EActors: an actor-based programming framework for Intel SGX

Dr.-Eng. V. A. Sartakov
01.02.2x20

Imperial College London
Why do we need another framework?

The framework
Fundamentals
Messaging
System Components
Benchmark
Examples

Future plans

Conclusion
Software Guard eXtensions (SGX) enclaves enable trusted execution in untrusted environment:

- Protect cold-boot [1], platform reset [2] and DMA attacks [3]
- Remove an OS and a hypervisor from the Trusted Computing Base (TCB)
- Special features: remote/local attestation, data sealing
Programming approach:

- Invocation of functions

Advantages:

- Low TCB
- Intuitive use

ECALL, OCALL:

- 50

sgx_mutex:

- 200
Programming approach:
- Invocation of functions

Advantages:
- Low TCB
- Intuitive use

Disadvantages:
- Inflexible partitioning
- High transition costs
  - ECALL, OCALL: $\approx 50 \times$
  - sgx_mutex: $\approx 200 \times$
Existing Approaches: LibOS/Shim Layer

Programming approach:
- Enclave the whole application

Frameworks:
- Haven [4], SCONE [5],
  Graphene-SGX [6], Panoply [7]

Advantages:
- Legacy
- Fast transitions (some)
Existing Approaches: LibOS/Shim Layer

Programming approach:
- Enclave the whole application

Frameworks:
- Haven [4], SCONE [5],
  Graphene-SGX [6], Panoply [7]

Advantages:
- Legacy
- Fast transitions (some)

Disadvantage:
- Monolithic design → Large TCB
Towards Multi-enclave Applications

A single process can host multiple enclaves

→ Mutually distrusted partitions

Examples:

- Instant message service
- Secure-multiparty computation

Programming model should offer:

- Fast enclave-to-enclave communication
- Minimal per-enclave TCB
- Flexible partitioning
Towards Actors-Based Trusted Computing

Actors:

- Non-blocking
- Use messages

→ Shared-nothing (no locks!)

→ Lightweight (flexible!)
Towards Actors-Based Trusted Computing

Actors:
- Non-blocking
- Use messages
  → Shared-nothing (no locks!)
  → Lightweight (flexible!)

Existing frameworks:
- Heavy runtime (Erlang, Java)
- Do not tailored for enclaves (CAF)
  → Need another framework
Why do we need another framework?

The framework
  Fundamentals
  Messaging
  System Components
  Benchmark
  Examples

Future plans

Conclusion
EActors: Actors-based Trusted Computing

- What is an Actor?
- How actors communicate?
- System support
General View

Components:
- $e$actors
- Enclaves
- Workers

Bindings:
- $e$actors to enclaves
- $e$actors to workers
- workers to CPUs
Programming with \textit{\textit{\textit{e}actors}}

An \textit{\textit{\textit{e}actor}}:

- Constructor
- Body function
- Private state

Building:

- \textit{\textit{\textit{e}actor}}’s source
- Deployment XML
- Framework

Output:

- Enclave’s binaries
- Untrusted binaries

```c
struct state { struct channel chan[2]; int first; }

void aping(struct actor* self) {
    if (self->state->first) {
        self->state->first = 0;
    } else {
        /* receive a pong */
        char* msg = recv(&self->channel[0]);
        if (msg == NULL)
            return;
    }
    /* send a ping */
    send(&self->channel[1], "ping");
}

void aping_ctr(struct actor* self) {
    self->state->first = 1;
    connect(self->channel[0]);
}
```
The node is a memory object:

- Header, Payload
- Allocated at startup
- Private or public

<table>
<thead>
<tr>
<th>Header</th>
<th>NULL</th>
<th>*next</th>
<th>Payload</th>
</tr>
</thead>
</table>

Null
Nodes – a Basis for Messaging

The node is a memory object:

- Header, Payload
- Allocated at startup
- Private or public
- Double-linked queues

API:

- pool: LIFO for empty nodes
- mbox: FIFO for message exchange
- push_to/pop_from tail/front
Message-based Communication

Send/receive:

1. **PING**: Dequeue a node
2. **PING**: Write (enc.) data
3. **PING**: Enqueue to a mbox
4. **PONG**: Dequeue from mbox
5. **PONG**: Read (dec.) data
6. **PONG**: Return the node
Message-based Communication

Send/receive:
1. PING: Dequeue a node
2. PING: Write (enc.) data
Message-based Communication

Send/receive:
1. PING: Dequeue a node
2. PING: Write (enc.) data
3. PING: Enqueue to a mbox

Enclave #1

PING

mbox

Enclave #2

PONG

POOL
Message-based Communication

Send/receive:
1. PING: Dequeue a node
2. PING: Write (enc.) data
3. PING: Enqueue to a mbox
4. PONG: Dequeue from mbox
5. PONG: Read (dec.) data
Message-based Communication

Send/receive:

1. PING: Dequeue a node
2. PING: Write (enc.) data
3. PING: Enqueue to a mbox
4. PONG: Dequeue from mbox
5. PONG: Read (dec.) data
6. PONG: Return the node
Message-based Communication

Send/receive:
1. PING: Dequeue a node
2. PING: Write (enc.) data
3. PING: Enqueue to a mbox
4. PONG: Dequeue from mbox
5. PONG: Read (dec.) data
6. PONG: Return the node
Message-based Communication

Send/receive:

1. PING: Dequeue a node
2. PING: Write (enc.) data
3. PING: Enqueue to a mbox
4. PONG: Dequeue from mbox
5. PONG: Read (dec.) data
6. PONG: Return the node
Connectors and Cargos

Nodes and queues are low-level communication primitives

- Multi-Producer Multi-Consumer
- Plain text
Connectors and Cargos

Nodes and queues are low-level communication primitives

- Multi-Producer Multi-Consumer
- Plain text

Cargos and Connectors are high-level communication primitives

- Unified interfaces for encrypted and non-encrypted messages
- Based on nodes and queues
- P2P message exchange
- Uses local-attestation for the key-exchange procedure
System Components::System Actors and EOS

System actors:

- $e$actor cannot use syscalls
- Multiple system $e$actors
- Message based interaction
System Components::System Actors and EOS

System actors:
- Eactor cannot use syscalls
- Multiple system Eactors
- Message based interaction

Eactors Object Store:
- Key-value store
- Can be private or public
- Can be encrypted or non-encrypted
- Persistence on demand
Ping-pong:
- 1,000,000 messages
- 16–512 KiB

SDK:
- 2 threads, ECALLs

EActors:
- 2 Actors, cargos
Ping-pong

SDK: 319 (1783 peak)
- 32KiB – L1 cache

SDK
EActors
Encrypted

Throughput (MiB/s)

0
5,000
10,000

Message size (Bytes)

16 64K 128K 256K 512K
Ping-pong

SDK: 319 (1783 peak)
- 32KiB – L1 cache

EActors: 9706

Throughput (MiB/s)

Message size (Bytes)
Ping-pong

SDK: 319 (1783 peak)
- 32KiB – L1 cache

EActors: 9706
Encrypted: 974
Some Examples and Demos

Sources:
https://github.com/ibr-ds/EActors/tree/master/examples

template  Simple hello-world actor
pingpong  non-encrypted messages
pingpong2 cargo-based messaging
pingpongLA Local attestation
  smc  Secure multi-party computation
eos   EActors object store
http  A simple web server with SSL
https://primate.ibr.cs.tu-bs.de
Plan

Why do we need another framework?

The framework

Fundamentals

Messaging

System Components

Benchmark

Examples

Future plans

Conclusion
EActors:: What is next?

- Hardening – Isolation for actors
- Auto partitioning
- Multi-enclave Applications
- Independent from Intel SGX SDK
Plan

Why do we need another framework?

The framework

Fundamentals

Messaging

System Components

Benchmark

Examples

Future plans

Conclusion
Takeaway

- EActors – an actor-based programming framework
- C, uses the Intel SGX SDK
- Targets multi-enclave use cases
- Provides system components
- High-performance communication primitives

Sources: https://github.com/ibr-ds/EActors

Thank you!


