1. BW scanning
2. Detect scanning
3. Report fake BW
4. Create consensus document

Directory authorities

Malicious relay (modified Tor)

Inflated!
Advertised BW

Plain-text

Attract more clients!

Tor clients
Entry
Middle
Exit
Destination

ACNet 2017
Attacks defeated by using SGX-Tor (Cont.)

- Access sensitive data
- Modify the code

Attacks defeated/mitigated by SGX-Tor
- Circuit demultiplexing [S&P06]
- Bandwidth inflation [PETS07, S&P13]
- Tagging attack [TON12, CCS12, S&P13]

SGX-Tor is an open source:
https://github.com/KAIST-INA/SGX-Tor
Performance evaluation

- SGX-Tor performance: WAN setting
  - Establish a private Tor network
  - For the realistic scenario, we consider the "locality of relays" (Asia, EU, U.S. West, U.S. East)

![Diagram showing throughput and time-to-first-byte for HTTP and HTTPS with file sizes of 10MB and 100MB.]

- 11.9% degradation
- 3.9% additional latency
Our Early Lessons on SGX research

• Performance overheads caused by using SGX
  – EPC Paging (limited memory space : < 200MB)
  – Context switch (for each OCALL)

• While building SGX-based system, we should
  – Reduce enclave size as much as possible
  – Minimize copying already encrypted data to EPC (e.g., SSL-encrypted packet)
  – Seal large data structures that are used infrequently
Our Early Lessons on SGX research (Cont.)

• Security issues while building SGX systems
  – Narrowing interface to reduce attack surface and sanity checking for input/output arguments
  – New attack scenarios caused by assuming malicious system software (e.g., bandwidth inflation by OS in SGX-Tor)

• As a result of our experience, we release SGX-ported OpenSSL as an open source
  – https://github.com/sparkly9399/SGX-OpenSSL
Conclusion

• We design and implement OpenSGX, fully functional and instruction-compatible SGX emulator

• Commoditization of TEE brings new opportunities for network applications

• Ongoing work: Apply SGX to Network Function Virtualization
  – Building a secure middlebox by leveraging SGX
  – Will be presented in **APNet 2017** (SGX-Box)
Our Early Lessons on SGX

• Misconceptions on SGX
  – SGX for desktop-like environment : Needs secure I/O channel (integration with hardware technology such as Intel IPT)
  – Need EPID support for remote attestation

• Malicious use of Intel SGX
  – Malware might be possible by abusing the isolation property
  – Fails on traditional signature-based AV programs
## Comparison: Intel SGX vs OpenSGX

<table>
<thead>
<tr>
<th></th>
<th>Intel SGX</th>
<th>OpenSGX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Hardware</td>
<td>Software Emulator</td>
</tr>
<tr>
<td><strong>Instructions</strong></td>
<td>16 ENCLS, 8 ENCLU</td>
<td>13 ENCLS, 8 ENCLU (Except debugging)</td>
</tr>
<tr>
<td><strong>Data structures</strong></td>
<td>Specified</td>
<td>☐</td>
</tr>
<tr>
<td><strong>Paging</strong></td>
<td>Page table</td>
<td>Direct mapping</td>
</tr>
<tr>
<td><strong>System software</strong></td>
<td>Not specified</td>
<td>User level emulation</td>
</tr>
<tr>
<td><strong>User level APIs</strong></td>
<td>SDK is available (Only for Windows)</td>
<td>☐</td>
</tr>
</tbody>
</table>
Defense against Iago attacks

- Iago attacks [ASPLOS’13]: Malicious OS tries to subvert trusted application by incorrect behavior, e.g., adds incorrect EPC page for heap

```c
void *malloc(int size){
    if(cur_heap_ptr == heap_end) {
        stub->mcode = EAUG;
        exit(trampoline);
    }
    enclu(EACCEPT, ...);
}
```

```
void *malloc_tramp(){
    sys_add_epc();
}
```

```
int sys_add_epc(){
    ...
}
```
Memory State of OpenSGX Program

User process (single address space)

Package Info
Entry point
Measurement
Key ...

SGX Lib
Trampoline
Stub

Enclave Program
Code
Lib
Data
Stack
Heap

Code
Lib
Data
Stack
Heap

System call (e.g., sys_sgxinit())

SGX OS Emulation

QEMU SGX

ENCLU
(e.g., EENTER)

ENCLS
(e.g., EINIT)

ENCLU
(e.g., EEXIT)

Privilege boundary

System calls boundary
Conclusion

- We design and implement OpenSGX, fully functional and instruction-compatible SGX emulator

- As a showcasing application, we develop SGX-enabled Tor to enhance the security and privacy

- OpenSGX offers opportunity to leverage all components of SGX for the research
  - Hardware semantics (e.g., encryption scheme of MEE)
  - System software, enclave loader and user-level APIs
  - Redesigning unforeseen security applications (e.g., Tor)
Trend: Commoditization of TEE

- **Trusted Execution Environment (TEE):** Hardware technology for trusted computing
  - **OS (untrusted)**
  - **Application (untrusted)**
  - **Secure container**
  - **Tor code**

- **Intel SGX:** A promising TEE technology for generic applications
  - Native performance in the secure mode
  - Available on Intel Skylake and Kaby lake CPU

Integrity checking

- Prevents behavior modification

Cannot access data, flow control

- Protects the secrecy of the program
Tor network: Threat model (Cont.)

- Careful admission
- Behavior monitoring

Directory authorities

Tor client

Destination

Anonymity Broken!
Tor network: Threat model (Cont.)

Out-of-scope: network-level adversary (controls a large fraction of network)

1. Currently runs ~10,000 relays
2. Large-scale traffic correlation is believed to be verify difficult in practice

- Careful admission
- Behavior monitoring

• Having a large number of relays

Directory authorities

Tor client

Anonymity Broken!
Tor network: Threat model (Cont.)

However, Tor is still vulnerable to many types of attacks under its traditional threat model

2. Large-scale traffic correlation are believed to be verify difficult in practice
Limitations of Tor

Problem 1. Tor relays are semi-trusted
   - Authorities cannot fully verify the behaviors of them

Problem 2. Even attackers control a few Tor relays, they can
   - Access internal information (circuit identifier, cell header, ...)
   - Modify the behavior of relays (DDoS, packet tampering, ...)

<Low-resource attacks>

- Malicious circuit creation [Security09, CCS11]
- Sniper attack [NDSS15]
- Bad apple attack [LEET11]
- Tagging attack [ICC08, TON12, CCS12, S&P13]
- Bandwidth inflation [PETS07, S&P13]
- Controlling HSDir [S&P13]
- Harvesting hidden service descriptors [S&P13]
- Circuit demultiplexing [S&P06]
- Website fingerprinting [Security15]

Modifying the behavior

Both

Accessing internal information
To address the problems on Tor,

1) Fundamental trust bootstrapping mechanism
2) Advanced trust model to verify untrusted remote parties are required
SGX-Tor: Leveraging Intel SGX on Tor

→ Reduces the power of an attacker who currently gets the sensitive information by running Tor relays

→ Raises the bar for Tor adversary to a traditional network-level adversary (only passively see the TLS bytestream)
SGX-Tor: Threat Model

- Only trusts the underlying SGX hardware & Tor code itself
- Do not address network-level adversaries: who can perform large-scale traffic analysis
- Out of scope: Vulnerabilities in Tor codes, SGX side channel attacks
  → Mitigated by recent SGX research: Moat [CCS16], SGX-Shield [NDSS17], T-SGX [NDSS17]
New functionality: Automatic admission

- Integrity verification of relays (Directory authority → Onion Router)
  - Automatically admits “unmodified” and “SGX-enabled” relays
  - Improved trust model: current implicit trust model turns into the explicit trust model

**NOTE:** Tor uses the same binary for directory authorities, Tor relays, and client proxies
Incremental deployability

• **SGX-Tor’s basic assumption:** “All relays and authorities are SGX-enabled”

• **SGX-Tor supports interoperability**
  – Allows admission of non-SGX relays without remote attestation
  – SGX-enabled clients can get the list of SGX-Tor relays from SGX-enabled authorities
Implementation detail

• Engineering efforts
  – Support for Windows/Linux (based on Intel SGX SDK)
  – SGX-ported libraries: OpenSSL, libevent, zlibc
  – SGX-Tor is an open source: Available at https://github.com/KAIST-INA/SGX-Tor

• Trusted Computing Base (TCB) size
  – 3.8x smaller (320K LoC vs 1,228K LoC) than Graphene (open source library OS for SGX)
Evaluation

1) What kind of sensitive data of Tor is protected by SGX-Tor?
2) What is the performance overhead of running SGX-Tor?
3) How compatible and incrementally deployable is SGX-Tor with the current Tor network?

• Environmental setup
  – SGX CPUs: Intel Core i7-6700 (3.4GHz) and Intel Xeon CPU E3-1240 (3.5GHz)
  – Configuration: 128MB Enclave Page Cache (EPC)
  – Running Tor in Windows, Firefox as a Tor browser (in the client proxy)
  – Establish a private Tor network using chutney
What is protected by SGX-Tor?

<table>
<thead>
<tr>
<th></th>
<th>Current Tor</th>
<th>Network-level adversary</th>
<th>SGX-Tor</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP header</td>
<td>Visible</td>
<td>Visible</td>
<td>Visible</td>
</tr>
<tr>
<td>TLS-encrypted bytestream</td>
<td>Visible</td>
<td>Visible</td>
<td>Visible</td>
</tr>
<tr>
<td>Cell</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>Circuit ID</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>Voting result</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>Consensus document</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>Hidden service descriptor</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>List of relays</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
<tr>
<td>Private keys</td>
<td>Visible</td>
<td>Not visible</td>
<td>Not visible</td>
</tr>
</tbody>
</table>
Performance evaluation (Cont.)

- End-to-end client performance of SGX-Tor (using Tor browser)
  - Web latency: Visiting Alexa Top 50 websites
  - Hidden service: HTTP file server (downloading 10MB)

![Cumulative Distribution Functions]

- SGX-Tor: 13.2s
- Original: 12.2s

7.4% additional latency

- SGX-Tor: 1.30Mbps
- Original: 1.35Mbps

3.3% degradation

APNet 2017
Compatibility with vanilla Tor

- Long-running: Admit SGX-Tor relays in the vanilla Tor
  - Collected results for two weeks

Serves Tor traffic well

Actually selected by multiple Tor users

Listed in the consensus document

* From https://collector.torproject.org/
Conclusion

• We design and implement SGX-Tor by leveraging commodity TEE and demonstrate its viability
  – Gives moderate performance overhead
  – Shows its compatibility and possibility of incremental deployment

• SGX-Tor enhances the security and privacy of Tor by
  – Defending against existing attacks on Tor
  – Bringing changes to the trust model of Tor
  – Providing new properties: automatic admission

• Available at github! (https://github.com/KAIST-INA/SGX-Tor)
On-Going Work

• Apply SGX to Network Function Virtualization
  – Running middleboxes on actual SGX-enabled CPU
  – Will be presented in APNet 2017 (SGX-Box)

• Enhancing the security and privacy of software-defined inter-domain routing