USENIX Security '23

A Verified Confidential Computing as a Service Framework for Privacy Preservation

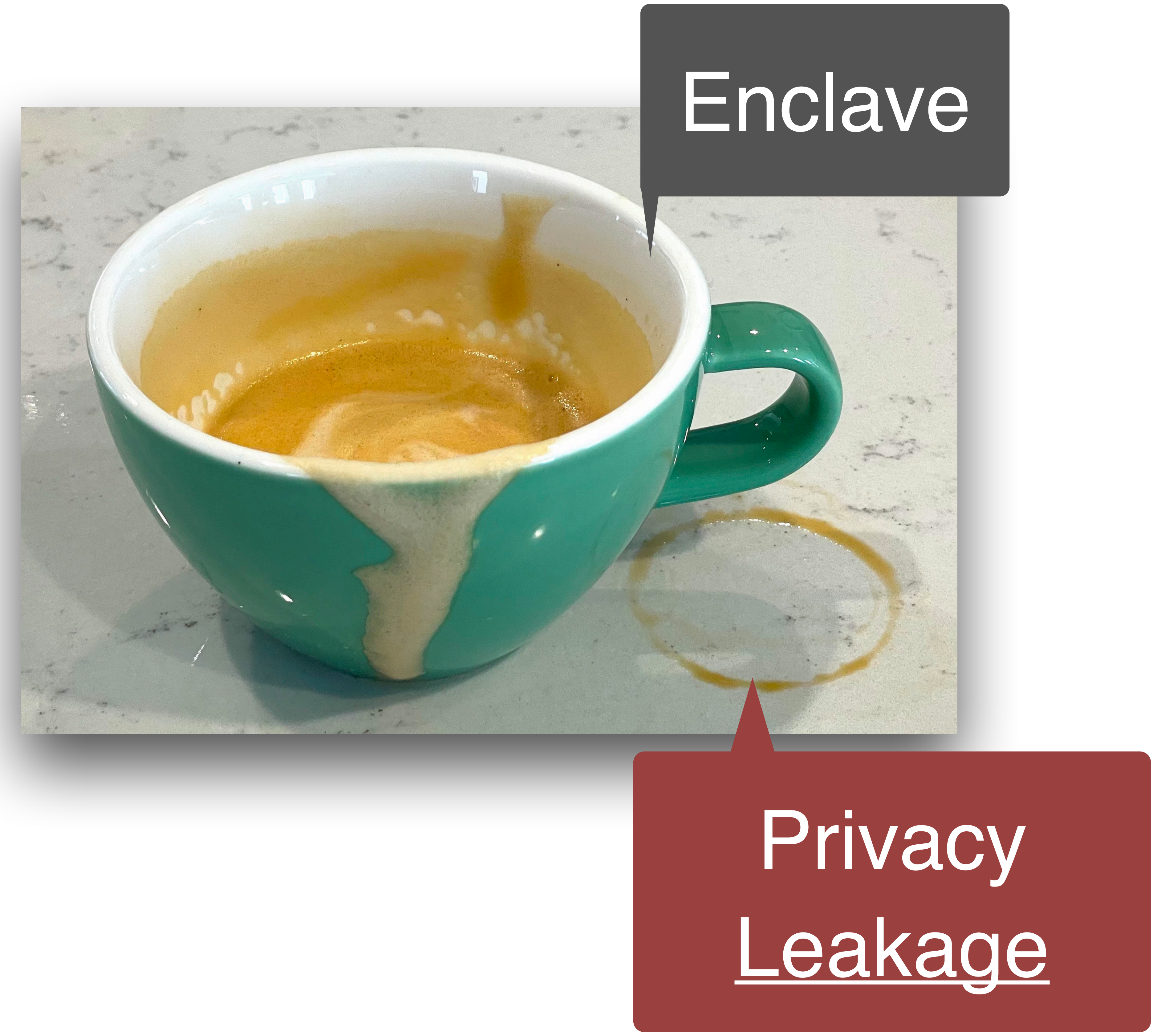
Hongbo Chen[1], Haobin Hiroki Chen[1], Mingshen Sun[2], Kang Li[3], Zhaofeng Chen[3], and XiaoFeng Wang[1]

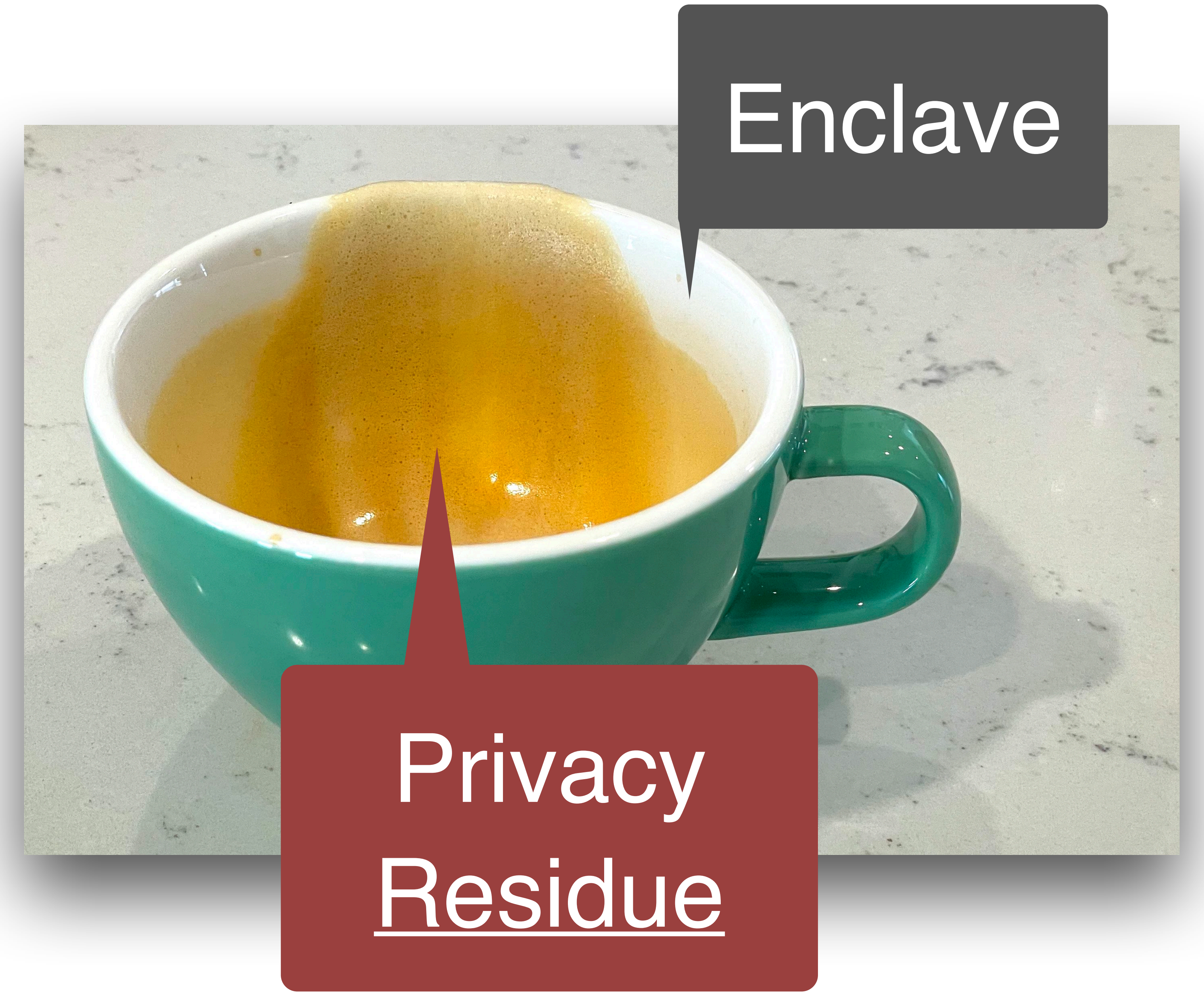
[1] Indiana University Bloomington, [2] Independent Researcher, [3] CertiK

Introduction & Background

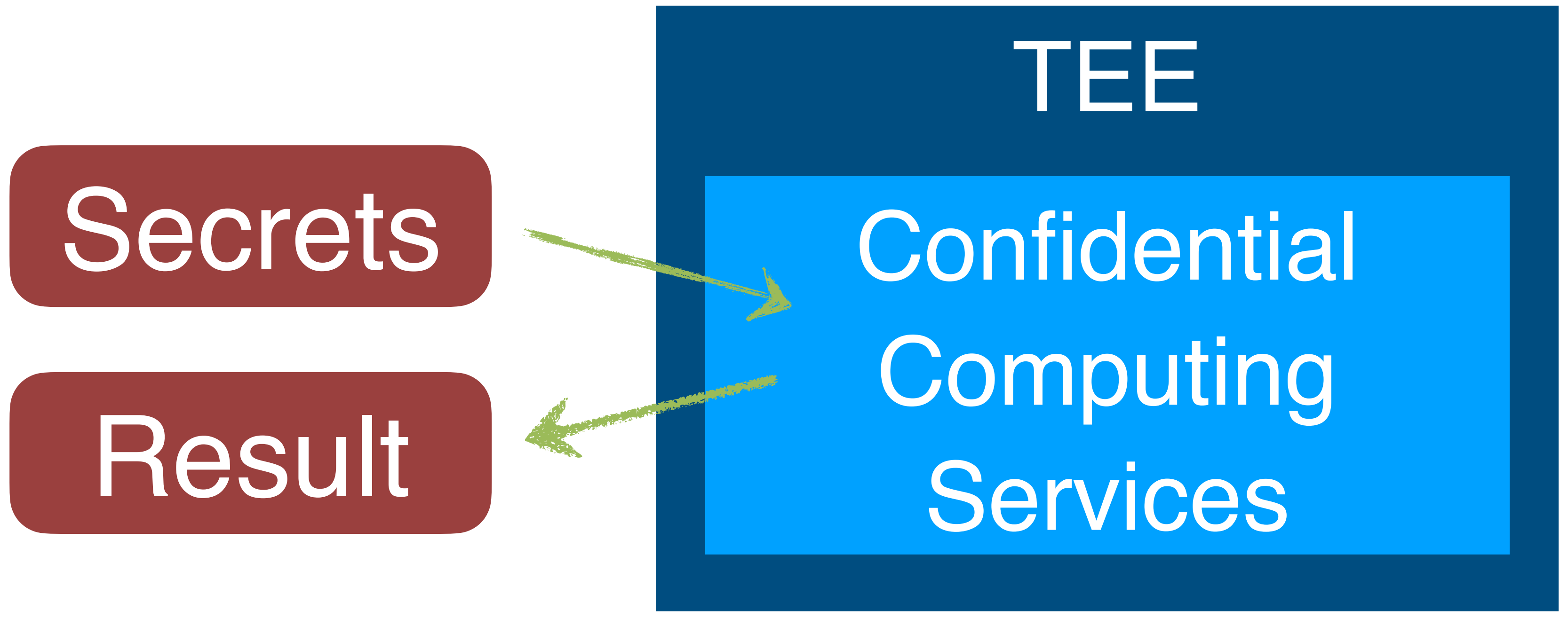
Coffee Incidents

Privacy Incidents



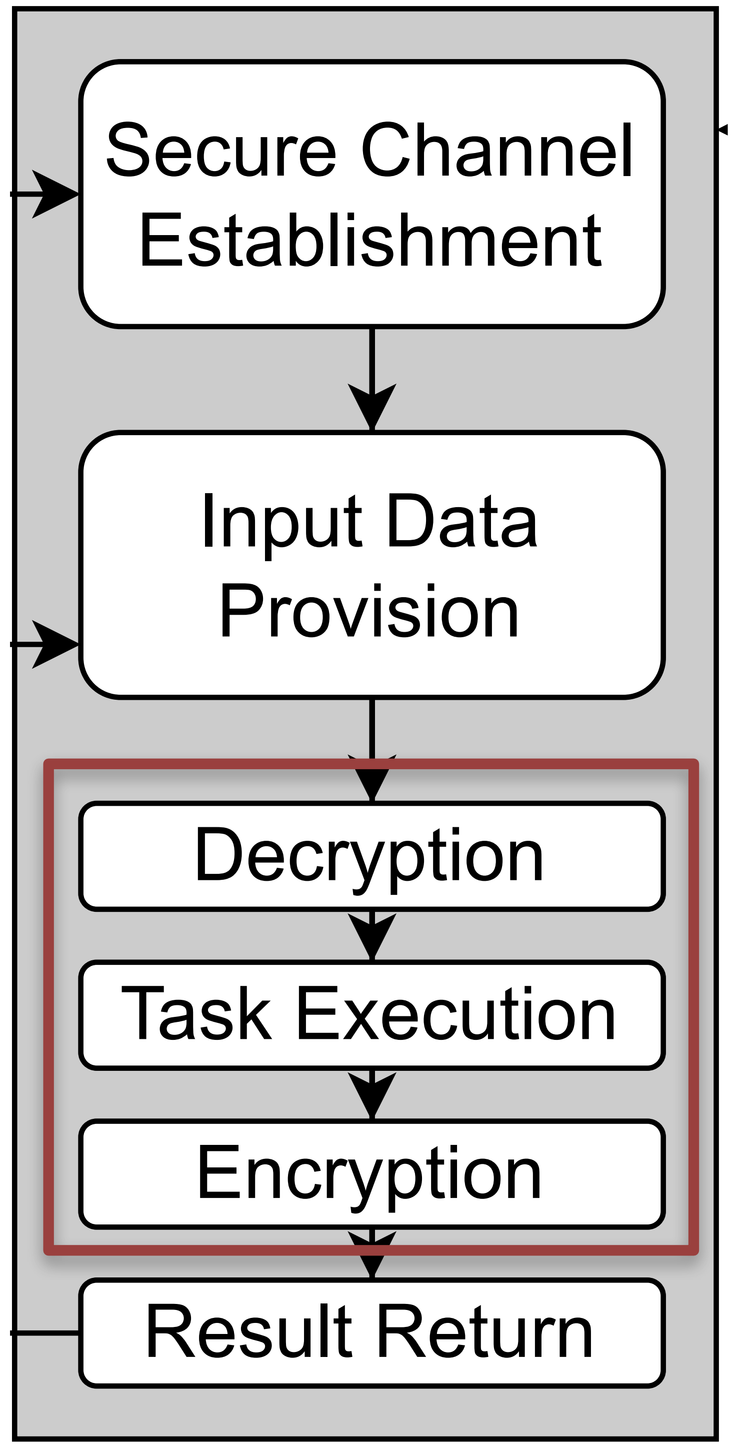


TEE's Abilities and Inabilities



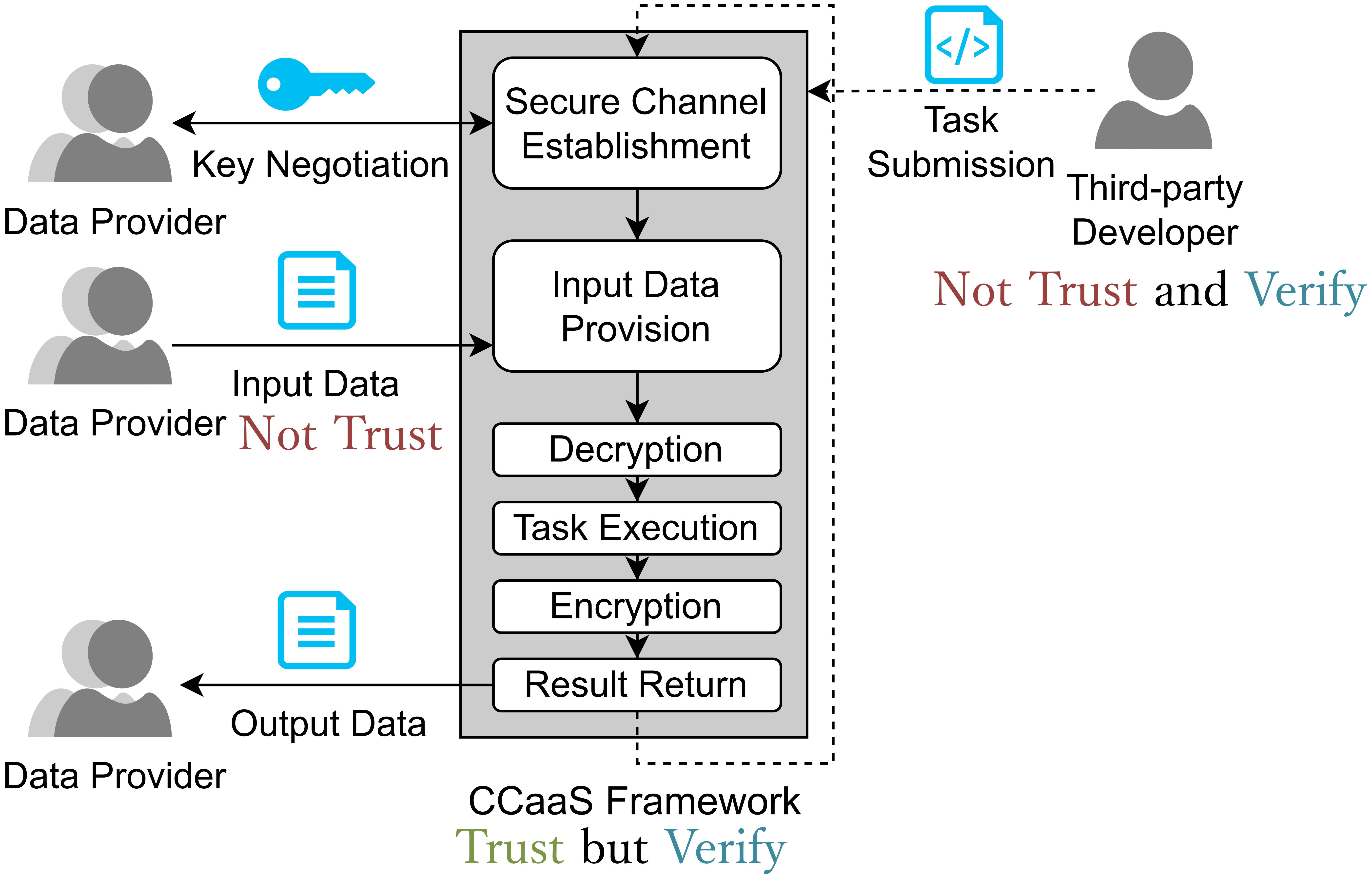
- ✓ Attestation — Guarantee code integrity
- ✓ Isolation → Prevent outside attackers
- ✓ Encryption — Protect data confidentiality

When Confidential Computing Become a Service

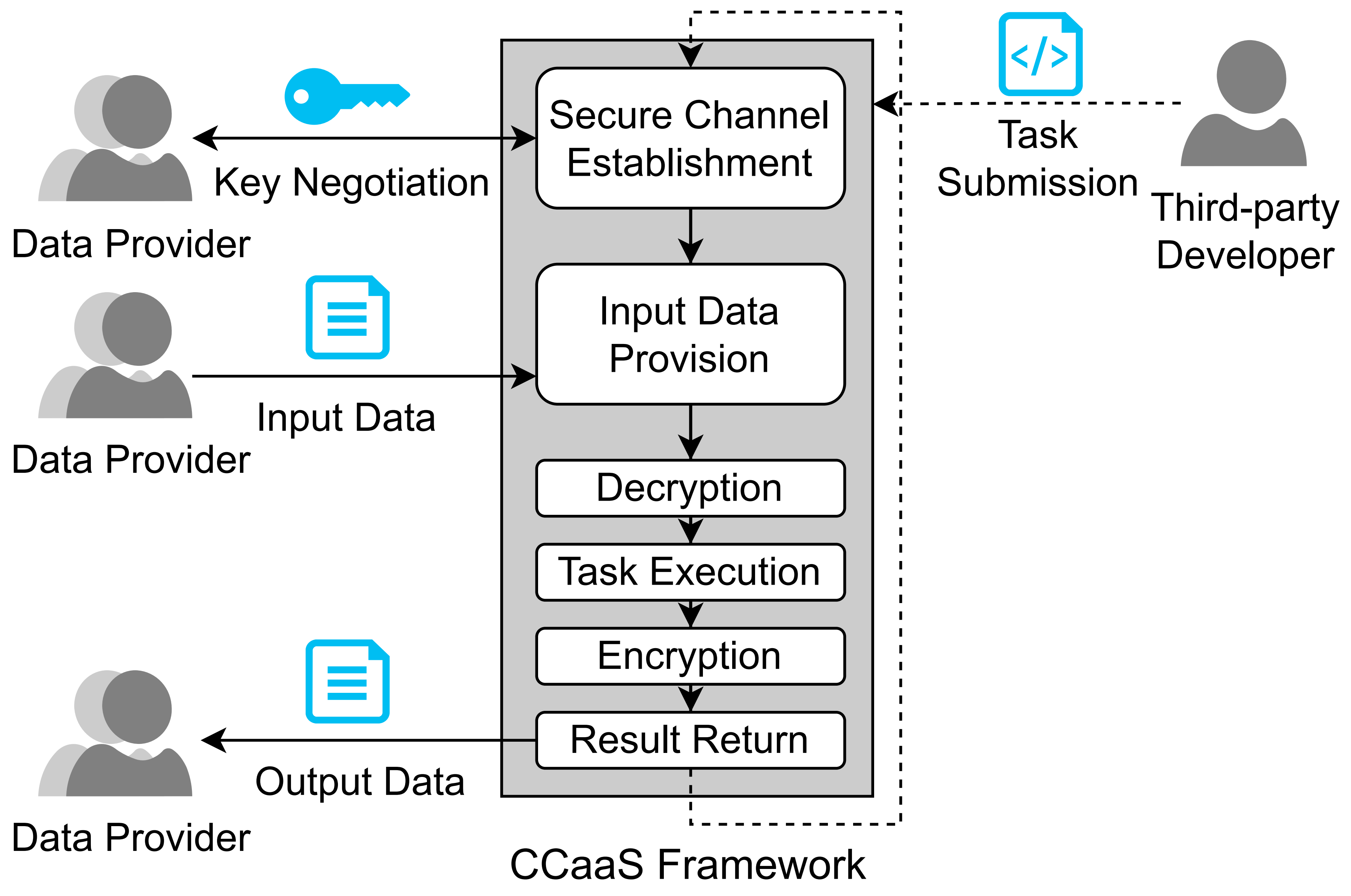


CCaaS Framework

CCaaS for Multiple Data Providers



TEE's Abilities and Inabilities



- **Attestation**: guarantee identity of code
  ⇨ *cannot prove the trustworthiness*
- **Isolation**: prevent outside attackers
  ⇨ *cannot prevent data leakage*
- **Encryption**: protect data safety
  ⇨ *cannot avoid secrets withheld*

Our goal: prove to the user that the enclave service cannot threaten their private information.

Proof of Being Forgotten (PoBF)

No Leakage

All secret and secret-tainted values are within a confined zone during computation.

+

No Residue

After the computation (e.g., serving a user), no secret is found in the enclave.

Theoretical Foundation: Enclave Model

Table 1: Generalized model of secure enclaves.

Type	Sym.	Definition
Natural	n	$\in \mathbb{N}$
String	str	$\in \mathbb{S}$
Bool	b	$::= \texttt{True} \vert \texttt{False}$
Value	v'	$::= \texttt{ConcreteN}(n) \vert \texttt{ConcreteB}(b) \vert \texttt{Any}$
Sec. Tag	vt	$::= \texttt{Secret} \vert \texttt{NotSecret} \vert \texttt{Nonsense}$
TagValue	v	$::= (v', vt)$
Mode	mo	$::= \texttt{EnclaveMode} \vert \texttt{NormalMode}$
Location	l	$::= \texttt{Stack}(n) \vert \texttt{Ident}(str) \vert \texttt{RV}$
Enc. Tag	et	$::= \texttt{Zone} \vert \texttt{NonZone}$
Cell	c	$::= \texttt{Nomral}(v) \vert \texttt{Enclave}(et, v) \vert \texttt{Unused}$
Result	r	$::= \texttt{Ok}(X) \vert \texttt{Err}(e)$
Error	e	$::= \texttt{Invalid} \vert \texttt{NoPrivilege}$
Storable	me	$::= \texttt{List}\ (l, c)$

Table 2: Enclave program syntax.

Term	Sym.	Definition
Exp.	e	$::= l \vert v' \vert \texttt{UnaryOp}(e) \vert \texttt{BinaryOp}(e1, e2)$
Proc.	p	$::= \texttt{Nop} \mid \texttt{Eenter} \mid \texttt{Eexit} \mid \texttt{Asgn}\, l := e$ $\mid \texttt{If}\ e\ \texttt{Then}\ p1\ \texttt{Else}\ p2 \mid \texttt{While}\ e\ \texttt{Do}\ p$ $\mid p1;\ p2$

Theoretical Foundation: NoLeakage Theorem

A procedure's execution does not leak secret.

↑

- Its initial state is secure;
- All memory writes are within the Zone;
- It aborts when error occurs;

Theoretical Foundation: NoResidue Theorem

If the Zerorize procedure is executed at the end of a function, then no sensitive data residue is left in the enclave.

↑

`zerorize` Clears the values stored in the confined zone.

Theoretical Foundation: Checked by Coq

✓ Mechanically Checked by Coq

Realizing the secure enclave service.

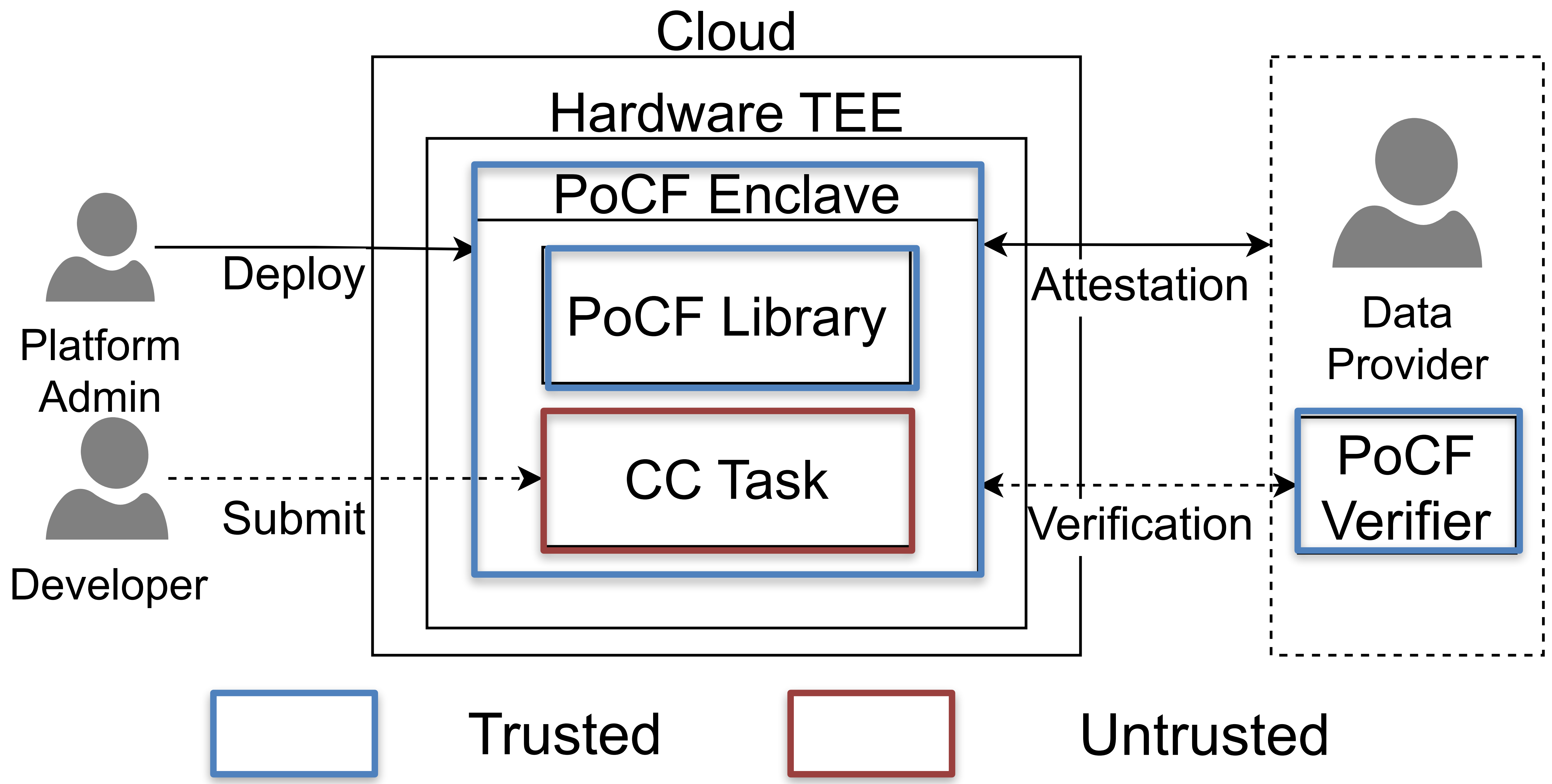
Design Goals

Security: No Leakage | No Residue | Verifiable

Auxiliary:

- Minimal code modification
- Various hardware TEE support

PoBF-Compliant Framework (PoCF)



System Overview

Our Artifacts:

- PoCF Library (TEE-Agnostic)
- PoCF Enclave (TEE-Specific)
- PoCF Verifier

Submitted by 3rd Party Developer:

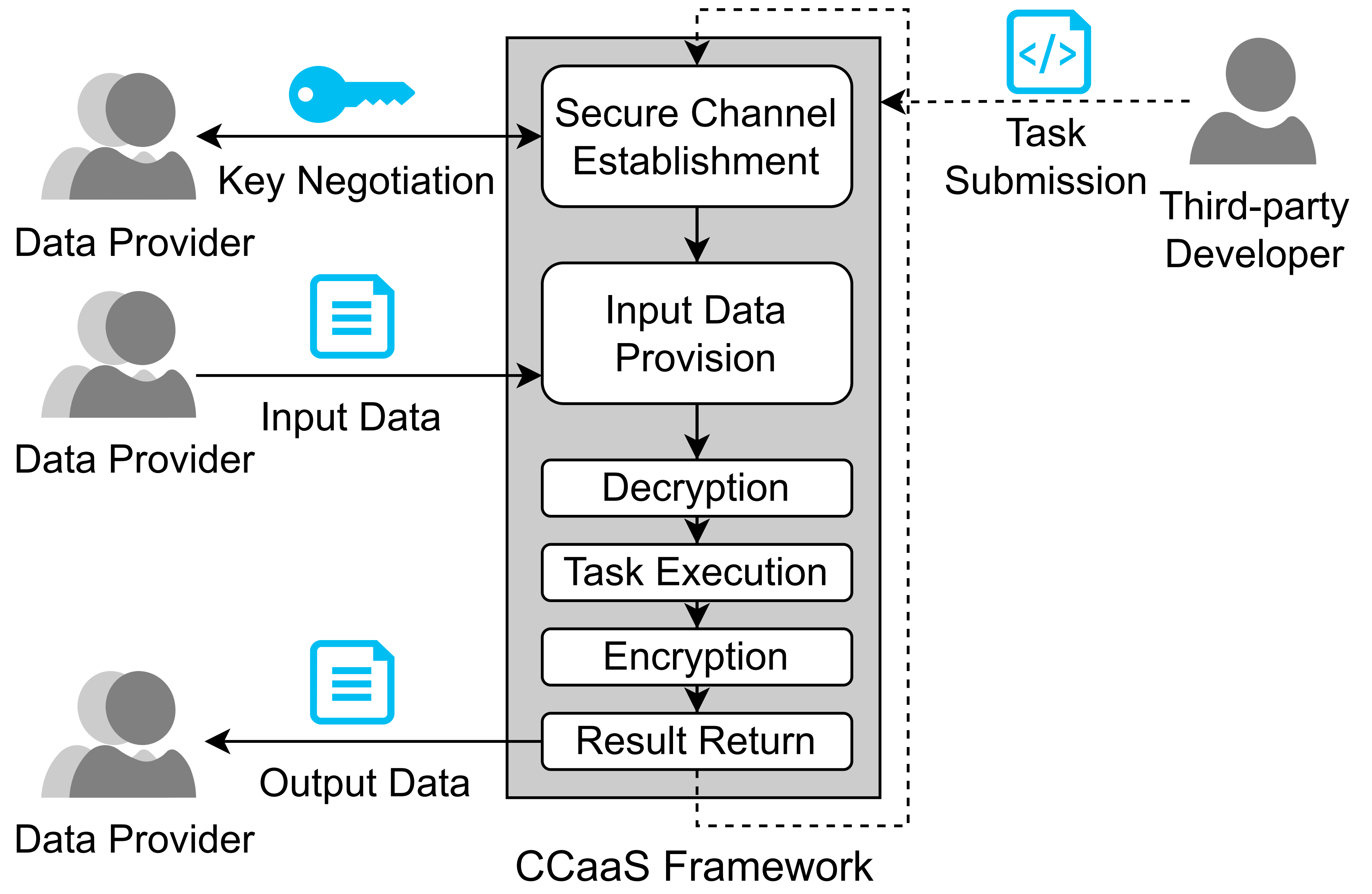
- CC (Confidential Computing) Task

Pillar of PoCF: Workflow Integrity

Control Flow Integrity

Type Safety ➜ Memory Safety

Workflow Integrity?



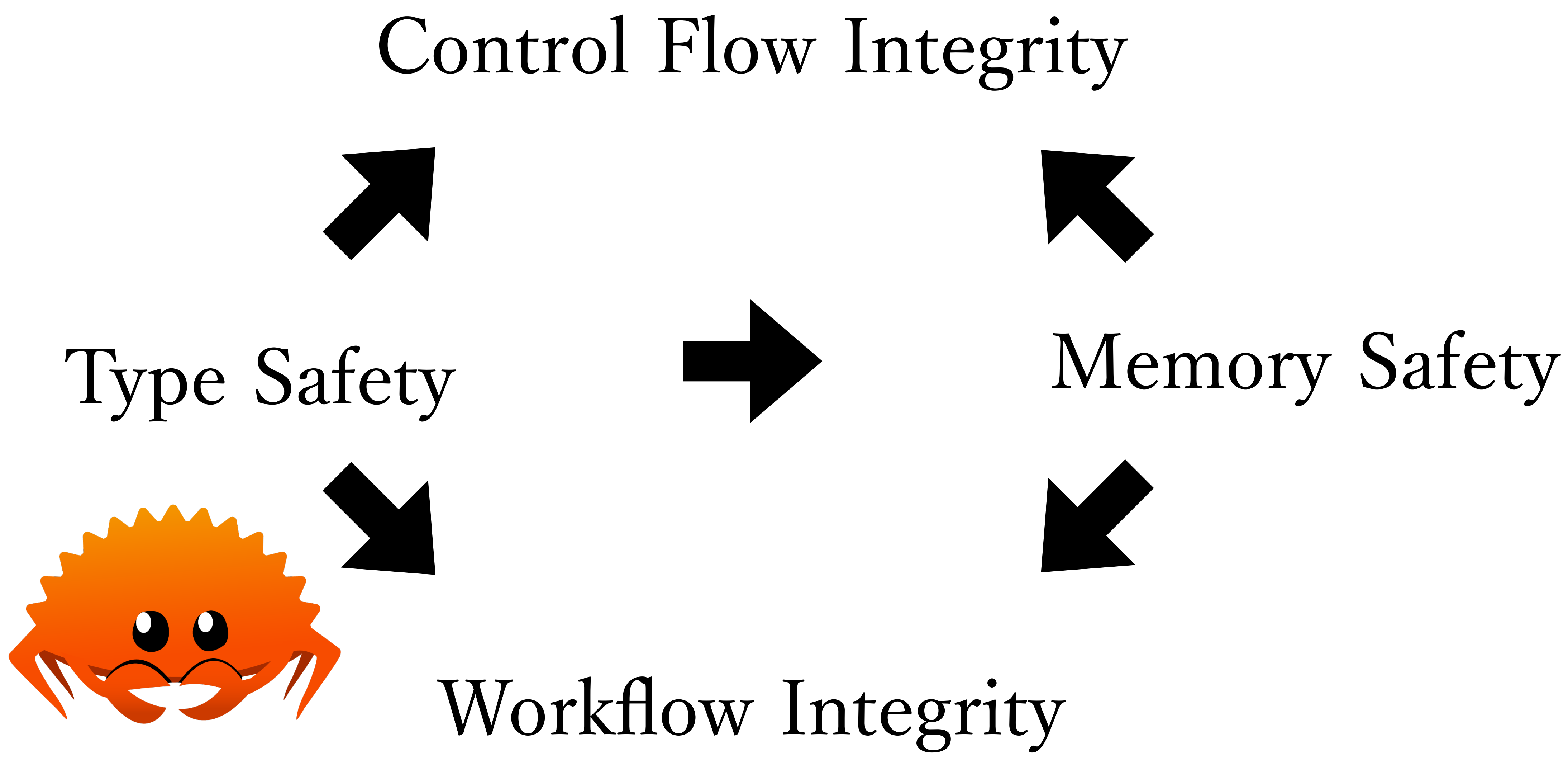
Typestate Specification

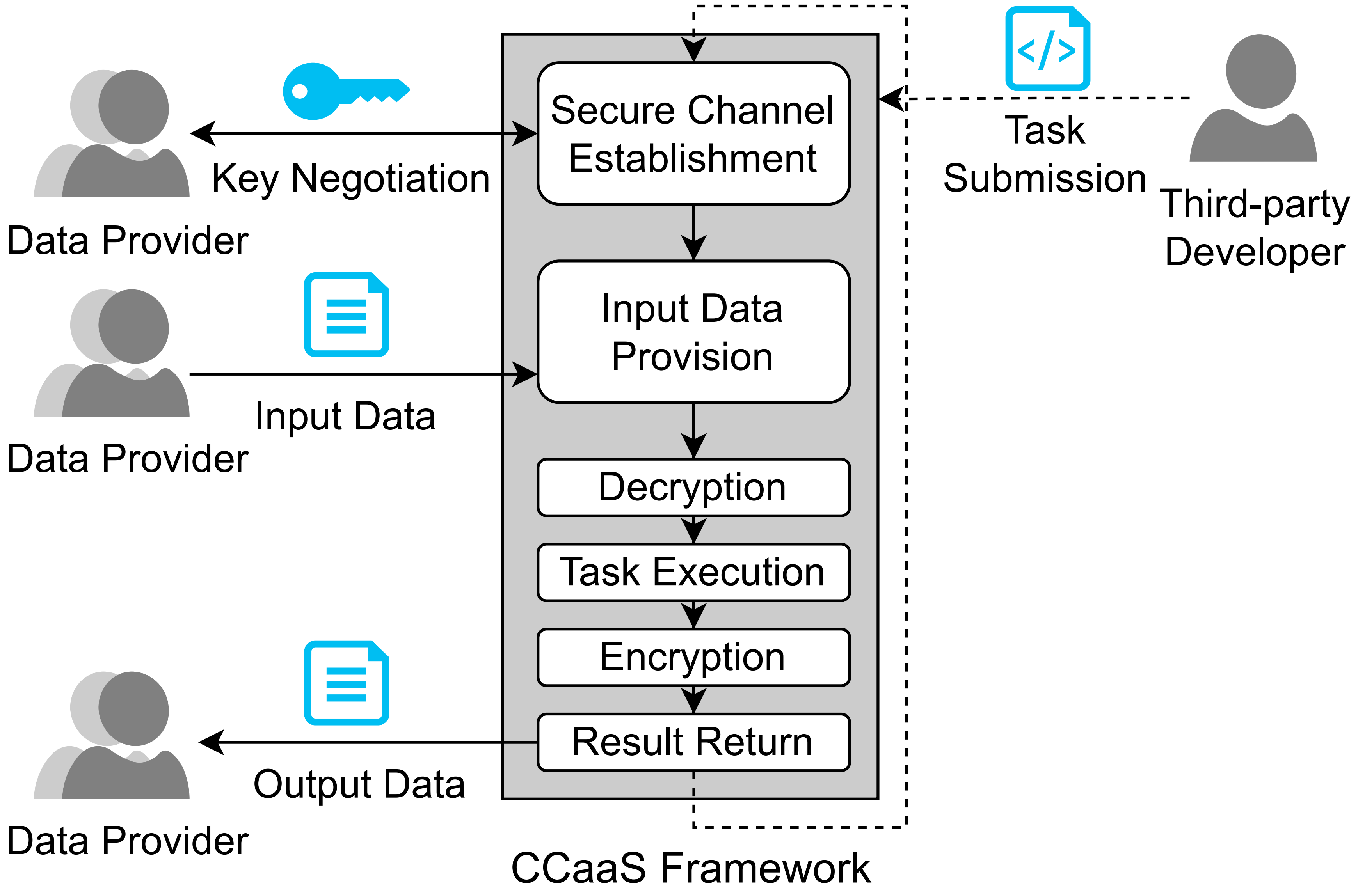
- ✓ Specified.
- ✓ Enforced by Rust.
- ✓ Verified by Prusti.
- ✓ Statically checked.
- ✓ Finally, workflow integrity guaranteed with minor runtime cost!

Listing 1: Typestate abstraction and specification.

```
pub struct Task<S, K, D> where
    S: TaskState + DataState + KeyState,
    K: Zeroize + Default, D: EncDec<K>,
    <S as DataState>::State: DState,
    <S as KeyState>::State: KState,
{
    data: Data<<S as DataState>::State, D, K>,
    key: Key<K, <S as KeyState>::State>,
    _state: S,
}

pub trait TaskState {
    #[pure]
    fn is_initialized(&self) -> bool {false}
    #[pure]
    fn is_finished(&self) -> bool {false}
    // Other similar functions are omitted.
}

pub struct Initialized;
#[refine_trait_spec]
impl TaskState for Initialized {
    #[pure]
    #[ensures(result == true)]
    fn is_initialized(&self) -> bool {true}
}

#[ensures((&result)._state.is_allowed_once())]
// Other similar specifications are omitted
pub fn cc_compute(self) ->
    Task<ResultEncrypted,Invalid,EncryptedOutput>;
```

Workflow Integrity by Rust & Typestate





NoResidue Instrumentation

✓Heap: modified Memory Allocator

✓Global: not mutable

✓Stack and Registers: Instrumentation

No Residue

Listing 1: Typestate abstraction and specification.

```
pub struct Task<S, K, D> where
    S: TaskState + DataState + KeyState,
    K: Zeroize + Default, D: EncDec<K>,
    <S as DataState>::State: DState,
    <S as KeyState>::State: KState,
{
    data: Data<<S as DataState>::State, D, K>,
    key: Key<K, <S as KeyState>::State>,
    _state: S,
}

pub trait TaskState {
    #[pure]
    fn is_initialized(&self) -> bool {false}
    #[pure]
    fn is_finished(&self) -> bool {false}
    // Other similar functions are omitted.
}

pub struct Initialized;
#[refine_trait_spec]
impl TaskState for Initialized {
    #[pure]
    #[ensures(result == true)]
    fn is_initialized(&self) -> bool {true}
}

#[ensures((&result)._state.is_allowed_once())]
// Other similar specifications are omitted
pub fn cc_compute(self) ->
    Task<ResultEncrypted,Invalid,EncryptedOutput>;
```

NoLeakage Verification

✓Edge function calls: does not leak secret.

- E.g., OCALL in SGX and call to the hypervisor in SEV

✓Static taint analysis

- Key's tracking: typestate
- Data tracking: MIRAI's taint analysis

No Leakage

PoCF Verifier

PoCF: Publicly Available

Verifiable

- Once CC Task Submitted: the deployer verifies it.

  1. Pass Verification: PoCF Enclave compiled.

- Data providers:

  1. Obtain the source code.
  2. Conduct verification.
  3. Calculate measurement.
  4. Feed data.

- Trusted builder: proprietary code.

Evaluation

Summary of Evaluation Results

1. PoCF reaches its design goals.
2. PoCF incurs negligible overhead in CPU-bound tasks.
3. PoCF exhibits degradation in IO-bound tasks (lack of IO optimizations).
4. The data flow tracking tool is not very accurate.

Questions?

ARTIFACT EVALUATED
usenix ASSOCIATION
AVAILABLE

ARTIFACT EVALUATED
usenix ASSOCIATION
FUNCTIONAL

ARTIFACT EVALUATED
usenix ASSOCIATION
REPRODUCED

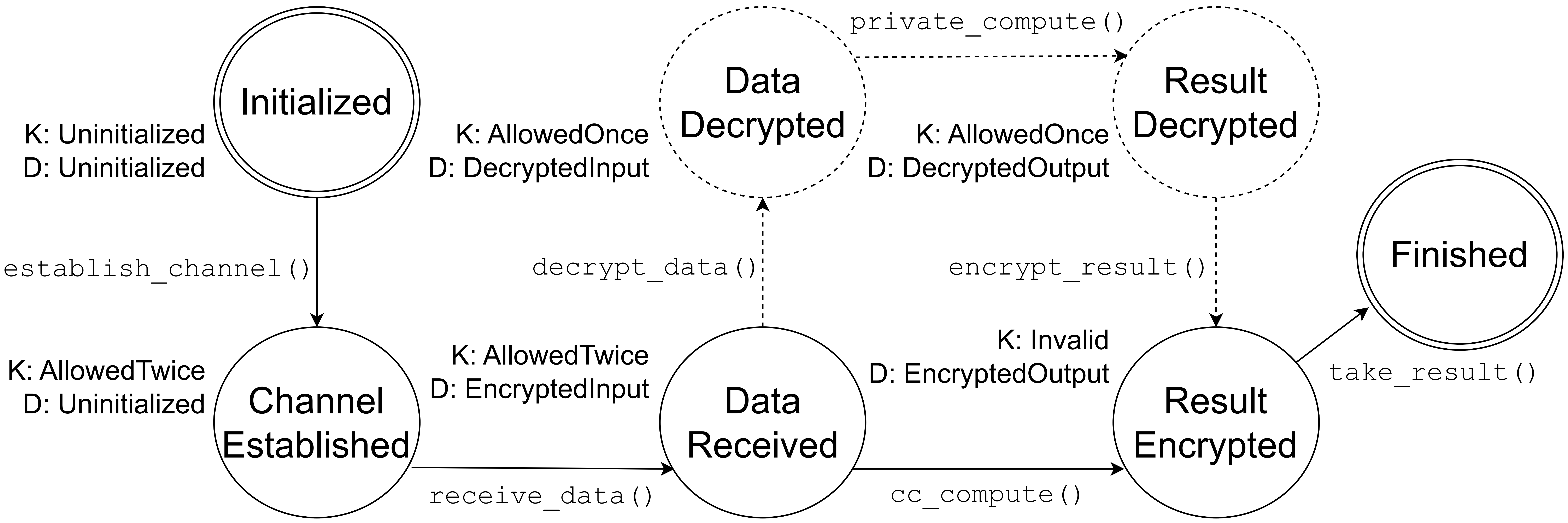
You're welcome to try and star our artifact!

Github: ya0guang/PoBF

Thanks!

Backup Slides

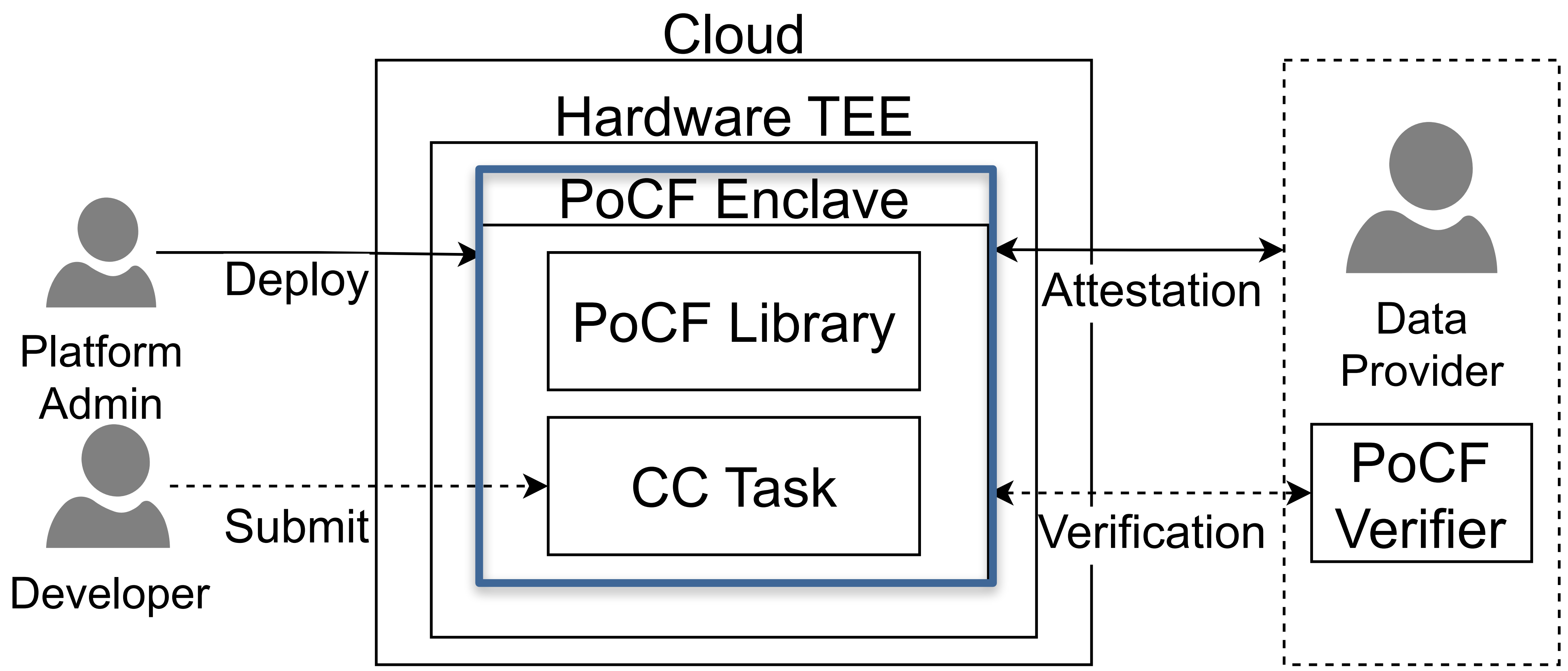
PoCF Library: TEE-Agnostic State Machine



PoCF Enclave: TEE-Specific Enclave Service

- Intel SGX
  - DCAP & EPID Attestation
  - Teaclave (Rust) SGX SDK
  - ECALL & OCALL

- AMD SEV on Azure
  - Azure Attestation Service
  - Standard Library



Effortless Porting

- Verifier invocations wrapped.
- Seamless use of standard library

```rust
#[macro_export]
macro_rules! ocall_log {
      ($str: expr) => {
          let s = alloc::format!($str);
          log(s)
      };
      ($formator:expr, $($arg:expr),+ $(,)?) => {

          let s = alloc::format!($formator, $($arg),+);
          log(s)
      };
  }

#[macro_export]
macro_rules! println {
      () => {
          ocall_log!("\n")
      };
      ($($arg:expr),+ $(,)? ) => {
        $(
            #[cfg(mirai)]
            verify!(does_not_have_tag!($arg, SecretTaint));
        )*
          ocall_log!($($arg),+);
      }
  }
```

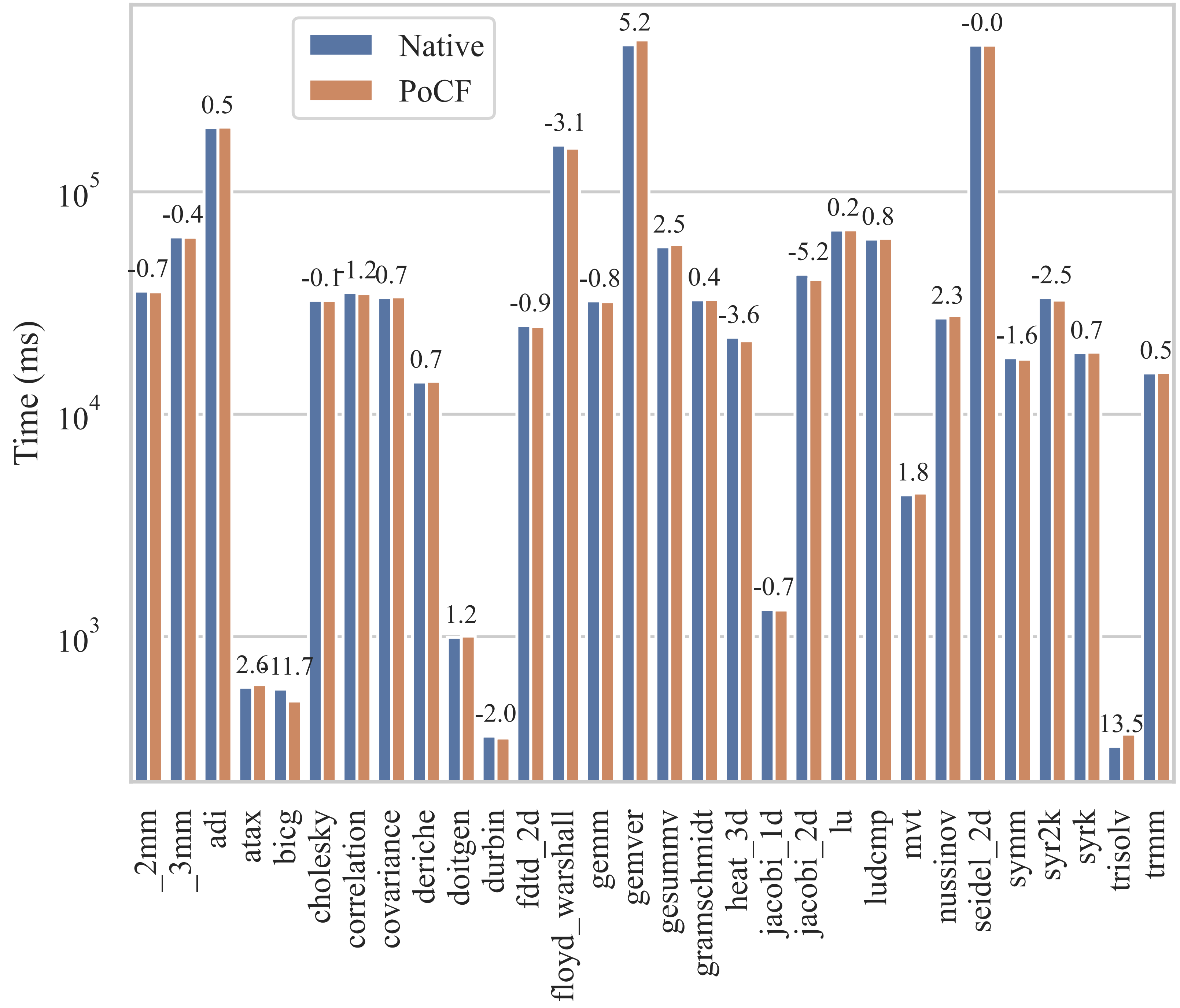
Taint Analysis: Accuracy of MIRAI

Table 4: The precision test of MIRAI categorized by Rust features.

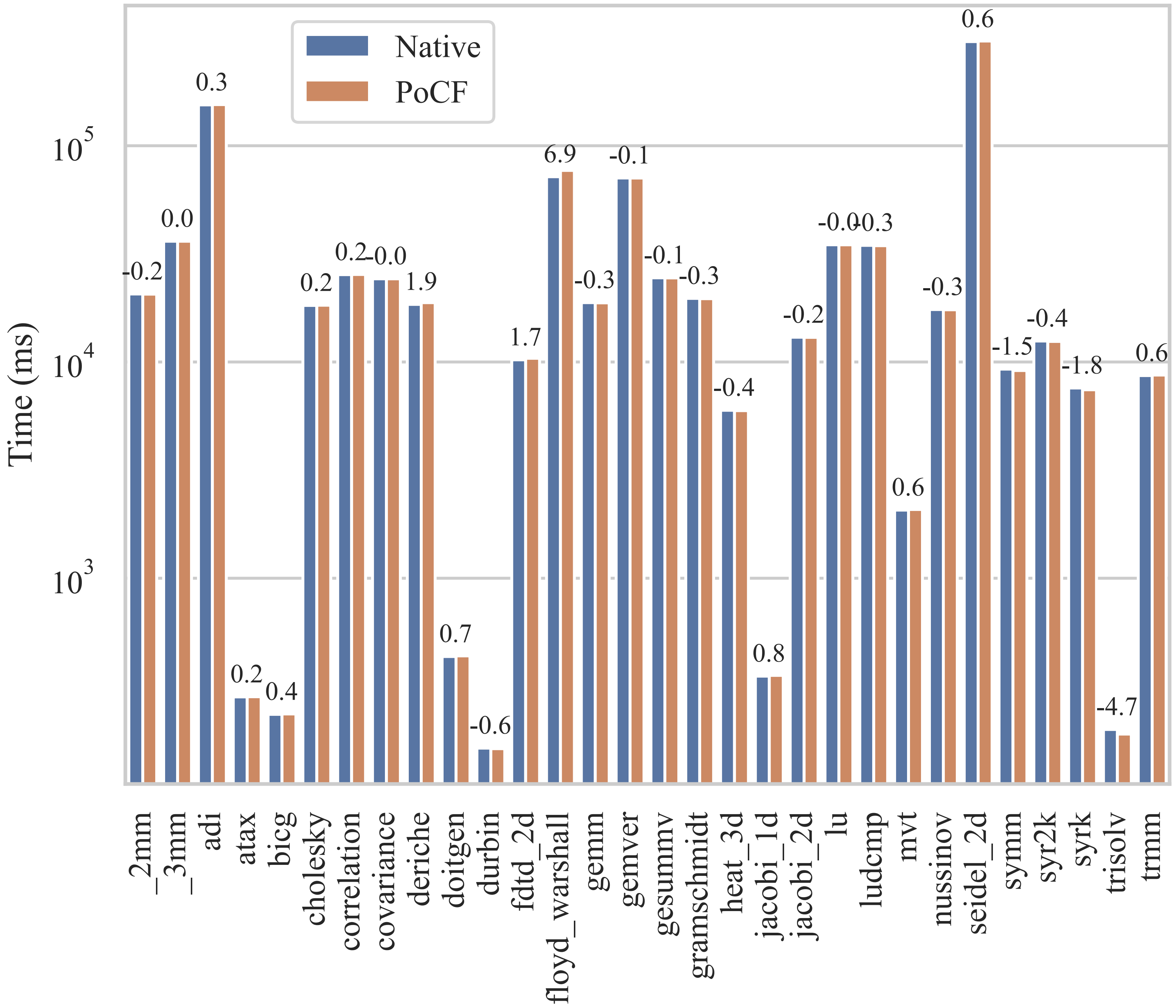
Test Name	Covered Rust Features	Expected	Actual	Missed Feature(s)
untrusted_input	Traits, generics, and arrays	✓	✓	/
control_flows	Loops, branches, and pattern matches	✗: 1; ○: 5	○: 6	/
ownership_transfer	Ownership and borrow check	✗: 2	✗: 2	/
pointers	Smart and raw pointers	✗: 4	✗: 1	Leakage by Rc<T>, Box<T>, and *const T.
complex_structs	Collections and structs	✗: 4	✗:1	Tag propagation from field to the whole struct

All the tests were analyzed by MIRAI using its strictest analysis level, i.e., MIRAI_FLAG=diag=paranoid.
✓: No data leakage; ✗: Has data leakage; ○: Possible data leakage. The number behind "✗" or "○" denotes the number of data leakages.

Microbenchmark: Polybench

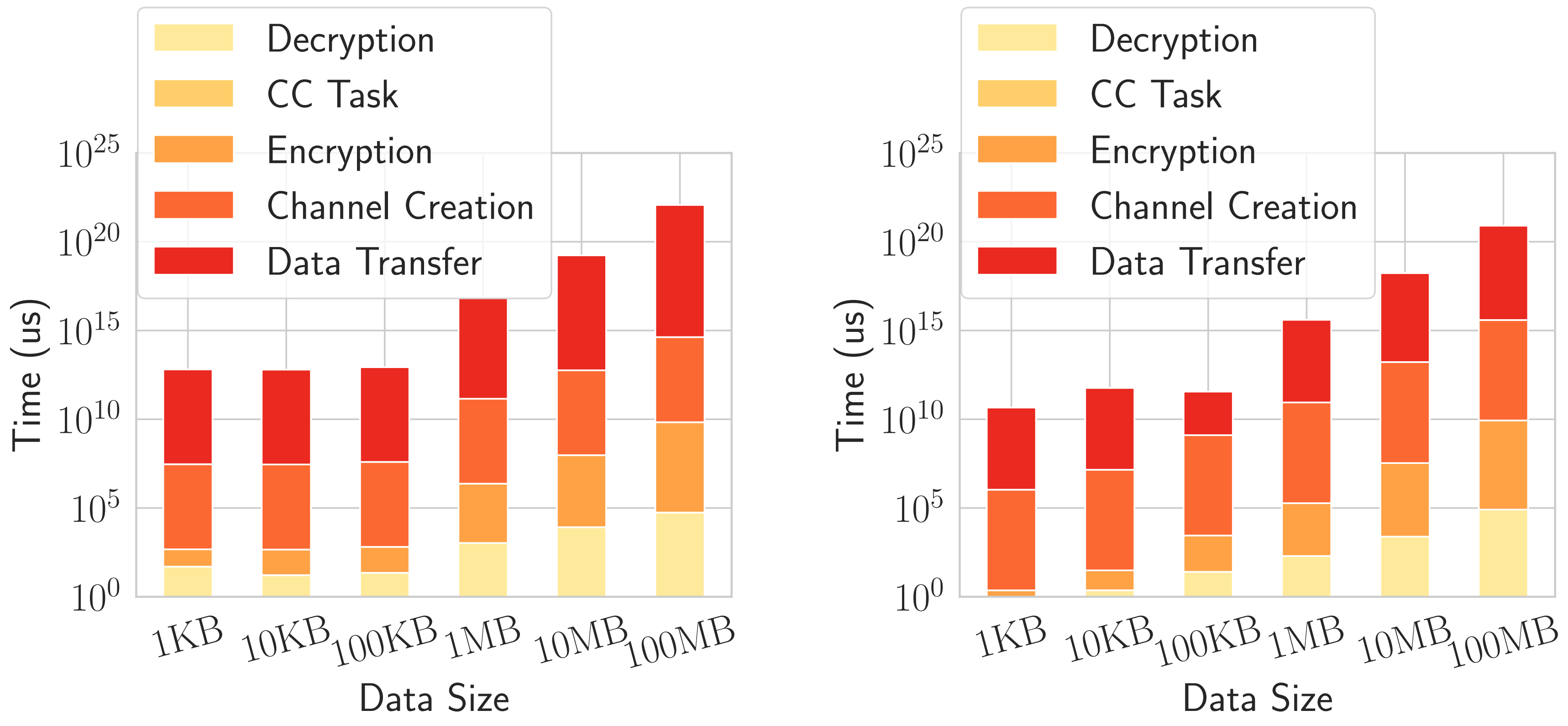


(a) Polybench: Performance of PoCF and Native on SGX.



(b) Polybench: Performance of PoCF and Native on SEV.

Microbenchmark: Overhead Analysis



(a) Cost breakup of PoCF on SGX. (b) Cost breakup of PoCF on SEV.

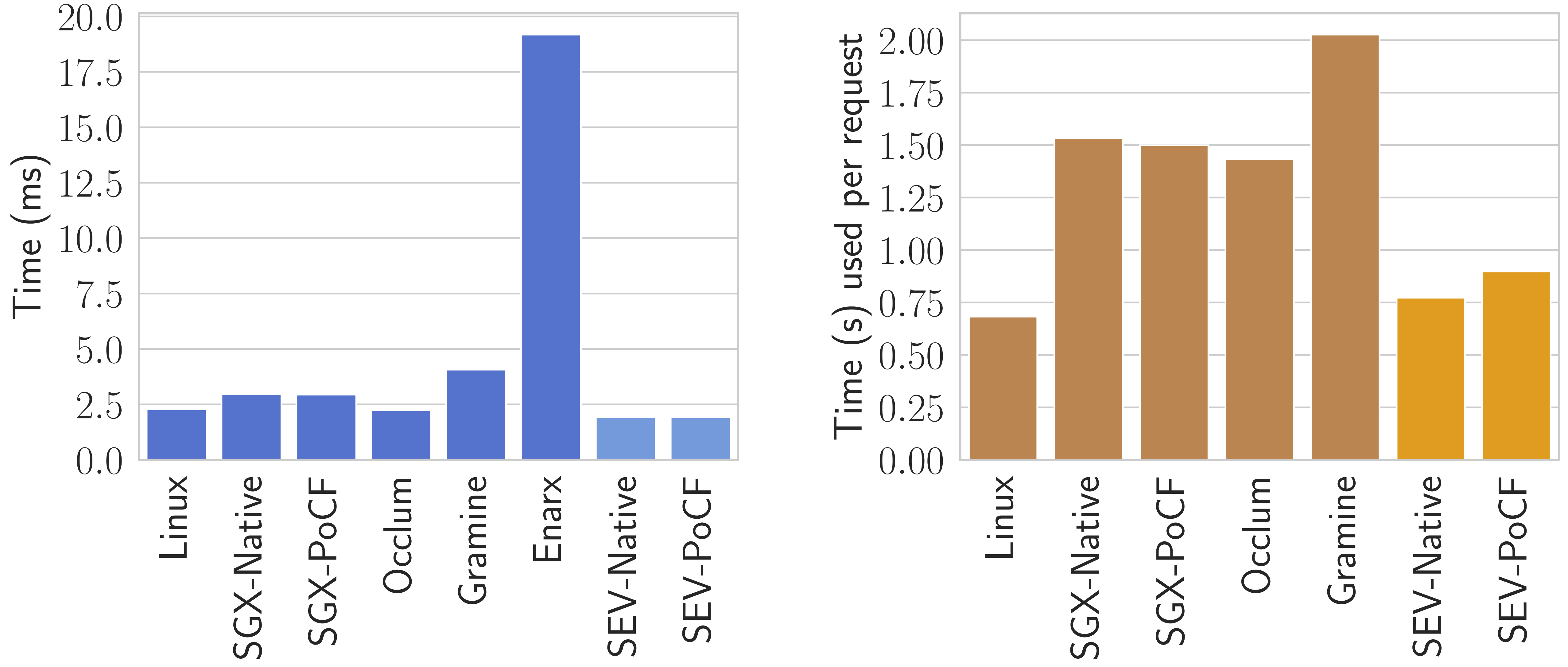
Figure 5: Identity task: Performance breakup of PoCF.

Table 5: Identity Task: Time (ms) under Different Data Sizes.

Config	1KB	10KB	100KB	1MB	10MB	100MB
NATIVE X	275.8	281.1	296.3	536.7	3026.5	28018.3
P W/O T X	278.3	280.4	298.6	541.1	3033.9	28022.9
P W/ T X	277.3	287.4	301.7	545.0	3043.7	28215.0
NATIVE V	489.1	487.3	449.7	495.6	502.0	923.3
POCF V	489.5	492.3	454.4	499.8	506.5	934.8

P: PoCF without data flow tracking; T data flow tracking; X: SGX; V: SEV

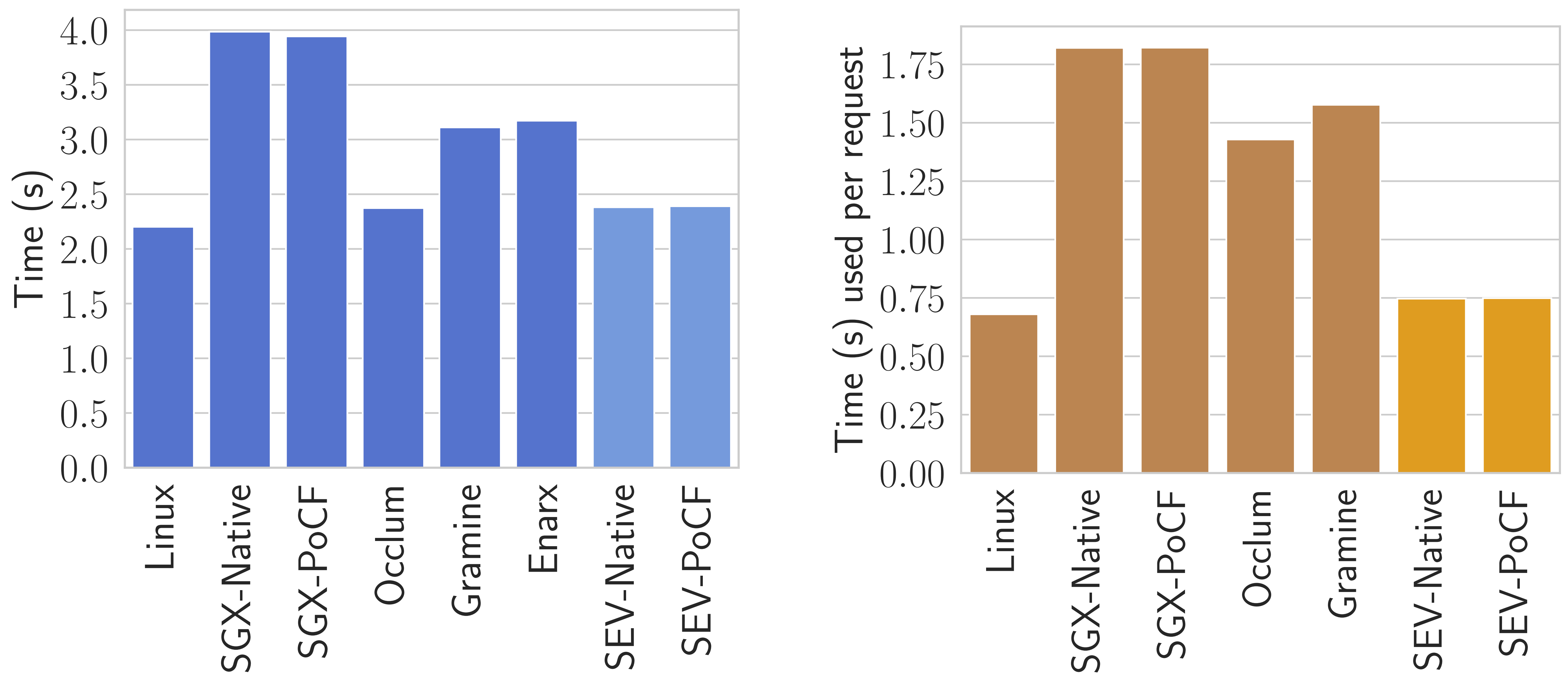
Macrobenchmark: AI Inference



(a) Single-threaded. (b) Multi-threaded.

Figure 7: Macrobenchmark: AI inference execution time.

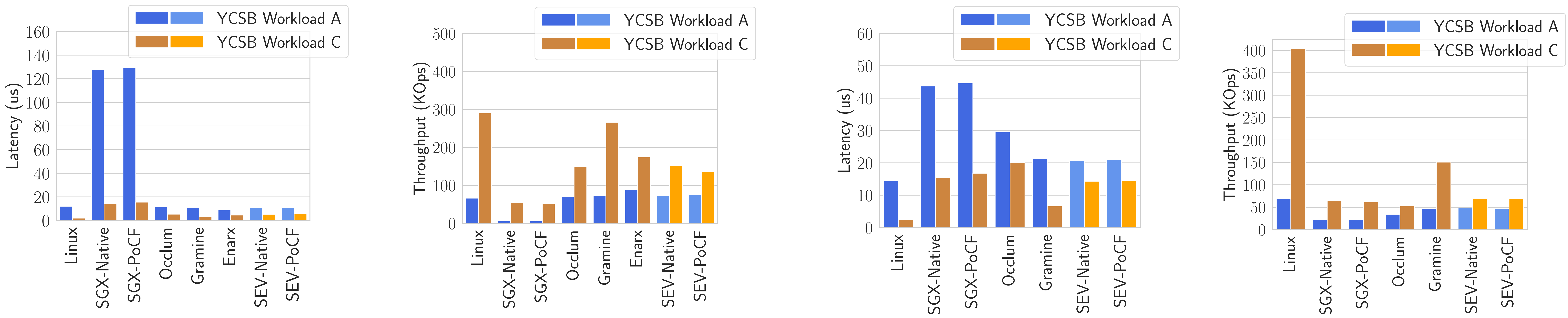
Macrobenchmark: FASTA



(a) Single-threaded. (b) Multi-threaded.

Figure 8: Macrobenchmark: FASTA execution time.

Macrobenchmark: In-memory KVDB



(a) Single-Thread Latency.

(b) Single-Thread Throughput.

(c) Multi-Thread Latency.

(d) Multi-Thread Throughput.