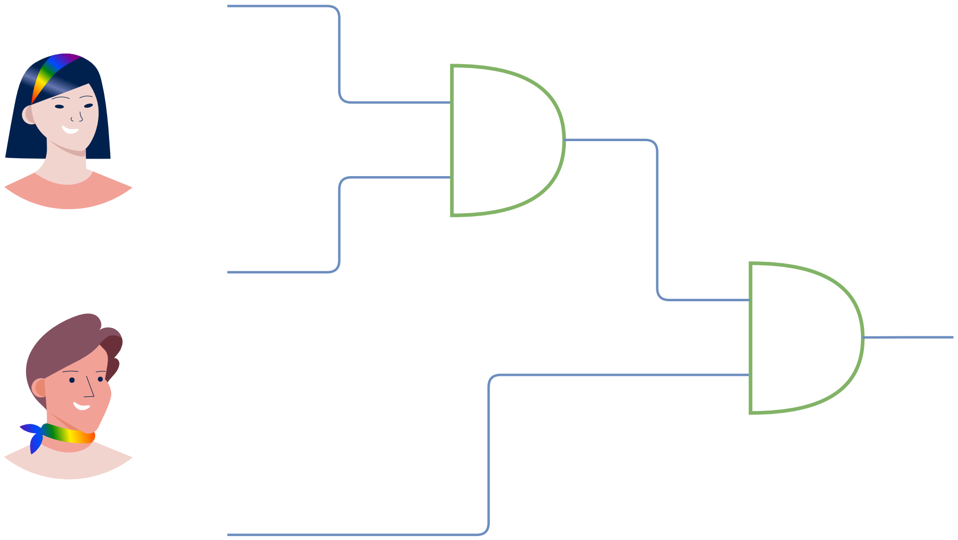
SEEC: Memory Safety Meets Efficiency in Secure Two-Party Computation

Robin Hundt Nora Khayata Thomas Schneider



Feedback

- don't spend so much time on memory
- slide 9: don't go into lazy iterator details
- slide 11: don't use preprocessing, but better compile
- practice more
- slide 12: leave out throughput
- slide 12: don't mention that you wouldn't execute AES-CBC in MPC
- slide: mention that I'm benchmarking GMW B
- bench slides: one conclusion for each slide
- slide 17: runtime numbers
- slide 18: remove eval mode
- slide 18: have take home message, memory safety and efficiency
- slide 18 references
- slide 18: reemphasize that sub-circuits don't increase depth
- slide 18: rename to summary
- kasra: slides are overloaded
  - map what I'm saying to
  - heavy on acronyms
- more clear steps
- legend for acronyms in benchmark

Thomas: Slide 1: "Joint work with Nora Khayata and Thomas Schneider" Also say you built SEEC during your Mas

Slide 3: Where do the 70 sth. percent occur in the figure? Couldn't find that.

Slide 4: ALSZ13

Slide 6/7: Call function func rather than process (process resembles a process/thread to me)

Slide 7: Put headlines above left (???) and right (graph-based) column

Slide 10: Remove "Figure 1:" and "Figure 2:"

Slide 11: Couldn't relate abbreviations (FG) and (IS) to rest of these columns "Stored MT Streaming" is not clear to

Slide 12: You can remove the Frameworks with which you compare from the previous overview slide on SEEC (sa

Slide 12: Explicitly mention your MPC benchmarking tool somewhere (either here or in the SEEC overview slide b

Slide 14: Remove , at the end of legends (for ABY, MOTION, MP-SPDZ) also in other slides => just put a white bo this was previous work and from which year

Slide 15: Write out acronyms DL / FG / IS / SL at the bottom again (audience might have forgotten these already b

Slide 16: Remove 2, 5 intermediate values on x axis as you are not using them.

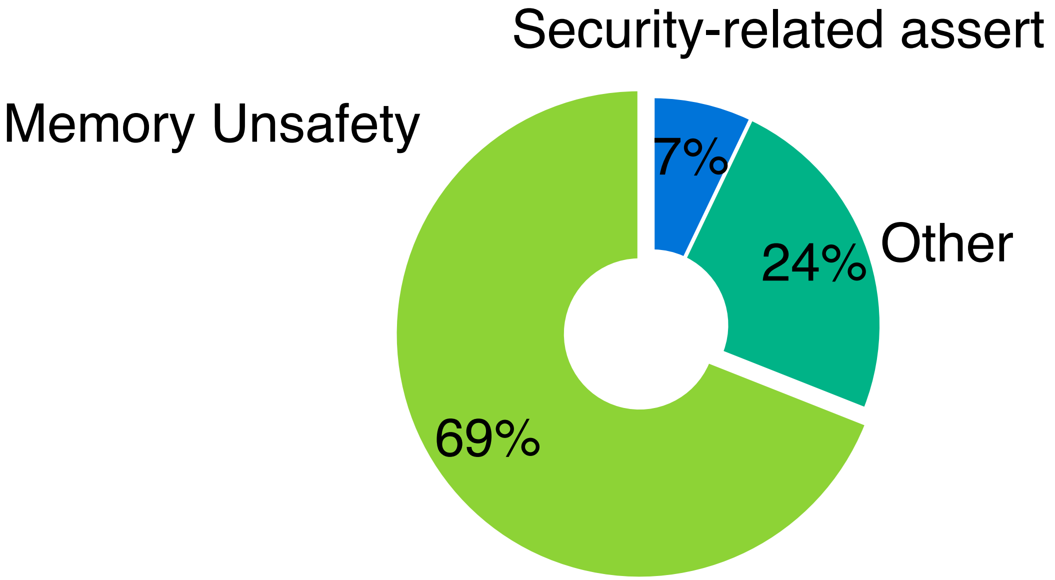
Slide 17: Fast-LAN => LAN10G, LAN => LAN1G, WAN => WAN100M (self-speaking names)

Slide 18: 16x - 1,983x (round to whole numbers to avoid confusing)

Slide 19: Do NOT say that SEEC is work in progress (this could kill our CCS submission) Completely move Future submission and save time.

Motivation

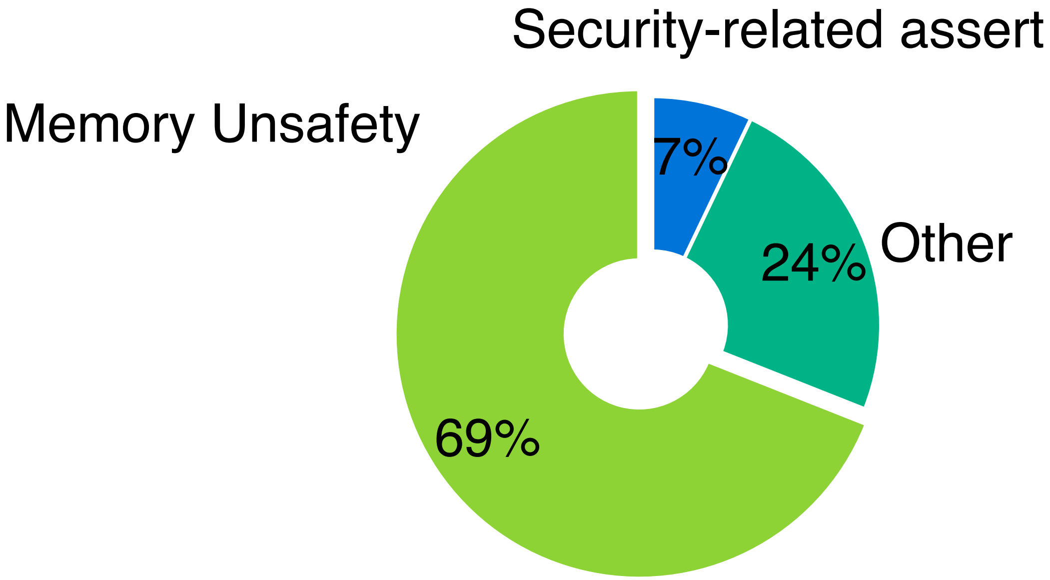
Safety



Source: The Chromium Projects - Memory Safety

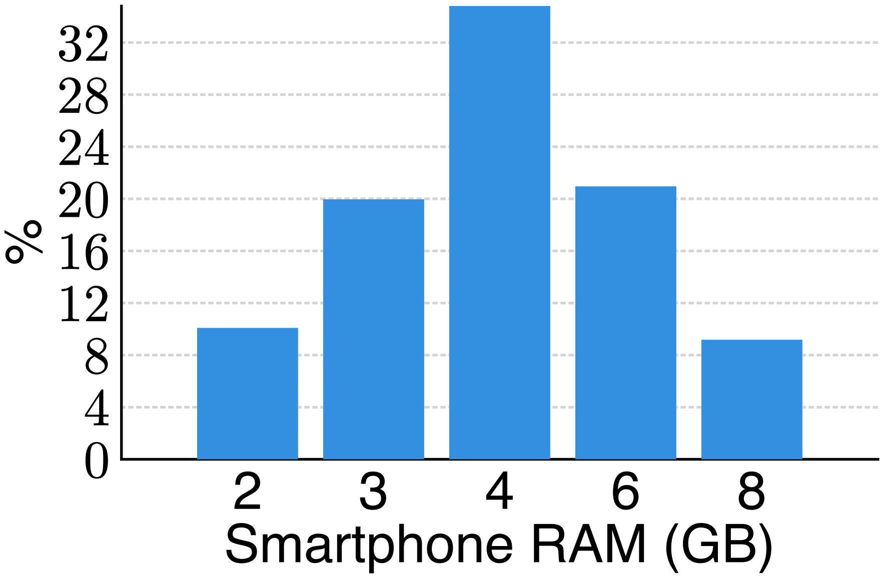
Motivation

Safety



Source: The Chromium Projects - Memory Safety

Efficiency



Source: scientiamobile, 2022

Notes: Motivation

- In recent years, Memory Safety of Applications and Programming Languages has received increasing interest. Due to an increasing dependence of our privacy on the security of digital systems, memory safety as one piece of secure systems, is becoming more and more important.
- However, experience hash shown time and time again, that high-impact vulnerabilities due to memory unsafety are virtually unavoidable in large projects written in C/C++.
- A recent examination of the high severity security impacting bugs in Chromium revealed that 70 % are due to memory unsafety. And, this is corroborated by other large projects, such as Windows and Android.
- This is relevant, as MPC applications are networked services, potentially exposed to the Internet, and which are usually written in C/C++.
- The Memory Safety of these applications will becomer more important as MPC progesses from research to real-world deployments.
- The reason C/C++ are so often used in MPC, is the performance and efficiency of the resulting implementations.

[NEXT SLIDE]

[SEEC Executes Enormous Circuits (SEEC)]

- High-Level eDSL / FUSE [BHK+23]
- (SIMD) Sub-Circuits
- Function (In-)Dependent Setup
- Extensibility w/o forking
- Cross-Platform

Notes: [SEEC Executes Enormous Circuits (SEEC)]

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

- Okay, so what do we contribute with SEEC?
- Because we wanted to achieve a memory safe and efficient MPC framework, which also provides good performance and a nice developer experience, we choose the programming language Rust for our implementation.
- It's a memory safe language, with performance similar to C/C++, control over memory allocations without garbage collection, and a good developer experience due to fantastic tooling.

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: [ALSZ13], Silent OT [BCG+19]

encrytpogroup/mpc-bench

* Partial Implementation

TECHNISCHE UNIVERSITÄT DARMSTADT

Functions in Traditional Programs

```
fn func(args: [bool; 2]) -> bool {
  // ... calculate return
}

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

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

Notes: Functions in Traditional Programs

- in traditional programs, functions are an important tool for building abstractions and organizing code
- they also reduce the size of the binary, inlining every function would result in a tremendous overhead
  - yet, this is exactly what we often do in MPC

Circuit Reuse in Secure Programs

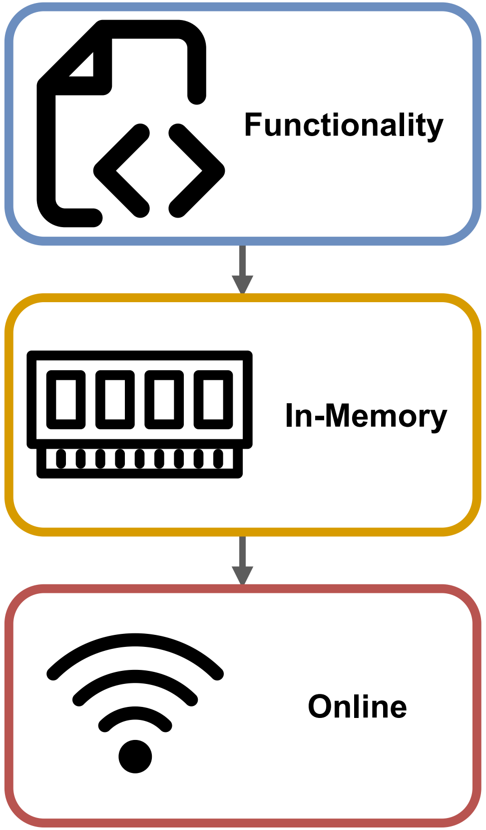
```
fn func(args: [bool; 2]) -> bool {
  // ... calculate return
}

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

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

```
fn func(args: [SBool; 2]) -> Sbool {
  // ... calculate return
}

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

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

Notes: Circuit Reuse in Secure Programs

- ideally when using MPC to securely evaluate a functionality, we want to express it in a high-level way
  - this should include the capability for using functions
  - code should be fairly close to traditional code, to ease development of MPC applications
  - some slight differences such as the changed types here, are okay; and likely necessary
- crucially, we want to not only use functions for organization, but also for reduced memory consumption during the MPC protocol's evaluation
- this is important for the real world deployment of MPC, where we might operate on memory-constrained devices or have very large inputs
- with our work SEEC, we have implemented and extensively benchmarked and compared one possible solution

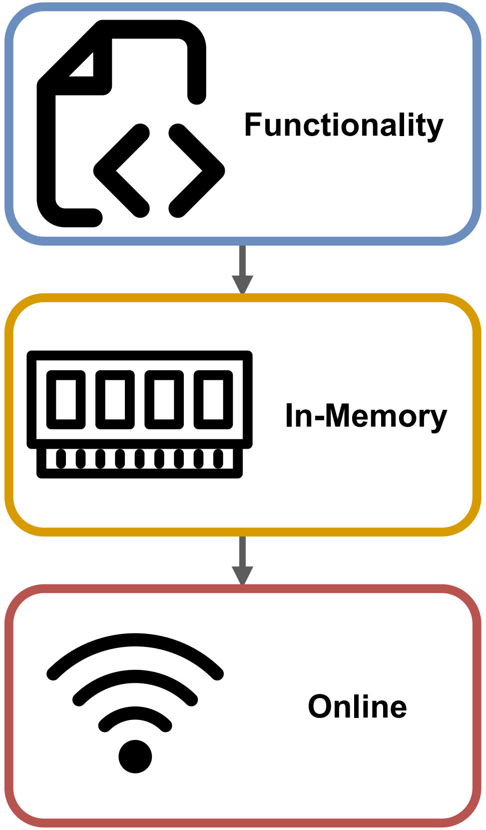
Sub-Circuits in GMW: Challenges




```
func(a);
func(b);
```

`a` and `b` are indepent

Sub-Circuits in GMW: Challenges



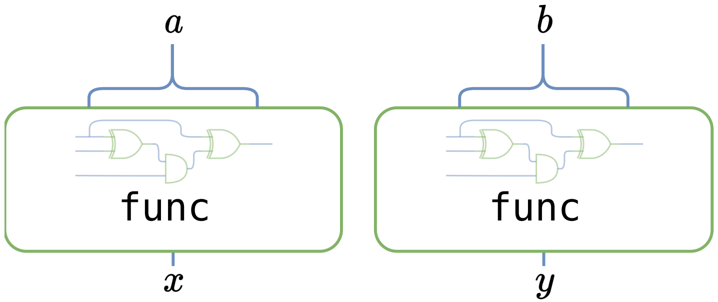

```
func(a);
func(b);
```

a and b are indepent

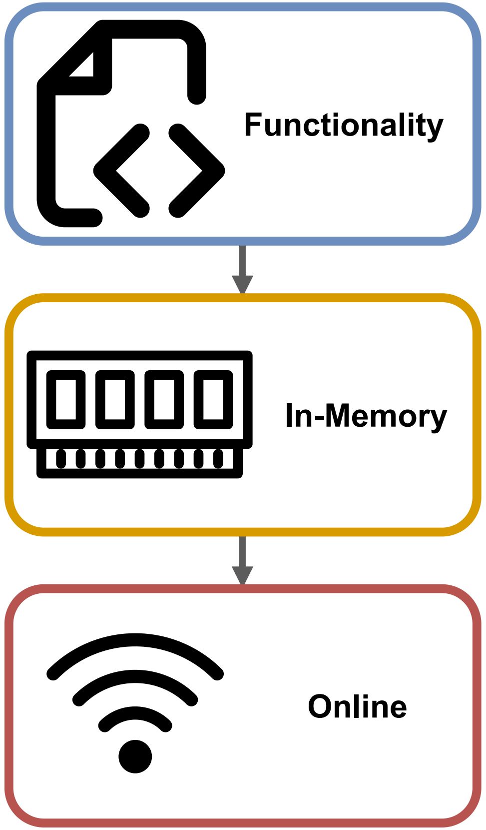
Bytecode VM

```
func:
  # ...
  ret

call func
call func
```

Graph based



Sub-Circuits in GMW: Challenges



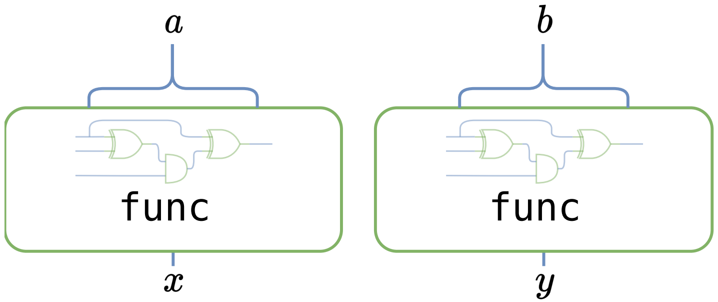

```
func(a);
func(b);
```

`a` and `b` are indepent

Bytecode VM

```
func:
  # ...
  ret

call func
call func
```

Graph based



→ Increased rounds
→ `func` only once in memory

→ Concurrent evaluation
→ Increased Memory

Notes: Sub-Circuits in GMW: Challenges

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

```
TODO: Maybe animate this slide
```

SEEC: eDSL Enables Efficient Circuit Reuse

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

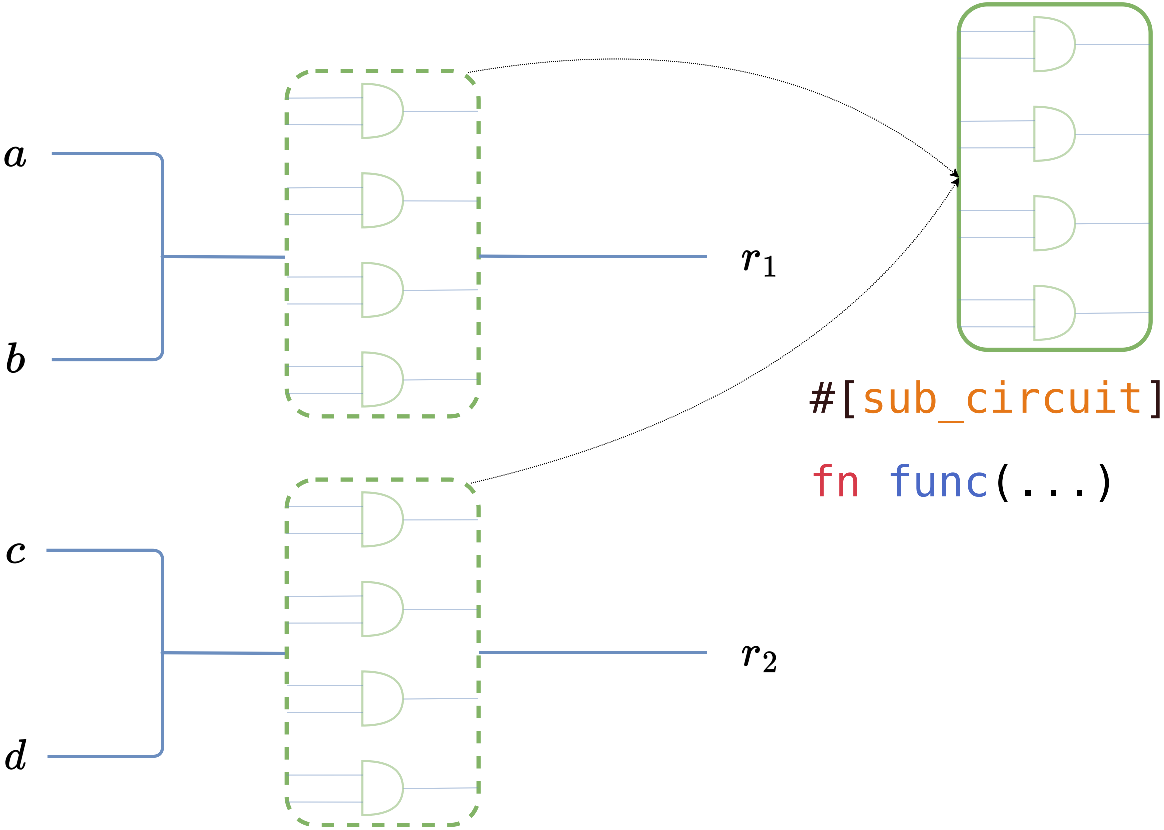
```
#[sub_circuit]
fn func(a: Vec<Secret>, b: Vec<Secret>)
    -> Vec<Secret> {
  a.into_iter().zip(b).map(|(el_a, el_b)| {

    el_a & el_b

  }).collect()
}
```

SEEC: eDSL Enables Efficient Circuit Reuse

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

    el_a & el_b

  }).collect()
}
```

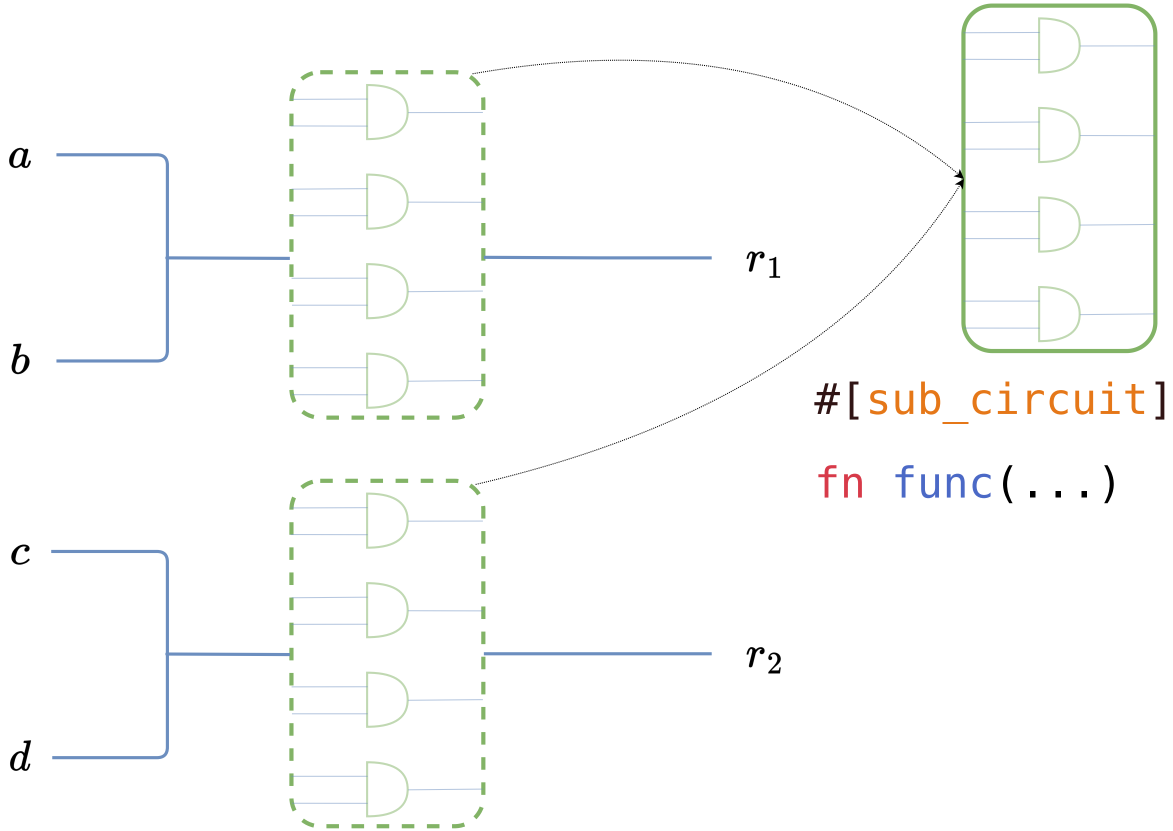
```
let (a, b, c, d) = init_data();
// func is called as normal
function.
let r1 = func(a, b);

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

SEEC: Sub-Circuits Are Not Inlined




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

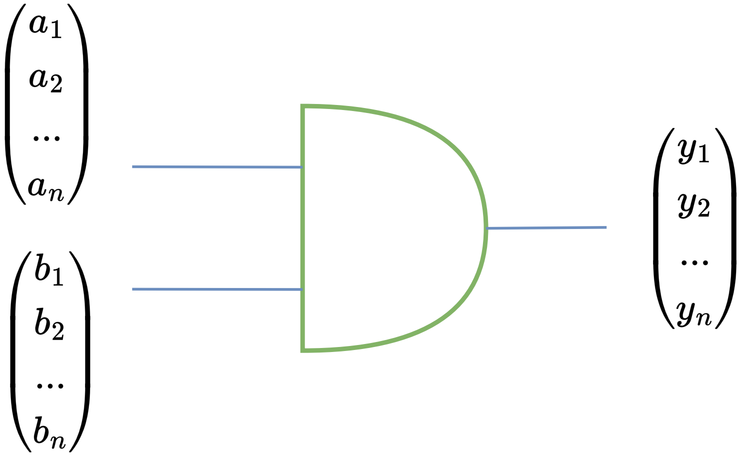
SEEC: Sub-Circuits Are Not Inlined



Online

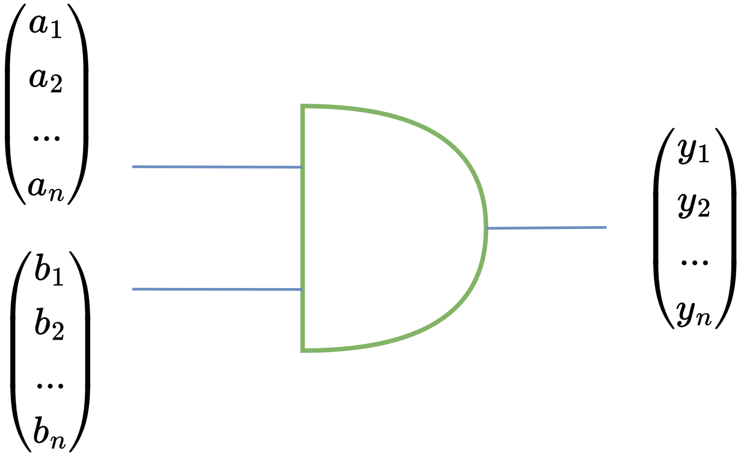
- Layer iteration **as if** inlined (DL)
  - No increase in depth
- Partial and concurrent evaluation

Single Instruction, Multiple Data

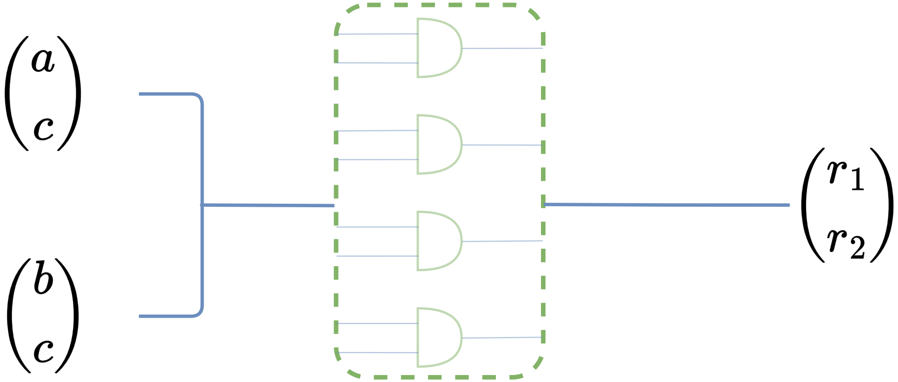


Traditional SIMD, e.g., in MOTION [BDST22].

Single Instruction, Multiple Data



Traditional SIMD, e.g., in MOTION [BDST22].



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 (ED)

Streaming MTs (SMT)

SEEC: Optimizations

Static Layers (SL)

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

Early Deallocation (ED)

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

Streaming MTs (SMT)

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 (ED)

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

Streaming MTs (SMT)

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

Notes: SEEC: Optimizations

```
- TODO: Explain optimizations
- text ist zu erschlagend
- spacing etwas erhöhen, text vllt. etwas kleiner
- maybe auch vorherige scritte ausgrauen
```

Evaluation

Frameworks

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

Environment

LAN-0.25ms / LAN-1.25ms
WAN-100ms

$1, 2, \ldots, 32$ Threads

Heaptrack[1]

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

Evaluation

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Frameworks

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

Environment

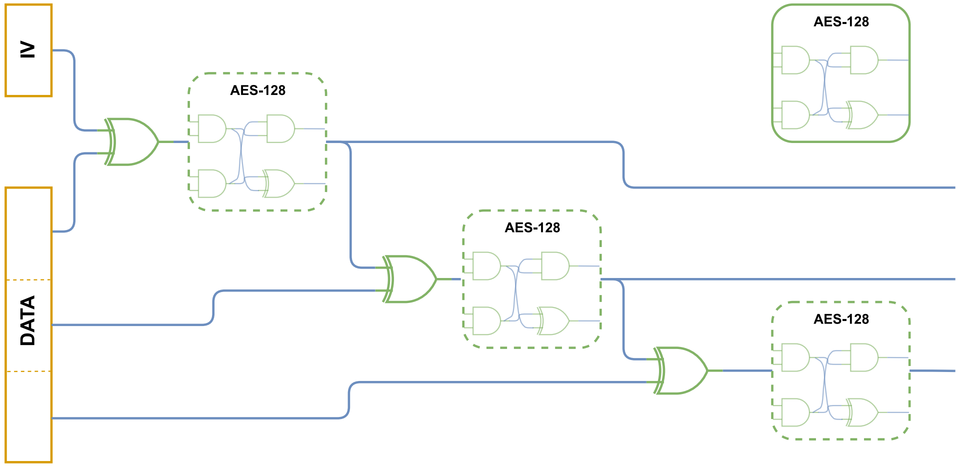
LAN-0.25ms / LAN-1.25ms
WAN-100ms

$1, 2, \ldots, 32$ Threads

Heaptrack[1]

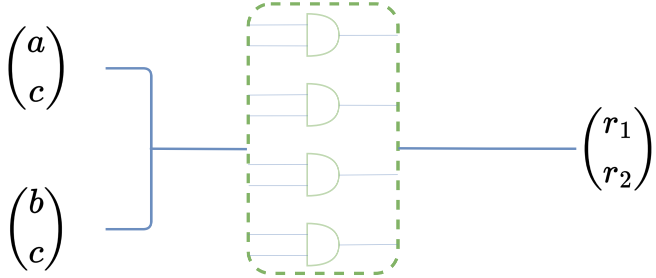
Circuits

Sub-Circuit



SIMD

- AES-128
- SHA-256



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

TECHNISCHE UNIVERSITÄT DARMSTADT

Notes: Evaluation

ENCRYPTO CRYPTOGRAPHY AND PRIVACY ENGINEERING

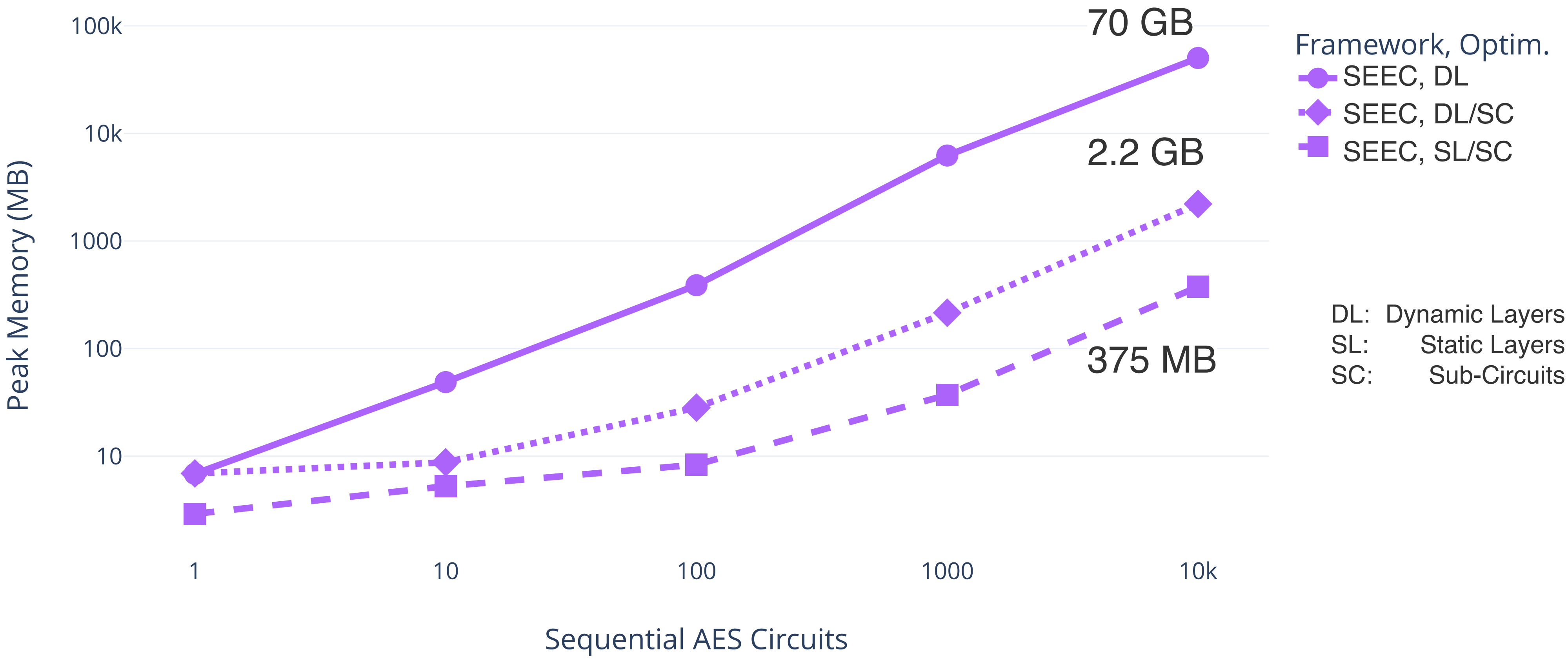
- Why did we choose these frameworks?

- explain net settings
- hardware: two simx servers
- mention bench tool

- bench sub-circuit via AES CBC circuit
- bench SIMD via parallel AES and SHA-256 circuits

TODO make clear that we use 2-party semi-hones GMW

AES-CBC: Reduced Memory via Sub-Circuits



DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits

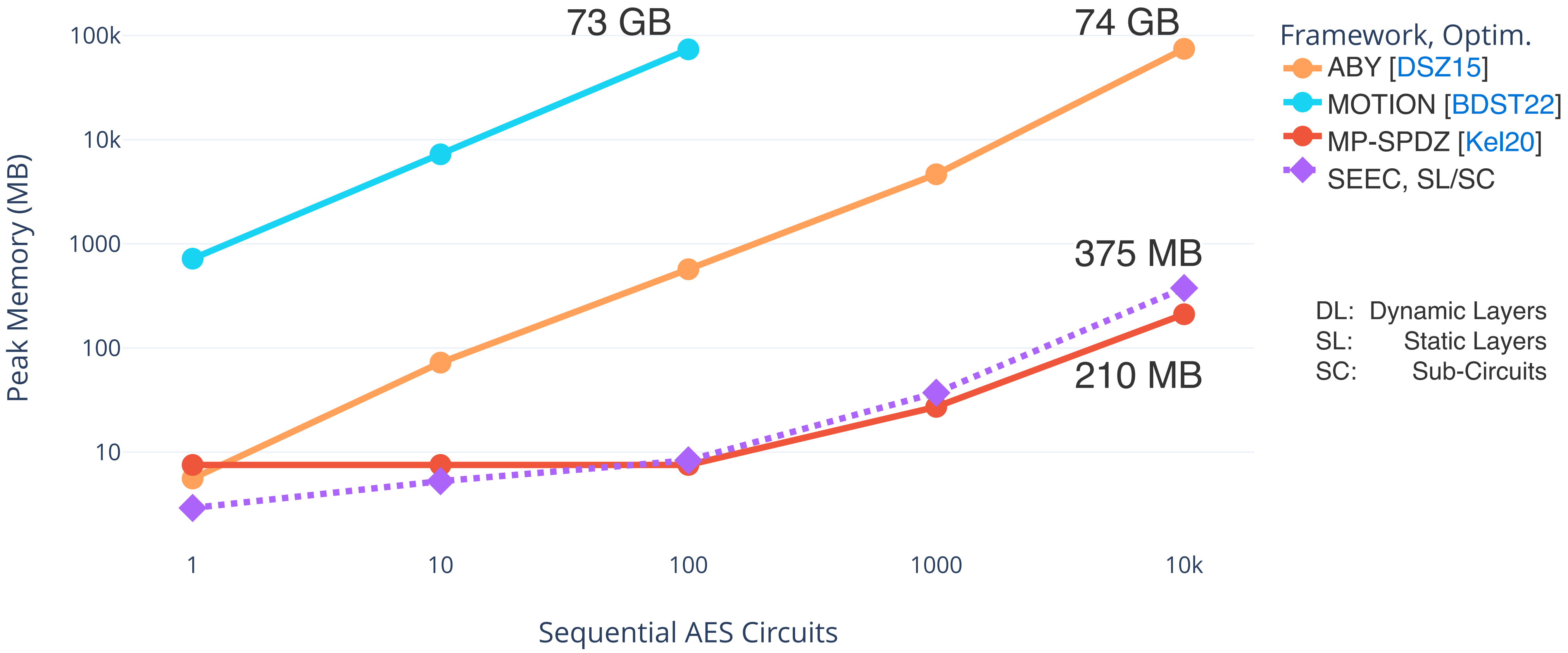
Notes: AES-CBC: Reduced Memory via Sub-Circuits

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

```
- 50 GB
- 2.2 GB
- 375 MB
```

AES-CBC: Reduced Memory via Sub-Circuits

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

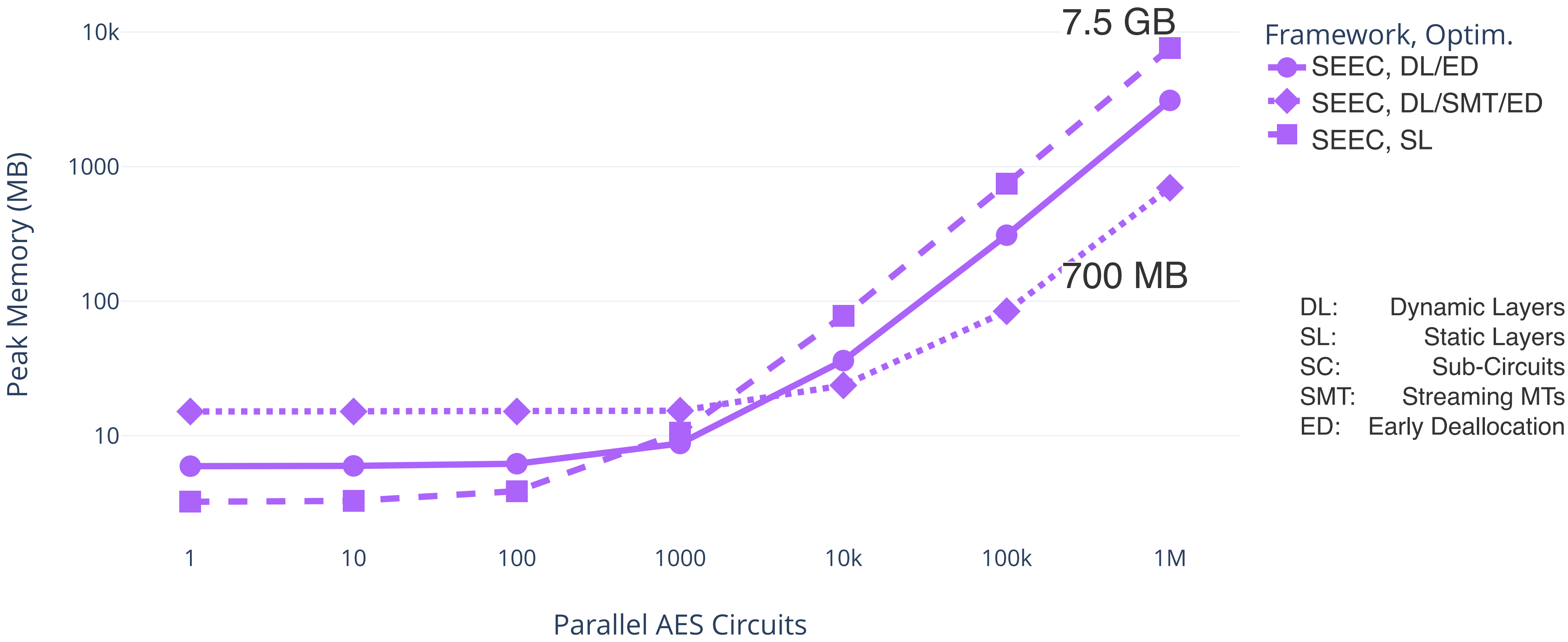


DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits

Notes: AES-CBC: Reduced Memory via Sub-Circuits

```
- at 10k:
  - 210 MB for MP-SPDZ
  - 375 MB for SEEC
- MOTION at 100:
  - ~73 GB
```

AES: Reduced SIMD Memory Usage

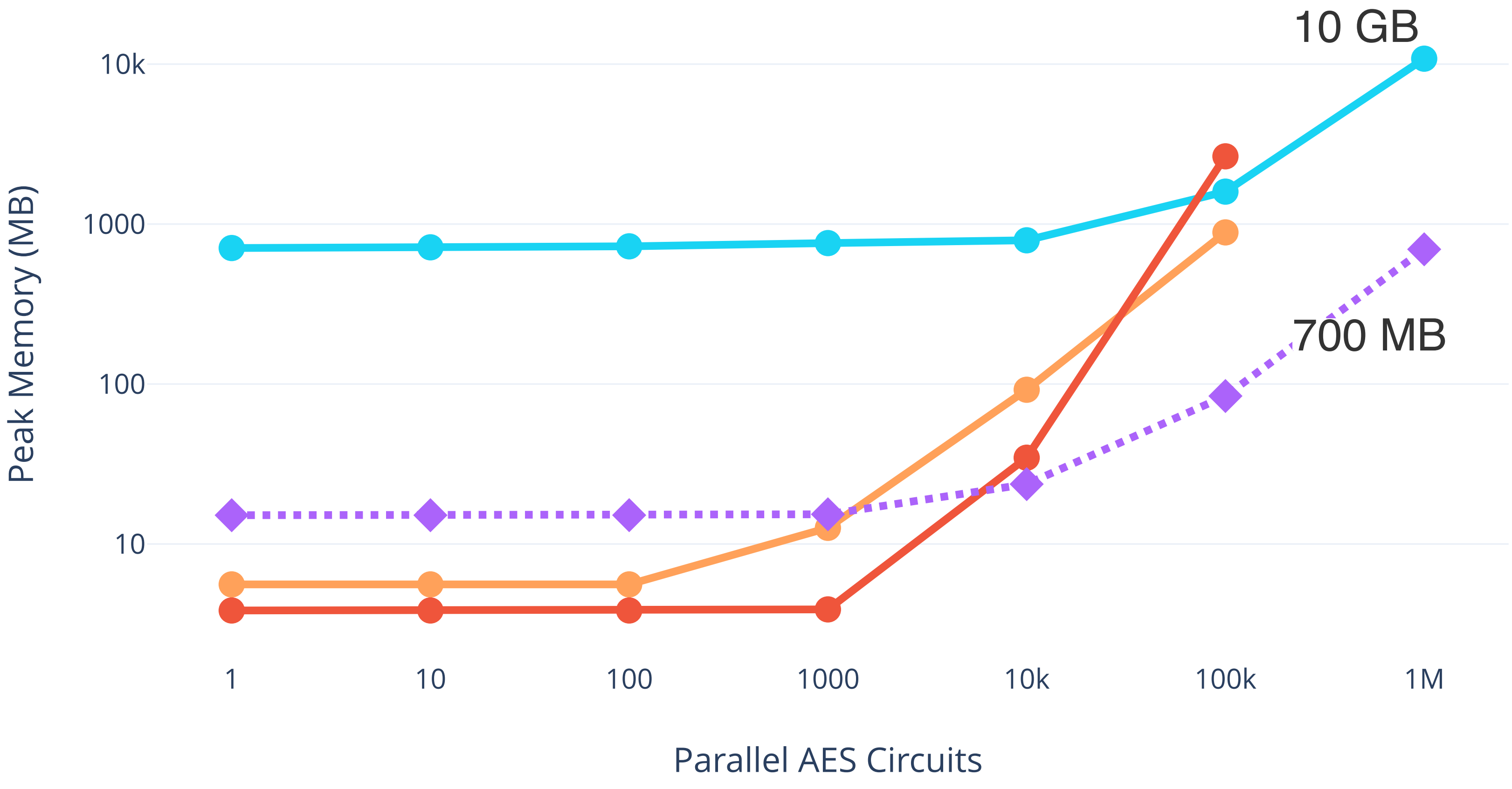


DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits
SMT: Streaming MTs
ED: Early Deallocation

Notes: AES: Reduced SIMD Memory Usage

```
- just SL: No opts: 7.5 GB
- FG: 3.1 GB
- SEEC: ~ 700 MB
```

AES: Reduced SIMD Memory Usage

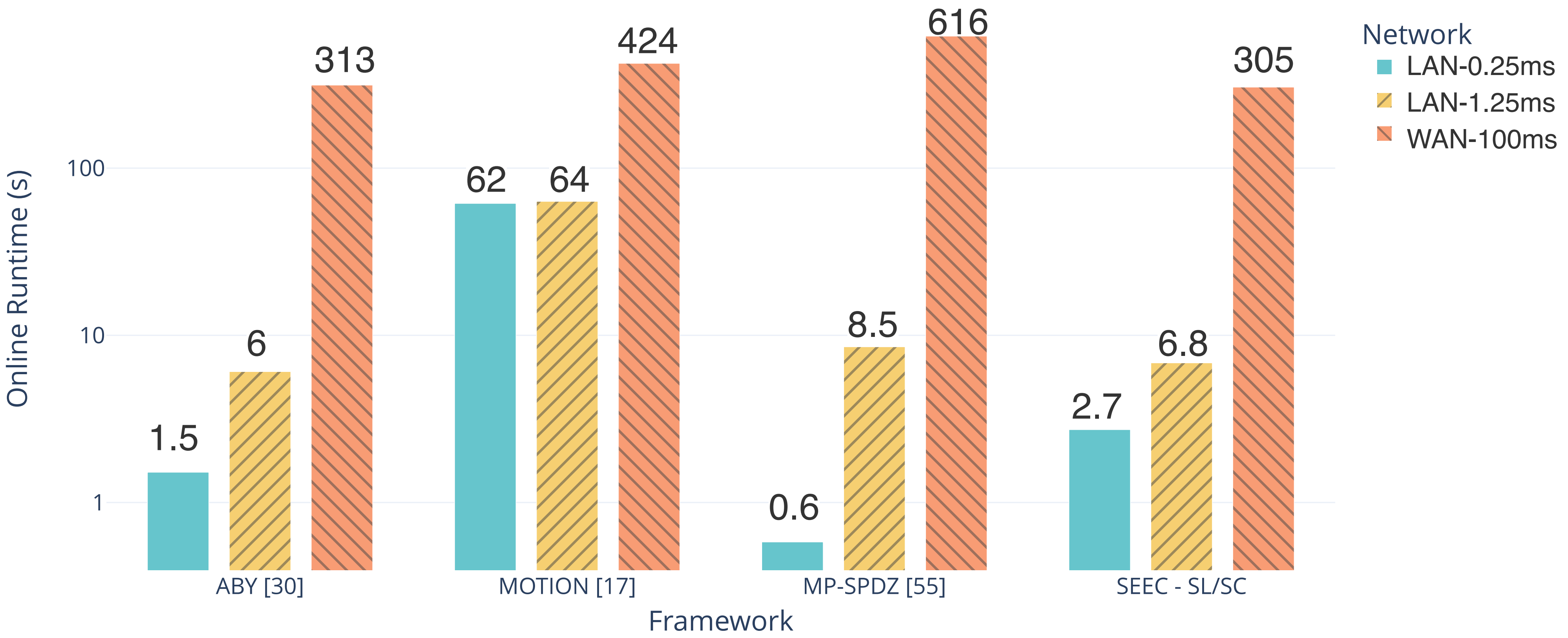


DL: Dynamic Layers
SL: Static Layers
SC: Sub-Circuits
SMT: Streaming MTs
ED: Early Deallocation

Notes: AES: Reduced SIMD Memory Usage

```
- at 1M:
  - MOTION: 10 GB
  - SEEC: ~ 700 MB
```

AES-CBC Runtime: Effect of Latency



Notes: AES-CBC Runtime: Effect of Latency

```
100 chained AES blocks
- WAN: MOTION 423 s and MP-SPDZ 616 s
```

Summary

Sub-Circuits

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

SIMD

Up to 15× - 1,983× less memory than MOTION [BDST22].

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

Notes: Summary

```
Mem. Reduction via Sub-Circuits
- support for loops and register allocation of gate outputs can lead to better
memory efficiency in some cases (MP-SPDZ)
- however sub-circuits are more versatile, as they can reduce memory consumption
of sub-circuit calls at unrelated places of the main circuit
SIMD
- Significantly better SIMD memory consumption
  - largely due to FG and IS optimizations
- Async. Eval. of MOTION has bad perf. for scalar circs but good for massively
parallel circs (high SIMD size)
- LBL execution of ABY, MP-SPDZ and SEEC is better for scalar circs
- Network setting can have non-obvious impacts on online perf.
- Predicatbility
- Realiability
```

Questions?

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING

Made with

typst draw.io SVG REPO

References

[ALSZ13] 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

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

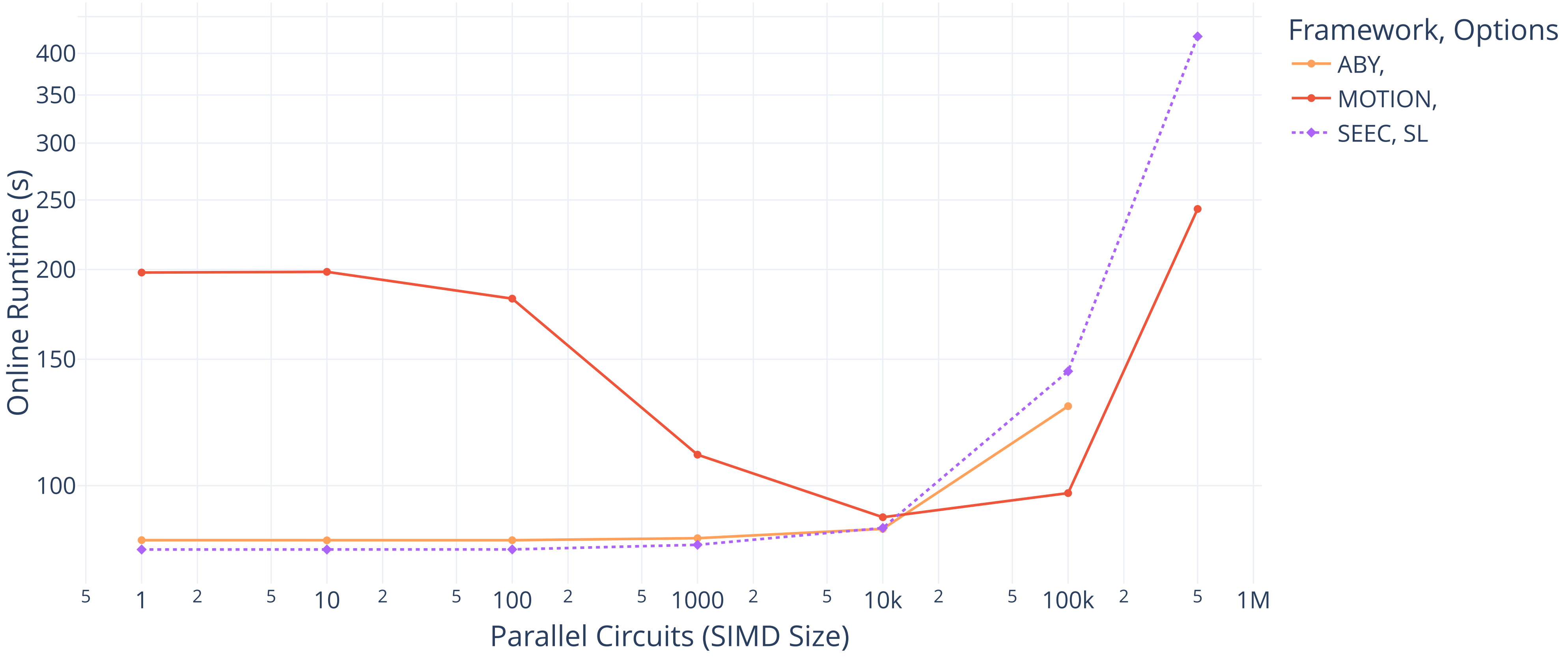
Benchmarking Tool

```
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"]
```

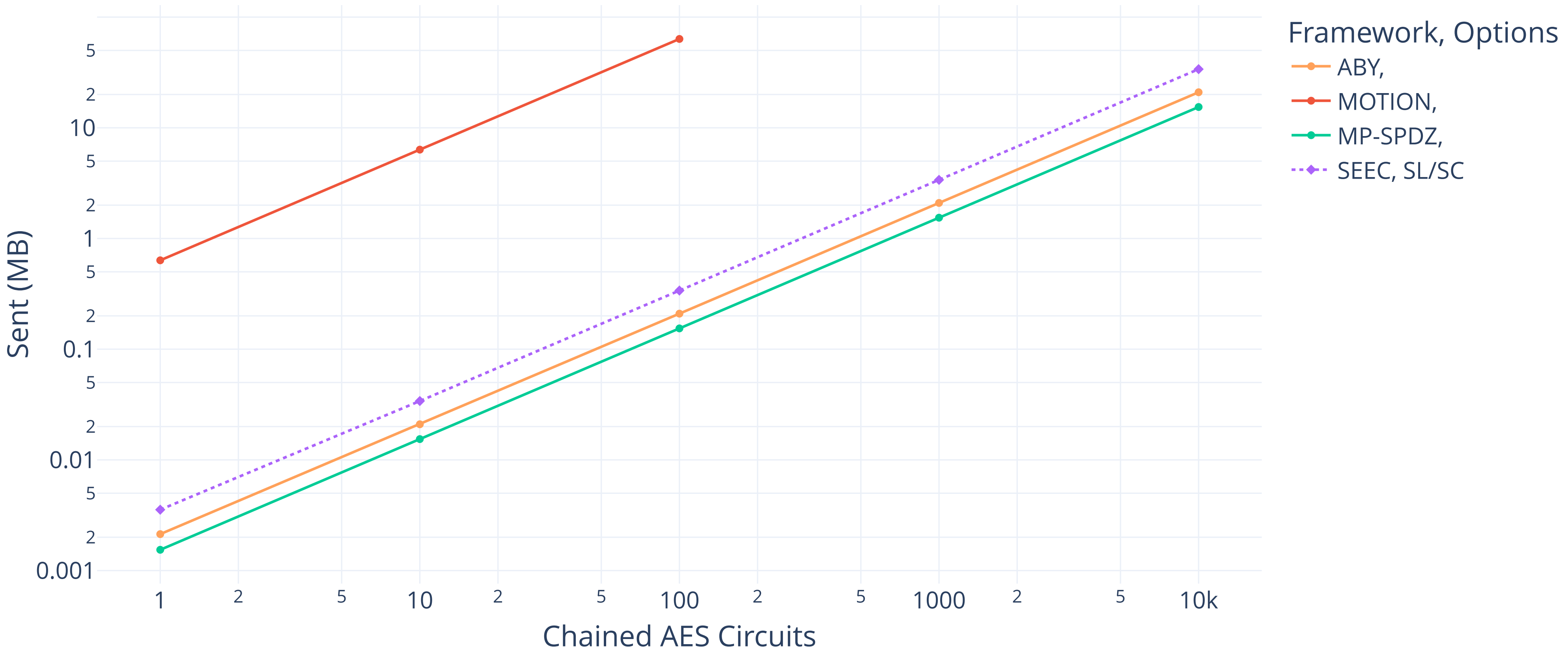
```
[[bench]]
framework = "MOTION"
tag = "motion_aes_no_setup"
target = "aes128"
flags = ["--insecure-setup"]
cores = [0,1]
[bench.args]
"--num-simd" = ["1", "10", "100",
"1000", "10000", "100000", "1000000"]
```

encryptogroup/mpc-bench

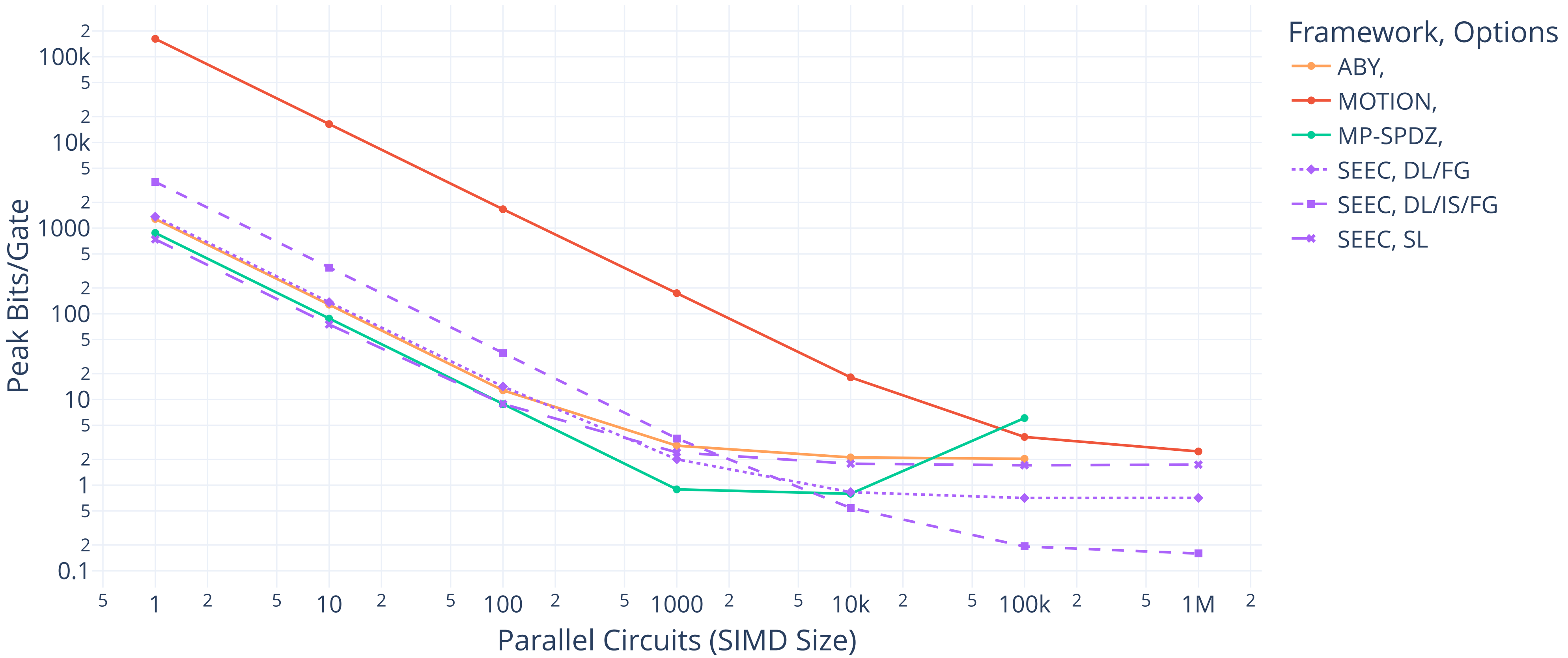
SHA-256: Effect of Nagle's Algorithm



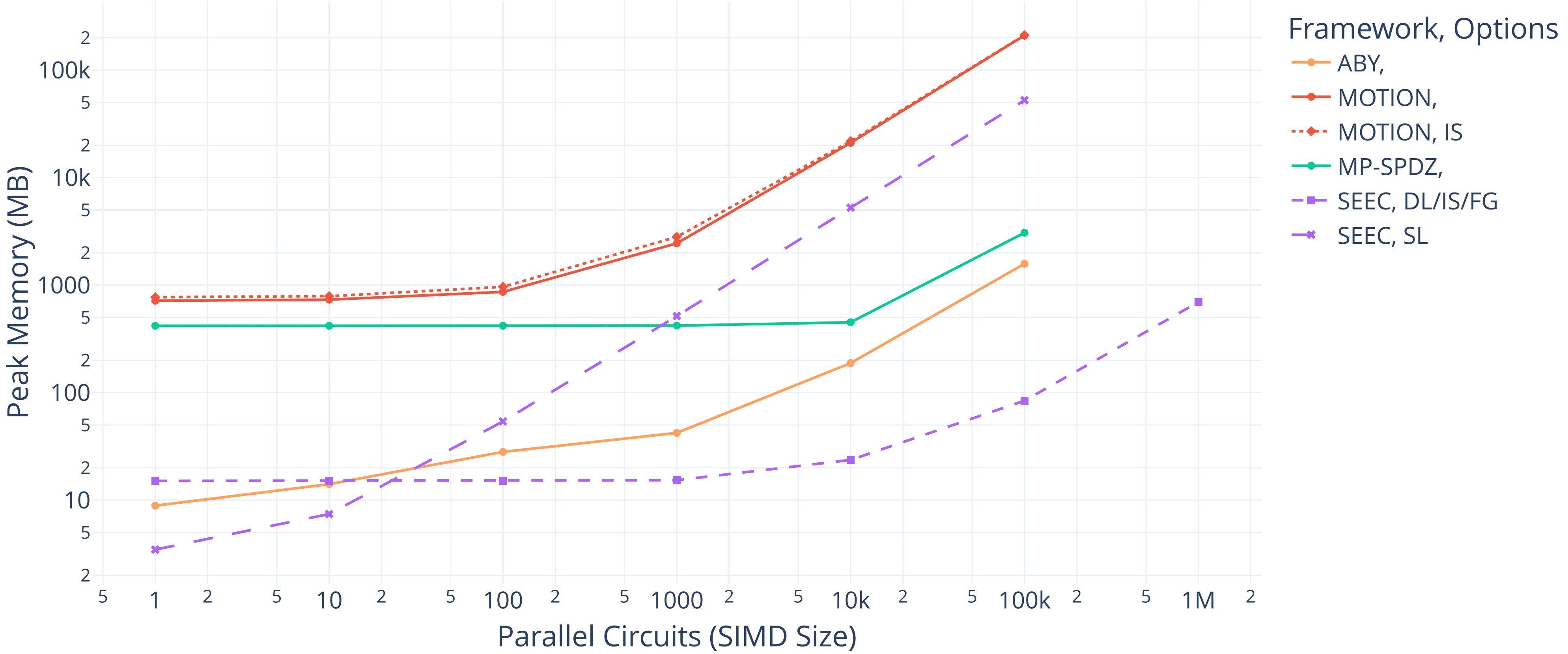
AES-CBC: Async. Communication Overhead



SIMD AES: Peak Bits per Gate

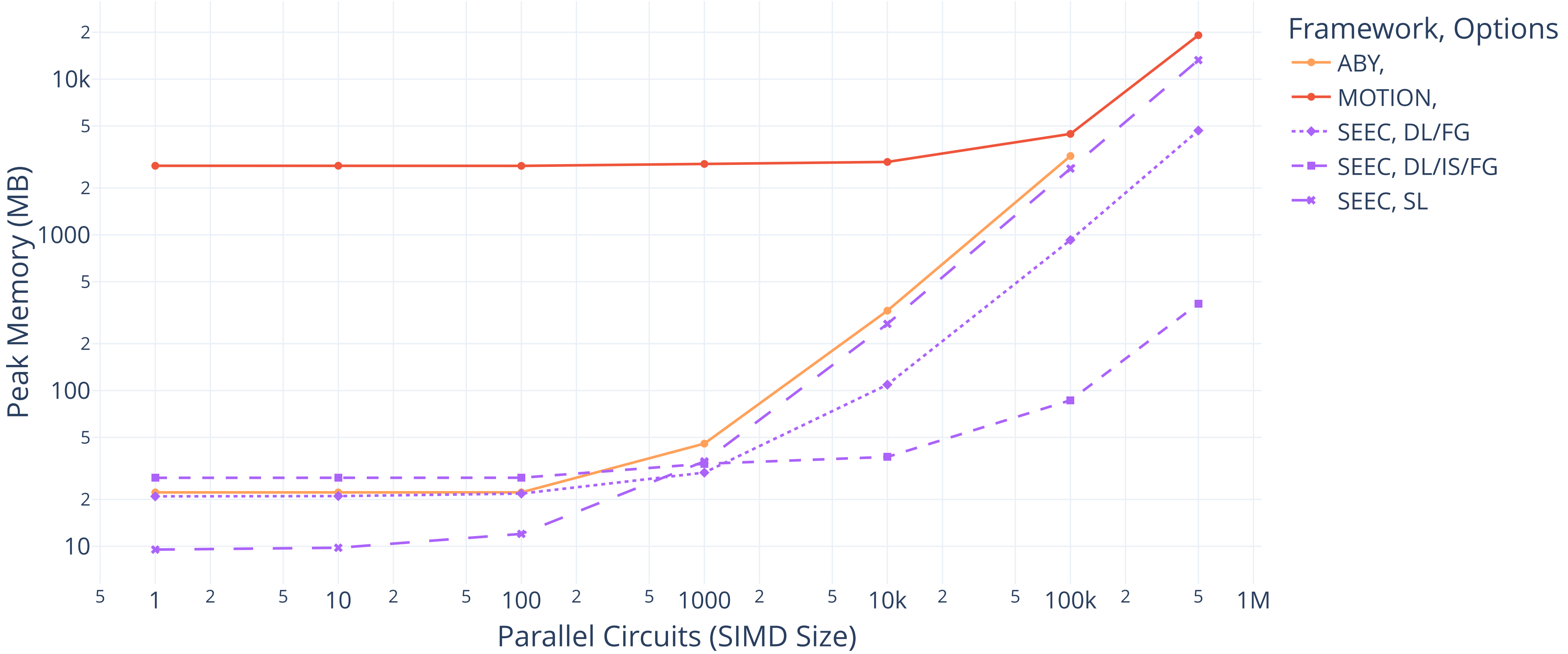


SIMD AES: Impact of Setup



SHA-256: Reduced SIMD Memory Usage

ENCRYPTO
CRYPTOGRAPHY AND PRIVACY ENGINEERING



TECHNISCHE UNIVERSITÄT DARMSTADT

SEEC: System Architecture