Practical and Efficient in-Enclave Verification of Privacy Compliance

Weijie Liu, Wenhao Wang, Hongbo Chen, Xiaofeng Wang, Yaosong Lu,
Kai Chan, Xinyu Wang, Qintao Shen, Yi Chen, Haixu Tang

Data security in Confidential Computing

- Confidential Computing as a Service

Data security in Confidential Computing

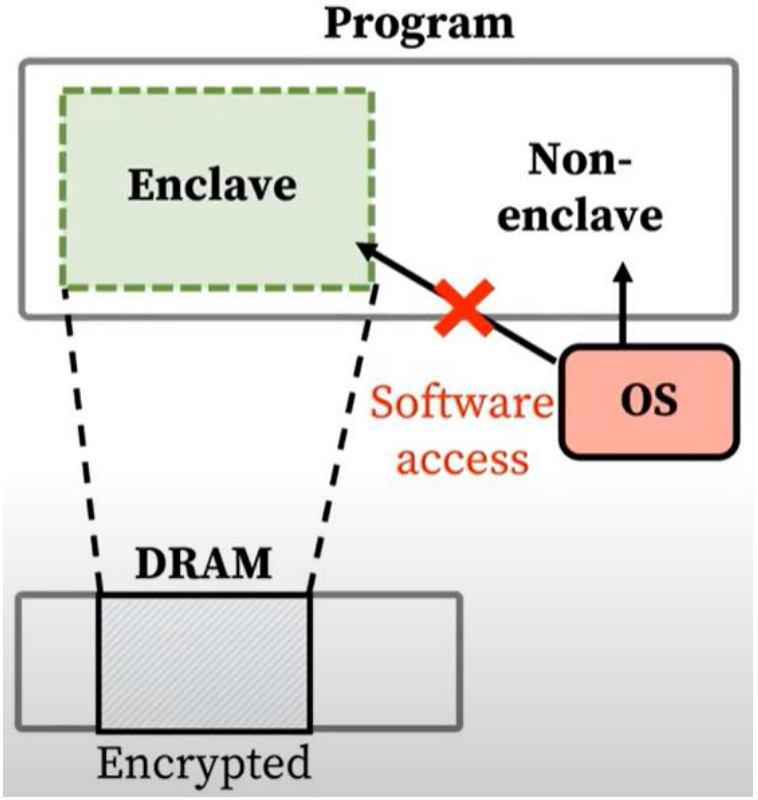
- Confidential Computing as a Service

Intel SGX is designed for Confidential Computing

- Data confinement
  - Enclave – an isolated and encrypted computing environment

Intel SGX is designed for Confidential Computing

- Data confinement
  - Enclave – an isolated and encrypted computing environment



SGX is designed for Confidential Computing

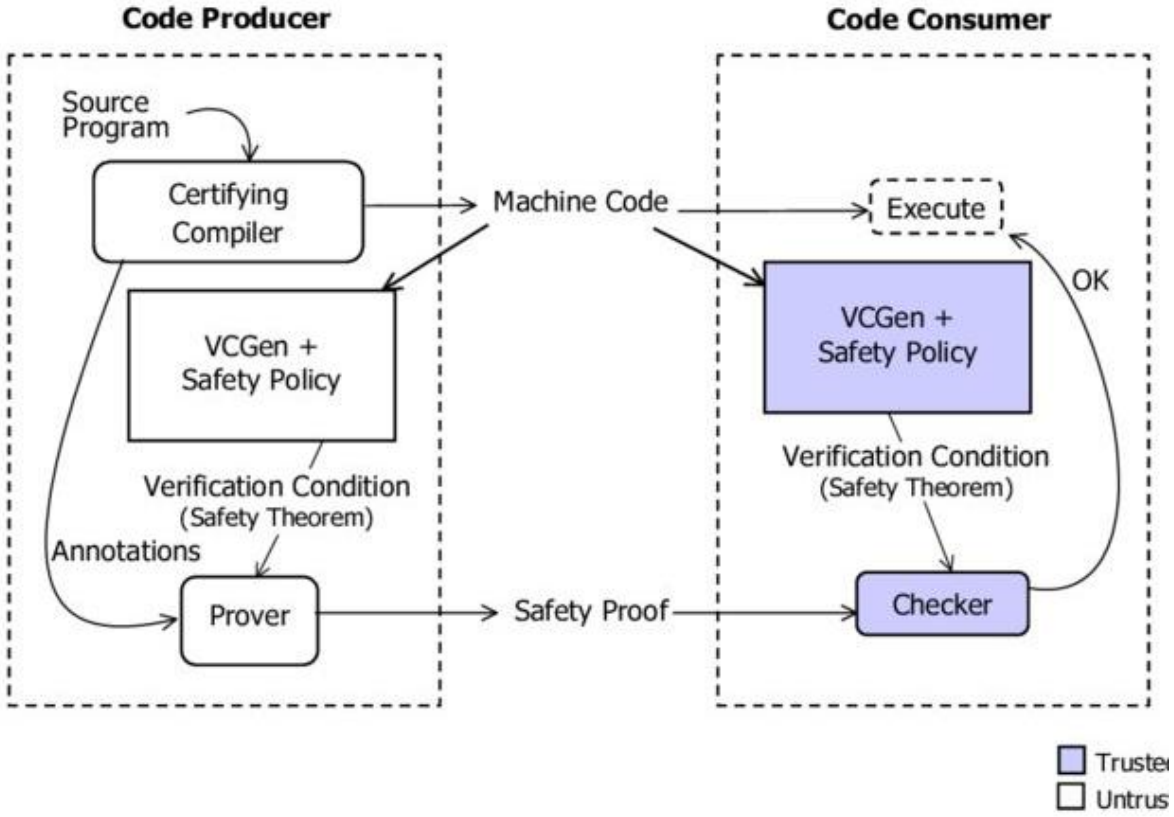
- Data confinement
  - Enclave – an isolated and encrypted computing environment
- Remote Attestation
  - Verifying a signed report – a measurement hash
  - Availability of the measurement – the program should be **public**

SGX does not protect data from untrusted code

- Programs may have exploitable bugs, or they may write information out of the enclave through corrupted pointers easily.
- Also, things become problematic when the program itself is private and cannot be exposed.

Existing program verification approaches

- Formal method
  - Traditional Proof-Carrying Code



Existing program verification approaches

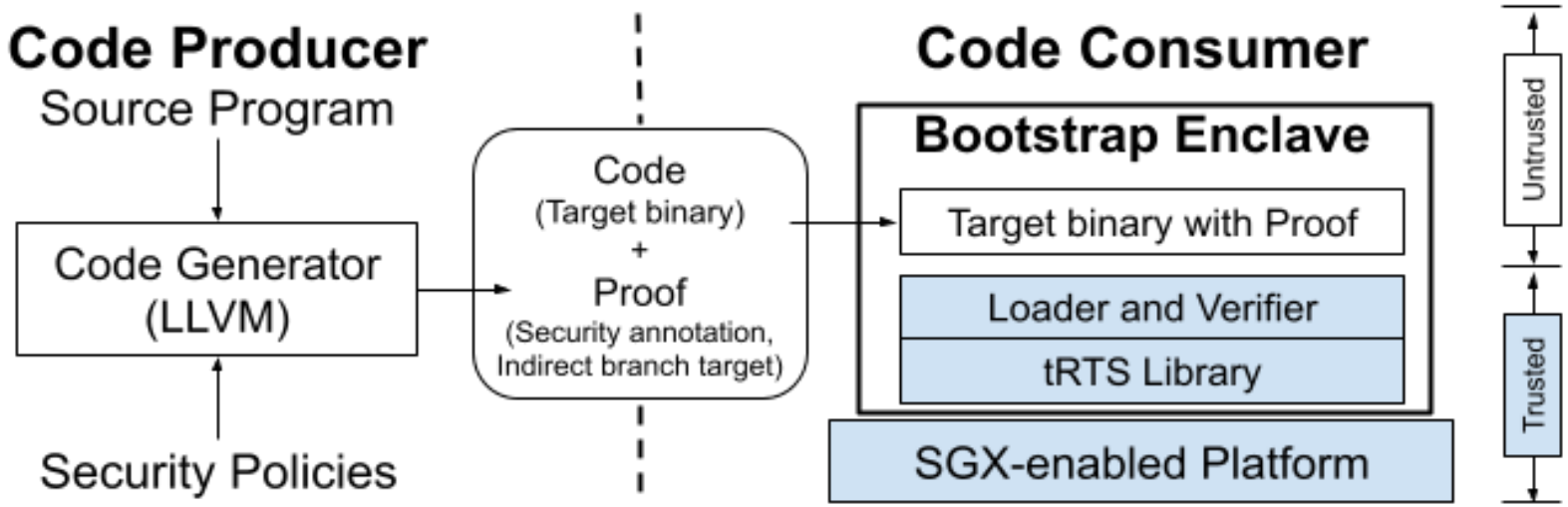
- Formal method
  - Traditional Proof-Carrying Code
    - Difficult to scale to real-world software
    - Large TCB
    - Lack of SGX runtime support

Idea

- Software-based Fault Isolation
  - More practical, but not efficient
- Proof-Carrying Code
  - Pushing the heavy-lifting part of program generation to the outside of the TCB

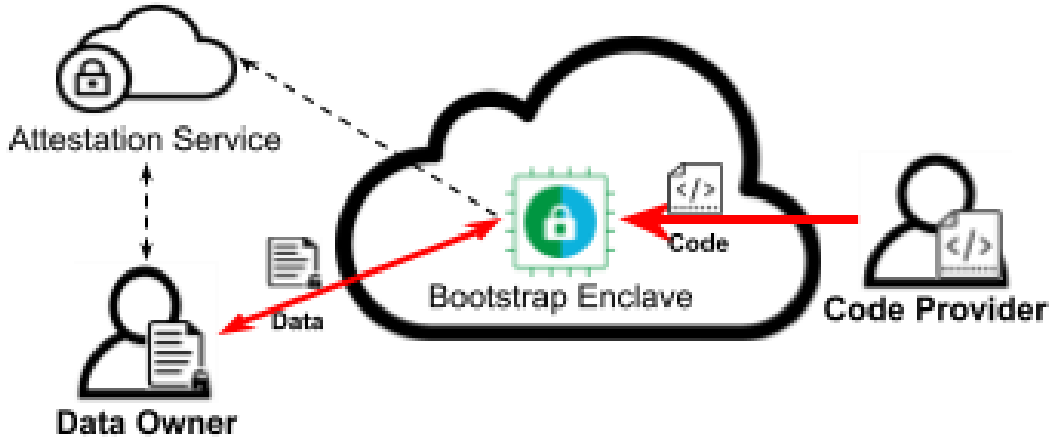
Idea

- Software-based Fault Isolation
- Proof-Carrying Code



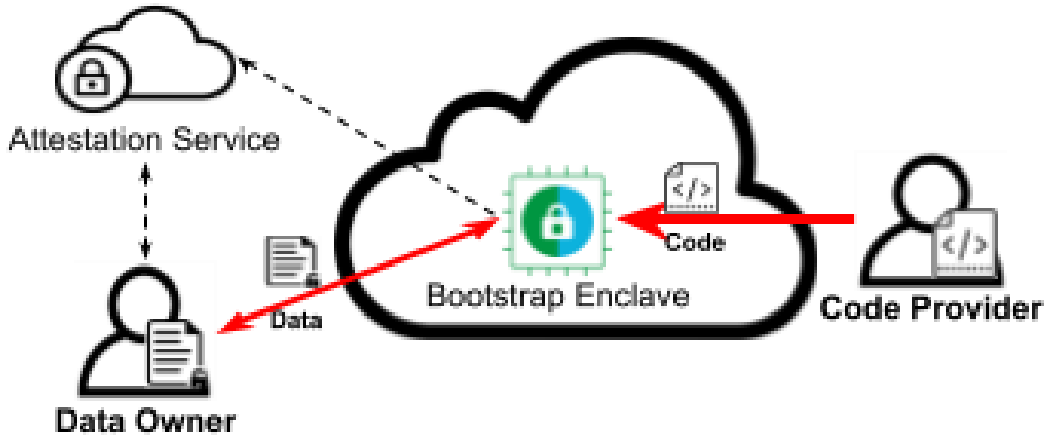
Design

- Model
  - Delegated and flexible enclave code verification (DEFLECTION)



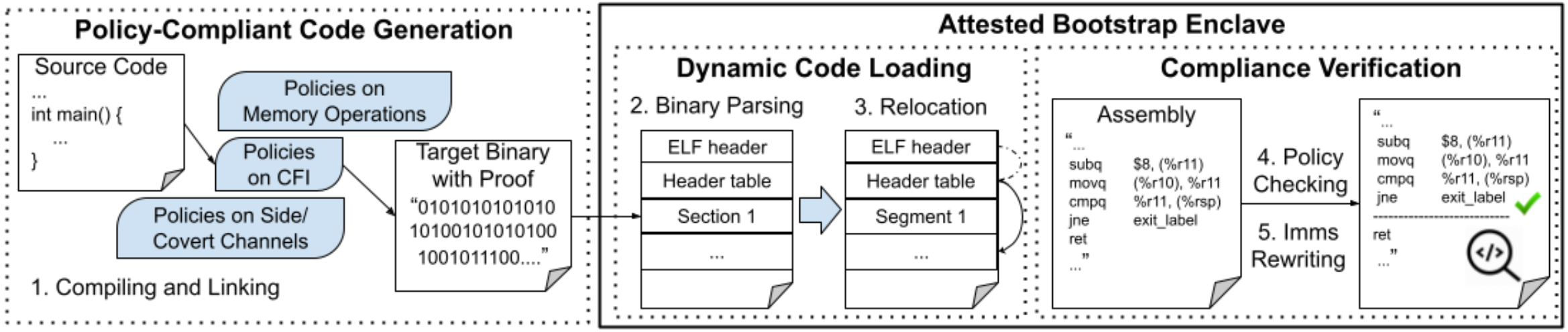
Design

- Model
  - Delegated and flexible enclave code verification (DEFLECTION)
  - Service code (target binary) is not trusted.
  - SGX hardware, its attestation protocol, and all underlying cryptographic primitives are trusted.



Design

- Workflow
  - Automated code instrumentation – by our **compiler tool-chain**
  - Attested bootstrapping – by the **loader**
  - Runtime security policy enforcement – by the **verifier** and **rewriter**



Automated program instrumentation

- Security policies
  - Enclave entry/exit control (P0)
  - Memory leak control (P1-P4)
  - Control-flow management (P5)
  - AEX-based side channel mitigation (P6)

Automated program instrumentation

- Security policies
  - Memory leak control
    - Preventing explicit out-of-enclave memory stores

```
1  pushq   %rbx      ;save execution status
2  pushq   %rax
3  leaq    [reg+imm], %rax ;load the operand
4  movq    $0x3FFFFFFFFFFFFFFF, %rbx   ;set bounds
5  cmpq    %rbx, %rax
6  ja      exit_label
7  movq    $0x4FFFFFFFFFFFFFFF, %rbx   ;set bounds
8  cmpq    %rbx, %rax
9  jb      exit_label
10 popq    %rax
11 popq    %rbx
12 movq    reg, [reg+imm]
```

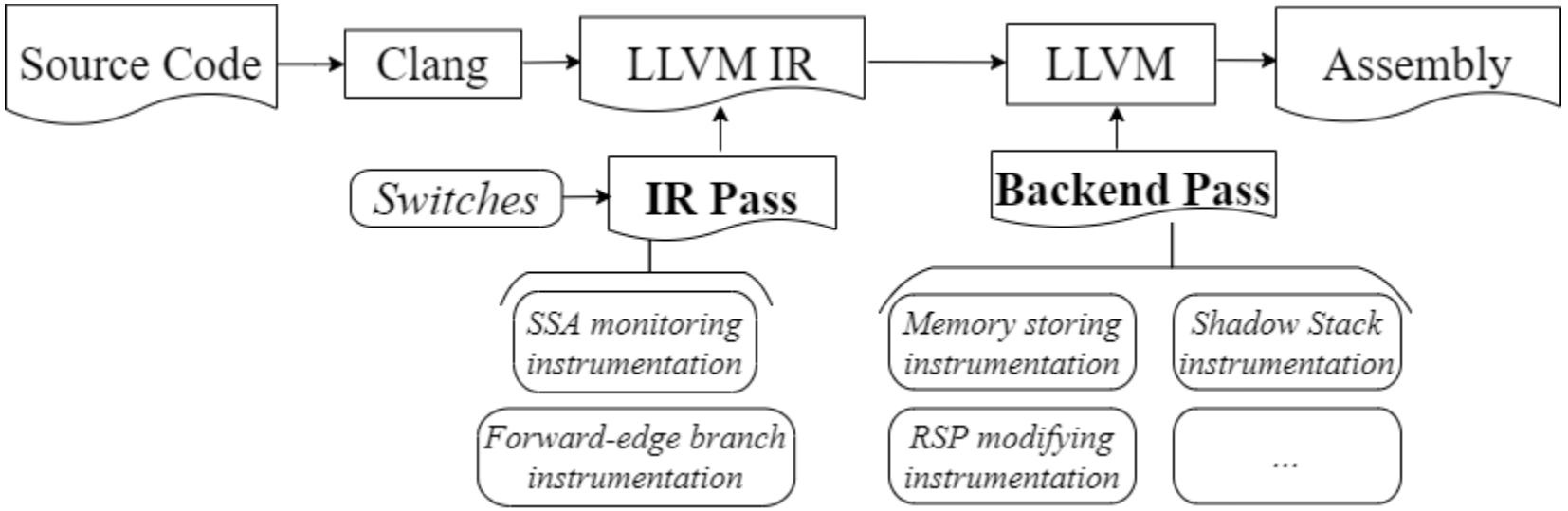
Memory storing

Automated program instrumentation

- Security policies
  - Memory leak control
    - Preventing explicit out-of-enclave memory stores (P1)
    - Preventing implicit RSP spills (P2)
    - Preventing unauthorized change to SSA/TLS (P3)
    - Preventing runtime code modification (P4)

Automated program instrumentation

- Security policies
- Code generation
  - IR level switch
  - Target level passes

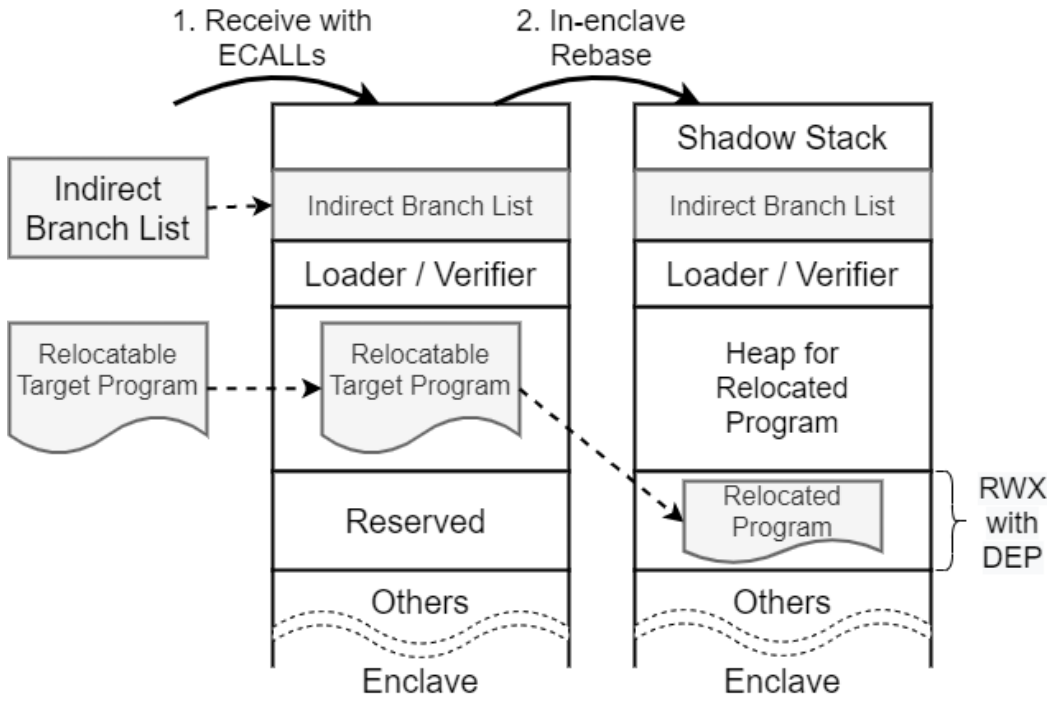


Attested bootstrapping

- Bootstrap enclave creation
- Attestation and key exchange

Attested bootstrapping

- Bootstrap enclave creation
- Attestation and key exchange
- Dynamic code loading



Runtime security policy enforcement

- Just-enough disassembling and scanning
  - Capstone
    - Recursive descent disassembling
    - Diet mode

Runtime security policy enforcement

- Just-enough disassembling
- Immediate operand rewriting
- Verification

```
1  pushq   %rbx      ;save execution status
2  pushq   %rax
3  leaq    [reg+imm], %rax  ;load the operand
4  movq    $0x3FFFFFFFFFFFFFFF, %rbx   ;set bounds
5  cmpq    %rbx, %rax
6  ja      exit_label
7  movq    $0x4FFFFFFFFFFFFFFF, %rbx   ;set bounds
8  cmpq    %rbx, %rax
9  jb      exit_label
10 popq    %rax
11 popq    %rbx
12 movq    reg, [reg+imm]
```

Security analysis

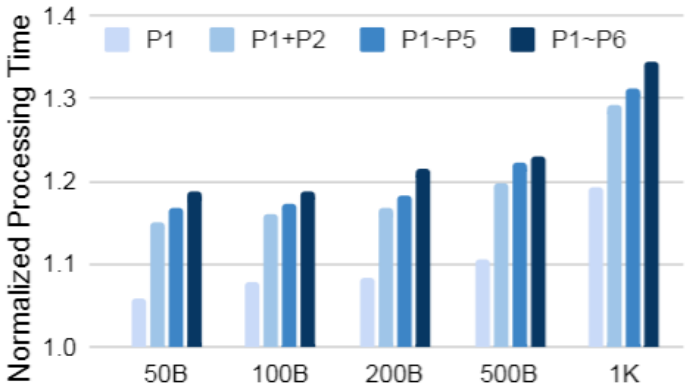
- TCB
  - Loader/Verifier – 1.3 kLoCs
  - Capstone base – 9.1 kLoCs
  - Total binary size – 3.5 MB

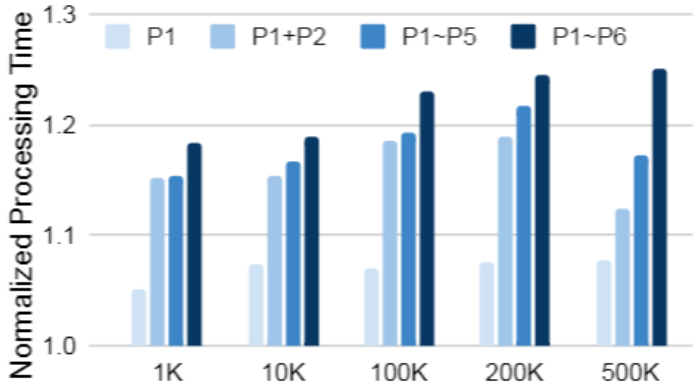
Security analysis

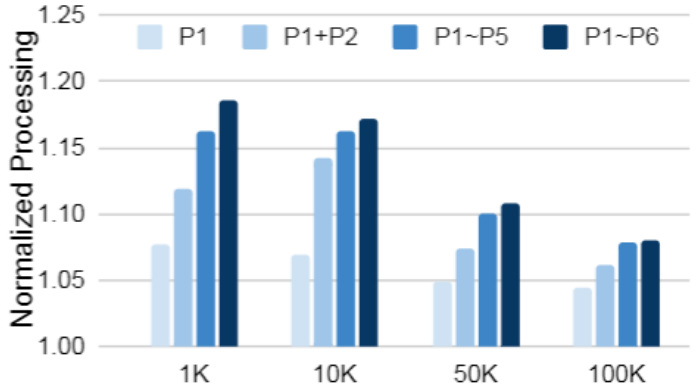
- TCB
  - Loader/Verifier – 1.3 kLoCs
  - Capstone base – 9.1 kLoCs
  - Total binary size – 3.5 MB
- Possible leakage
  - Bridge functions (P0)
  - Memory write (P1-P5)
  - Side/Covert channel (P6)
    - Hyperrace – our previous work on IEEE S&P’18

Performance

- Real-world applications
- Benchmarks
  - nBench – 0.3%~25% (P1-P5)
  - HTTPS server – 14% on average (P1-P6)

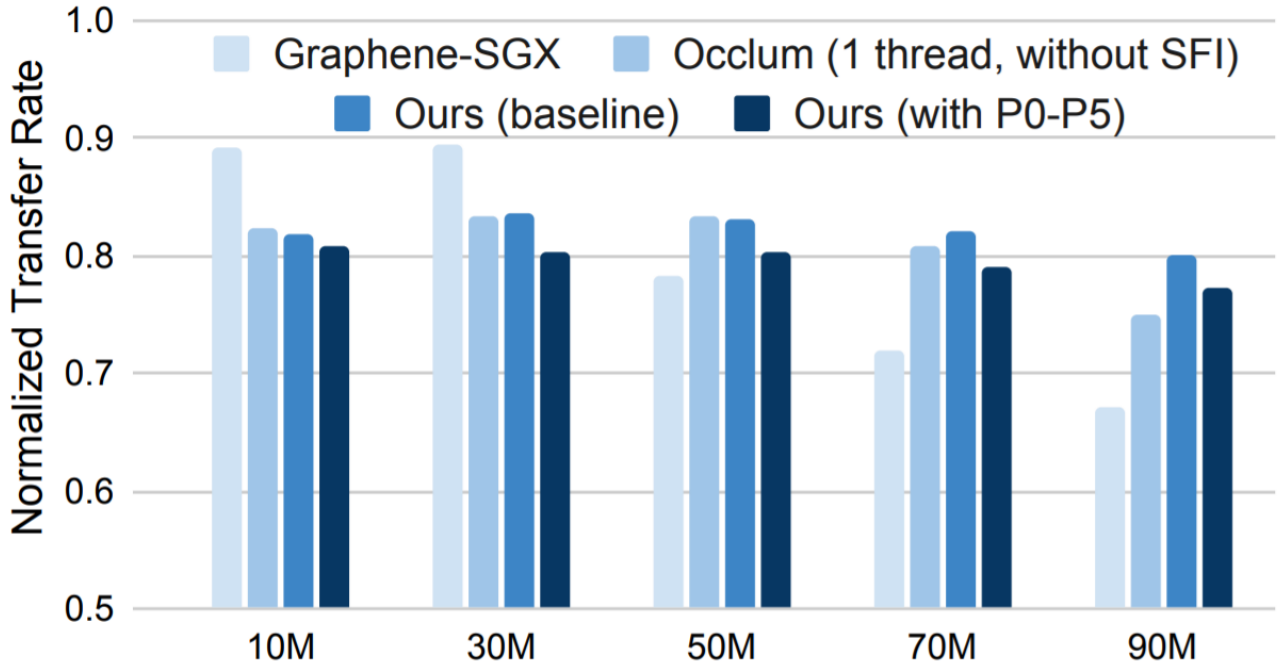






Performance

- Real-world applications
- Benchmarks
- Comparison with Graphene-SGX/Occlum



Benefit over state-of-the-arts

- Rely less on CPU hardware features
- Smaller TCB
- Side channel mitigation

Summary

- Deflection is practical and efficient.
- Deflection is relatively flexible.

Thanks

- https://github.com/StanPlatinum/Deflection