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

Robin William Hundt
Technical University of Darmstadt
Darmstadt, Germany

Nora Khayata
Technical University of Darmstadt
Darmstadt, Germany

Thomas Schneider
Technical University of Darmstadt
Darmstadt, Germany

## ABSTRACT

Secure Multi-Party Computation (MPC) allows multiple parties to perform privacy-preserving computation on their private data. MPC protocols based on secret sharing have high throughput which makes them well-suited for batch processing, where multiple instances of a function are evaluated in parallel. This is valuable for privacy-preserving cloud services where the service provider collects multiple requests to scale to many users. However, practical implementations of secret sharing-based MPC protocols mainly focus on runtime and communication efficiency, so the memory overhead of protocol implementations is often overlooked. Established techniques to reduce the memory overhead for constant-round garbled circuit protocols cannot be directly applied to secret sharing-based protocols because they would increase the round complexity. Additionally, state-of-the-art implementations of secret sharing-based MPC protocols are in C/C++ and may exhibit memory unsafety and leaks with potentially undefined behavior.

We present SEEC: SEEC Executes Enormous Circuits, a framework for secret sharing-based MPC with a novel approach to address memory efficiency and safety without compromising on runtime and communication efficiency. We realize SEEC in Rust for memory-safety at close-to-native speed. To reduce the memory footprint, we develop an in-memory representation for sub-circuits. Thus, we never inline sub-circuit calls during circuit evaluation, a common issue that blows up memory usage in MPC implementations. We compare SEEC with the state-of-the-art secret sharing-based MPC frameworks ABY (NDSS'15), MP-SPDZ (CCS'20), and MOTION (TOPS'22) and achieve competitive or better performance.

## 1 INTRODUCTION

Privacy-enhancing technologies (PETs) are essential to secure the fundamental right of privacy in an increasingly digital world. An important PET building block is secure multi-party computation (MPC). It allows two or more distrusting parties to jointly compute a public function on their private data. MPC ensures that each party learns only the output of the functionality and nothing more [11]. With increasing computational and network resources together with protocol improvements, MPC is starting to be deployed in the real-world [1]. There are two main classes of MPC protocols:
(1) Constant round protocols like *Yao's garbled circuits* (GCs) [18] evaluate costly symmetric cryptographic operations in the online phase. With state-of-the-art protocol optimizations [16], GCs provide free XORs and a single AND gate has communication $1.5\kappa$ bits, where $\kappa$ is the symmetric security parameter. These protocols have constant rounds and low latency, but low throughput.
(2) *Secret sharing-based* MPC protocols like GMW [11], SPDZ [7], and ABY2.0 [15] allow to precompute as many cryptographic operations as possible in an input-independent setup phase. The setup uses Oblivious Transfer (OT)-extension [3], SilentOT [4], or Homomorphic Encryption [7]. This yields a very efficient online phase without expensive cryptographic operations, but needs interaction between the parties for every layer of AND gates. The total communication can be as low as 2 bits per AND gate [15]. Secret sharing-based MPC protocols have high throughput, which makes them very well-suited for batch processing [10]. Batch processing is useful in outsourcing scenarios that are commonly used in privacy-preserving machine learning (PPML) [14]. Outsourcing allows using efficient passively (semi-honestly) secure protocols, even if clients are malicious because the MPC protocol is run only between the outsourcing servers. However, finding non-colluding parties to run the outsourcing servers in practice is challenging, so the number of parties must be kept as small as possible [9]. Many practical works, e.g., in MPC-based PPML, assume two parties [14].

State-of-the-art implementations of general-purpose MPC protocols, including secret sharing-based protocols such as ABY [8], MP-SPDZ [12], and MOTION [5], are developed in C/C++ and focus on optimizing runtime and communication efficiency. These frameworks can be prone to memory leakage and more general memory-safety issues such as data races, buffer overflows, and use-after-free errors. All previous works on *memory-efficient* MPC were restricted to constant-round GC protocols and propose pipelining [13] and compact sequential circuit representations [17]. These optimizations are not directly translatable to multi-round secret sharing-based MPC protocols, because they would increase round complexity and need careful scheduling of the online phase.

**Our Contributions.** We present SEEC, the first memory-efficient and memory-safe framework for secret sharing-based MPC protocols in Rust. SEEC is available as open-source[1].
(1) **Improved memory consumption:** We optimize memory consumption for secret sharing-based protocols through a generalized in-memory representation with sub-circuits and Single Instruction Multiple Data (SIMD). We implement two sub-circuit iteration algorithms, the *dynamic* and *static* layer iteration. Both iteratively compute the current global evaluation layer, accumulating the evaluation layers inside the sub-circuits. The dynamic layer iterator (DL) computes the next layer ad-hoc during runtime while the static layer iterator (SL) precomputes all evaluation layers during compile-time. Both iterators reduce the memory footprint while maintaining low round complexity during evaluation. Sub-circuits work together with SIMD, a runtime optimization known from state-of-the-art MPC frameworks [5, 8, 12] which requires no data-flow dependencies between sub-circuit instances.
(2) **State-of-the-art protocols and modular MPC phases:** We implement the Boolean ABY2.0 [15] protocol with function-dependent preprocessing and efficient vector and matrix sub-protocols.

[1] https://encrypto.de/code/SEEC

We also implement Boolean (B) and Arithmetic (A) GMW with conversions for mixed-mode computation. For the setup phase, SEEC supports OT-based multiplication triple generation using OT extension [3] and Silent OT [4]. We provide Rust implementations of [3, 4] and integrate libOTe, an efficient C++ library of various OT protocols. SEEC supports Boolean Bristol Fashion circuits [2] and mixed circuits in the FUSE intermediate representation (IR) [6]. FUSE IR contains sub-circuit calls that are preserved when executing them in SEEC. We present the design of SEEC in Fig. 1. It abstracts over the different phases of MPC, enabling support for both function-independent and function-dependent preprocessing phases and adding new phases for protocol advancements in the future. Using Rust's communication channels abstractions, SEEC offers various implementations for communication between the parties with minimal communication overhead. These include in-memory communication, TCP, and TLS communication. We improve over the state-of-the-art MOTION [5], which provides abstractions in C++, but has high communication overhead.

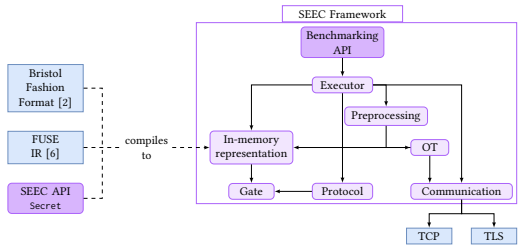


**Figure 1: Architecture of the SEEC Framework.**

(3) **Reproducible benchmarks:** We compare SEEC with the state-of-the-art secret sharing-based MPC frameworks ABY [8], MP-SPDZ [12], and MOTION [5]. We have developed a benchmarking tool that enables easily reproducible results of all four frameworks which is of independent interest. We present the results for peak memory consumption for a) the MNIST neural network from [14], and b) highly parallelized AES evaluations for batch processing.

*a) MNIST inference:* Neural networks often comprise multiple layers of matrix multiplication. For repeating matrix multiplications with identical input dimensions, we use sub-circuits to reduce the circuit size and memory consumption of the evaluation. We benchmark the MNIST network [14] with our mixed A+B GMW implementation. Our results in Fig. 2 show the difference in memory consumption between using sub-circuits and inlining every occurrence of a matrix multiplication and activation function, i.e., having one single circuit without sub-circuits. For both dynamic and static layer iteration, using sub-circuits decreases the peak memory necessary to evaluate the network by up to 2×. We anticipate larger differences for more complex neural networks.

*b) Batch processing of AES:* We compare the memory consumption for evaluating the setup and online phase of massively parallel circuits with state-of-the-art frameworks in Fig. 3. For SEEC, we compare two variants: Dynamic layer iteration with additional memory optimizations (DL/IS/FG) and static layer iteration with precomputed layers (SL). While SEEC DL has high memory overhead due to the setup phase, SEEC SL provides the best memory consumption. For 100k parallel AES evaluations, our peak memory consumption is 84 MB, 18.8× less than the next best ABY [8].

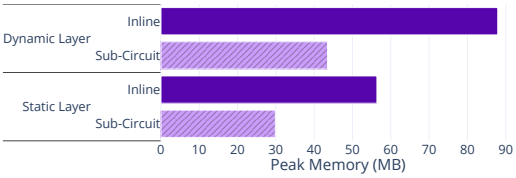


**Figure 2: Peak memory (MB) for evaluating the MNIST neural network [14] with mixed A+B GMW in SEEC (online phase).**

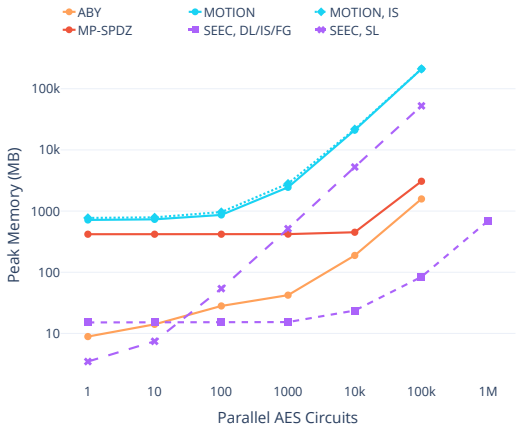


**Figure 3: Peak memory consumption of parallel AES evaluations in the LAN setting *including oblivious transfer (OT)-based setup.* MOTION, IS uses an interleaved setup.**

## REFERENCES


[1] MPC Alliance. https://www.mpcalliance.org, 2020.
[2] V. A. Abril, P. Maene, N. Mertens, D. Sijacic, and N. Smart. Bristol fashion MPC circuits. https://nigelsmart.github.io/MPC-Circuits/, 2019.
[3] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.
[4] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.
[5] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko. MOTION - A framework for mixed-protocol multi-party computation. *TOPS*, 2022.
[6] L. Braun, M. Huppert, N. Khayata, T. Schneider, and O. Tkachenko. FUSE - Flexible file format and intermediate representation for secure multi-party computation. In *ASIACCS*, 2023.
[7] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
[8] D. Demmler, T. Schneider, and M. Zohner. ABY – A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
[9] K. EdalatNejad, W. Lueks, J. P. Martin, S. Ledésert, A. L'Hôte, B. Thomas, L. Girod, and C. Troncoso. DatashareNetwork: A decentralized privacy-preserving search engine for investigative journalists. In *USENIX Security*, 2020.
[10] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT*, 2017.
[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.
[12] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *CCS*, 2020.
[13] L. Malka. VMCrypt: Modular software architecture for scalable secure computation. In *CCS*, 2011.
[14] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *S&P*, 2017.
[15] A. Patra, T. Schneider, A. Suresh, and H. Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*, 2021.
[16] M. Rosulek and L. Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In *CRYPTO*, 2021.
[17] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *S&P*, 2015.
[18] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, 1986.

## ACKNOWLEDGMENTS

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI).

It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230. The authors want to thank Raine Nieminen for his initial inputs that led to this project. The authors used Grammarly to revise the text in the paper to correct any typos, grammatical errors, and awkward phrasing.