*E*Actors: an actor-based programming framework for Intel SGX

Dr.-Eng. V. A. Sartakov

01.02.2x20

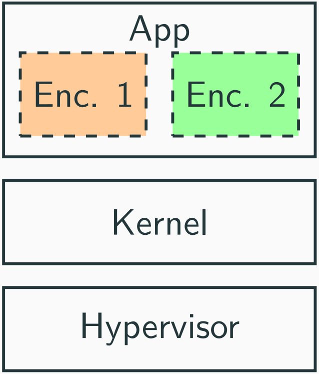
Imperial College London

Plan

New System Component for Trusted Execution

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



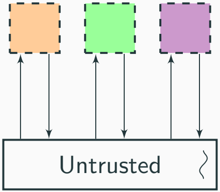
Intel SGX Software Development Kit

Programming approach:

- Invocation of functions

Advantages:

- Low TCB
- Intuitive use



Intel SGX Software Development Kit

Programming approach:

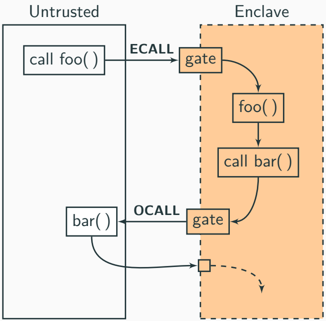
- Invocation of functions

Advantages:

- Low TCB
- Intuitive use

Disadvantages:

- Inflexible partitioning
- High transition costs
  - ECALL, OCALL: ≈50×
  - sgx_mutex: ≈200×



Existing Approaches::LibOS/Shim Layer

Programming approach:

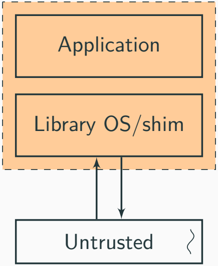
- Enclave the whole application

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:

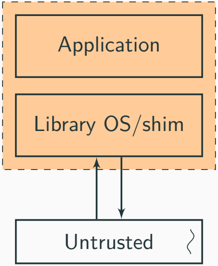
- Haven [4], SCONE [5], Graphene-SGX [6], Panoply [7]

Advantages:

- Legacy
- Fast transitions (some)

Disadvantage:

- Monolithic design → Large TCB



Towards Multi-enclave Applications

A single process can host multiple enclaves

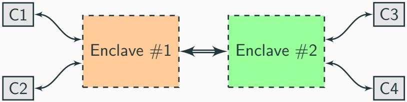
→ Mutually distrusted partitions

Examples:

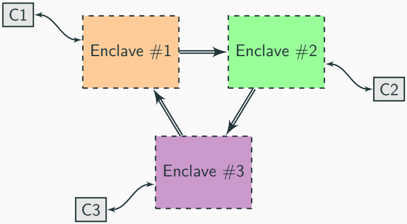
- Instant message service
- Secure-multiparty computation

Programming model should offer:

- Fast enclave-to-enclave communication
- Minimal per-enclave TCB
- Flexible partitioning



Partitioned instant message service



Secure multi-party computation

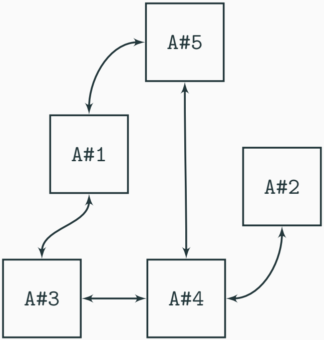
Towards Actors-Based Trusted Computing

Actors:

- Non-blocking
- Use messages

→ Shared-nothing (no locks!)

→ Lightweight (flexible!)



Towards Actors-Based Trusted Computing

Actors:

- Non-blocking
- Use messages

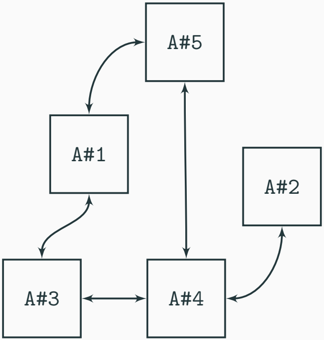
→ Shared-nothing (no locks!)

→ Lightweight (flexible!)

Existing frameworks:

- Heavy runtime (Erlang, Java)
- Do not tailored for enclaves (CAF)

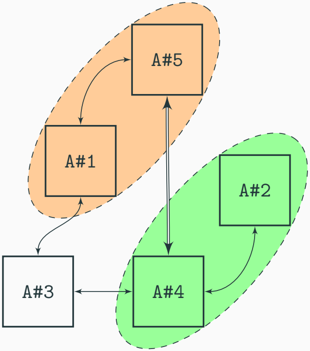
→ **Need another framework**



Plan

*E*Actors: Actors-based Trusted Computing

- What is an *Actor*?
- How actors communicate?
- System support



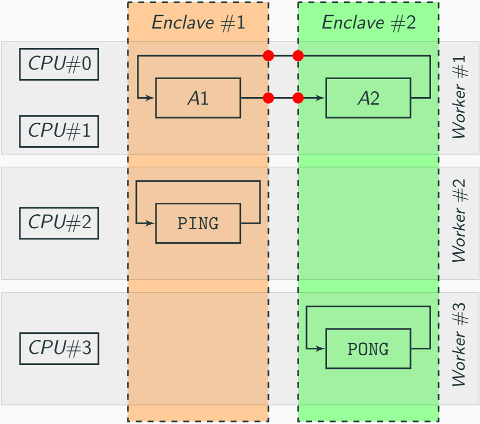
General View

Components:

- *eactors*
- Enclaves
- Workers

Bindings:

- *eactors* to enclaves
- *eactors* to workers
- workers to CPUs



Programming with $_e$*actor*s

An $_e$*actor*:

- Constructor
- Body function
- Private state

Building:

- $_e$*actor*'s source
- Deployment XML
- Framework

Output:

- Enclave's binaries
- Untrusted binaries

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

void aping(struct actor* self) {
  if(self->state->first) {
    self->state->first = 0;
  } else {
  ^^I/* 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]);
}
```

Nodes – a Basis for Messaging

The `node` is a memory object:

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

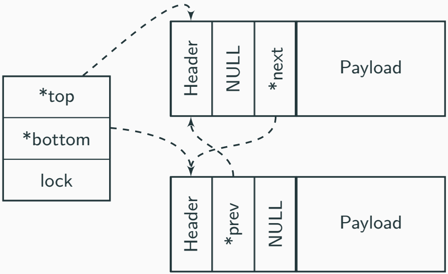
Nodes – a Basis for Messaging

The `node` is a memory object:

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

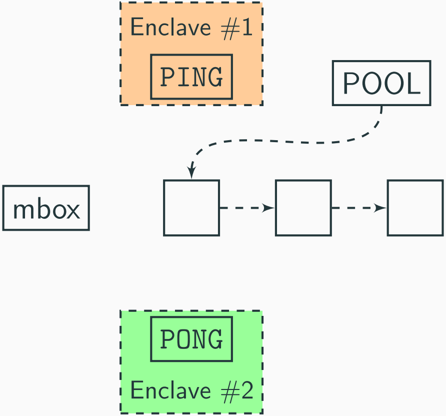
API:

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



Message-based Communication

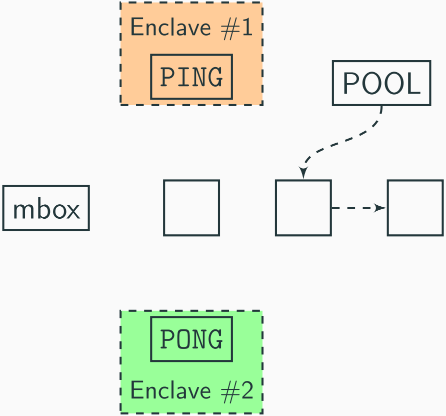
Send/receive:



Message-based Communication

Send/receive:

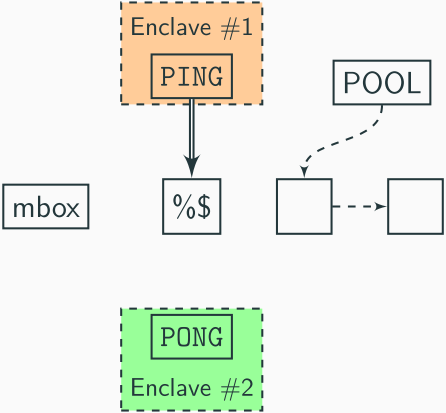
1. PING: Dequeue a node
2. PING: Write (enc.) data



Message-based Communication

Send/receive:

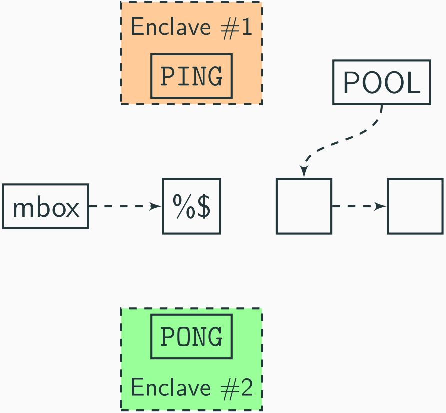
1. `PING`: Dequeue a node
2. `PING`: Write (enc.) data
3. `PING`: Enqueue to a mbox



Message-based Communication

Send/receive:

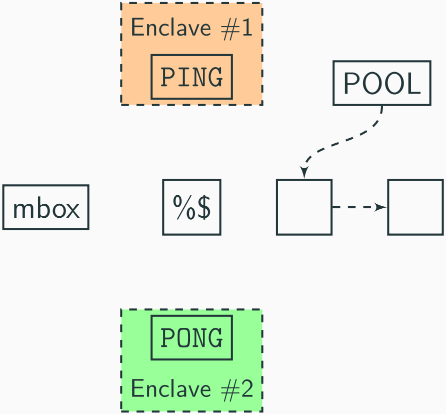
1. `PING`: Dequeue a node
2. `PING`: Write (enc.) data
3. `PING`: Enqueue to a mbox
4. `PONG`: Dequeue from mbox
5. `PONG`: Read (dec.) data



Message-based Communication

Send/receive:

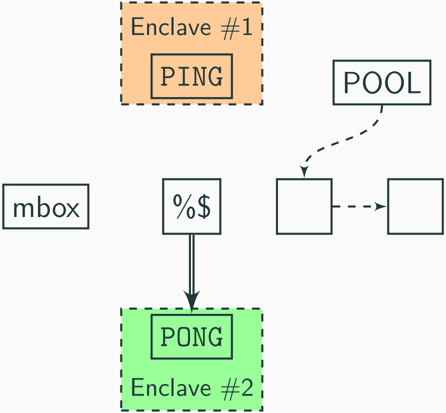
1. `PING`: Dequeue a node
2. `PING`: Write (enc.) data
3. `PING`: Enqueue to a mbox
4. `PONG`: Dequeue from mbox
5. `PONG`: Read (dec.) data
6. `PONG`: Return the node



Message-based Communication

Send/receive:

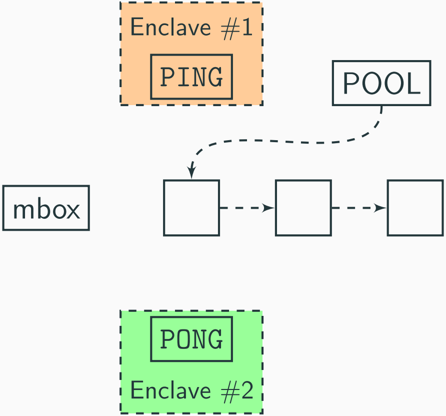
1. `PING`: Dequeue a node
2. `PING`: Write (enc.) data
3. `PING`: Enqueue to a mbox
4. `PONG`: Dequeue from mbox
5. `PONG`: Read (dec.) data
6. `PONG`: Return the node



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

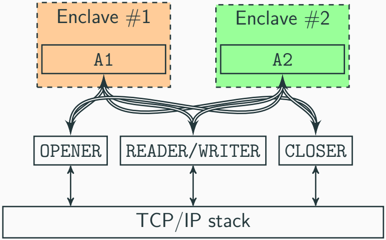
Cargos and Connectors are high-level communication primitives

- 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:

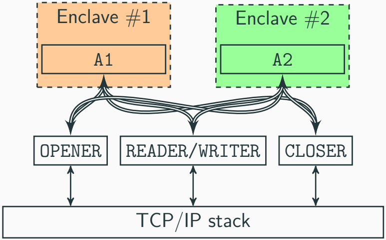
- $_e$*actor* **cannot** use syscalls
- Multiple system $_e$*actor*s
- Message based interaction



System Components::System Actors and EOS

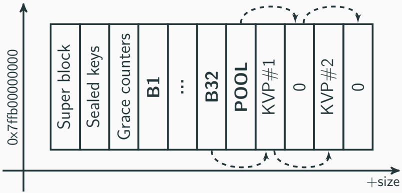
System actors:

- $_e$*actor* **cannot** use syscalls
- Multiple system $_e$*actor*s
- Message based interaction



*E*actors Object Store:

- Key-value store
- Can be private or public
- Can be encrypted or non-encrypted
- Persistence on demand



Ping-pong

Ping-pong:

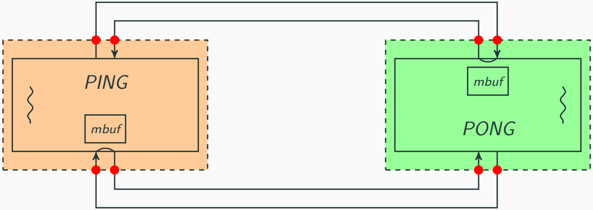
- 1,000,000 messages
- 16–512 KiB

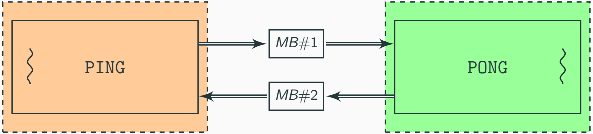
SDK:

- 2 threads, ECALLs

*E*Actors:

- 2 Actors, cargos

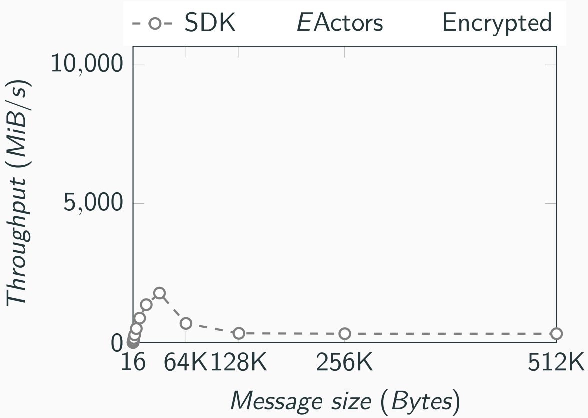




Ping-pong

SDK: 319 (1783 peak)

- 32KiB – L1 cache

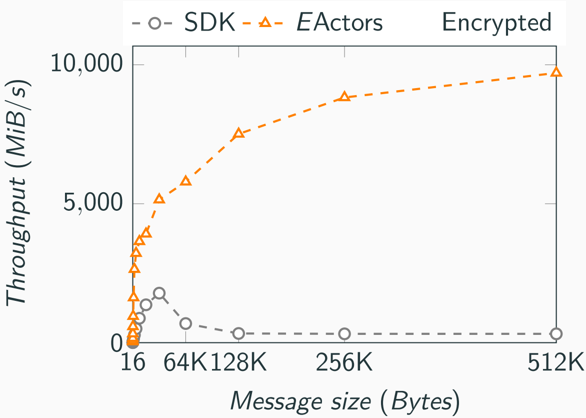


Ping-pong

SDK: 319 (1783 peak)

- 32KiB – L1 cache

*E*Actors: 9706



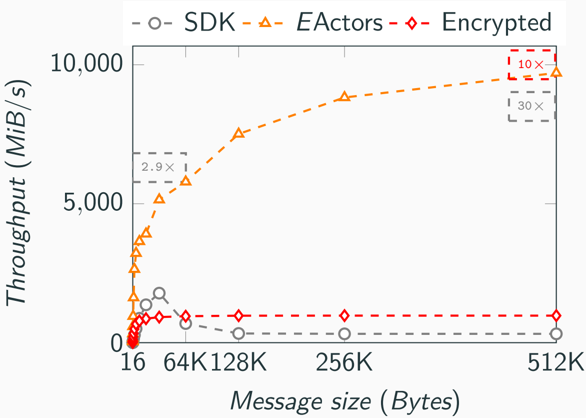
Ping-pong

SDK: 319 (1783 peak)

- 32KiB – L1 cache

*E*Actors: 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

*E*Actors:: What is next?

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

Plan

Takeaway

- *E*Actors – 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!

References i

- J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.
- A. Boileau, "Hit by a bus: Physical access attacks with firewire," *Presentation, Ruxcon*, vol. 3, 2006.
- B. Böck and S. B. Austria, "Firewire-based physical security attacks on windows 7, efs and bitlocker," *Secure Business Austria Research Lab*, 2009.
- A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

- S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell *et al.*, "SCONE: Secure Linux Containers with Intel SGX." in *OSDI*, 2016, pp. 689–703.
- C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.
- S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "PANOPLY: Low-TCB Linux Applications With SGX Enclaves," in *Proc. of the Annual Network and Distributed System Security Symp.(NDSS)*, 2017.