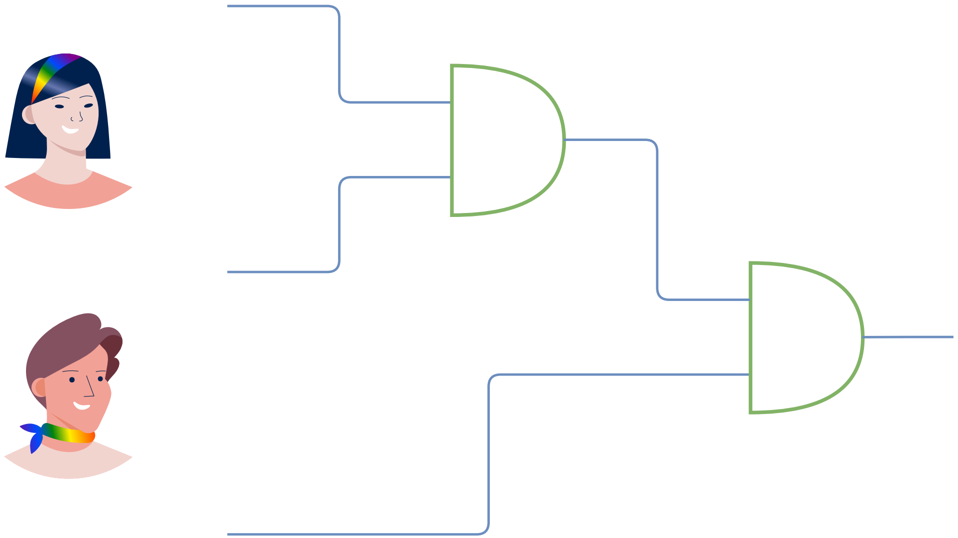
SEEC: Memory Safety Meets Efficiency in Secure Two-Party Computation

Robin Hundt



Agenda

`#[sub_circuit]`

Memory Safety

SEEC

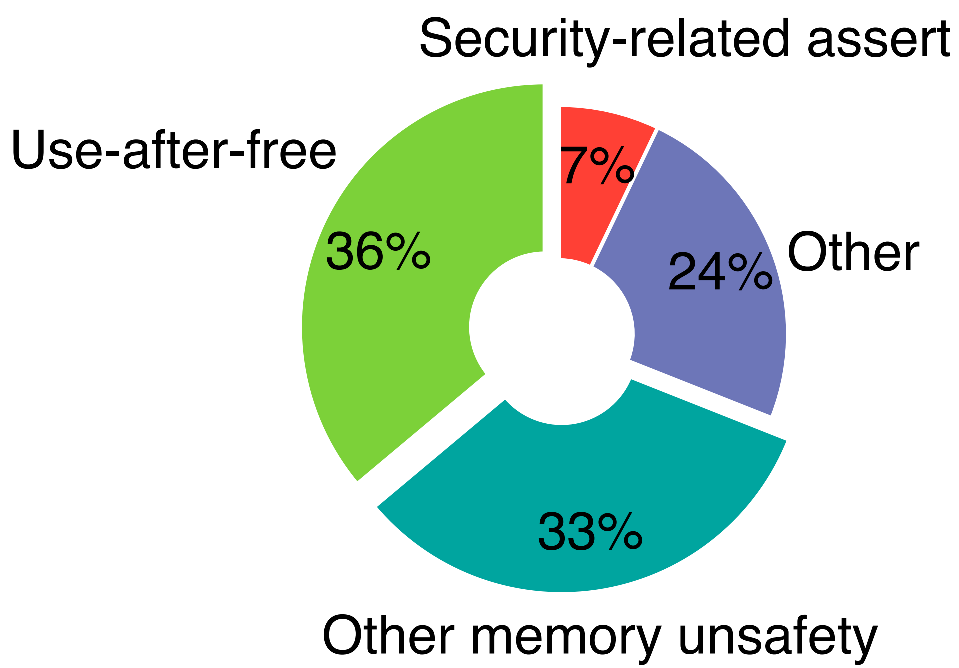
Functions in MPC

Sub-Circuits

Benchmarks

Motivation

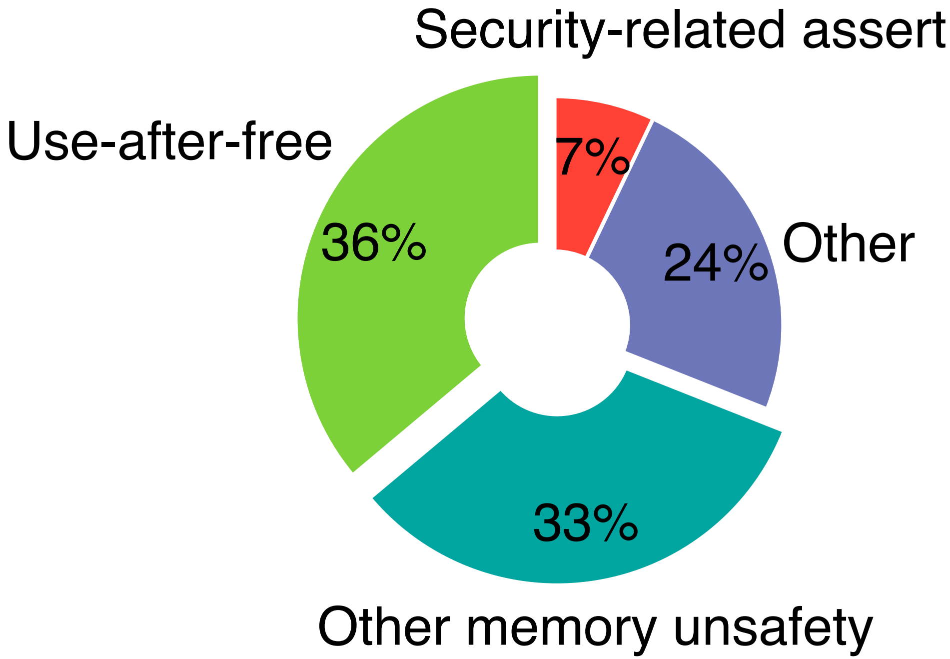
Safety



Source: The Chromium Projects - Memory Safety

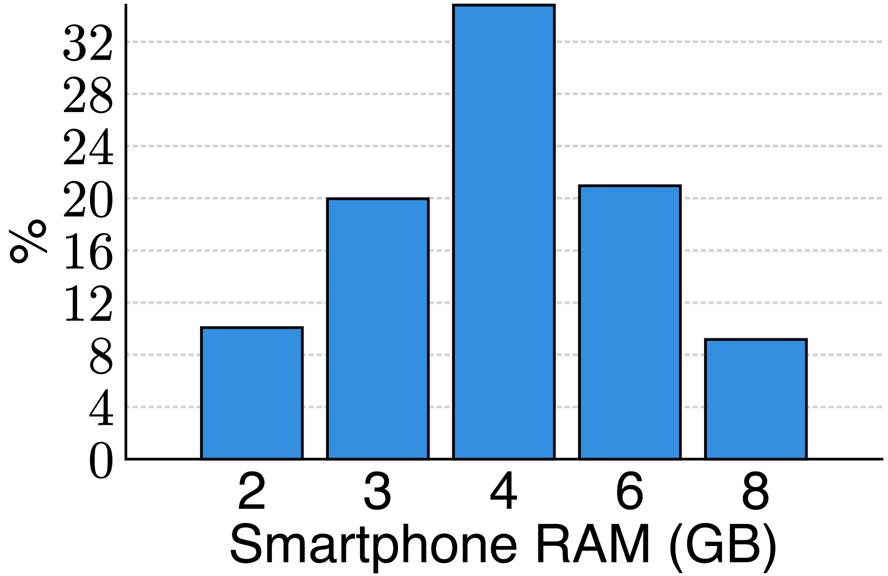
Motivation

Safety



Source: The Chromium Projects - Memory Safety

Efficiency



Source: scientiamobile

SEEC Executes Enormous Circuits (SEEC)

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

2PC GMW (A/B) [GMW87,Bea92]
2PC GMW (A+B) [DSZ15]
ASTRA (B*) [CCPS19]
ABY2.0 (B*) [PSSY21]
OT: [ASLZ13], Silent OT [BCG+19]

High-Level eDSL
(SIMD) Sub-Circuits
Function (In-)Dependent Setup
Extensibility w/o forking
Cross-Platform

ABY [DSZ15]
MP-SPDZ [Kel20]
MOTION [BDST22]
SEEC

* Partial Implementation

Functions in Traditional Programs

```
fn process(args: [bool; 2]) -> bool {
  // ... process the arguments
}

let [a, b, c] = read_data();

let result_0 = process([a, b]);
// use result_0 for next process call
let result_1 = process([result_0, c])
```

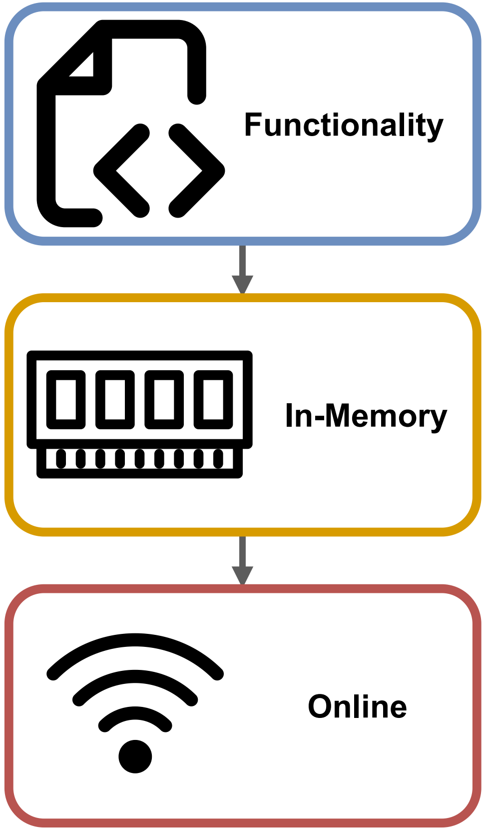
Circuit Reuse in Secure Programs

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

```
fn process(args: [bool; 2]) -> bool {
  // ... process the arguments
}

let [a, b, c] = read_data();

let result_0 = process([a, b]);
// use result_0 for next process call
let result_1 = process([result_0, c])
```

```
fn process(args: [SBool; 2]) -> SBool {
  // ... process the arguments
}

let [a, b, c] = read_data();

let result_0 = process([a, b]);
// use result_0 for next process call
let result_1 = process([result_0, c])
```

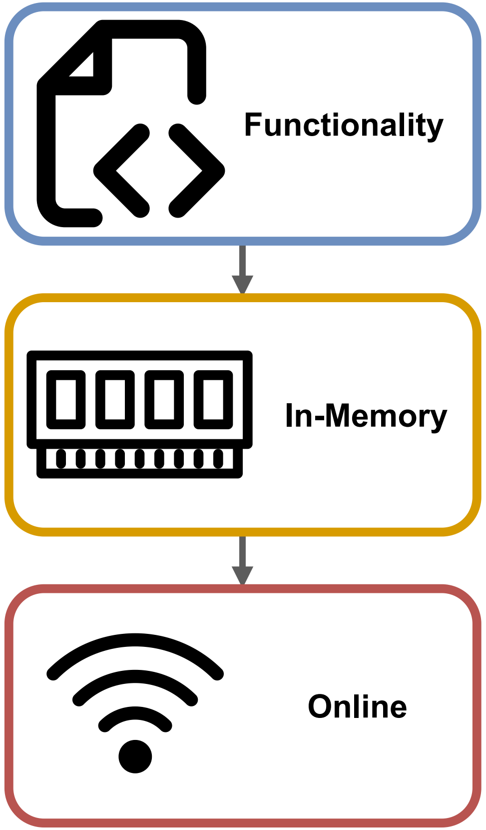
Sub-Circuits in GMW: Challenges




```
process(a);
process(b);
```

a and b are indepent

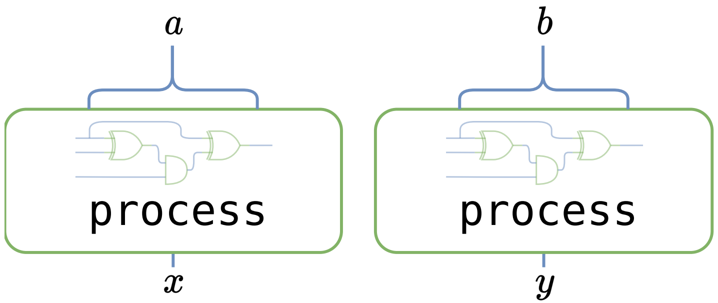
Sub-Circuits in GMW: Challenges



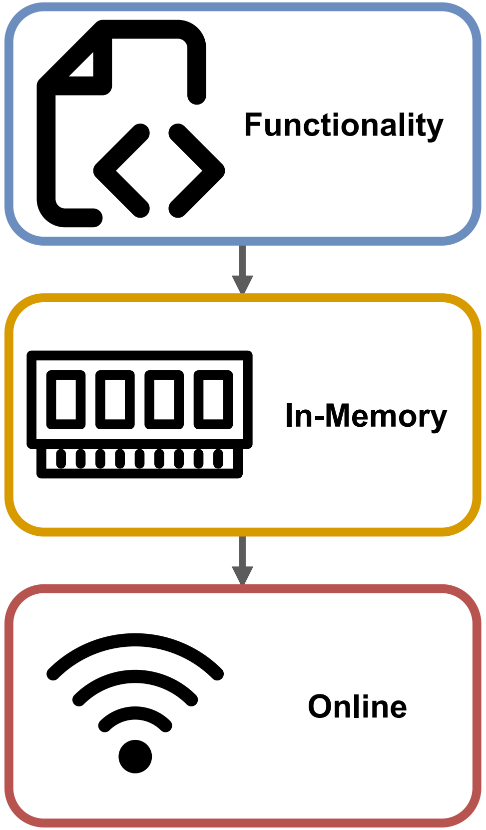

```
process(a);
process(b);
```

a and b are indepent

```
process:
  # ...
  ret

call process
call process
```



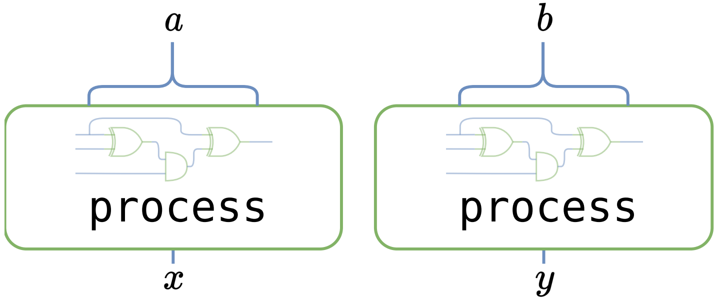
Sub-Circuits in GMW: Challenges




```
process(a);
process(b);
```

a and b are indepent

```
process:
  # ...
  ret

call process
call process
```



→ Increased rounds
→ `process` only once in memory

→ Concurrent evaluation
→ Increased Memory

SEEC: eDSL Enables Efficient Circuit Reuse

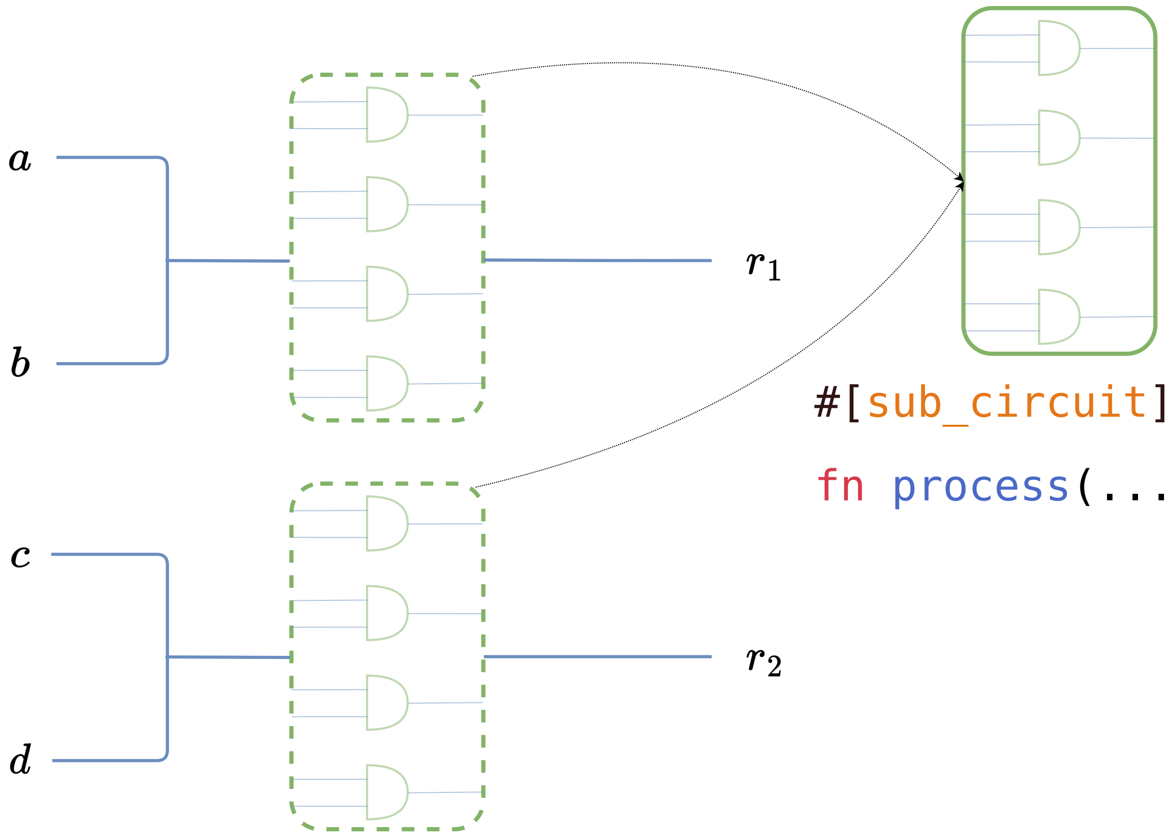
```
#[sub_circuit]
fn process(a: Vec<Secret>, b: Vec<Secret>)
    -> Vec<Secret> {
  a.into_iter().zip(b).map(|(el_a, el_b)| {

    el_a & el_b

  }).collect()
}
```

SEEC: eDSL Enables Efficient Circuit Reuse

```rust
#[sub_circuit]
fn process(a: Vec<Secret>, b: Vec<Secret>)
    -> Vec<Secret> {
  a.into_iter().zip(b).map(|(el_a, el_b)| {

    el_a & el_b

  }).collect()
}
```

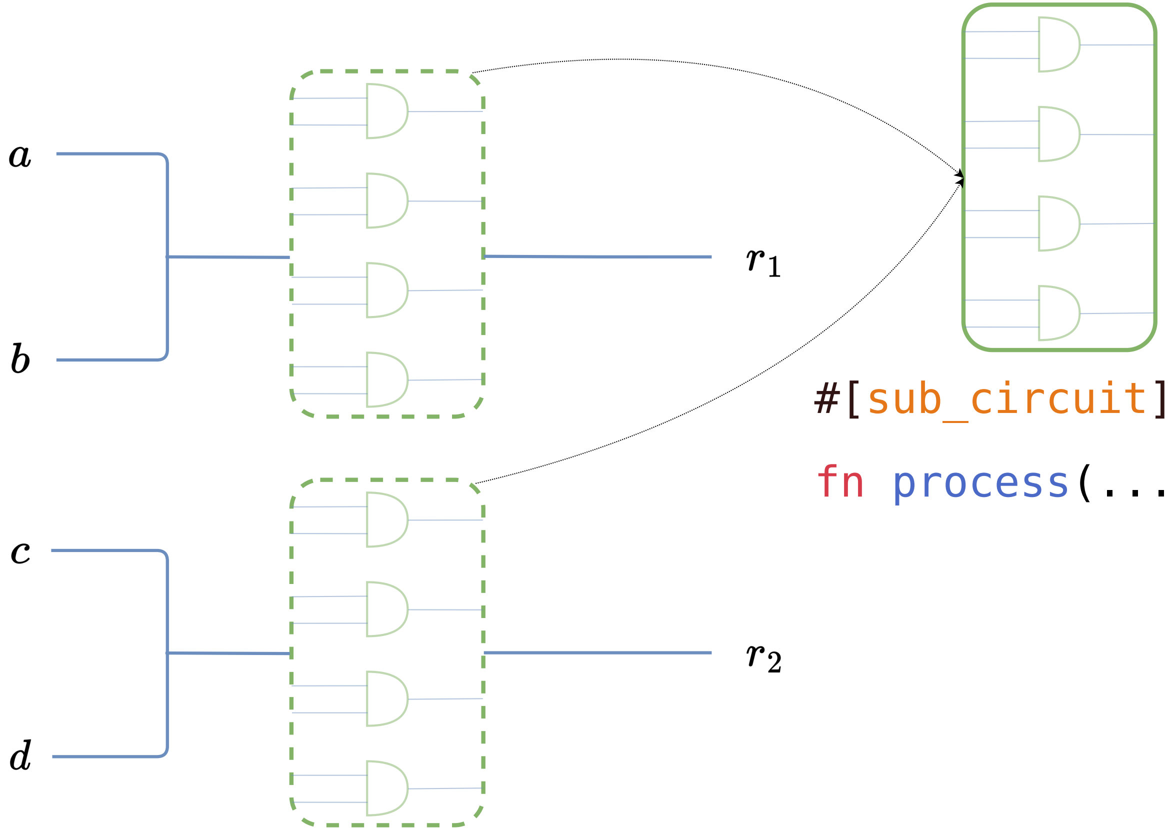
```rust
let (a, b, c, d) = init_data();
// process is called as
function.
let r1 = process(a, b);

let r2 = process(c, d);
```

SEEC: Sub-Circuits Are Not Inlined




```
#[sub_circuit]
fn process(...)
```

SEEC: Sub-Circuits Are Not Inlined




```
#[sub_circuit]
fn process(...)
```

Online

- Layer iteration **as if** inlined (DL)
- Partial and concurrent evaluation

Single Instruction, Multiple Data

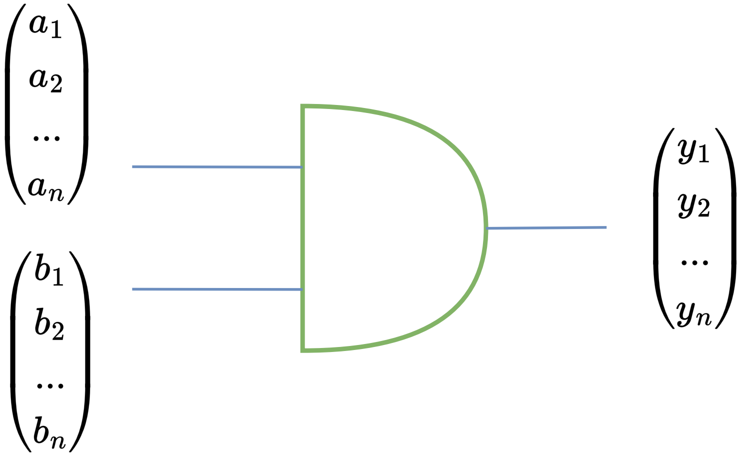


Figure 1: Traditional SIMD, e.g., in MOTION [BDST22].

Single Instruction, Multiple Data

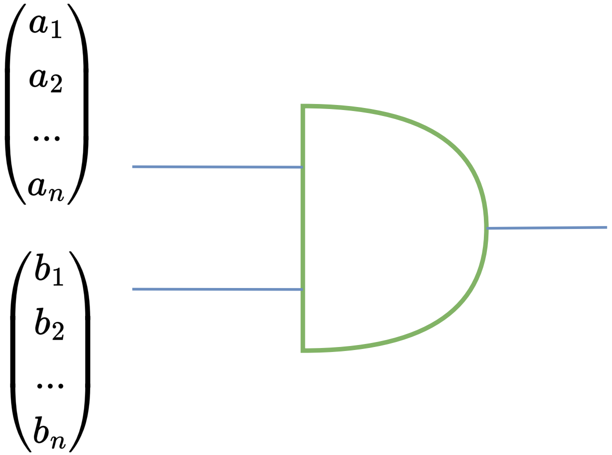


Figure 1: Traditional SIMD, e.g., in MOTION [BDST22].

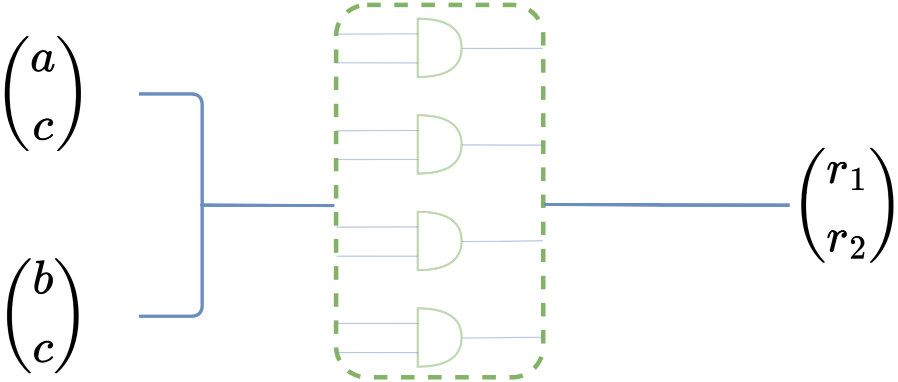


Figure 2: SIMD Sub-Circuits in SEEC.

SEEC: Optimizations

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (FG)

Stored MT Streaming (IS)

SEEC: Optimizations

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (FG)

- Unneeded gate outputs are freed
- Only applies to SIMD circuits

Stored MT Streaming (IS)

TECHNISCHE UNIVERSITÄT DARMSTADT

SEEC: Optimizations

Static Layers (SL)

- Transforms Dynamic Layer (DL) representation
- Layers are precomputed for every call site
- Precomputed layers are stored deduplicated

Early Deallocation (FG)

- Unneeded gate outputs are freed
- Only applies to SIMD circuits

Stored MT Streaming (IS)

- MTs are computed and stored in a file
- Online: read on-demand in batches from the file

Evaluation

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Frameworks

- ABY [DSZ15]
- MP-SPDZ [Kel20]
- MOTION [BDST22]
- SEEC (SL / FG / IS)

Environment

Fast-LAN / LAN / WAN

$1, 2, \ldots, 32$ Threads

Heaptrack[1]

[1] https://github.com/KDE/heaptrack

Evaluation

Frameworks

- ABY [DSZ15]
- MP-SPDZ [Kel20]
- MOTION [BDST22]
- SEEC (SL / FG / IS)

Environment

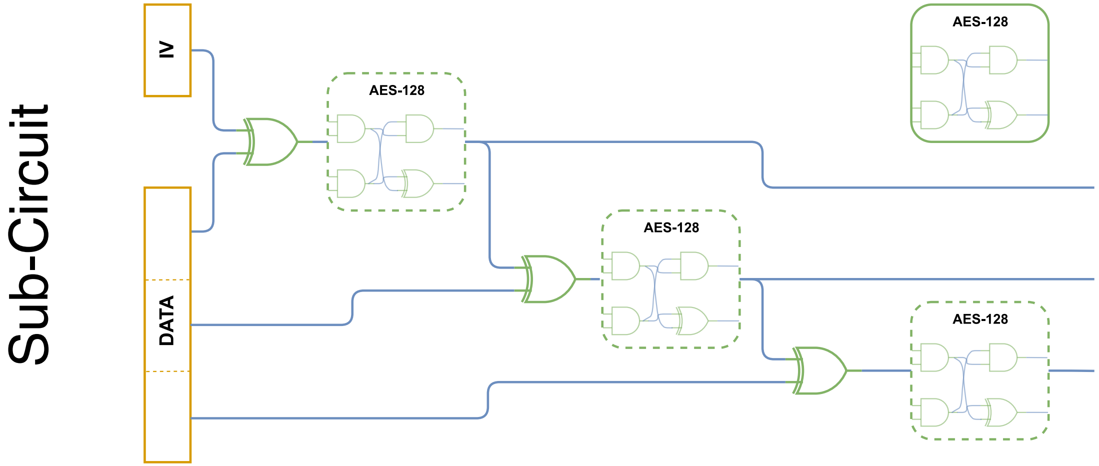
Fast-LAN / LAN / WAN

$1, 2, \ldots, 32$ Threads

Heaptrack[1]

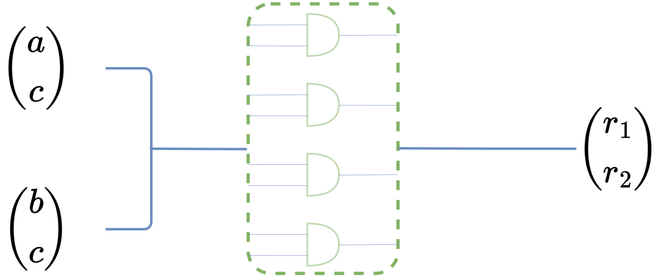
[1] https://github.com/KDE/heaptrack

Circuits

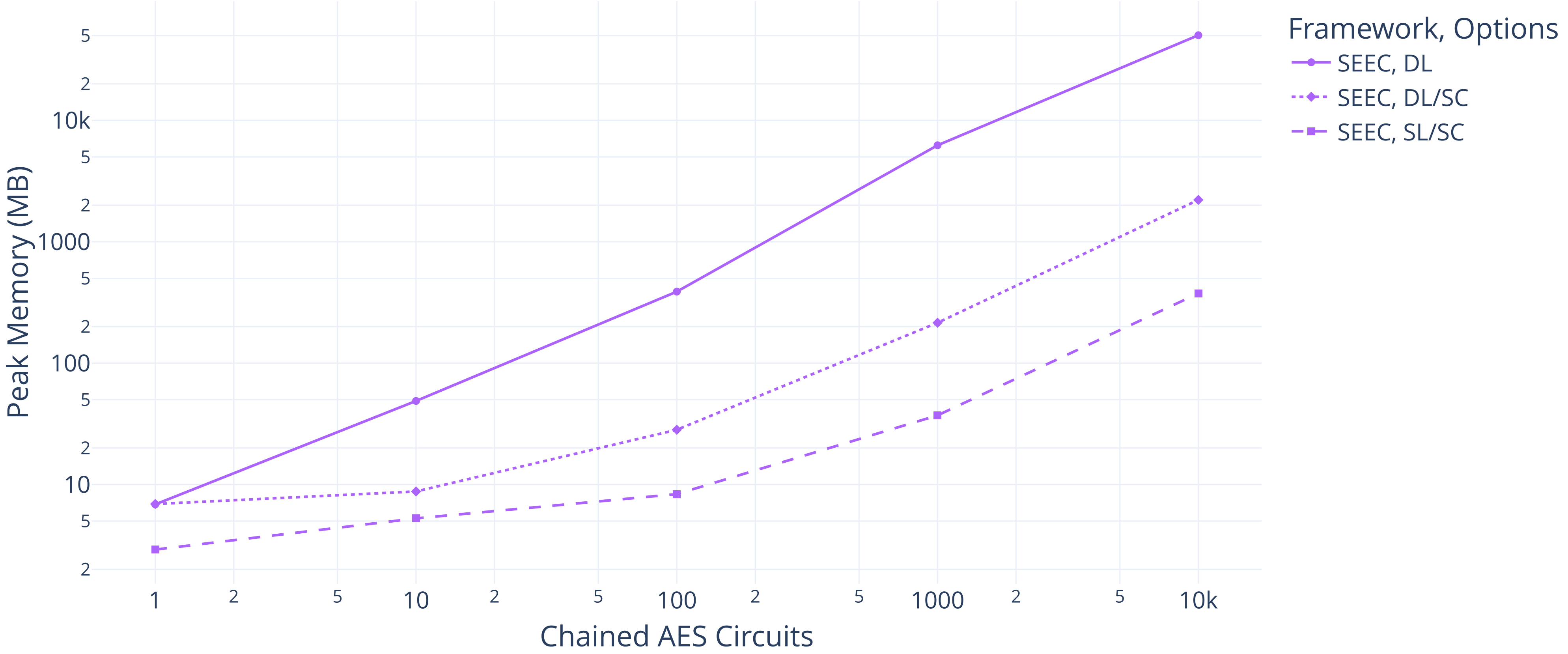


SIMD

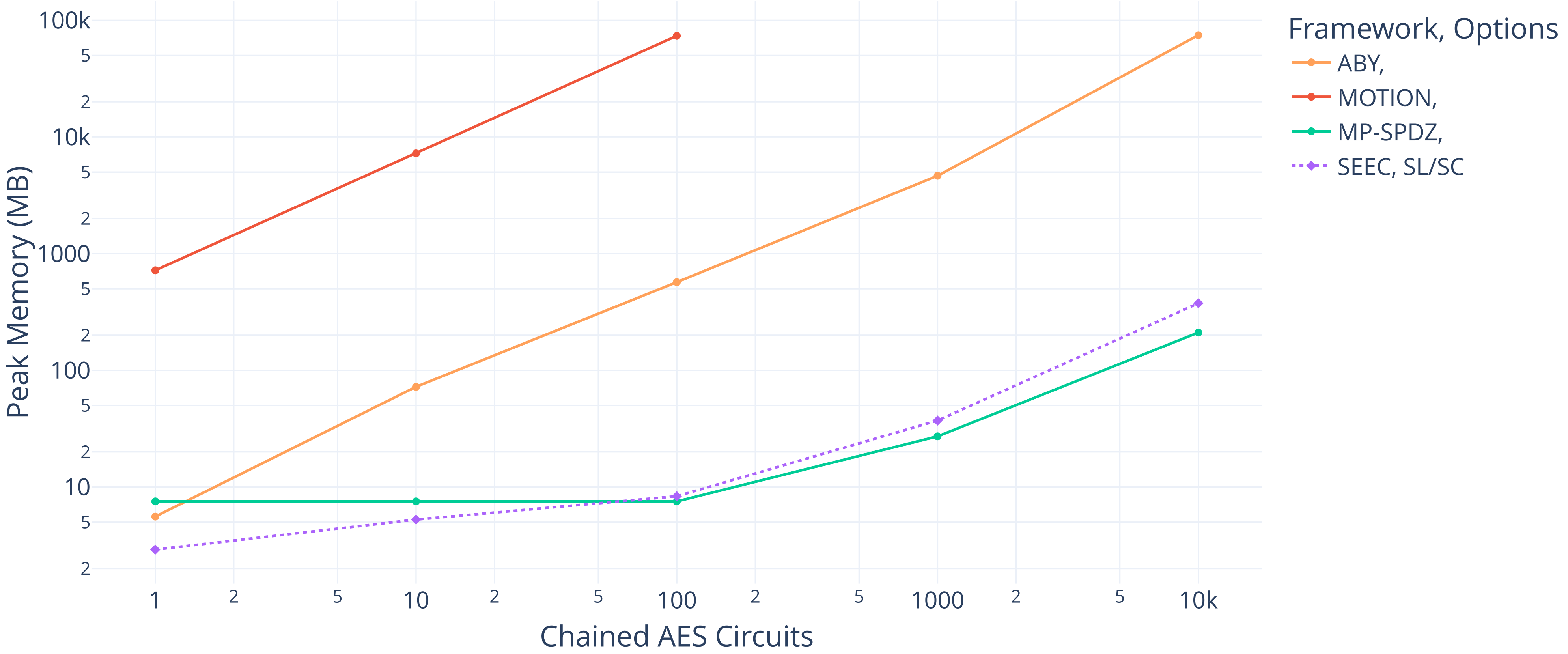
- AES-128
- SHA-256



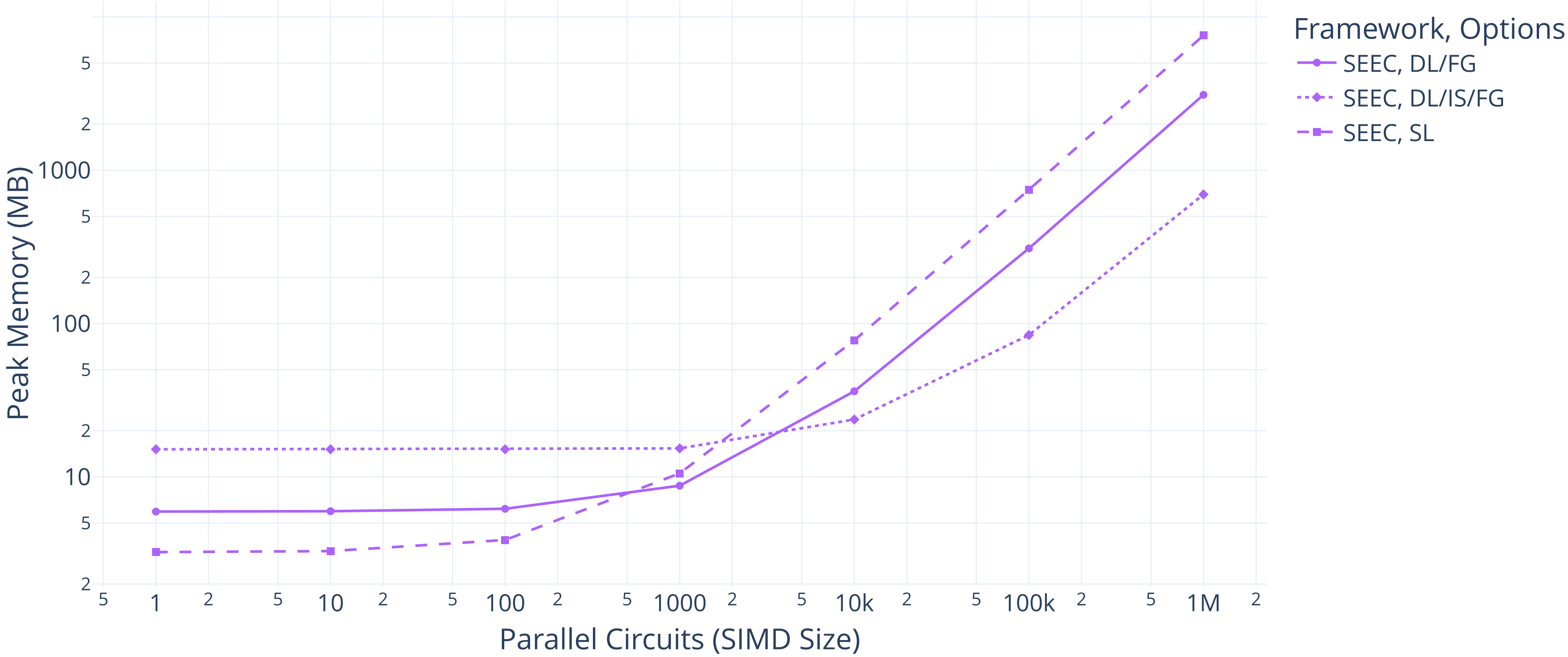
AES-CBC: Reduced Memory via Sub-Circuits



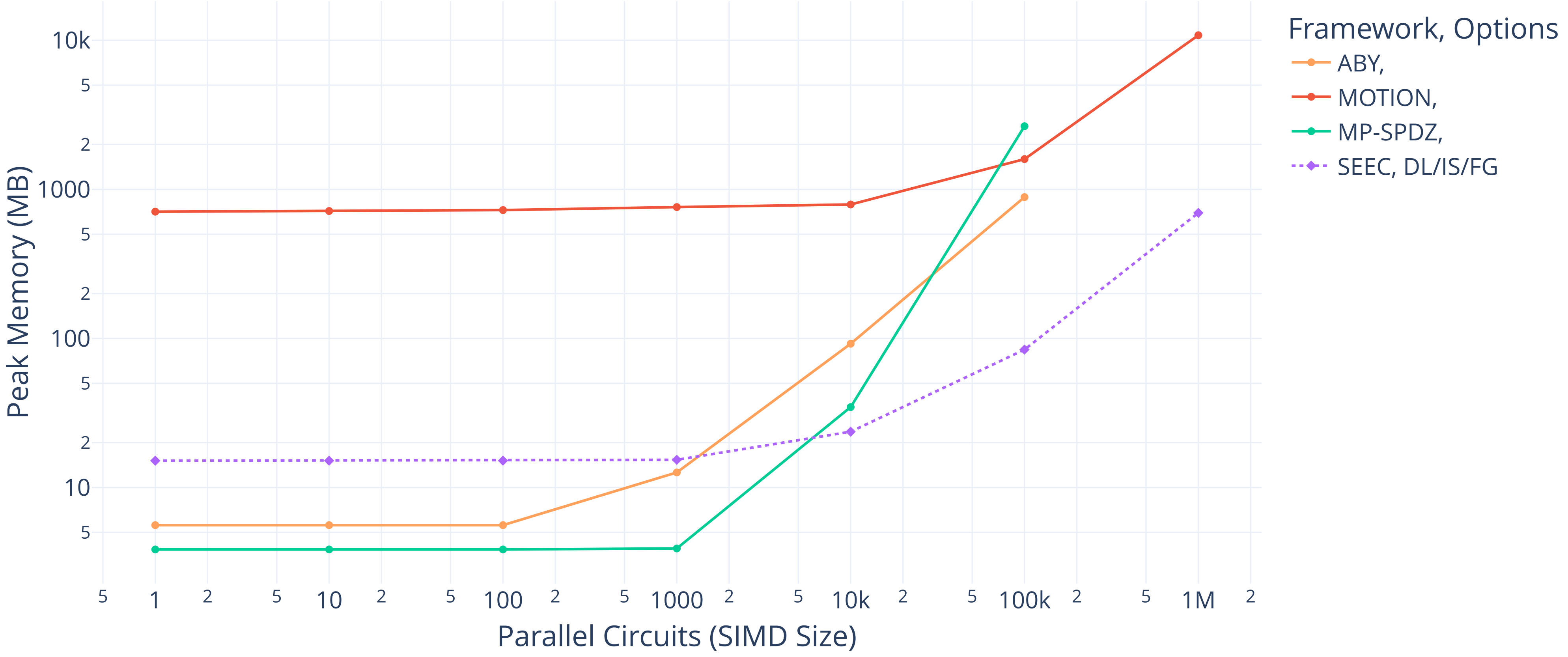
AES-CBC: Reduced Memory via Sub-Circuits



AES: Reduced SIMD Memory Usage



AES: Reduced SIMD Memory Usage



AES-CBC Runtime: Effect of Latency

Discussion

ENCRYPTO CRYPTOGRAPHY AND PRIVACY ENGINEERING

Sub-Circuits

```
#[sub_circuit]
fn process(...)
```

SIMD

Up to 15.54× - 1,983× less memory than MOTION [BDST22].

Evaluation Mode

Scalar → Layer-by-Layer

SIMD → Asynchronous

	Predictability	Reliability
ABY	✓	✗
MP-SPDZ	✗	✓
MOTION	✗	✓
SEEC	✓	✓

Future Work

- Expanding `Secret` API
- SIMD `#[sub_circuit]` macro
- Usability improvements

- Protocol composability
- Optional register storage
- Sub-Circuit SIMD-vectorization

- Sub-Circuit output deallocation

- OT-based interleaved setup
- Interleaved function dependent preprocessing

- Asynchronous Evaluation
- QUIC Channels
- Multi-Party + Malicious Protocols

Questions?

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Made with

typst draw.io SVG REPO

References

[ASLZ13] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. “More Efficient Oblivious Transfer and Extensions for Faster Secure Computation”. In: CCS, 2013.

[BCG+19] E. BOYLE, G. COUTEAU, N. GILBOA, Y, ISHAI, L. KOHL, P. RINDAL, and P. SCHOLL. “Efficient two-round OT extension and silent non-interactive secure computation.” In CCS, 2019.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In STOC, 1987.

[Bea92] D. BEAVER. “Efficient Multiparty Protocols Using Circuit Randomization”. In CRYPTO, 1992.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. “ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”. In NDSS, 2015.

[CCPS19] ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction”. In: CCSW@CCS, 2019.

[Kel20] M. KELLER. “MP-SPDZ: A Versatile Framework for Multi-Party Computation”. In CCS, 2020.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. “ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation”. In USENIX Security, 2021.

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. “MOTION - A Framework for Mixed-Protocol Multi-Party Computation”. In TOPS, 2022.

[BHK+23] L. BRAUN, M. HUPPERT, N. KHAYATA, T. SCHNEIDER, O. TKACHENKO. “FUSE - Flexible File Format and Intermediate Representation for Secure Multi- Party Computation”. In AsiaCCS 2023.

Appendix

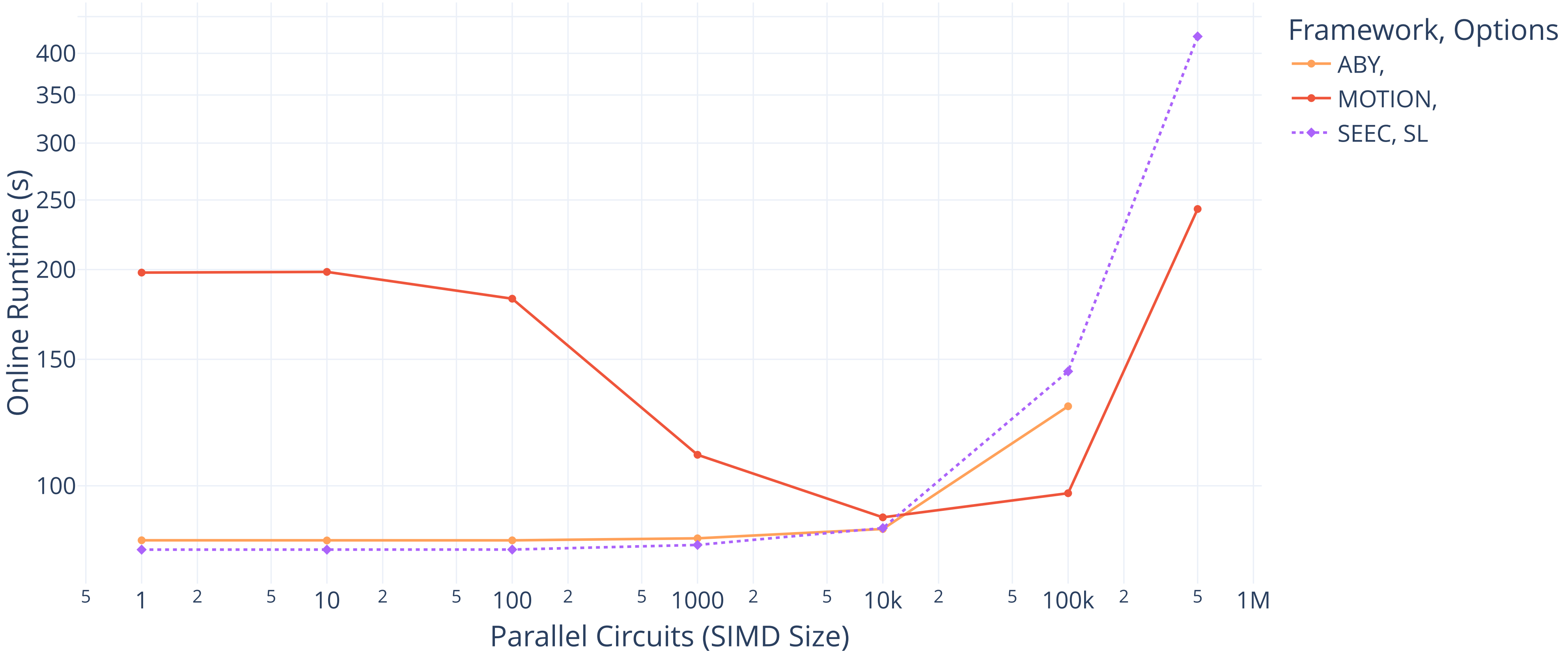
Benchmarking Tool

```
net_settings = ["RESET", "LAN", "WAN"]
repeat = 5

[[bench]]
framework = "SEEC"
target = "bristol"
tag = "seec_aes_ctr_no_setup"
compile_flags = ["../../../circuits/
advanced/aes_128.bristol"]
flags = ["--insecure-setup"]
cores = [0,1]
[bench.compile_args]
"--simd" = ["1", "10", "100", "1000",
"10000", "100000", "1000000"]
```

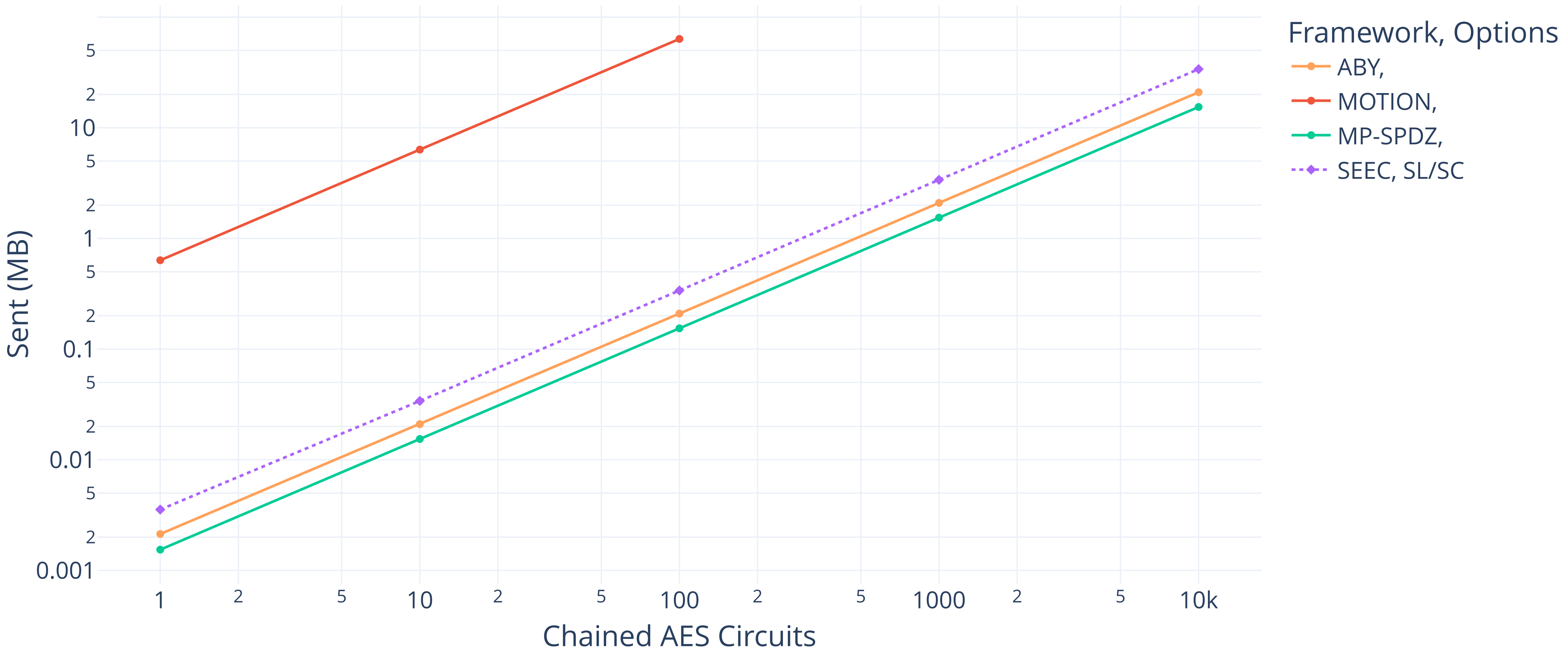
```
[[bench]]
framework = "MOTION"
tag = "motion_aes_no_setup"
target = "aes128"
flags = ["--insecure-setup"]
cores = [0,1]
[bench.args]
"--num-simd" = ["1", "10", "100",
"1000", "10000", "100000", "1000000"]
```

encryptogroup/mpc-bench

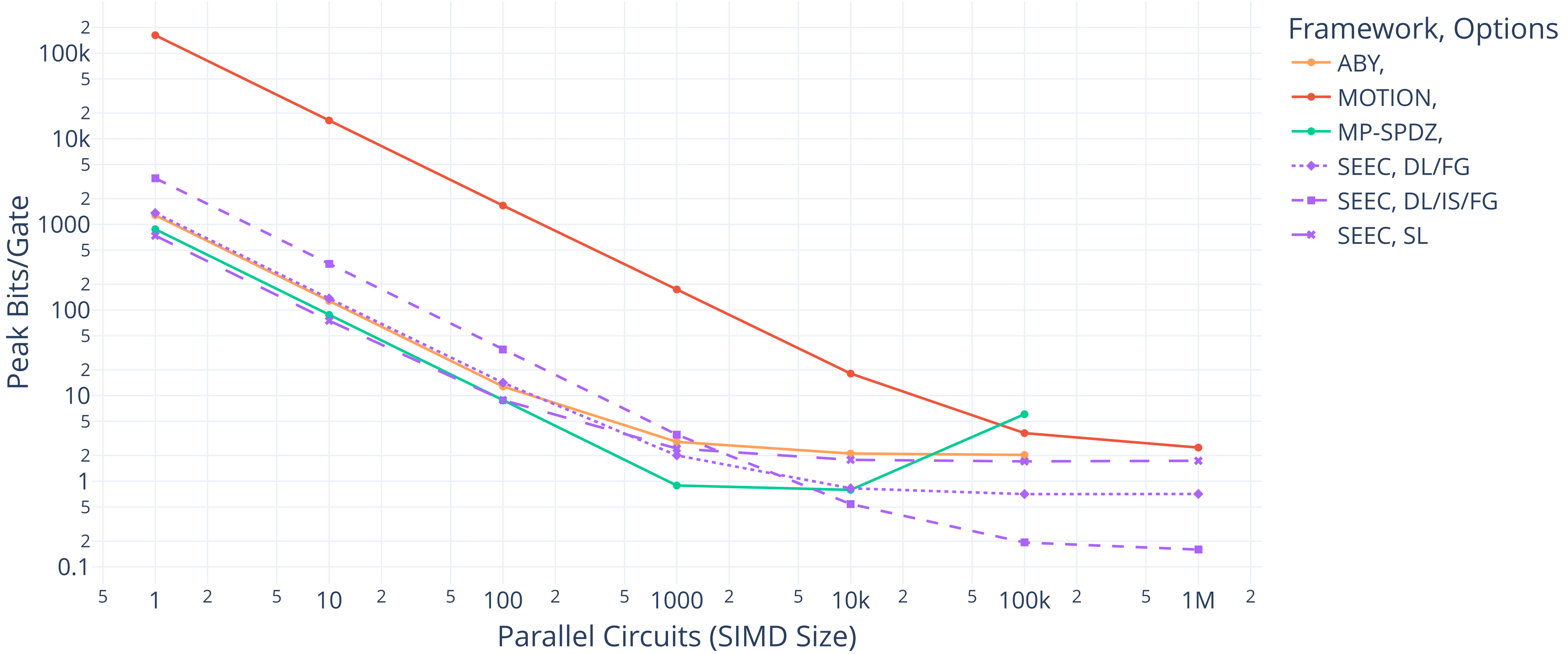
SHA-256: Effect of Nagle's Algorithm



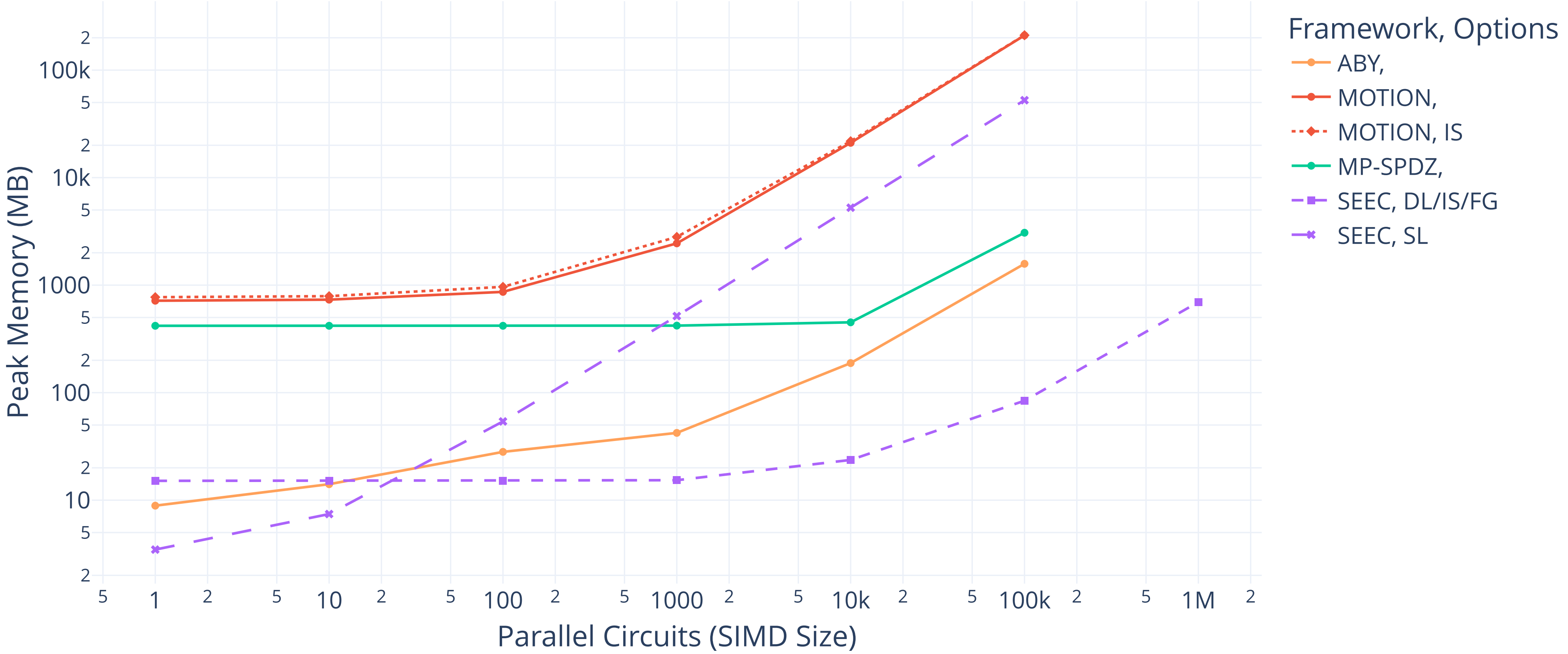
AES-CBC: Async. Communication Overhead



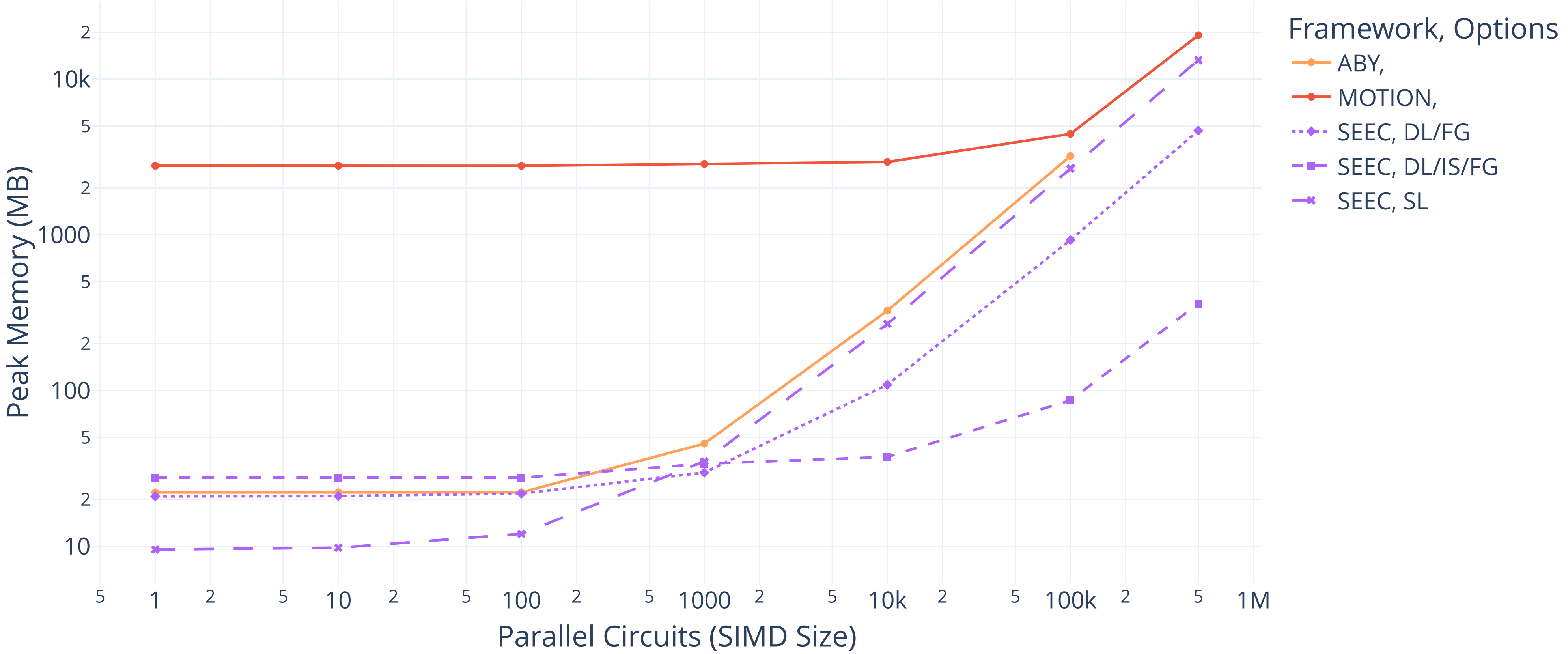
SIMD AES: Peak Bits per Gate



SIMD AES: Impact of Setup



SHA-256: Reduced SIMD Memory Usage



SEEC: System Architecture