TECHNISCHE
UNIVERSITÄT
DARMSTADT

Advancing MPC: From Real-World Applications to LUT-Based Protocols

at the Department of Computer Science
of the Technical University of Darmstadt

Doctoral Thesis

submitted in fulfillment of the requirements for the degree of
Doctor of Engineering (Dr.-Ing.)

by

Hossein Yalame, M.Sc.

Advisors: Prof. Dr.-Ing. Thomas Schneider
Prof. Dr. Arpita Patra

Date of submission: 24.06.2024
Date of defense: 10.09.2024

D 17
Darmstadt, 2024

Hossein, Yalame
Advancing MPC: From Real-World Applications to LUT-Based Protocols
Darmstadt, Technical University of Darmstadt
Publication year of the dissertation on TUprints: 2024
URN: : urn:nbn:de:tuda-tuprints-281688
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/28168
Date of defense: 10.09.2024

Erklärung

Hiermit versichere ich, Hossein Yalame, M.Sc., die vorliegende Doctoral Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Die deutsche Übersetzung der Zusammenfassung wurde mit dem Tool DeepL erstellt. Ich habe ChatGPT-4 verwendet, um den Text zu überarbeiten und Tipp- sowie Grammatikfehler zu korrigieren. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, 24.06.2024

Hossein Yalame, M.Sc.

Abstract

Secure multi-party computation (MPC) is a cryptographic protocol that allows multiple parties to collaboratively compute a public function using their private inputs, ensuring the confidentiality of these inputs while revealing only the final output. This technology is crucial in a variety of fields, including privacy preserving machine learning (PPML) and the emerging field of federated learning (FL). However, the current approaches for MPC incur significant communication and computational overhead, leading to increased communication and runtime costs in comparison to non-private counterparts.

This thesis explores the feasibility and potential improvements of MPC for real-world applications, focusing on two critical aspects:

1) This work demonstrates how MPC provides feasible privacy-preserving solutions in practical applications.

2) It identifies new methods that significantly improve the computational and communication efficiency of MPC.

Practical Privacy-Preserving Services Many machine learning (ML) services depend on training data that contains sensitive information from various sources. MPC in PPML allows multiple parties to collaboratively work on their shared data without revealing their individual inputs to each other.

Building on existing practical privacy-preserving services, our research explores the efficiency of such services by focusing on clustering—an important unsupervised ML technique for grouping data. Our comprehensive review and analysis of 59 studies dedicated to privacy-preserving clustering reveal information leakage in the majority of these studies (49 out of 59). We implement and evaluate four efficient and fully private protocols: Cheon et al. (SAC'19), Mohassel et al. (PETS'20), Meng et al. (CCSW'21), and Bozdemir et al. (ASIACCS'21), with each protocol enhancing the privacy of a different clustering algorithm. Additionally, we conduct benchmarks of these protocols to evaluate their clustering quality, communication efficiency, and computational overhead, thus providing a detailed comparison of their effectiveness.

Expanding upon our exploration of privacy-preserving clustering, our research extends to FL, a distributed ML method that inherently protects data privacy by allowing clients to jointly train a global model through an aggregator without exposing their training data. FL not only improves privacy but also benefits from the computational power and data of potentially millions of clients concurrently. However, FL is vulnerable to poisoning attacks from malicious clients introducing false data, and to inference attacks by malicious aggregator(s) who can deduce information about clients' data from their models. To address these issues, our research involves a critical analysis and identification of vulnerabilities in the only existing solution (Liu et al., IEEE TIFS'21) that addresses both poisoning attacks and inference attacks simultaneously, leading to the introduction of FLAME. FLAME is designed to protect

against both poisoning and inference attacks. Through our extensive evaluations across various ML applications and datasets, FLAME effectively prevents poisoning attacks without compromising the accuracy of the model. Moreover, we develop, implement, and benchmark specialized two-party computation (2PC) protocols within FLAME, ensuring the privacy of client training data and protection against inference attacks on their models.

This part of the thesis is based on the following three publications:

[HMSY21] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, pp. 225–248. CORE Rank **A**. Appendix A.

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2021/025. USENIX Association, 2022, pp. 1415–1432. CORE Rank **A***. Appendix B.

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), pp. 1407–1409. CORE Rank **A**. Appendix C.

MPC Protocols & Optimizations Generic MPC protocols efficiently evaluate Boolean or arithmetic circuits using two main approaches: 1) low-latency, utilizing the garbled circuit (GC) protocol for constant-round solutions, and 2) high-throughput, employing the Goldreich-Micali-Wigderson (GMW) protocol to minimize communication and improve parallelization. A notable feature of these protocols is their division into two phases: an input-independent setup phase that allows for pre-computing expensive cryptographic primitives such as oblivious transfer (OT), resulting in an extremely fast online phase once inputs are available.

The challenge with GC-based protocols lies in the computational overhead of cryptographic operations, like advanced encryption standard (AES), during the online phase. For instance, the most efficient scheme today requires computing 9 fixed-key AES operations for each AND gate (Rosulek and Roy, CRYPTO'21). While GMW-based protocols avoid cryptographic operations in the online phase, they are impeded by the communication rounds needed, which depend on the depth of the circuit being evaluated.

To improve the efficiency of GC-based protocols, we explore the use of vector AES (VAES), a recent innovation by Intel that enables the parallel computation of multiple AES operations. Our aim is to utilize these cutting-edge hardware capabilities to optimize GC-based protocols for practical applications.

For the GMW-based protocols, we aim to achieve practical efficiency through function-dependent pre-processing, which yields substantial improvements by using knowledge of the underlying function to be evaluated. This approach is particularly beneficial in contexts like machine learning as a service (MLaaS), where a single function is repeatedly evaluated with different inputs. Our developments, ABY2.0 and FLUTE, improve the online communication

efficiency of the GMW protocol in a two-party setting. We also design efficient protocols for the secure evaluation of multi-input AND gates and look-up tables (LUTs), alongside novel constructions based on LUTs to enhance private function evaluation (PFE) efficiency. Moreover, we automatically generate circuits optimized for these protocols.

Another area of our research addresses the common MPC assumption of symmetric trust, which assumes that all parties are either semi-honest or malicious. However, this assumption may not align with the asymmetric nature of trust found in real-world scenarios, shaped by reputation, power dynamics, and incentives. To accommodate this, we extend our 2PC framework from ABY2.0 to an honest-majority three-party computation (3PC) setting, designed to handle scenarios where only one party is malicious while the other parties remain semi-honest. This extension considers cases where the malicious party acts as a helper in ABY2.0 or serves as the evaluating party in the 3PC setting.

This part of the thesis is based on the following five publications:

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, pp. 131–145. CORE Rank **A**. Appendix D.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, pp. 2165–2182. CORE Rank **A***. Appendix E.

[BHS+23] A. BRÜGGEMANN, R. HUNDT, T. SCHNEIDER, A. SURESH, H. YALAME. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, pp. 515–533. CORE Rank **A***. Appendix F.

[DGS+23] Y. DISSER, D. GÜNTHER, T. SCHNEIDER, M. STILLGER, A. WIGANDT, H. YALAME. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 14438. LNCS. Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/LUC. Springer, 2023, pp. 3–37. CORE Rank **A**. Appendix G.

[BSS+24] A. BRÜGGEMANN, O. SCHICK, T. SCHNEIDER, A. SURESH, H. YALAME. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

This thesis is dedicated to advancing the practical application of MPC in real-world scenarios. It focuses on optimizing low-level building blocks and developing efficient privacy-preserving solutions that are specifically designed for practical use cases.

Zusammenfassung

Sichere Mehrparteienberechnung (im Englischen *secure multi-party computation*, MPC) ist ein kryptografisches Protokoll, das es mehreren Parteien ermöglicht, gemeinsam eine öffentliche Funktion unter Verwendung ihrer privaten Eingaben zu berechnen und dabei die Vertraulichkeit dieser Eingaben zu gewährleisten, während nur das Endergebnis offengelegt wird. Diese Technologie ist in einer Reihe von Bereichen von entscheidender Bedeutung, einschließlich des datenschutzfreundlichen maschinellen Lernens (im Englischen *privacy preserving machine learning*, PPML) und des aufkommenden Bereichs des föderierten Lernens (im Englischen *federated learning*, FL). Die derzeitigen Ansätze für MPC erfordern jedoch einen erheblichen Kommunikations- und Rechenaufwand, was zu erhöhten Kommunikations- und Laufzeitkosten im Vergleich zu nicht-privaten Gegenstücken führt.

In dieser Arbeit werden die Machbarkeit und das Verbesserungspotenzial von MPC für reale Anwendungen untersucht, wobei der Schwerpunkt auf zwei kritischen Aspekten liegt:

1) Diese Arbeit zeigt, wie MPC in praktischen Anwendungen praktikable Lösungen zur Wahrung der Privatsphäre bietet.

2) Sie zeigt neue Methoden auf, die die Rechen- und Kommunikationseffizienz der MPC deutlich verbessern.

Praktische Dienste zum Schutz der Privatsphäre Viele Dienste des maschinellen Lernens (ML) sind auf Trainingsdaten angewiesen, die sensible Informationen aus verschiedenen Quellen enthalten. MPC im PPML ermöglicht es mehreren Parteien, gemeinsam an ihren gemeinsamen Daten zu arbeiten, ohne ihre individuellen Eingaben einander preiszugeben

Basierend auf bestehenden praktischen Diensten zur Wahrung der Privatsphäre untersucht unsere Forschung die Effizienz solcher Dienste, indem sie sich auf das Clustering konzentriert – eine wichtige unüberwachte ML-Technik zur Gruppierung von Daten. Unsere umfassende Überprüfung und Analyse von 59 Arbeiten, die sich dem datenschutzbewahrenden Clustering widmen, zeigen Informationslecks in der Mehrheit dieser Arbeiten auf (49 von 59). Wir implementieren und bewerten vier effiziente und vollständig private Protokolle: Cheon et al. (SAC'19), Mohassel et al. (PETS'20), Meng et al. (CCSW'21) und Bozdemir et al. (ASIACCS'21), wobei jedes Protokoll die Privatsphäre bei einem anderen Clustering-Algorithmus verbessert. Zusätzlich führen wir Benchmarks dieser Protokolle durch, um ihre Clustering-Qualität, Kommunikationseffizienz und Rechenüberlast zu bewerten und so einen detaillierten Vergleich ihrer Wirksamkeit zu bieten.

Aufbauend auf unserer Untersuchung des datenschutzbewahrenden Clusterings, erweitert sich unsere Forschung auf FL, eine verteilte ML-Methode, die den Datenschutz inhärent gewährleistet. Kunden können durch einen Aggregator ein globales Modell trainieren, ohne ihre Trainingsdaten offenzulegen. FL verbessert nicht nur die Privatsphäre, sondern profitiert auch von der Rechenleistung und den Daten potenziell Millionen von Kunden gleichzeitig. Jedoch ist FL anfällig für Poisoning-Angriffe durch bösartige Kunden, die falsche Daten

einführen, und für Inference-Angriffe durch bösartige Aggregatoren, die Informationen aus den Modellen der Kunden ableiten können. Um diese Probleme anzugehen, beinhaltet unsere Forschung eine kritische Analyse und Identifizierung von Schwachstellen in der einzigen bestehenden Lösung (Liu et al., IEEE TIFS'21), die sowohl Vergiftungsangriffe als auch Inferenzangriffe gleichzeitig behandelt, was zur Einführung von FLAME führt. FLAME ist darauf ausgelegt, sowohl gegen Poisoning- als auch Inference-Angriffe zu schützen. Durch unsere umfangreichen Bewertungen über verschiedene ML-Anwendungen und Datensätze hinweg verhindert FLAME effektiv Poisoning-Angriffe, ohne die Genauigkeit des Modells zu beeinträchtigen. Darüber hinaus entwickeln, implementieren und bewerten wir spezialisierte Protokolle zur Zwei-Parteien-Berechnung (im Englischen secure two-party computation, 2PC), die die Privatsphäre der Trainingsdaten der Kunden und den Schutz gegen Inference-Angriffe auf deren Modelle gewährleisten.

Dieser Teil der Arbeit basiert auf den folgenden drei Veröffentlichungen:

[HMSY21] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, S. 225–248. CORE Rank **A**. Appendix A.

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), S. 1407–1409. CORE Rank **A**. Appendix C.

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2021/025. USENIX Association, 2022, S. 1415–1432. CORE Rank **A***. Appendix B.

MPC-Protokolle & Optimierungen Generische MPC-Protokolle bewerten Boolesche oder arithmetische Schaltkreise unter Verwendung von zwei Hauptansätzen: 1) geringe Latenz, die das Garbled-Circuit-Protokoll (GC) für Lösungen mit konstanter Runde nutzt, und 2) hoher Durchsatz, der das Goldreich-Micali-Wigderson-Protokoll (GMW) verwendet, um die Kommunikation zu minimieren und die Parallelisierung zu verbessern. Ein wichtiges Merkmal dieser Protokolle ist ihre Aufteilung in zwei Phasen: eine eingabeunabhängige Setup-Phase, die das Vorrechnen von aufwendigen kryptografischen Primitiven wie dem oblivious transfer (OT) ermöglicht, gefolgt von einer extrem schnellen Online-Phase, sobald die Eingaben verfügbar sind.

Die Herausforderung bei auf GC basierenden Protokollen liegt im Rechenaufwand kryptografischer Operationen, wie dem Advanced Encryption Standard (AES), während der Online-Phase. Beispielsweise erfordert das effizienteste heutige Schema die Berechnung von 9 festgelegten AES-Operationen für jedes AND-Gatter (Rosulek und Roy, CRYPTO'21). Während GMW-basierte Protokolle kryptografische Operationen in der Online-Phase vermeiden, werden sie durch die benötigten Kommunikationsrunden, die von der Tiefe des zu bewertenden Schaltkreises abhängen.

Um die Effizienz von auf GC basierenden Protokollen zu verbessern, erforschen wir die Verwendung von Vector AES (VAES), einer jüngsten Innovation von Intel, die die parallele Berechnung mehrerer AES-Operationen ermöglicht. Unser Ziel ist es, diese modernen Hardware-Fähigkeiten zu nutzen, um GC-basierte Protokolle für praktische Anwendungen zu optimieren.

Bei den auf GMW basierenden Protokollen zielen wir darauf ab, praktische Effizienz durch funktionsspezifische Vorverarbeitung zu erreichen, die erhebliche Verbesserungen erbrachte, indem das Wissen über die zu bewertende Funktion genutzt wurde. Dieser Ansatz ist besonders vorteilhaft in Kontexten wie maschinellem Lernen als Dienstleistung (im Englischen machine learning as a service, MLaaS), wo eine einzelne Funktion wiederholt mit verschiedenen Eingaben bewertet wird. Unsere Entwicklungen, ABY2.0 und FLUTE, haben die Online-Kommunikationseffizienz des GMW-Protokolls in einer 2PC deutlich verbessert. Wir haben auch effiziente Protokolle für die sichere Bewertung von Multi-Input-AND-Gattern und Lookup-Tabellen (LUT) entworfen, zusammen mit neuen Konstruktionen basierend auf LUTs, um die Effizienz der privaten Funktionsbewertung (im Englischen private function evaluation, PFE) zu erhöhen. Darüber hinaus haben wir automatisch Schaltkreise generiert, die für diese Protokolle optimiert sind.

Ein weiterer Bereich unserer Forschung befasst sich mit der gängigen Annahme symmetrischen Vertrauens bei MPC, die davon ausgeht, dass alle Parteien entweder halb-ehrlich sind oder zu bösartigem Verhalten fähig sind. Diese Annahme entspricht jedoch möglicherweise nicht der asymmetrischen Natur des Vertrauens, wie sie in realen Szenarien vorkommt, die durch Reputation, Machtverhältnisse und Anreize geprägt sind. Um dies zu berücksichtigen, erweitern wir unser 2PC-Framework von ABY2.0 zu einer Drei-Parteien-Berechnung (im Englischen three-party computation, 3PC) mit ehrlicher Mehrheit, das für Szenarien konzipiert ist, in denen nur eine Partei bösartig ist und die anderen Parteien halb-ehrlich sind. Diese Erweiterung berücksichtigt Fälle, in denen die bösartige Partei als Helfer in ABY2.0 agiert oder als auswertende Partei im 3PC-Szenarien dient.

Dieser Teil der Arbeit basiert auf den folgenden fünf Veröffentlichungen:

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, S. 131–145. CORE Rank **A**. Appendix D.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, S. 2165–2182. CORE Rank **A***. Appendix E.

[BHS+23] A. BRÜGGEMANN, R. HUNDT, T. SCHNEIDER, A. SURESH, H. YALAME. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, S. 515–533. CORE Rank **A***. Appendix F.

[DGS+23] Y. DISSER, D. GÜNTHER, T. SCHNEIDER, M. STILLGER, A. WIGANDT, H. YALAME. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Bd. 14438. LNCS. Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/LUC. Springer, 2023, S. 3–37. CORE Rank **A**. Appendix G.

[BSS+24] A. BRÜGGEMANN, O. SCHICK, T. SCHNEIDER, A. SURESH, H. YALAME. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

Diese Dissertation widmet sich der Förderung der praktischen Anwendung von MPC in realen Szenarien. Sie konzentriert sich auf die Optimierung von grundlegenden Bausteinen und die Entwicklung effizienter datenschutzbewahrender Lösungen, die speziell für praktische Anwendungsfälle konzipiert sind.

My Contributions

This thesis comprises eight scientific publications, four A* and four A-ranked papers, co-authored by me, my PhD advisor Thomas Schneider, and many researchers from both academia and industry: Aditya Hegde (Johns Hopkins University, USA), Helen Möllering (McKinsey & Company, Germany), Arpita Patra (Indian Institute of Science Bangalore, India), Ajith Suresh (Technology Innovation Institute, UAE), Jean-Pierre Münch (Technical University of Darmstadt (ENCRYPTO), Germany), Thien Duc Nguyen (Technical University of Darmstadt (System Security Lab), Germany), Phillip Rieger (Technical University of Darmstadt (System Security Lab), Germany), Huili Chen (Amazon, USA), Hossein Fereidooni (KOBIL, Germany), Samuel Marchal (VTT, Finland), Markus Miettinen (Frankfurt University of Applied Sciences, Germany), Azalia Mirhoseini (Stanford University, USA), Shaza Zeitouni (Technical University of Darmstadt (System Security Lab), Germany), Farinaz Koushanfar (University of California San Diego, USA), Ahmad Reza Sadeghi (Technical University of Darmstadt (System Security Lab), Germany), Andreas Brüggemann (Technical University of Darmstadt (ENCRYPTO), Germany), Robin Hundt (Technical University of Darmstadt (ENCRYPTO), Germany), Yann Disser (Technical University of Darmstadt (Department of Mathematics), Germany), Daniel Günther (Technical University of Darmstadt (ENCRYPTO), Germany), Maximilian Stillger (Technical University of Darmstadt (ENCRYPTO), Germany), Arthur Wigandt (Technical University of Darmstadt (Department of Mathematics), Germany), and Oliver Schick (Technical University of Darmstadt (ENCRYPTO), Germany). I thank all my co-authors for these very successful collaborations and detail my own contributions below.

The idea of conducting a Systemization of Knowledge (SoK) on privacy-preserving clustering [**HMSY21c**] was originally proposed by me. This publication emerged from Aditya Hegde's internship with the ENCRYPTO group in 2020, which was co-supervised by Helen Möllering and myself. Through our research efforts, we established that among 59 examined works, only 10 fully ensured privacy protection. Additionally, I identified four clustering protocols that not only fully preserved privacy but also exhibited efficiency, making them ideal for benchmark analysis. My evaluation included a thorough examination of their security and privacy features, alongside an assessment of their performance based on asymptotic communication and computational complexity. I also significantly contributed to shaping the experimental evaluation's design. This work was conducted in collaboration with Aditya Hegde, Helen Möllering, and Thomas Schneider.

The foundational idea behind the attack detailed in [**SSY23**] was developed by me in collaboration with Thomas Schneider and Ajith Suresh. In this work, I illustrated the shortcomings of an existing solution in the literature, specifically its inability to preserve privacy. My analysis revealed that this approach inadvertently exposes the complete gradient vector of all participating users in an unencrypted format to one of the involved parties, thus highlighting a significant privacy vulnerability in the existing solution.

In the development of FLAME [**NRC+22**], I played a pivotal role in designing the mixed 2PC protocols, which were essential for ensuring secure aggregation and effective clipping. Phillip

Rieger took the lead on the entire implementation process and subsequent benchmarking efforts, while Helen Möllering led the protocol design for the HDBSCAN approximation and provided guidance on implementation aspects. I provided expert guidance on the aspects of 2PC relevant to our project, in addition to assisting with the generation of circuits for 2PC protocols. Furthermore, Helen Möllering contributed to systematizing related work on backdoor and inference attacks. Beyond the technical design and implementation, I made substantial contributions to the systematic review of existing literature on secure aggregation within the context of FL, along with a critical analysis of their limitations. This work was conducted in collaboration with Thien Duc Nguyen, Phillip Rieger, Helen Möllering, Huili Chen, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Shaza Zeitouni, Farinaz Koushanfar, Ahmad Reza Sadeghi, Helen Möllering, and Thomas Schneider.

The work presented in [**MSY21**] was derived from Jean-Pierre Münch's Master's thesis, which I co-supervised alongside Thomas Schneider. My role in this project involved conducting a thorough review of various MPC frameworks to identify opportunities for enhancement through the integration of new vector AES (VAES). Jean-Pierre Münch focused primarily on the incorporation of VAES into the MPC frameworks, a process for which I offered detailed advice on how to effectively apply VAES across different frameworks. I was involved in setting up the benchmarking process to evaluate our improvement.

The initiative for the work detailed in [**PSSY21a**] was ignited by my original idea to utilize multi-input AND gates within 2PC. Following this concept, I developed new circuit designs for a range of important primitives, including parallel prefix adder (PPA), bit extraction, comparison, and equality test, all employing multi-input AND gates to improve efficiency. Moreover, I created a depth-optimized version of the AES circuit. In addition to these technical contributions, I analyzed the communication and round complexity for our newly established circuits over different ℓ-bit ring sizes, providing a detailed evaluation of the improvements our work introduced. This work was done in collaboration with Arpita Patra, Thomas Schneider, and Ajith Suresh.

Motivated by the innovative use of multi-input AND gates, I introduced the approach for evaluating LUTs as elaborated in [**BHS+23**]. My contribution was the development of a novel method for LUT evaluation characterized by its unique communication efficiency: the setup phase communication is independent of the number of LUT outputs, while the online phase communication remains unaffected by the number of LUT inputs. I utilized hardware synthesis tools to create optimized LUT circuit designs. This work was carried out with Andreas Brüggemann, Robin Hundt, Thomas Schneider, and Ajith Suresh.

In [**DGS+23**], I introduced the idea of evaluating universal circuits (UCs) by employing LUT-based MPC protocols. My analysis focused on the application of our innovative LUT-based PFE constructions in cutting-edge areas, such as intellectual property (IP) protection. To demonstrate the practicality of these concepts, I used IP libraries available in the Synopsys Design Compiler, a leading commercial ASIC synthesis tool, to generate LUT-based circuit netlists capable of executing complex tasks, including floating-point operations, by developing

new technology libraries designed to create more optimized LUT-based circuits compared to those found in the literature. In collaboration with our student assistant, Joachim Schmidt, I also carried out extensive benchmarking to evaluate the performance of our newly developed constructions. This work was conducted with Yann Disser, Daniel Günther, Thomas Schneider, Maximilian Stillger, and Arthur Wigandt.

The foundational concept behind the work presented in [**BSS+24**] originated from a productive dialogue between Ajith Suresh and myself. I examined the current body of research concerning MPC in asymmetric trust model. Furthermore, I was responsible for setting up our implementation and, with the valuable support of our student assistant, Maximilian Stillger, conducted comprehensive benchmarking activities. These efforts were aimed at comparing our contributions with recent advancements in asymmetric MPC settings and highlighting the innovations of our approach. This work was done in collaboration with Andreas Brüggemann, Oliver Schick, Thomas Schneider, and Ajith Suresh.

Contents

1 Introduction

In the rapidly evolving digital era, the ability to collect and analyze data is crucial for advancements across various areas, including healthcare, business development, and smart cities. This process, however, presents the significant challenge of increasing privacy risks associated with large-scale data collection. With the growing sensitivity around privacy concerns, particularly regarding sensitive information like financial and health records, the necessity to safeguard both personal and business interests becomes critical. This need is further highlighted by stringent legal regulations such as the EU's General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), which require strict compliance standards for data protection and impose significant penalties for violations. Consequently, the main challenge we face is the development and implementation of systems capable of protecting sensitive user data while also allowing businesses and consumers to benefit from data access and utilization.

Machine learning (ML) plays a crucial role in this landscape, serving as an essential tool to handle vast datasets. Advances in ML technology not only enhance business operations internally but also enable companies to provide these advanced ML capabilities to external clients as a service, known as machine learning as a service (MLaaS). Consider, for example, a scenario where a MLaaS service provider, such as Google or Amazon, provides applications like credit scoring [1] or fraud detection [2] to online stores. These applications, built on pre-trained ML models, enable online stores to make decisions, such as whether to approve or reject invoice-based orders.

This process requires the sharing of customer data by the online store and the sharing of ML models by the MLaaS service provider. While the goal is to solve a specific ML challenge, it presents a significant dilemma: the unwillingness of both parties to share their own data. The online store may have concerns about revealing its clients' identities to the MLaaS service provider, reflecting broader concerns over data privacy. Simultaneously, the service provider wants to protect its ML models, considering them as critical intellectual property (IP) that supports its market position. This careful balancing of safeguarding both data privacy and model confidentiality highlights the challenges faced in deploying MLaaS solutions.

To address both the previously mentioned issue and the challenges associated with processing distributed data, privacy-enhancing technologies (PETs) can be used. These technologies enable the sharing of data while preserving its privacy, facilitating secure computation

[1] https://cloud.google.com/customers/credit-ok
[2] https://aws.amazon.com/fraud-detector/

over the data. While several PETs are explored in the literature, four have proven to offer particularly efficient solutions: secure multi-party computation (MPC), fully homomorphic encryption (FHE), zero knowledge (ZK), and differential privacy (DP).

In this thesis, we focus on privacy-preserving protocols using MPC. MPC enables a group of parties to securely compute a joint function on their private data. It ensures that only the final results are revealed, preventing disclosure of any data beyond what can be derived from the output. We focus on MPC because it offers several practical advantages: FHE enables computations on encrypted data but is computationally intensive due to the complex cryptographic operations involved. ZK is used for verifying correctness without revealing data but is limited to specific proofs. DP differs from MPC as it guarantees the privacy of the output, whereas MPC focuses on the privacy of the computation process.

Use Cases of MPC MPC is used in a variety of real-world applications, including financial services [ASA22], private clustering [**HMSY21c**; **KMSY21**], privacy preserving machine learning (PPML) [CCPS19; **BCD+20**; PS20], and federated learning (FL) [BIK+17; **NRC+22**; **MSS+23**; **BMP+24**].

The development of MPC over the last four decades has been significant. In the 1990s, it was mainly considered an interesting theoretical concept. By the time my doctoral studies started, early versions capable of handling small-scale tasks with MPC were already in place. Now, there is a wide range of open-source MPC tools available, drawing interest from large companies such as Meta[3], Bosch[4], and Microsoft[5], as well as smaller companies like Cosmian[6] and Galois[7]. These organizations are exploring MPC for practical applications. A notable recent effort is the creation of the MPC alliance[8], a collaboration of industry leaders with solid academic foundations, who recognize MPC as an important technological tool in their work.

Our Contributions In this thesis, we begin by evaluating the performance of existing MPC protocols. Our work involves optimizing and implementing these protocols, demonstrating how they can be combined, and illustrating their effectiveness in various applications, including multiple clustering algorithms (cf. Sect. 2.1.1) and FL (cf. Sect. 2.1.2). We further improve MPC efficiency by incorporating recent advancements in hardware acceleration techniques like Intel's vector AES (VAES) (cf. Sect. 3.1.1), function-dependent preprocessing (cf. Sect. 3.1.2), and utilizing asymmetric trust settings (cf. Sect. 3.1.4). We develop a new series of hybrid protocols for two-party computation (2PC) and three-party computation (3PC) that significantly improve the efficiency of the input-dependent online phase over state-of-the-art work. Additionally, we make significant progress in private function evaluation (PFE) and successfully implement these advancements (cf. Sect. 3.1.3).

[3] https://github.com/facebookresearch/fbpcf
[4] https://github.com/carbynestack/carbynestack
[5] https://github.com/mpc-msri/EzPC
[6] https://github.com/Cosmian/CipherCompute
[7] https://github.com/GaloisInc/swanky
[8] https://www.mpcalliance.org/

Open Access All eight research papers included in this thesis are available as open access via the Cryptology ePrint Archive[9]. Additionally, we provide five open-source demonstrators and prototype implementations for [**HMSY21c**; **MSY21**; **BHS+23**; **DGS+23**; **BSS+24**]. These implementations are based on our research findings, enabling other researchers to fairly compare their work with ours, build upon our results, and use our developments to further improve the practicality of MPC.

Additional Contributions Although not included in this thesis due to their focus on engineering efforts, there are two noteworthy contributions from our research:

An important highlight is our collaboration with Intel Labs and the development of the MP2ML framework [**BCD+20**]. This collaboration results in the ability to transform high-level neural network inference code into efficient privacy-protecting protocols. By making this framework open source and widely available, we demonstrate our commitment to practical implementations of PETs.

Another significant achievement of our research emerges from our collaboration within the Private AI Collaborative Research Institute[10], involving VMWare Research, Aptos Labs, Technology Innovation Institute, and DFINITY Foundation. We develop a method to use quantization to minimize client-server communication in secure FL [**BMP+24**]. This innovation is recognized with the Runner-Up Distinguished Paper Award at the 2nd IEEE Conference on Secure and Trustworthy Machine Learning (SaTML'24).

[9] https://eprint.iacr.org/
[10] https://www.private-ai.org/

2 Practical Privacy-Preserving Services

Machine learning (ML) technologies are increasingly being integrated across various fields, such as autonomous driving [CLH+23], medical diagnosis [LHR23], and natural language processing [LYF+23], where they play a crucial role in analyzing and interpreting vast datasets. In addition to these applications, leading technology companies, including Apple, Facebook, Google, and Microsoft, have gathered large data collections to utilize for commercial analysis and profit [GSCM07]. However, growing privacy concerns among data owners have begun to present significant challenges to the traditional methods of data collection. These privacy issues highlight the need for innovative approaches to train ML models, thereby establishing a basis for privacy preserving machine learning (PPML) [MZ17; MR18; RRK+20a; **HMSY21c**; **PSSY21a**]. Although existing PPML techniques, which utilize secure multi-party computation (MPC) [Lin20] and/or homomorphic encryption (HE) [AAUC18], offer promising solutions, they often come with substantial communication and computational overheads.

In response to the growing demand for methods that are both cost-effective and capable of preserving privacy, Google has developed federated learning (FL) [KMY+16]. FL enables users to train models directly on their devices. Following this local training phase, a central server aggregates these individual updates to refine the global model, using techniques such as FedAvg [MMR+17]. A key benefit of FL is that it keeps user data on the device itself, thereby enhancing user trust by ensuring data privacy. This method has quickly become popular across both academic [HLS+20] and industrial research communities [BEG+19], leading to the implementation of numerous real-world applications such as video analytics[1].

However, FL faces its own set of challenges, especially concerning vulnerabilities to *privacy inference* and *poisoning* attacks from malicious actors within the system. Privacy inference attacks, which may be conducted by a compromised model aggregator, seek to extract sensitive information from the model updates or gradients shared by users [MSCS19; NSH19]. On the other hand, poisoning attacks occur when corrupt users send manipulated models to the central aggregator with the intention of disrupting the training process. This disruption can take the form of either a decrease in overall model accuracy or the insertion of a backdoor that triggers altered predictions under specific conditions [FCJG20; SHKR22a]. Addressing both types of attacks simultaneously is very challenging: strategies to mitigate poisoning attacks often involve analyzing each user's updates individually, while measures against privacy inference attacks focus on safeguarding the privacy of aggregated models, thereby eliminating the possibility of inspecting individual updates.

[1] https://developer.nvidia.com/blog/federated-learning-clara/

After reviewing the current state of research, it becomes clear that there is a notable gap in evaluating how practical existing methods for privacy-preserving ML and FL truly are. This concern mainly centers around the computational demands of these approaches and their efficacy in protecting data privacy. This observation leads us to a critical question:

> *How feasible are secure computation techniques in providing privacy within ML and FL settings?*

This chapter outlines our contributions to address the research question by utilizing efficient two-party computation (2PC) frameworks. These frameworks aim to mitigate privacy concerns within ML and FL areas and evaluate the practicality of using 2PC in these fields. Specifically, in the realm of ML, we concentrate on clustering—an unsupervised learning method that organizes data into separate groups.

In the first part of this chapter (cf. Sect. 2.1.1), we systematically analyze 59 papers focusing on privacy-preserving clustering to evaluate their practical applicability [**HMSY21c**]. Our analysis finds that only 10 of these works provide full privacy protection, ensuring no information is leaked beyond the intended output. Additionally, we select, implement, and benchmark the four most advanced and efficient protocols in fully privacy-preserving clustering [CKP19; MRT20; BCE+21; MPOT21], analyzing them based on communication overhead and computational efficiency. This review enables us to identify and discuss their potential shortcomings for deployment in real-world scenarios.

In the second part of this chapter (cf. Sect. 2.1.2), we explore the privacy solution proposed by the only work [LLX+21] that addresses both privacy inference and poisoning attacks in FL. We analyze and thoroughly review this work in [**SSY23**], which informs our subsequent development of FLAME [**NRC+22**]. FLAME [**NRC+22**] employs 2PC to tackle both privacy inference and poisoning attacks concurrently. We conduct a detailed evaluation of FLAME, testing its resistance against various poisoning attacks across multiple datasets and application scenarios. Our testing verifies that FLAME effectively mitigates the impact of poisoning while preserving model accuracy. Additionally, we demonstrate that FLAME can achieve practical run-time performance.

2.1 Our Contributions

2.1.1 First Systematization of Knowledge for Privacy-Preserving Clustering

Clustering, an unsupervised ML technique, organizes a dataset into separate groups. It is widely used for various purposes, such as market division based on customer preferences [CCGR97] and categorizing images of unhealthy organs in medical imaging [MS99]. These uses often involve handling sensitive data, such as proprietary business details or confidential medical records. Our research concentrates on situations involving multiple parties, such as businesses or healthcare institutions, that lack mutual trust to share their

data about clients or patients. Despite this mistrust, there is a mutual interest in clustering their aggregated data to achieve better insights and more accurate results, as a larger dataset typically improves clustering effectiveness. Over the years, 59 papers have contributed to the field of privacy-preserving clustering by using secure computation techniques. This section presents *the first systematic* review and empirical evaluation of these contributions, analyzing them from both theoretical perspectives and practical implementations, as detailed in the following publication available in Appendix A.

[HMSY21] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, pp. 225–248. CORE Rank **A**. Appendix A.

Evaluating Privacy-Preserving Clustering Protocols In our analysis, we systematically review and evaluate the various techniques and protocols developed for privacy-preserving clustering. We examine the strengths and limitations of these approaches with respect to several critical aspects: i) the underlying plaintext clustering algorithm, ii) the security model, iii) the specific scenarios for which the protocols were designed, iv) the types of data distributions handled, v) the secure computation techniques utilized, vi) the levels of privacy, and vii) efficiency achieved. A comprehensive comparison table (cf. [Table. 3] in Appendix A) summarizes our review of all 59 works focused on privacy-preserving clustering. From this in-depth review, we identify the following main criteria for selecting privacy-preserving clustering protocols:

1. *Privacy:* A protocol dedicated to privacy-preserving clustering achieves *full privacy protection* if it reveals no information beyond what can be derived from the protocol's output.
2. *Efficiency:* We consider a privacy-preserving clustering approach efficient if it can handle increasing workload efficiently. This includes how well the protocol scales in terms of communication and computational time relative to the dataset's size N, the number of clusters K, and the dimensionality d of the input data.

Lesson Learned Through our comprehensive analysis, we identify only ten protocols that meet stringent privacy standards without any information leakage. Within this select group, just four protocols are determined to provide efficient solutions (cf. Table. 1 in Appendix A). Among these, the MPC-based DBSCAN [BCE+21] was already implemented. We implement the other three protocols: HE-based Mean-shift [CKP19], MPC-based K-means [MRT20], and hybrid HE-MPC-based hierarchical clustering [MPOT21]. We then benchmark all four protocols, conducting a detailed comparison of their asymptotic runtime, communication, and the number of rounds required, presenting these findings in Table 5 in Appendix A. From an asymptotic perspective, the MPC-based K-means [MRT20] protocol is found to be the most efficient in terms of communication and runtime. According to our benchmarks, [MRT20] is

the most suitable choice for clustering large multi-dimensional datasets. Conversely, [CKP19] is preferable for scenarios where a single data owner with limited resources assigns clustering tasks to a more capable server across a network characterized by high latency and low bandwidth. For clustering smaller datasets, [BCE+21] appears to be the optimal choice, performing well on a variety of dataset types and also achieving low runtimes.

Impact We identify and theoretically and experimentally evaluate the most promising protocols for privacy-preserving clustering, clarifying their strengths and limitations. Additionally, our work highlights the research challenges that need to be tackled to make privacy-preserving clustering not just a theoretical concept but also a practical tool for real-world applications. We make all our implementations openly available under the MIT license at `https://encrypto.de/code/SoK_ppClustering`. Furthermore, our research was presented as a contributed talk at the Privacy in Machine Learning Workshop at the Conference on Neural Information Processing Systems (NeurIPS'21) [**HMSY21a**] and at the Privacy Preserving Machine Learning Workshop at the ACM Computer and Communications Security Conference (CCS'21) [**HMSY21b**].

2.1.2 First Privacy-Preserving and Robust Federated Learning

In this section of the thesis, we address both privacy inference and poisoning attacks in FL. Detailed information can be found in the following publications listed in Appendix B and Appendix C.

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: `https://ia.cr/2021/025`. USENIX Association, 2022, pp. 1415–1432. CORE Rank **A***. Appendix B.

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), pp. 1407–1409. CORE Rank **A**. Appendix C.

The work by Liu et al. [LLX+21] introduced a privacy-enhanced federated learning (PEFL) system designed to detect both poisoning and inference attacks within FL. PEFL is the first to address these types of attacks in FL using encrypted data, relying on HE. In our analysis, we carefully review the PEFL framework and identify a number of privacy issues. Specifically, we observe that key protocols within PEFL inadvertently reveal significant information about the user's data gradients to one of the computing servers, thus compromising privacy (see Sect. II in Appendix C for details).

Upon reviewing the shortcomings of PEFL, it is evident that there is no existing solution capable of effectively tackling the two main challenges in FL—poisoning and inference

attacks—simultaneously. As a result, we introduce the first FL framework that not only resists sophisticated poisoning attacks but also significantly enhances protection against inference attacks on the training data.

Privacy-preserving and Robust FL system We propose FLAME [NRC+22] to counteract poisoning attacks by employing a strategy that combines clustering with data clipping, further enhanced by the addition of differentially private noise. This approach effectively neutralizes poisoning efforts. To address inference attacks, FLAME incorporates 2PC, which restricts access to individual model updates, thereby preventing inference attacks by ensuring that two semi-honest aggregators cannot access sensitive local model information. The core components of FLAME, including distance calculation, clustering, data clipping, and model aggregation (detailed in Algorithm 1 in Appendix B), are all executed within a 2PC framework using ABY [DSZ15] to maintain client data privacy. To achieve this, we co-design all components of FLAME as efficient 2PC protocols. This requires representing all functions that need to be computed with 2PC as Boolean circuits (including generating those circuits using conventional logic synthesis tools) and using efficient combinations of the three 2PC protocols: Yao's garbled circuit (GC), arithmetic Goldreich-Micali-Wigderson (A-GMW), and boolean Goldreich-Micali-Wigderson (B-GMW) [DSZ15].

We generate a novel (previously not existing) circuit for square root computation needed for determining cosine and L_2-norm distances (cf. Algorithm 1 in Appendix B) using the Synopsys Design Compiler [10], a conventional logic synthesis tool. We carefully implement the circuit using the Verilog hardware description language (HDL) and compile it with the Synopsys Design Compiler in a highly efficient manner. We customize the flow of the commercial hardware logic synthesis tools to generate circuits optimized for GC, specifically using the Free-XOR optimization technique [KS08b] similar to [SHS+15]. We develop a technology library to guide the mapping of the logic to the circuit without defining specific manufacturing rules.

Efficiency We conduct a thorough evaluation of FLAME's efficiency and effectiveness across a variety of datasets, including word prediction, image classification, and IoT intrusion detection, as detailed in Section 7 in Appendix B. Our results demonstrate that FLAME effectively counters all known poisoning attacks while having a minimal impact on the accuracy of the aggregated model. Furthermore, we find that FLAME is practical for real-world applications. For instance, training a neural network with 2.7 million parameters and 50 clients on the CIFAR-10 dataset requires less than 13 minutes. Even when training the largest model we consider (Reddit) with 20 million parameters and involving 100 clients in each training iteration, the total runtime is 22,081 seconds (approximately 6 hours) per iteration with FLAME. Moreover, our experiments indicate that the communication overhead introduced by FLAME is manageable, showing only a threefold increase compared to the non-private FLAME.

Impact FL has been a groundbreaking development in the realm of large-scale distributed ML over the last decade, significantly influencing both academic research and industrial

applications. In our research, we introduce the first comprehensive solution, FLAME, designed to tackle the challenges of poisoning and privacy attacks in FL within a single unified framework, enabling private and robust distributed ML. By addressing both privacy and robustness concerns simultaneously, FLAME represents one of the initial steps towards wider deployment of FL in real-world applications. This advancement may inspire more clients to engage in FL, contributing to a richer pool of training data and improving the overall effectiveness and reliability of ML models developed under the FL paradigm.

Moreover, building on our attack [**SSY23**], a recent study [WZL24] has exposed privacy weaknesses in the work developed after FLAME [ZWM+22], leading to a lightweight and secure FL framework. This revelation highlights the essential lesson from our analysis: for solutions to be effective and private, researchers must provide provably secure solutions.

2.2 Related Work

In this section, we review the existing work in the fields of PPML (cf. Sect. 2.2.1) and FL (cf. Sect. 2.2.2), highlighting the major developments in these areas. Additionally, we present our contributions, demonstrating how our research builds upon and advances the current understanding of PPML and FL.

2.2.1 PPML

PPML is an active research field, with a strong emphasis on applying cryptographic techniques for privacy in supervised and deep learning models [LJLA17; MZ17; HTGW18; JVC18; **BCD+20**; KRC+20; MLS+20; RRK+20b; NC23]. As the availability of training data, often unlabeled, continues to grow, the relevance of privacy measures in unsupervised learning, such as clustering, has also grown.

In the domain of privacy-preserving clustering, a considerable amount of research has focused on the K-means algorithm. These studies primarily differ in the cryptographic methods they use. A group of these studies leverage HE, such as [JKM05; DT13; LBY14; SM17; JA18; YT19; WLW+20]. However, these approaches often exhibit slow performance due to the heavy cryptographic operations involved. Alternatively, other research adopts MPC strategies, offering more efficient computation [JW05; PGJ12; MRT20], though they require more communication compared to HE methods. Notably, various privacy-preserving K-means methods [VC03; JW05; JKM05; SMO07; GC16; YT19; WLW+20] leak intermediate values like the centroids, thereby not providing appropriate privacy protection. Considering both privacy and efficiency, the MPC-based approach proposed by Mohassel et al. [MRT20] is the most advanced and practical solution available.

A limited number of studies have explored privacy-preserving hierarchical clustering by utilizing HE and MPC [İKS+07; JPWU10; DT13; SM17; MPOT21]. Apart from Meng et al.'s work [MPOT21], these studies often do not include a formal security proof [İKS+07; JPWU10;

DT13] or lack a detailed complexity analysis, implementation, and benchmarks [SM17]. Moreover, the approaches in [JPWU10; DT13] face challenges with scalability, as their performance degrades exponentially with the increase in the number of participants.

In the realm of privacy-preserving density-based clustering, Cheon et al. [CKP19] introduced an outsourcing protocol for the Mean-shift algorithm that uses HE. Bozdemir et al. [BCE+21] developed an efficient, fully privacy-preserving protocol for the DBSCAN clustering algorithm, another density-based method. There have also been works to achieve privacy in DBSCAN clustering through the use of random blinding techniques, HE, or the involvement of partially trusted third parties [AE06; XHL+07; LXLH13; RBK17]. Nonetheless, these methods encounter various drawbacks: some protocols tend to leak information [KR07; RBK17], others require that each participant has access to certain plaintext data for input into the clustering process, making them unsuitable for outsourcing applications [KR07; XHL+07; JXJ+08; AGM+13; LXLH13; AG17]. Additionally, some approaches face scalability issues due to the dependence on heavy cryptographic operations and the need for intensive interaction among the parties involved [KR07; LXLH13; AG17].

To navigate the rapidly expanding field of research on PPML, various surveys have been conducted. Haralampieva et al. [HRP20] provided a comprehensive review of frameworks specifically for private image classification. Tanuwidjaja et al. [TCBK20] compiled a summary of work on privacy-preserving deep learning, covering both the challenges of implementing these methods and potential attacks on private deep learning systems. Ng and Chow [NC23] conducted an analysis of 53 papers on privacy-preserving neural networks from 2016 to 2022, focusing on approaches based on HE and MPC. All previous surveys concentrated on privacy-preserving supervised learning, where a training dataset with labeled samples (i.e., known input-output pairs) trains a model for future data record classification. Contrastingly, our research [**HMSY21c**] explores clustering, a widely used *unsupervised* ML technique. Clustering detects unknown patterns in unlabeled data, eliminating the need for a traditional "training" phase for a model.

Our Work In Sect. 2.1.1, we analyze the four most recent studies in privacy-preserving clustering that show no privacy leakage. Specifically, we look into the research by Mohassel et al. [MRT20] on privacy-preserving K-means, Meng et al. [MPOT21] on privacy-preserving hierarchical clustering, Bozdemir et al. [BCE+21] on privacy-preserving DBSCAN, and Cheon et al. [CKP19] on privacy-preserving Mean-shift. Our analysis involves comparing their benefits and drawbacks, along with in-depth experimental benchmarks to evaluate their performance and efficiency.

2.2.2 FL

Inference Attacks In FL, clients contribute to a collective learning process by sharing locally trained model updates with a central party. This process, while collaborative, introduces a risk of data leakage, as these shared local models could potentially reveal sensitive information. Even a semi-honest central server might infer confidential information from the private

training data by analyzing these local updates [NSH19; LWB+21; BDS+23]. To mitigate such inference attacks, secure aggregation methods have been developed, characterized by their handling of a large number of clients and drop-out tolerance. Secure aggregation protocols ensure that the central server only receives aggregated model updates, thus preventing the analysis of individual updates that could lead to inference attacks [BIK+17; CB17; BBG+20; **FMM+21**; CCKS22]. In this direction, our publication [**FMM+21**] (which is not part of this thesis) conducts a comparative analysis of several secure aggregation protocols, focusing on their efficiency and feasibility for real-world applications. More detailed comparisons among secure aggregation schemes for FL can be found in [MÖBC23].

Poisoning Attacks FL is vulnerable to manipulations by malicious clients [BBG19; BVH+20; FCJG20; SH21; RNMS22; ZPS+22]. These poisoning attacks can be classified into untargeted and targeted types. Untargeted attacks focus on reducing the global model's performance for a large number of test inputs, yielding a final global model with a high error rate [BBG19; FCJG20; SH21]. Targeted attacks, also known as backdoor attacks [BVH+20], activate attacker-defined triggers that cause a victim model to do targeted misclassifications, which can then be activated in the inference phase [BVH+20; RNMS22; ZPS+22]. Remarkably, the model's accuracy on inputs without the trigger remains unaffected.

Poisoning Defenses In FL, the simple approach of averaging parameters [MMR+17] from various clients' updates is vulnerable to outliers, which can significantly compromise the model's accuracy. To address this issue, Byzantine-robust defenses have been developed to improve the robustness of FL against potential attacks. One such defense is the Krum algorithm [BMGS17], which selects a single local update for the global model. This update is chosen because it has the closest similarity to the majority of other updates, based on the assumption that the majority of clients are honest. Specifically, it picks the update that is closest to the $n-m-2$ other updates, where n is the total number of clients, and m is the estimated number of malicious clients. Extending this concept, Multi-Krum [BMGS17] allows for the selection of multiple updates rather than just one. Another approach, known as Median [YCRB18], involves coordinate-wise aggregation. This method selects the median value for each parameter across all updates, which naturally mitigates the impact of outliers and thus enhances the robustness of the aggregation process. To further improve the detection of malicious updates, some strategies use an auxiliary dataset, or rootset, at the aggregator. This dataset is used to evaluate the performance of updates on a known baseline, assisting in identifying potentially harmful gradients [LXC+19; CFLG21; SH21]. FLTrust [CFLG21] uses this concept by comparing each received update to a baseline update calculated from the auxiliary dataset, using the ReLU-clipped cosine-similarity to assess similarity. Similarly, RSA [LXC+19] uses an ℓ_1-norm-based regularization for comparison, focusing on the deviation from the baseline update as a sign of reliability. FLDetector [ZCJG22] detects malicious clients by checking their model updates' consistency based on historical model updates. See [SHKR22b] for a comprehensive overview of different robust aggregations.

Our Work The poisoning defenses discussed so far are not compatible with secure aggregation protocols in a straightforward manner or lead to an intolerable overhead. In

Sect. 2.1.2 [**NRC+22**], we propose FLAME, that simultaneously considers both threats. Concretely, FLAME uses density-based clustering to remove updates with significantly different cosine distances (i.e., different directions) combined with clipping. Tab. 2.1 summarizes the key differences of FLAME from previous works. Briefly, in FLAME, FL clients use secret-sharing to securely outsource training local models to distributed servers based on MPC, thereby preserving model privacy. The clients can then leave, making FLAME robust against real-world issues such as drop-outs and requiring no interaction between clients.

Solution	Representative Work(s)	Technique	Model Privacy	Poisoning Resilience	Distributed Servers	No Client Interaction	Dropout Handling
Plain Aggregation	[MMR+17]	–	✗	✗	✗	✓	✓
Secure Aggregation	[BIK+17]	Masking	✓	✗	✗	✗	✓
	[CB17]	MPC	✓	✗	✓	✓	✓
	[**FMM+21**]	MPC	✓	✗	✓	✓	✓
Robust Aggregation	[CFLG21]	–	✗	✓	✗	✓	✓
	[LXC+19]	–	✗	✓	✗	✓	✓
	[SH21]	–	✗	✓	✗	✓	✓
FLAME (cf. Sect. 2.1.2)	[**NRC+22**]	MPC	✓	✓	✓	✓	✓

Table 2.1: High-level comparison of FLAME [**NRC+22**] and previous works. Since the body of literature is vast, comparison is made against only a representative subset for each category.

Follow-up Works Following the introduction of FLAME [**NRC+22**], a series of studies [DCL+21; HLX+21; MMM+22; DWL+23] have delved into MPC-based secure FL schemes. These approaches aimed to achieve a dual goal: preserving data privacy while enhancing the system's defense against Byzantine threats, thus securing data and preventing malicious entities from influencing model aggregation. Within our SAFEFL framework [**GMS+23**] (not part of this thesis), we conduct a thorough evaluation of these recent schemes within the FL domain, revealing that FLAME surpasses others in accuracy, as detailed in Table 3 of [**GMS+23**]. Furthermore, SAFEFL is designed to promote the development of future methodologies that are not only efficient but also robust against both privacy inference and poisoning attacks. Building upon SAFEFL, we introduce WW-FL [**MSS+23**] (not part of this thesis), a unified framework developed to support private and robust distributed ML at scale. WW-FL is based on a novel abstraction that also captures existing regular and hierarchical FL architectures in a *hybrid* manner.

The most recent of our contributions is ScionFL [**BMP+24**] (not part of this thesis), which represents a significant advancement by combining three essential aspects: (1) communication efficiency through quantization, (2) data privacy via secure aggregation, and (3) robustness through an advanced poisoning defense mechanism. This novel poisoning defense in ScionFL follows the approach initiated by FLAME in using density-based clustering, clipping, and noise addition, with significant variations, which we emphasize in the following:

1. *Magnitude Boundary.* FLAME's clipping is done with respect to the median Euclidean distance of the local updates to the previous global model. However, especially with non-iid

data and in early training phases, each training iteration may exhibit significant differences even for consecutive iterations. Hence, using the recent average norm (assuming the majority of updates is benign) as in [**BMP+24**] intuitively gives a better estimation for a benign magnitude in the current training state.

2. *Filtering.* FLAME compares cosine similarity in a pair-wise fashion among individual updates, i.e., it computes $\frac{n \cdot (n-1)}{2}$ cosine distances per iteration, where n represents the number of clients participating in the FL process. In contrast, [**BMP+24**] does n. While [**BMP+24**] sorts local updates based on cosine similarity, FLAME uses secure clustering with low cubic complexity [BCE+21]. FLAME only accepts updates assigned to the largest cluster, which can lead to an exclusion of benign updates and thus significantly slowing down the training by removing valuable benign contributions. In contrast, [**BMP+24**] removes only a fixed number of updates, thereby enabling an efficient trade-off that reduces the attack's effect to a tolerable level (even if a few malicious contributions are not filtered out) with a low false positive rate.

3. *Differential privacy (DP):* After the clipping, FLAME aggregates the updates and adds noise in the cleartext to create a differentially private new global model. We do not consider DP in [**BMP+24**].

3 MPC Protocols & Optimizations

This chapter explores the optimization of secure multi-party computation (MPC) protocols in terms of both computation and communication. The literature on MPC distinguishes between two main types of corruption scenarios: i) *semi-honest*, where the corrupt parties are expected to follow the protocol steps, and ii) *malicious*, where corrupt parties may deviate from the protocol in various ways. MPC protocols are broadly classified into i) *low-latency* (e.g., [MRZ15; GRW18; RR21]) and ii) *high-throughput* (e.g. [ABF+17; CCPS19; RS20]) protocols. Low-latency protocols employ garbled circuit (GC) [Yao86; BMR90; KS08b; ZRE15; RR21], achieving constant-round solutions through symmetric cryptographic operations in the online phase. Conversely, the latter uses secret sharing (SS)-based solutions, which do not require cryptographic operations during the online phase.

Focusing on low-latency protocols, the introduction of the half-gates garbling scheme by Zahur, Rosulek, and Evans in [ZRE15], along with their proposal of a matching lower bound of 2κ bits for the cost of garbling an AND gate within their linear garbling model, marked significant progress in GC-based protocols. Subsequently, Rosulek and Roy [RR21] moved beyond this lower bound by introducing a technique known as slicing and dicing. Their approach led to a more efficient garbling scheme. Specifically, they demonstrated that in their scheme, XOR gates are free [KS08b], while AND gates require only $1.5\kappa + 5$ bits. However, these schemes impose considerable computational overheads: for the half-gates method [ZRE15], each AND gate requires four symmetric cryptographic operations for the garbler and two for the evaluator. In contrast, the "slicing and dicing" technique [RR21] requires six cryptographic operations for garbling and three for evaluation per AND gate.

Despite their efficiency in terms of communication, high-throughput protocols require a number of communication rounds that is linear with the multiplicative depth of the circuit. In seeking practical solutions, several works [DPSZ12; KSS13] have explored a model involving an input-independent setup phase. During this phase, parties generate a lot of correlated randomness (e.g., Beaver multiplication triples [Bea91]), which is then used in the online phase to make the computation of parties' inputs more efficient. Despite considerable efforts to optimize the setup phase [KOS16; KPR18; RST+22], improving the online phase has remained largely unexplored since 2013 [SZ13].

This chapter focuses on improving both GC-based and SS-based MPC protocols. For GC-based protocols, we use recent advancements in hardware technologies to reduce computation complexity. On the other hand, for SS-based protocols, we develop a new set of protocols to lower the number of rounds needed and to achieve more efficient high-throughput solutions.

3.1 Our Contributions

3.1.1 Reducing Computation Overhead of GC-based Protocols

In Yao's protocol [Yao86], the garbler (one of the two parties) generates a GC that corresponds to a Boolean circuit by encrypting the truth table of each gate. The evaluator (the other party) then decrypts (evaluates) the circuit received from the garbler. The cryptographic protocol known as oblivious transfer (OT) [IKNP03] is used for transferring the necessary keys corresponding to the evaluator's inputs. The most efficient solution for GCs combines free-XOR [KS08b] and half-gates [ZRE15] (at the time of this contribution [**MSY21**])/three-halve-gates [RR21] (first implemented in 2022 [HKST22]). This optimization means that transferring 2/1.5 ciphertexts is required for each AND gate, while XOR gates, which require only XOR operations, do not require any communication, essentially making them free. Additionally, fixed-key advanced encryption standard (AES) operations are required to evaluate AND gates [BHKR13; GKWY20].

The addition of the AES instruction set (AES-NI) extension into the x86 instruction set architecture has significantly improved the performance and efficiency of AES operations. This development allows for AES-128 to be computed at about 1.3 cycles per byte on a single processor core. Intel has further pushed the boundaries in this field by introducing vector AES (VAES) instructions [DGK19], which are part of the Ice Lake microarchitecture. These VAES instructions enable the parallel processing of several AES rounds across various data blocks, each using unique round keys.

In this section of the thesis, we explore how using VAES can make GC-based MPC more efficient, as detailed in the following publication available in Appendix D.

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, pp. 131–145. CORE Rank **A**. Appendix D.

We focus on using VAES to improve the computation overhead within GC-based protocols and their respective applications. Initially, VAES was mainly used to solve specific design issues in computer architecture, particularly for Pseudo-Random Functions (PRFs) and Pseudo-Random Generators (PRGs) [DG19], where the sequence of AES operations is fixed ahead of time. We expand its use to the design and implementation of protocols, like those in GC, where the sequence of AES operations can change and is not known in advance. To get the most benefit from the hardware, it is important to group together enough AES tasks in each batch of operations.

Batch Identification Our approach groups enough tasks together in batches for the computer processor to minimize the delay caused by individual tasks and maximize the processor's throughput. We develop methods to create these batches, even when dealing with complex task sequences that don't follow a straightforward path, using a technique called *dynamic*

batching. This technique involves deferring some tasks and tracking additional details about them. The core idea behind dynamic batching is to delay the execution of operations until they are needed and to process all pending operations in a batch when one is referenced as an input dependency. We refine this approach by introducing detailed tracking methods for more complicated scenarios, such as managing two distinct types of AES-related tasks and employing techniques like early evaluation (cf. Section 4.1 in Appendix D).

Batch Computation We then adjust our strategy to identify and organize independent tasks early on by layering them and directly leveraging single instruction multiple data (SIMD) capabilities for better performance. Once we organize these tasks into batches, our next step is to compute the AES operations. This involves tackling the work on two levels: at a lower level, we give the compiler as many opportunities for optimization as possible while maintaining high-level code simplicity. At a higher level, we adopt a memory-oriented approach to simplify our code, reducing dependence on specific hardware. This technique addresses code duplication concerns and uses platform-independent instructions. To achieve this, we implement our own versions of AES operations instead of relying on existing libraries. This choice is driven by the need to reduce the extra time it takes to call these external libraries and the flexibility to fine-tune our code, for instance, to generate certain inputs right where they are needed (cf. Section 4.2 in Appendix D).

Application We implement our techniques in various frameworks, focusing on different areas of secure computation. This includes the ABY framework [DSZ15] designed for semi-honest two-party computation (2PC), the EMP-AGMPC [WRK17c] framework for malicious MPC, and Microsoft's CrypTFlow2 [RRK+20a] framework for privacy preserving machine learning (PPML).

Among these, both ABY and EMP-AGMPC benefit significantly from our approach of grouping tasks into dynamic batches. Specifically, ABY is improved with more efficient garbling techniques [GLNP15; ZRE15; GKW+20] and takes advantage of SIMD capabilities to better organize these batches. On the other hand, EMP-AGMPC focuses on using memory in a more abstract way to streamline its processes.

Efficiency After integrating our improvements into these frameworks, we evaluate the performance enhancements. Initially, by organizing tasks into batches without needing extra hardware, the ABY framework's runtime increases by up to 230%. Then, by applying the VAES technology, we see additional improvements in performance across nearly all the frameworks. For ABY, this means an extra $25-100\%$ increase in efficiency; for EMP-AGMPC, up to a 30% increase; and for Cryptflow2, an overall $6-20\%$ improvement in performance. We make our modified versions of these frameworks available for everyone to use and to serve as a reference point for future developments at `https://encrypto.de/code/VASA`.

Impact We focus on improving three major frameworks: the ABY framework [DSZ15] used for semi-honest 2PC; the EMP-AGMPC framework [WRK17c] designed for malicious MPC; and Microsoft's PPML framework CrypTFlow2 [RRK+20a]. By integrating VAES technology into their implementations, without modifying the foundational protocols themselves— we improve their efficiency and usability.

After publishing our results online, we received insightful feedback and historical context from Shay Gueron, the original designer of VAES at Intel. His input added valuable depth to our work, highlighting the practical significance of our application of VAES in MPC settings. This exchange emphasizes the impact of our research in bringing real-world improvements to MPC frameworks.

3.1.2 Improving Efficiency of SS-based Protocols

In 2PC secure against semi-honest adversaries, the potential for improving the online phase of SS-based protocols had not been explored in depth for over ten years [SZ13]. This part of the thesis makes a significant contribution in leveraging this potential, highlighted by two key publications listed in Appendix E and Appendix F:

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, pp. 2165–2182. CORE Rank **A***. Appendix E.

[BHS+23] A. BRÜGGEMANN, R. HUNDT, T. SCHNEIDER, A. SURESH, H. YALAME. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, pp. 515–533. CORE Rank **A***. Appendix F.

First, we introduce ABY2.0 [**PSSY21a**], a high-performance framework designed for 2PC. We primarily concentrate on improving the efficiency of the online phase in ABY2.0 over state-of-the-art work [DSZ15]. Additionally, we expand our methods to support multi-input multiplication gates without increasing *the online communication* compared to state-of-the-art solution [DSZ15], meaning that the communication needed does not depend on the number of inputs. We develop several essential building blocks to improve online phase efficiency, including scalar product, comparison, equality check, and the AES S-box. The key contributions of our work can be summed up as follows:

2PC and Mixed Protocol Conversions We propose an optimized 2PC protocol that requires only the transmission of two ring elements for each multiplication operation in the online phase. This is 2× better than the state-of-the-art [DSZ15], which required four ring elements for the same operation. Additionally, our protocol handles multiplication gates with multiple inputs (N-input) with the same minimal online cost of two elements and just a single round

of interaction. This efficiency is particularly notable when compared to the state-of-the-art approach [ON20], which required a communication of two ring elements per input, amounting to a total of $2N$ elements for N-input gates. Furthermore, we introduce a series of conversions that improve upon existing techniques [DSZ15]. Our solution decreases the number of rounds needed for most conversions from two to just one.

Building Blocks We develop depth-optimized circuits and constructions designed to improve efficiency for several fundamental building blocks: i) scalar product, ii) comparison, iii) equality testing, and iv) AES S-box. Our main achievements include:

- Depth-Optimized Circuits: parallel prefix adders (PPAs) provide a depth-optimized way to add two binary numbers of ℓ-bits. The most effective PPAs achieve a depth of $\log_2(\ell)$ [BL01]. We develop a PPA that uses combinations of two, three, and four input AND gates to further optimize depth. Specifically, for a 64-bit ring, our design achieves a depth that is 2× more efficient compared to existing designs [MR18].
- Scalar product: Our protocol features online communication that is *independent* of the vector dimension n, representing a notable advancement in 2PC efficiency. Specifically, our method only needs to exchange two ring elements as opposed to $4n$ elements in [DSZ15].
- Comparison: Our new protocol improves the online communication of [ON20] by a factor of 6× and reduces the number of online rounds from 4 to 3 for 64-bit length.
- Equality testing: Our new protocol for checking the equality of two l-bit values, improves the online rounds of [PSTY19] from $\log_2(l)$ to $\log_4(l)$. By incorporating our improved equality check into the state-of-the-art circuit-based private set intersection protocol [PSTY19], we achieve approximately a 2.35× decrease in online communication.
- AES S-box: By applying our protocol for 3-input AND gates to the construction of AES S-box, we reduce the online rounds from 4 [BP12], down to 3. This increases the efficiency of AES rounds by a factor of 1.33×. On a high level, we start with the three-layer construction of [BP12; BMP13], and focus on improving the middle layer by replacing some of the 2-input AND gates with 3-input AND gates.

In our subsequent work, we introduce FLUTE [**BHS+23**], a protocol designed for efficiently evaluating look-up tables (LUTs). This protocol adopts the same secret sharing as our ABY2.0 protocol [**PSSY21a**]. We successfully improve the online phase in terms of both communication and rounds, without increasing the total communication compared to the current state-of-the-art SS-based LUT protocol of [DKS+17]. The following is our main highlight:

A Novel LUT Evaluation We examine the evaluation of LUTs, which previous studies [IKM+13; DKS+17] have approached from two angles. Some focused on protocols like OTTT [IKM+13] and its refinement OP-LUT [DKS+17], which reduced the online communication but require a large amount of total communication. Others, like SP-LUT [DKS+17], minimized total communication but required more online communication. We offer the best of both worlds with our protocol, achieving low online communication similar to OP-LUT while maintaining total communication at levels comparable to SP-LUT. Instead of viewing LUTs merely as tables mapping a number of inputs to outputs, we propose seeing them as a generalized notion of inner products over a Boolean domain (see Section 3.2.1 of Appendix F [**BHS+23**]). With FLUTE, the setup communication is *not affected* by the number of LUT outputs, and the online communication is *independent* of the number of LUT inputs.

LUT Circuit Generation We generate the circuits for FLUTE [**BHS+23**] using a hardware synthesis toolchain similar to the one in [DKS+17]. To do so, we use Yosys [WGK13] and the ABC tool [Ber]. Yosys [WGK13] serves as the open-source framework for front-end processing of our Verilog hardware description language (HDL), mapping it into a network of low-level logic operations in an intermediate format. The ABC tool [Ber] then structures this network into Directed Acyclic Graphs (DAGs) and maps it into LUTs in a depth-optimized fashion. For generating LUTs of more complex functionalities, such as floating point operations, we use the hardware intellectual property (IP) libraries in the Synopsys Design Compiler (DC) [10].

Efficiency To validate the effectiveness of ABY2.0 [**PSSY21a**] and FLUTE [**BHS+23**], we conduct implementations and evaluations against their most direct competitors. In the case of ABY2.0, we apply it to both the training and inference phases of logistic regression and neural networks (NNs) within networks of bandwidths 25Gbps (local area network (LAN)) and 75Mbps (wide area network (WAN)). For logistic regression training, compared to SecureML [MZ17], our online runtime improvements ranged from 4.4× to 6.1× for LAN and from 1.5× to 1.95× for WAN. Inference for logistic regression sees online runtime improvements over SecureML by factors of 5.5× for LAN and 1.6× for WAN. NN training over SecureML [MZ17] shows improvements between 2.7× to 3.46× for LAN and 2.4× to 2.8× for WAN. For NN inference, we achieve an improvement in online runtime over SecureML by 3.26× for LAN and 2.37× for WAN.

With FLUTE, we showcase its capability and practicality through the evaluation of various circuits, including those for floating point operations. Compared to the state-of-the-art SS-based LUT evaluation [DKS+17] and the SP-LUT method, FLUTE improves the online communication by over 100× for floating point operations, while only increasing the overall communication by less than 4% on average. Moreover, FLUTE achieves a 3× improvement in online communication compared to the best previous LUT evaluation methods that focused on optimizing the online phase, namely OTTT [IKM+13] and OP-LUT [DKS+17].

Impact After publishing our ABY2.0 [**PSSY21a**], it received more than 230 citations from the research community to date and encourages more work in the field of privacy-preserving technologies. Notably, ABY2.0 was adopted by the India Urban Data Exchange (IUDX [BTM23]) for practical and scalable MPC solutions. Furthermore, SP-LUT from [DKS+17] has been applied in several key studies, such as [RRK+20a] for secure inference comparisons, [RBS+22] for mathematical functions on floating-point numbers, and [RRG+21] for functions on fixed-point numbers. The researchers in [RRG+21] specifically pointed out the crucial role of LUTs in improving their protocol's efficiency. Notably, FLUTE [**BHS+23**] can serve as an effective alternative to SP-LUT in these applications.

In addition, we highlighted ABY2.0 in various venues, including a contributed talk at the Privacy in Machine Learning Workshop (PriML) at NeurIPS'21 [**PSSY21c**], the Privacy Preserving Machine Learning (PPML) in Practice Workshop at ACM CCS'21 [**PSSY21d**], and the PPML workshop at the IACR CRYPTO'21 Conference [**PSSY21b**].

We make FLUTE available as an open-source project, coded in Rust because of its excellent performance, memory safety, and ease of safe parallel processing. This release includes the Boolean 2PC from ABY2.0 and the semi-honest silent OT extension from [BCG+19a], now in Rust for the first time. The code, under the MIT License, is accessible at `https://encrypto.de/code/FLUTE`. Rust's robust safety measures are particularly important for developing secure protocols that are reliable in practical applications.

3.1.3 Improving PFE Protocols

MPC protocols enable the effective implementation of private function evaluation (PFE), which involves evaluating a private function on private data through the use of a publicly known universal circuit (UC). A UC is a Boolean circuit with a size of $\Theta(n \log n)$ [Val76], designed to simulate any Boolean function up to a given size n. The concept of UCs was first introduced by Valiant [Val76], who proposed the initial two UC designs with asymptotic sizes of $\sim 5n \log n$ and $\sim 4.75n \log n$. The most efficient design to date, developed by Liu et al. [LYZ+21], achieves a more efficient size of $\sim 3n \log n$.

Previously, all the developed UCs were designed to simulate Boolean gates that have two inputs and one output. In this part of our thesis, we expand the capabilities of UCs to also include the simulation of circuits made up of LUTs that have δ inputs and σ outputs. Details of this approach are presented in our following publication that can be found in Appendix G.

[DGS+23] Y. Disser, D. Günther, T. Schneider, M. Stillger, A. Wigandt, H. Yalame. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 14438. LNCS. Online: `https://ia.cr/2021/809`. Code: `https://encrypto.de/code/LUC`. Springer, 2023, pp. 3–37. CORE Rank **A**. Appendix G.

UC for evaluating LUTs with δ inputs and σ outputs We develop two approaches for PFE, termed LUT-based UC (*LUC*) and Varying UC (*VUC*). The first construction is called LUT-based UC (*LUC*) and has asymptotic size $1.5\delta\sigma n \log \sigma n$. While this construction can be used to fully hide a function, it unfortunately has the LUT's input dimension δ as prefactor for UC sizes. Many functions can be most efficiently represented as multi-input LUTs of various input dimensions, e.g., AES SBoxes can naturally be implemented with 8-input LUTs. However, mathematical operations like additions and multiplications consist of full adders and thus benefit from using 3-input LUTs. Combining a circuit that uses AES and arithmetic operations and evaluating it with the LUC construction brings a massive overhead as all 3-input LUTs need to be extended into an 8-input LUT, i.e., the potential of many 8-input LUTs is not fully used, but its expensive prefactor is omnipresent. On the other hand, if all operations are shown as 3-input LUTs, we need many more LUTs for AES operations compared to using 8-input LUTs.

For this reason, we propose a second construction called Varying UC (*VUC*) that has an asymptotic size of $\sim 1.5(\sigma n + \Delta)\log(\sigma n + \Delta)$, where Δ is defined as $\sum_{i=1}^{n} \frac{L_i^{in}-2}{2}$, with L_i^{in} representing the number of inputs for the i-th LUT in the circuit. Here, the actual size of the VUC constructions depends on the used LUTs's dimensions. However, this construction leaks the number of inputs of each individual LUT and thus leaks more information than the LUC construction. Deciding whether to use LUC or VUC involves balancing the need for performance with the requirement for function privacy. Examples of *VUC*'s application are detailed in Section 5.2 in Appendix G [**DGS+23**].

Synthesis and UC Compiler Previous studies on UC-based PFE [Val76; KS08a; KS16; GKS17; ZYZL19; AGKS20a; LYZ+21] typically approached the task of minimizing the input circuit (to create a small number of 2-input gates) and the task of constructing a smaller UC as two distinct steps. Our research demonstrates that by utilizing LUTs with multiple inputs and outputs, we can merge these steps into one streamlined process, leading to a reduction in overall circuit size.

Fig. 3.1 illustrates our compiler pipeline, as detailed in Subsection 6.2 of Appendix G [**DGS+23**]. In our circuit synthesis flow, similar to LUT generation for FLUTE [**BHS+23**] (cf. paragraph 3.1.2), we use the hardware synthesis tools Yosys [WGK13] and ABC [Ber].

We first convert high-level design elements into a more manageable intermediate form using Yosys [WGK13]. This intermediate representation is then formatted into the Berkeley Logic Interchange Format (BLIF) [Mis92], ensuring compatibility with both Yosys and ABC tools. Once we import the circuit into ABC, we apply a series of optimizations and transform the circuit into a form that can be represented by LUTs with δ inputs, using the FPGA technology mapping process `if` [CD16]. The circuit data is then saved in the Bench file format [Ber]. Subsequently, our UC compiler processes this data to produce the final UC file along with the programming bits required for it.

The last stage involves adapting the circuit for use with the ABY framework [DSZ15], thereby enabling the execution of the circuit within this framework. We choose ABY instead of our

ABY2.0 and FLUTE because all previous PFE implementations [KS16; GKS17; AGKS20a] are based on the ABY framework.

We optimize the process of LUT-based circuit generation by integrating LUTs that share some inputs and have several outputs. However, standard synthesis tools are not set up to directly create LUTs with multiple outputs. To overcome this limitation, we develop a post-processing step. This step takes the circuits with single-output LUTs, which are initially generated by the synthesis tool, and transforms them into circuits with multi-output LUTs.

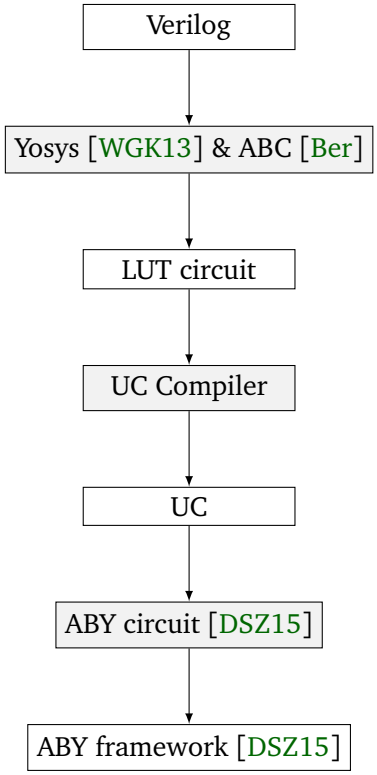


Figure 3.1: High-level overview of the compiler pipeline in our work [**DGS+23**].

Efficiency We evaluate three fundamental components often used in PFE applications: the full adder (FA), comparator (CMP), and multiplexer (MUX). We compare the circuit sizes using our newly developed LUC construction against conventional UC designs. Our results indicate that the LUC construction significantly reduces circuit size, leading to UC size reductions of 1.67× for FA, 2.67× for CMP, and 2× for MUX.

Moreover, we evaluate the performance of both our LUC and VUC constructions with different LUT sizes and compare these to the latest UC construction by Liu et al. [LYZ+21]. Our LUC construction achieves up to 2.18× smaller circuit sizes compared to Liu et al.'s approach. We also examine the runtime and communication requirements of our LUC construction using a network with 10 Gbit/s bandwidth and a 1 ms round-trip time. Our methods result in a faster

total runtime for sample circuits by $1.14-2\times$ and reduce the communication overhead by $1.12-2.25\times$. Notably, our VUC construction, which differs from LUC by revealing the number of inputs (*fan-in*) for each LUT, demonstrates up to $2.90\times$ smaller circuit sizes than Liu et al.'s UC, with total runtime improvements ranging from $1.1-2.85\times$ and communication enhancements between $1.06-2.96\times$. For a comprehensive efficiency analysis, refer to Tables 5 and 6 in Appendix G.

Impact We provide the first implementation of today's most efficient UC construction [LYZ+21]. The source code is available under the MIT license at https://encrypto.de/code/LUC. Additionally, we introduce two new constructions, *LUC* and *VUC*, along with their implementations.

In today's digital landscape, maintaining the confidentiality of processes handling private data is essential for both economic and security reasons. A prominent example is hardware IP protection. Circuit designs are critical assets for companies, containing significant intellectual property. Manufacturers producing these designs can potentially access and learn proprietary details, leading to IP theft and unauthorized cloning, resulting in substantial financial losses and a diminished competitive edge. Hardware logic locking [BGH+22] has emerged as a potential solution for hardware IP protection. This technique embeds a locking mechanism within the hardware to prevent unauthorized use. However, current methods of logic locking are not fully secure, as attackers can bypass or reverse-engineer these protections, compromising the security of the designs [SPJ19]. PFE offers a technical solution that protects both data and functions, suitable for applications like hardware IP protection. Our *LUC* and *VUC* constructions provide a more secure way of protecting IP.

3.1.4 Improving MPC Efficiency Leveraging Cheater's Identity

In MPC, the usual assumption is that all parties might be equally susceptible to either semi-honest or malicious corruption. Yet, the idea of *asymmetric trust*, where trust levels can vary among the participants, likely provides a more realistic perspective for many practical applications. Within the realm of PPML, recent works like MUSE [LMSP21] and SIMC [CGOS22] have drawn attention to a specific scenario called *client-malicious*. This scenario involves a malicious client interacting with a semi-honest server to perform machine learning (ML) inference tasks.

In this part of the thesis, we present the first protocols that extend the client-malicious setting to scenarios with an honest majority in three-party computation (3PC). This adaptation is crucial because it opens the door to developing more efficient protocols. We develop two versions of the protocol, each tailored to handle specific types of potential corruption, with more details provided in the following publication listed in Appendix H.

[BSS+24] A. BRÜGGEMANN, O. SCHICK, T. SCHNEIDER, A. SURESH, H. YALAME. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

Asymmetric Trust Scenario In situations where two parties are involved and one party is guaranteed not to act maliciously, we face an *asymmetric trust model*. This condition can reduce the costs associated with 2PC compared to scenarios where either party might be corrupt. Such an *asymmetric trust model* was considered in the MUSE [LMSP21], within the realm of PPML. The efficiency gains observed in MUSE sparked interest in this specific model, leading to the development of the faster SIMC [CGOS22]. In our research, we revisit techniques used in 3PC under the honest majority assumption, aiming to explore potential improvements when the identity of the potentially corrupt party is known in advance. We introduce two protocols tailored to two distinct corruption scenarios, with further details provided in the subsequent paragraphs. Fig. 3.2 depicts the two settings considered in this work and the settings in MUSE [LMSP21] and ABY2.0 [**PSSY21a**].

3PC with a Malicious Evaluator (cf. §5 in Appendix H) We introduce SOCIUM and build upon the client-malicious MUSE [LMSP21] by adapting it to a three-party context. This adaptation is based on the malicious 3PC protocol SWIFT [KPPS21], with a unique addition to MUSE [LMSP21]: we involve a third party to serve as a helper. Unlike the protocols in MUSE [LMSP21] and SIMC [CGOS22] that operate over fields, our protocols are designed to work over rings. This decision is made to better align with the functionalities of modern CPU architectures, thus promising improved performance. Our primary goal is to explore the potential benefits of this setup:

> *Is it possible for an additional semi-honest helper to reduce the costs of a two-party scenario, specifically when dealing with a malicious client and a semi-honest server?*

Our investigations confirm that including a semi-honest helper significantly improves efficiency. Specifically, SOCIUM, with this additional semi-honest helper, achieves communication improvements for ML inference ranging from $14-60\times$ compared to SIMC [CGOS22], all without using resource-intensive homomorphic encryption (HE).

3PC with a Malicious Helper (cf. §4 in Appendix H) In this part of our work, we introduce AUXILIATOR to investigate scenarios where an additional helper is assumed to behave maliciously. This approach improves our ABY2.0 [**PSSY21a**] work from Sect. 3.1.2 by adding a malicious third party. Drawing upon the semi-honest 3PC protocol ASTRA [CCPS19], we maintain equivalent efficiency in the online phase while exploring the consequences of this setup. Our central question is:

> *Can including an additional helper, assumed to be malicious, reduce the costs in a semi-honest two-party protocol?*

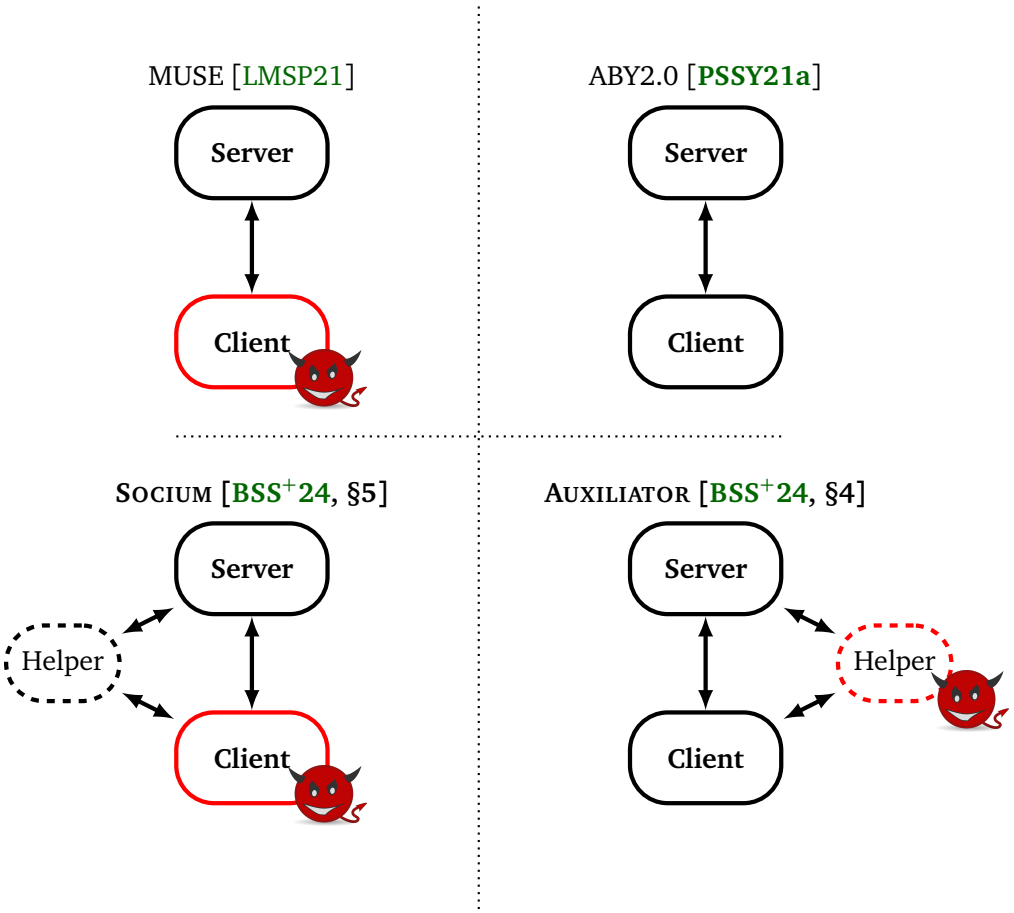


Figure 3.2: Comparison of the semi-honest 2PC setting in ABY2.0 [**PSSY21a**] and the client-malicious 2PC setting in MUSE [LMSP21] against our 3PC settings with either a malicious client or a malicious helper. Black circles represent semi-honest parties, while red circles represent malicious parties.

Our findings confirm that AUXILIATOR not only addresses this question affirmatively but also significantly enhances the efficiency of our semi-honest 2PC setups by integrating a malicious helper. Specifically, by extending ABY2.0 with a malicious third party, AUXILIATOR demonstrates a remarkable improvement in setup communication, achieving $2-3$ orders of magnitude better efficiency for ML inference.

Efficiency We compare the performance of SOCIUM for ML inference against SIMC [CGOS22] within a *client-malicious* context as detailed in §6.3 of Appendix H. Our evaluations are carried out in both the LAN setting with 10Gbit/s bandwidth and 1ms round trip time (RTT), and the WAN setting with 100Mbit/s bandwidth and 100ms RTT. SOCIUM outperforms SIMC, being 1.07-2.31× faster in LAN and 1.56-1.62× faster in WAN scenarios. Additionally, we evaluate AUXILIATOR's performance by comparing it with the semi-honest 2PC protocol

ABY2.0 [**PSSY21a**] as detailed in §6.2 of Appendix H. This comparison underlines the benefits of adding a non-colluding helper, even if untrusted, as AUXILIATOR significantly outperforms ABY2.0 in terms of communication overhead, achieving at least two orders of magnitude improvement. SOCIUM has up to 8.57× better performance than the malicious secure SWIFT protocol. AUXILIATOR has similar communication as the semi-honest ASTRA protocol.

Impact We provide *the first* open-source implementations of ASTRA [CCPS19], SWIFT [KPPS21], and our new protocols AUXILIATOR and SOCIUM. Our implementation is based on the MOTION [BDST22] framework and is accessible at `https://encrypto.de/code/MOTION-FD`. This is notably the first time function-dependent preprocessing has been integrated into an MPC framework. Our implementation is particularly tailored for real-world PPML scenarios. It supports varying trust levels among participants and allows for flexible computation-communication trade-offs. This adaptation enhances efficiency, leading to lower monetary costs and improved throughput. Furthermore, our approaches are applicable to a wide range of real-world applications, including private data analytics and consensus mechanisms in blockchain technology.

3.2 Related Work

In this section, we conduct a thorough review of key developments and enhancements in five domains within this chapter: GC improvements (cf. Sect. 3.2.1), SS-based protocol improvements (cf. Sect. 3.2.2), UC-based (PFE) protocols (cf. Sect. 3.2.3), circuit generation (cf. Sect. 3.2.4), and protocols for symmetrical and asymmetrical trust settings (cf. Sect. 3.2.5). This section positions our contributions within the existing body of work and highlights how our research addresses existing challenges. We refer the reader to [Lin20; CP22] for a comprehensive analysis of related work on MPC.

3.2.1 GC Improvements

Protocol-Level Previous research has mainly focused on improving GC protocols by finding ways to reduce communication requirements and minimize the use of computationally heavy operations [NPS99; KS08b; ZRE15; WRK17a; RR21; ACHY23; Hea24]. There has also been a trend towards automating the conversion of high-level programming languages into GC circuits, often achieved by adapting advanced hardware logic synthesis tools to make circuit creation for GC easier [DDK+15; SHS+15; RJHK19]. In line with these efforts, we develop LLVM-Circ [**HST+21**] (not part of this thesis), a new compilation tool designed to automate circuit generation for GC. By integrating with the LLVM compiler infrastructure, LLVM-Circ efficiently translates high-level code into optimized GCs.

Implementation-Level At the implementation level, significant work has been directed towards improving the performance of specific operations within GCs, particularly focusing on refining the processes of garbling and evaluating individual gates [BHKR13; GLNP15]. Efforts to parallelize the evaluation of GCs have adopted both coarse- and fine-grained approaches [HEKM11; HMSG13; BBL+14; BK15]. Coarse-grained strategies [BBL+14; BK15] typically use multiple threads to process distinct parts of the same GC, sometimes favoring parallel processing capabilities over the efficiency of communication. This includes, for instance, bypassing the free-XOR optimization [KS08b] to utilize specialized hardware such as GPUs or Intel Quick Assist Technology [HMSG13]. Fine-grained approaches [BBL+14; BK15], similar to our approach (cf. Sect. 3.1.1 [**MSY21**]), have predominantly focused on using a layering technique to allocate tasks among different threads, aiming to fully utilize the parallel processing power available in today's CPUs. There has also been exploration into separating the garbling from the evaluation roles within GC computation, either by dividing the circuits [BK15] or by synchronizing these tasks with garbling and evaluation operations [HEKM11], which are methodologies distinct from ours. Furthermore, studies have explored leveraging field-programmable gate array (FPGA) technology to accelerate GC procedures [JKSS10a; JKSS10b; HRGK18; HK19], aiming to exploit the capabilities of these devices to accelerate GC operations.

Our Work In our research (cf. Sect. 3.1.1 [**MSY21**]), we distinguish our work by focusing not on protocol-level enhancements, but rather on advancing the actual implementations and frameworks that use these protocols [DSZ15; WRK17c; RRK+20a].

3.2.2 SS-based Protocols Improvements

Two-Input Multiplication The works in [DPSZ12; KSS13] introduced efficient SS-based methods for dishonest majority MPC over fields. This approach was later adapted for operations over rings in [CDE+18b]. The core of SS-based methods is the creation of Beaver multiplication triples [Kil88; Bea91] during the initial setup phase, which are then used to efficiently process two-input multiplication gates in the subsequent online phase. These works initiated a rich line of research on increasingly efficient MPC protocols in the preprocessing model, e.g., [DSZ15; DKS+17; PS20; RRG+21; RBS+22; KPPS23]. For 2PC, the state-of-the-art [DSZ15] required two public reconstructions between the participants for each multiplication gate during the online phase. In ABY2.0 [**PSSY21a**] (Sect. 3.1.2), we reduce this to just one public reconstruction per gate. [BNO19] also achieved this reduction to a single reconstruction in the online phase through function-dependent preprocessing. However, this adjustment introduced the need for extra communication—specifically, the exchange of four ring elements—during the preprocessing phase.

Multi-Input Multiplication. The study in [ON20] increased the number of inputs for multiplication, moving from just two inputs to any number of inputs, by extending the use of Beaver triples. This extension reduced the number of rounds needed during the online phase. However, [ON20] resulted in online communication that scales with the number of inputs

(fan-in) of the multiplication gates. In contrast, in ABY2.0 [**PSSY21a**], we keep the online communication at just 2 ring elements, regardless of the multiplication gate's fan-in.

Mixed-Protocol Conversions. Mixed 2PC protocols leverage the strengths of both GC-based and SS-based approaches, making them highly effective for various privacy-preserving applications. The first framework combining these approaches was TASTY [HKS+10], which integrated GC with HE. Following this, ABY [DSZ15] introduced a more efficient framework tailored for the semi-honest setting, which utilized the best features of arithmetic sharing, Boolean sharing, and GC. Subsequently, the ABY framework was expanded to include scenarios with three and four parties, assuming an honest majority, as demonstrated by [MR18; RS20]. In the multi-party setting, MOTION [BDST22] and Manticore [CDG+23] introduced semi-honest protocols that support arithmetic sharing, Boolean sharing, BMR [BLO16], and conversions between any two representations.

LUTs In the GC-based setting, Fairplay [MNPS04] implemented Yao's GC protocols to evaluate gates with up to 3-input gates. The TASTY framework [HKS+10] implemented multi-input garbled gates including garbled-row reduction [PSSW09]. [PAP22] proposed an MPC protocol that works on circuits with multi-input/multi-output gates instead of working on circuits with 2-input gates. Recently, [HKN24] proposed garbled circuits with multi-input/multi-output gates.

The idea of using LUTs in SS-based protocols was first introduced by [IKM+13]. Their method, known as OTTT, represented an entire circuit with a single LUT, which resulted in suboptimal performance for complex circuits. This LUTs approach was further explored by [DZ16] for evaluating AES S-boxes with malicious security. The combination of OTTT with a preprocessing step was proposed by [DZ16] and later adapted by [DKS+17] for semi-honest security. [DKS+17] also introduced two versions of LUT protocols as alternatives to OTTT: OP-LUT, which focused on optimizing online communication, and SP-LUT, which reduced the total communication. TinyTable [DNNR17] presented a method for using LUTs with security against malicious parties. However, [DKS+17] found that TinyTable had similar performance issues as OTTT due to its setup process. Extending this idea, [KOR+17] adapted TinyTable for multi-party settings using secret-sharing techniques.

Our Work For a multiplication gate with N inputs, our ABY2.0 approach [**PSSY21a**] results in a *constant* overhead of only 2 ring elements and requires a single round of interaction. This represents a significant improvement compared to the method in [ON20], which demands the exchange of $2N$ ring elements. When considering the complexity in terms of rounds, the straightforward strategy of multiplying N elements two at a time leads to $\log_2(N)$ rounds needed for the online phase. This method, in terms of overall communication, requires $4(N-1)$ ring elements for [DSZ15] and $2(N-1)$ ring elements for [BNO19]. Our improvements over these previous works are detailed in Tab. 3.1.

To evaluate a LUT with δ inputs and σ outputs, FLUTE [**BHS+23**] (Sect. 3.1.2) requires a setup communication of $(|\mathsf{MT}| + 4) \cdot (2^\delta - \delta - 1)$ bits and an online communication of

Table 3.1: Comparison of ABY2.0 [**PSSY21a**] and existing works for 2PC protocols for computing 2, 3, and 4-input multiplication gates. Best values for the online phase are marked in bold.

Protocol	Ref.	Setup	Online	
		Comm [bits]	Comm [bits]	Rounds
MULT	[DSZ15]	$2\ell(\kappa+\ell)$	4ℓ	**1**
	[BNO19]	$2\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
	[ON20]	$2\ell(\kappa+\ell)$	4ℓ	**1**
	ABY2.0 [PSSY21a]	$2\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
MULT3	[DSZ15]	$4\ell(\kappa+\ell)$	8ℓ	2
	[BNO19]	$4\ell(\kappa+\ell)$	4ℓ	2
	[ON20]	$8\ell(\kappa+\ell)$	6ℓ	**1**
	ABY2.0 [PSSY21a]	$8\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
MULT4	[DSZ15]	$6\ell(\kappa+\ell)$	12ℓ	2
	[BNO19]	$6\ell(\kappa+\ell)$	6ℓ	2
	[ON20]	$22\ell(\kappa+\ell)$	8ℓ	**1**
	ABY2.0 [PSSY21a]	$22\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**

2σ bits. In comparison, the OTTT method [IKM+13; DZ16] involved $(|\mathsf{MT}|+4)\cdot(\delta-1)\cdot 2^{\delta}\sigma$ bits of communication during the setup phase and 2δ bits during the online phase. Here, $|\mathsf{MT}|$ represents the cost for generating a Boolean multiplication triple. The OP-LUT strategy [DKS+17] had a setup communication of $|\binom{2^{\delta}}{1}\text{-}\mathsf{OT}^1_{2^{\delta}\sigma}| - \delta$ bits and 2δ bits for the online phase. Meanwhile, the SP-LUT method [DKS+17] involved setup communication of $|\binom{2^{\delta}}{1}\text{-}\mathsf{rOT}^1_{\sigma}|$ bits and online communication of $\delta + 2^{\delta}\sigma$ bits. Here, $|\binom{n}{1}\text{-}\mathsf{OT}^1_{l}|$ denotes the cost for one OT with n messages and l-bit inputs, and $|\binom{n}{1}\text{-}\mathsf{rOT}^1_{l}|$ denotes the cost for the same but with a random OT.

Fig. 3.3 provides a visual comparison of FLUTE against prior works, focusing on setup and online communication costs. As shown, FLUTE achieves online communication efficiency on par with OP-LUT [DKS+17] and OTTT [IKM+13; DZ16], while maintaining total communication efficiency comparable to SP-LUT [DKS+17], which was recognized for its minimal overall communication costs. Additionally, Fig. 3.3 illustrates how the efficiency of previous methods shifts when their traditional OT extension schemes in the style of IKNP [IKNP03], are substituted with the latest advancements in silent OT extension techniques [BCG+19b].

3.2.3 UC-based PFE Protocols

The most adaptable and current method in PFE involved the use of UCs [Val76; KS08a; KS16; GKS17; ZYZL19; AGKS20a; LYZ+21], with a complexity of $\Theta(n \log n)$ [Val76], where n represents the size of the private function being simulated. Initially, Valiant [Val76] proposed two efficient UC structures: one with a 2-way and another with a 4-way recursive design, measuring approximately $\sim 5n\log n$ and $\sim 4.75n\log n$ in size, respectively. Cook and Hover [CH85] developed a UC optimized for depth, capable of emulating Boolean circuits of size n and depth d, with a size of $\mathcal{O}(n^3 d/n)$ and a depth of $\mathcal{O}(d)$. Kolesnikov and

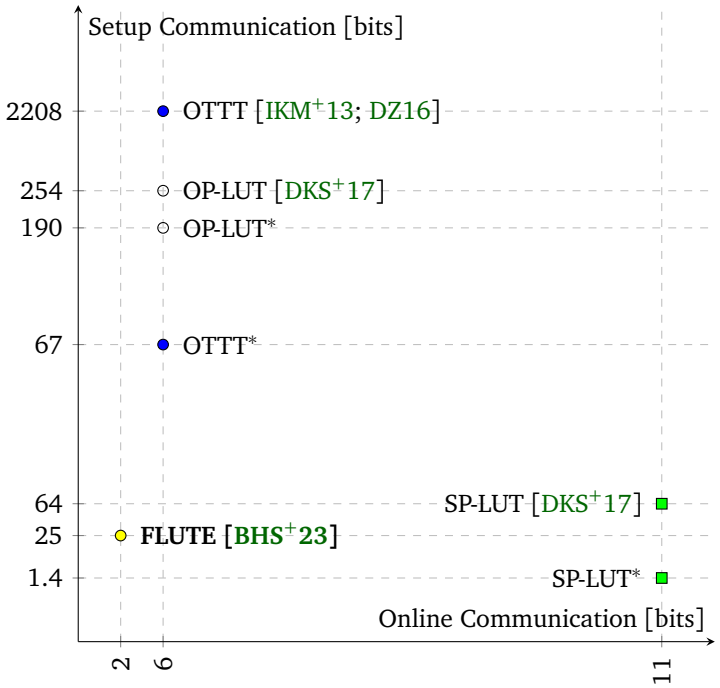


Figure 3.3: Comparison of setup and online communication (in bits) for previous LUT protocols and FLUTE, focusing on LUT with 3 inputs and 1 output. The * symbol denotes modified LUT protocols incorporating silent OT [BCG+19a].

Schneider [KS08a] were the first to provide a practical UC implementation, though it had a non-optimal asymptotic size of $\mathcal{O}(n \log^2 n)$.

The primary aim in this field has been to minimize the overall size of the UC and its prefactor. Research by Günther et al. [GKS17] modularized Valiant's construction, implemented the more efficient 4-way split construction, and provided a generic edge-embedding algorithm for k-way split constructions. They also showed that the 3-way split construction within Valiant's framework is less efficient than the 2-way split construction. Alhassan et al. [AGKS20b] proposed and implemented a scalable hybrid UC construction that combines Valiant's 2-way and 4-way split constructions with Zhao et al.'s improvements [ZYZL19], which brought Valiant's 4-way split design down to approximately $\sim 4.5n \log n$. More recently, Liu et al. [LYZ+21] further refined Valiant's approaches, particularly the 2-way split design, reducing its size to around $\sim 3n \log n$. They also demonstrated that a $k = 2$-way split is the most efficient for their updated UC construction, closely approaching their theoretical lower bound of $\sim 2.95n \log n$.

Our Work We are *the first* to implement the design by Liu et al. [LYZ+21], which forms the foundation for our UC constructions. Prior to our work, all advancements in the field were centered around UCs designed for Boolean circuits that have 2-input and 1-output gates. In

our research, we expanded upon the UC framework established by Liu et al. [LYZ+21] to support LUTs.

3.2.4 Boolean Circuit Synthesis

Compilers that translate high-level code to binary circuits offer a higher level of abstraction when running MPC in the binary domain. Works in this area include Fairplay [MNPS04], Fairplay-MP [BNP08], CBMC-GC [FHK+14], ObliVM [LWN+15], PCF [KMSB13], and HyCC [BDK+18]. Another approach is the deployment of existing hardware synthesis tools that take code in a HDL as input. Examples of that include TinyGarble [SHS+15], TinyGMW [DDK+15], SynCirc [**PSSY22**], and FLUENT [**GSSY24**]. As demonstrated in [DKS+17], this approach can be extended to LUTs by utilizing and re-purposing LUT-based synthesis tools [WGK13; Ber].

There is a range of specialized commercial FPGA synthesis tools available, including the Intel HLS Compiler[1], Microchip SmartHLS[2], and AMD Xilinx Vivado[3]. These tools are designed to create LUT-based circuits that are optimized for the specific characteristics of their respective devices, such as the size and number of LUTs that are physically available on the FPGA.

Our work We successfully generate LUTs with up to 8 inputs, a task that exceeds the capabilities of most standard commercial synthesis tools. The Berkeley Logic Synthesis and Verification tool, ABC [Ber], excels by enabling the mapping of circuits to LUTs with a variable number of inputs, using a technology known as priority cuts [Cho07]. ABC is unique in its experimental implementation of various mapping and optimization techniques, including strategies for optimal-delay mapping using directed acyclic graphs (DAGs). What makes ABC particularly suited to our project is its flexibility in allowing users to set the maximum number of LUT inputs, independent of the specific constraints of any FPGA design. Reflecting the findings in [DKS+17], we limit the LUT inputs to 8. To construct our LUT-based circuits, we use the Yosys-ABC synthesis tool [WGK13; Ber], alongside the Synopsis Design Compiler [10] for the LUT-Mapping process.

Further Application of LUT Synthesis Building on our synthesis pipeline and the use of LUTs in MPC [**BHS+23**] and PFE [**DGS+23**], we expand our research to include the application of LUT-based circuits in addressing cache side-channel attacks [**MSS+24**] (not part of this thesis). Specifically, we find that circuits hardened using LUTs are not only more efficient performance-wise but also maintain the same level of protection against cache side-channel attacks as those hardened with Boolean circuits. To achieve this, we develop HyCaMi, a framework that combines LUT synthesis with detailed analysis of side-channel vulnerabilities.

[1] https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html

[2] https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/smarthls-compiler

[3] https://www.xilinx.com/products/design-tools/vivado.html

3.2.5 Protocols for symmetrical and asymmetrical trust settings

In recent research [GRW18; BCPS20], the concept of function-dependent preprocessing has been identified as a method to simplify the complexity of the online phase, thereby ensuring optimal performance when actual inputs are provided. Function-dependent preprocessing has proven particularly useful in situations like *machine learning as a service (MLaaS)*, where the same computational processes are executed repeatedly. This repetition allows for the effective batching of preprocessing tasks across several uses of the protocol.

However, the typical assumptions of semi-honest or malicious parties assume a uniform level of trust across all participants, an assumption that often does not align with the varied trust levels present among participants in many practical scenarios. Particularly in the field of PPML, there is a growing interest in adapting to what is referred to as a *fixed-corruption* model. In this model, the potential for parties to be corrupted is predetermined and known, diverging from the assumption that any party could potentially act maliciously [MR18; PS20; KPPS21]. Protocols designed for semi-honest participants may not provide adequate security in such contexts [LMSP21], while those designed for malicious security model incur unnecessary performance costs.

While it is common practice to allow the adversary to arbitrarily corrupt any subset of the parties while still respecting the underlying corruption threshold, constrained settings where only a certain subset of parties are allowed to behave maliciously can allow for simpler and more efficient protocols. A well-known example of this is Yao's two-party GC [Yao86]. While the original work is secure only against semi-honest corruption, several works [WRK17b; WRK17c; YWZ20] have enhanced the security against malicious corruption by incorporating additional verification mechanisms. Protecting against a malicious garbler, in particular, incurs significant additional computational and communication costs (cf. [Lin20] for an overview of different techniques). However, it is well-known that the original scheme is inherently secure against a malicious evaluator when using oblivious transfer secure against a malicious receiver. Thus, an asymmetric corruption scenario for two parties in which one party is guaranteed not to act maliciously can lower MPC costs as compared to schemes that allow either party to be corrupted. Such an asymmetric trust model was considered in MUSE [LMSP21] in the context of PPML, named *client-malicious setting*, where a semi-honest server and a malicious client perform MPC operations to conduct ML inference. The efficiency improvements realized in MUSE have prompted further development in this area, leading to the creation of more efficient protocols like SIMC [CGOS22] and its successor, SIMC2.0 [XHZ+23]. In parallel to these developments, the recent three-party model pMPL [SWW+22] examined scenarios where one party was considered inherently more trustworthy than the others.

Our work In [**BSS+24**] (Sect. 3.1.4), we revisit the three-party secure computation techniques in the honest majority setting (3PC) and investigate the potential advancements that could be made if the corrupt party's identity is known. We provide two protocols that address two different corruption scenarios. Our protocols are designed using the input-independent preprocessing paradigm to ensure a very efficient online phase. Our approach utilizes a

mixed-protocol [KPRS22] approach combined with *function-dependent* preprocessing [BNO19; BCPS20], in line with ASTRA [CCPS19] and SWIFT [KPPS21].

Tab. 3.2 compares the communication costs for performing a multiplication operation in our protocols with those in other relevant 3PC protocols. Maliciously secure 3PC protocols over rings typically use one of two approaches: Distributed Zero-Knowledge Proofs (DZKP) [BBC+19] and triple sacrificing [CDE+18a]. The DZKP approach achieves sublinear communication complexity at the expense of increased computational complexity, while the triple sacrificing approach maintains low computational complexity but requires higher communication overhead. Tab. 3.2 shows both variants for the malicious case.

Table 3.2: Comparison of communication cost per multiplication for our protocols proposed in [**BSS+24**] and related 3PC protocols over the ring $\mathbb{Z}_{2^\ell}$. σ denotes the statistical security parameter.

Verification Method	Security	Corruption	Protocol	Communication	
				Setup	Online
–	Semi-honest	any	Araki et al.[AFL+16]	0	3ℓ
		any	ASTRA [CCPS19]	ℓ	2ℓ
Distributed Zero Knowledge	malicious	any	Boyle et al.[BGIN19]	0	3ℓ
		any	SWIFT [KPPS21]	3ℓ	3ℓ
		evaluator	**Socium [BSS+24, §5]**	$\mathbf{2}\ell$	$\mathbf{3}\ell$
		helper	**Auxiliator [BSS+24, §4]**	$\mathbf{1}\ell$	$\mathbf{2}\ell$
Tripe Sacrifice Method	malicious	any	ABY^3 [MR18]	12ℓ	9ℓ
		any	SWIFT [KPPS21]	$9(\ell+\sigma)$	3ℓ
		evaluator	**Socium [BSS+24, §5]**	$\mathbf{5}(\ell+\sigma)$	$\mathbf{3}\ell$
		helper	**Auxiliator [BSS+24, §4]**	$\mathbf{4}(\ell+\sigma)$	$\mathbf{2}\ell$

4 Conclusion and Future Work

In the final chapter, we summarize the key findings of this thesis in Sect. 4.1 and outline potential avenues for future research in Sect. 4.2.

4.1 Summary

This thesis makes significant contributions to the study of practical MPC protocols for real-world applications, focusing on the design and improvement of these protocols.

We conduct a thorough review and analysis of 59 papers in privacy-preserving clustering (cf. Sect. 2.1.1), revealing that only 10 out of 59 papers offer full privacy for real-world applications. Following this, we benchmark and compare four efficient protocols that securely implement four clustering algorithms [CKP19; MRT20; BCE+21; MPOT21]. Our comparison focuses on clustering quality, communication, and runtime to evaluate their suitability for real-world applications.

As another important step toward making secure multi-party computation (MPC) more practical in real-world applications, we take a close look at PEFL [LLX+21] and identify multiple privacy issues (cf. Sect. 2.1.2). We then thoroughly analyze poisoning attacks and introduce FLAME (cf. Sect. 2.1.2), *the first* federated learning (FL) defense that protects both privacy and security in a general adversarial setting. Our detailed evaluation across various machine learning (ML) applications and datasets shows that FLAME effectively counters poisoning attacks without compromising the accuracy of the main task on clean data. Moreover, we design, implement, and benchmark efficient two-party computation (2PC) protocols for FLAME to ensure the privacy of clients' training data and to prevent inference attacks on client updates.

We show how to improve the efficiency of garbled circuit (GC)-based MPC protocols by using vector AES (VAES) technology (cf. Sect. 3.1.1). We apply this technology to various MPC frameworks such as ABY [DSZ15], EMP-AGMPC [WRK17c], and Microsoft's CrypTFlow2 [RRK+20a] for privacy preserving machine learning (PPML). Our evaluation shows that the use of VAES significantly improves performance in these frameworks.

To improve the performance of secret sharing (SS)-based MPC protocols, we introduce ABY2.0 (cf. Sect. 3.1.2) and FLUTE (cf. Sect. 3.1.2). In ABY2.0, we develop a new 2PC protocol that securely evaluates circuits over a ring, including those with N-input multiplication gates where online communication is improved from $4N$ in [DSZ15] to just 2 and is independent

of the fan-in. Our new sharing methods allow for conversions between mixed protocols, improving upon the existing state-of-the-art work ABY [DSZ15] in terms of both rounds and online communication. Furthermore, we design a set of efficient primitives for real-world applications, significantly outperforming previous works. Building on the sharing methods of ABY2.0, we also introduce FLUTE, a secure protocol for look-up table (LUT) evaluation, offering improved online communication and overall communication.

In Sect. 3.1.3, we broaden the use of universal circuits (UCs) to simulate circuits consisting of LUTs, with the goal of improving the performance of private function evaluation (PFE) protocols. Our new construction significantly reduces the size of the UC, achieving up to a 2.18× reduction for common functions (e.g., full adder, comparator, and multiplexer) compared to the best previous construction [LYZ+21]. Moreover, we generate efficient LUT-based circuits by using and modifying the hardware circuit synthesis tools Yosys, ABC, and Synopsis Design Compiler specifically for LUT mapping.

To bridge the gap between fully semi-honest and fully malicious scenarios in real-world applications, we introduce two novel protocols, AUXILIATOR and SOCIUM (cf. Sect. 3.1.4). These protocols are designed for practical application, allowing for various levels of trust among participants and offering flexible ways to balance the trade-off between computation and communication overheads. We show how the involvement of a malicious helper can improve the efficiency of semi-honest 2PC, and similarly, how a semi-honest helper can aid in 2PC when dealing with a malicious party.

4.2 Future Work

Finally, we give some ideas for future work.

4.2.1 Advancing Privacy in Real-World Applications

Practical Privacy-Preserving Clustering For real-world applications, it is essential that privacy-preserving clustering protocols are efficient in both runtime and communication, utilize memory efficiently, depend on parameters that are mostly independent of the input data, and are capable of clustering data from any distribution with high accuracy. Unfortunately, none of the state-of-the-art approaches satisfy all these requirements simultaneously. Furthermore, there's a clear need to evaluate the quality of clustering results obtained in a privacy-preserving manner. In future work, these issues can be addressed comprehensively, with a special emphasis on using newly optimized MPC protocols to offer efficient solutions for practical privacy-preserving clustering.

Fully-Private ML Inference PPML inference solutions vary in their privacy guarantees. While these solutions generally protect the data used for inference, they might reveal information about the ML model, which can be undesirable in practical applications. Homomorphic encryption (HE)-based methods secure the model completely, but this comes at the expense of increased computation overhead. Conversely, MPC-based techniques may disclose the structure or functional form of the ML model, though they have the advantage of the least computation overhead. Hybrid HE-MPC solutions, like MP2ML [**BCD+20**], leak specific aspects such as the type and dimension of each activation function. In future work, the use of PFE-based methods can be investigated to completely protect ML models for ML inference solutions, such as [SS08], particularly by applying optimized PFE protocols that utilize LUT-based circuits.

Privacy Vulnerabilities in FL Recent studies [WGF+22; BDS+23] have highlighted significant concerns regarding privacy vulnerabilities in FL, particularly when using secure aggregation (SA) with a single aggregator. These concerns reveal the effectiveness of privacy attacks, even when SA techniques are used across a large number of clients, demonstrating that such methods are inadequate for practical applications. The limitations of standard FL, as well as FL with SA, become evident in real-world scenarios, especially when the legal intricacies of sharing sensitive data across jurisdictions are considered. In our current research [**MSS+23**], we tackle these issues by developing a distributed aggregator framework. This framework, based on our improved MPC techniques, aims to securely and efficiently distribute aggregation tasks among multiple servers. Our objective is to ensure privacy, even in situations where some servers might collude, influenced by our previous works [**FMM+21**; **GMS+23**].

Data Extraction Attacks in FL Recent research indicates that unrestricted access to the aggregated model in FL can lead to the extraction of traces from the original training data due to inadequate privacy protections for the global model [PFA22]. Initially, FL was designed to enhance user engagement and improve model accuracy, rather than focusing on the model's privacy. Yet, as FL becomes crucial across various industries, such as healthcare, the need for privacy protections becomes clear. Consider the scenario where a group of five hospitals collaborates to train a model using data from their hospitals to discover treatments for a specific disease. While offering payment to hospitals for their data, the group would prioritize safeguarding the confidentiality of the resulting model due to its sensitivity, investment significance, and associated legal risks. The lack of proper privacy for the global model could allow a malicious participant to misuse or commercialize the model, undermining the joint efforts. The possibility of extracting data from the model underscores the urgent need for improved privacy measures. In our future work, we intend to merge our optimized MPC protocols with FL to ensure the global model's privacy and address the issue of data extraction in real-world FL applications.

4.2.2 More Efficient MPC Protocols and Primitives

Beyond PPML with Enhanced 2PC Protocols With the emergence of many real-world applications, such as privacy-preserving auctions [NPS99], private text classification [PZM+24], and private graph analytics [AFO+21], that use 2PC protocols, the idea of expanding the use of our advanced 2PC protocols to a wider range of applications is an interesting direction. Initially, our protocols are designed specifically for PPML applications. However, the potential for their use in other areas is significant and deserves further exploration. In our future work, we plan to conduct a comprehensive survey of potential applications and adapt our enhanced protocols to suit these diverse use cases.

Expanding to Additional Scenarios ABY2.0 and FLUTE are originally designed for the semi-honest security model and tailored for two-party scenarios. Extending them to handle malicious security, where adversaries can arbitrarily deviate from the protocol, and adapting them for broader multi-party settings presents valuable research directions. In our future work, we plan to adapt both ABY2.0 and FLUTE for use in malicious and multi-party scenarios by integrating a MAC (Message Authentication Code) checking technique to ensure security against malicious behavior. Specifically, we will employ a MAC checking procedure, similar to the one used in [CDE+18a]. By incorporating this MAC checking mechanism, we aim to strengthen the security of ABY2.0 and FLUTE against malicious adversaries, while also expanding their applicability to multi-party settings.

Improving Efficiency in PPML with FLUTE and LUTNet Combination Binary neural networks (BNNs) are a promising approach for improving the performance of PPML [RSC+19]. LUTNet [WDCC20] introduced a LUT-based neural network design, resulting in a significantly reduced logic size compared to the most advanced BNNs. Our future efforts will focus on using FLUTE for LUTNet to develop a more effective PPML solution. This strategy seeks to combine the compact architecture of LUTNet with FLUTE, aiming to improve efficiency in PPML.

Developing an Enhanced Compiler/Toolchain for FLUTE In our efforts to generate optimized circuits, we utilize conventional hardware synthesis tools instead of custom solutions. For our future work, we plan to develop a specialized compiler/toolchain to produce LUT circuits specifically optimized for FLUTE. This includes developing improved heuristics for LUT generation. This work seeks to refine the circuit generation process, enhancing the efficiency and effectiveness of FLUTE.

New Protocols for Asymmetric Security Models We focus on scenarios with asymmetric trust levels where only one party (out of three) is suspected of being malicious. Moving forward, we plan to extend our investigation to situations where two out of the three servers might exhibit malicious behavior. Moreover, we aim to broaden our research to include new methods and approaches applicable to a wider array of real-world applications. This expansion will extend beyond the current application in ML inference to include areas such as graph neural networks and decision trees.

Bibliography

[10] **"Synopsys Inc. Design compiler"**. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler. 2010.

[AAUC18] A. ACAR, H. AKSU, A. S. ULUAGAC, M. CONTI. **"A Survey on Homomorphic Encryption Schemes: Theory and Implementation"**. In: *ACM Computing Surveys* 51.4 (2018), 79:1–79:35.

[ABF+17] T. ARAKI, A. BARAK, J. FURUKAWA, T. LICHTER, Y. LINDELL, A. NOF, K. OHARA, A. WATZMAN, O. WEINSTEIN. **"Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2017, pp. 843–862.

[ACHY23] T. ASHUR, E. COHEN, C. HAZAY, A. YANAI. **"A New Framework for Garbled Circuits"**. In: *ACNS*. 2023.

[AE06] A. AMIRBEKYAN, V. ESTIVILL-CASTRO. **"Privacy Preserving *DBSCAN* for Vertically Partitioned Data"**. In: *Intelligence and Security Informatics*. Vol. 3975. LNCS. Springer, 2006, pp. 141–153.

[AFL+16] T. ARAKI, J. FURUKAWA, Y. LINDELL, A. NOF, K. OHARA. **"High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority"**. In: *Computer and Communications Security (CCS)*. ACM, 2016, pp. 805–817.

[AFO+21] T. ARAKI, J. FURUKAWA, K. OHARA, B. PINKAS, H. ROSEMARIN, H. TSUCHIDA. **"Secure Graph Analysis at Scale"**. In: *Computer and Communications Security (CCS)*. ACM, 2021, pp. 610–629.

[AG17] I. V. ANIKIN, R. M. GAZIMOV. **"Privacy Preserving DBSCAN Clustering Algorithm for Vertically Partitioned Data in Distributed Systems"**. In: *International Siberian Conference on Control and Communications*. IEEE, 2017, pp. 1–4.

[AGKS20a] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. **"Efficient and Scalable Universal Circuits"**. In: *Journal of Cryptology (JoC)* 33.3 (2020), pp. 1216–1271.

[AGKS20b] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. **"Efficient and Scalable Universal Circuits"**. In: *Journal of Cryptology (JoC)* 33.3 (2020), pp. 1216–1271.

[AGM+13] O. ARBELAITZ, I. GURRUTXAGA, J. MUGUERZA, J. M. PÉREZ, I. PERONA. **"An Extensive Comparative Study of Cluster Validity Indices"**. In: *Pattern Recognition* 46.1 (2013), pp. 243–256.

[ASA22] S. ATAPOOR, N. P. SMART, Y. T. ALAOUI. **"Private Liquidity Matching Using MPC"**. In: *Cryptographers' Track at the RSA Conference (CT-RSA)*. Vol. 13161. LNCS. Springer, 2022, pp. 96–119.

[BBC+19] D. BONEH, E. BOYLE, H. CORRIGAN-GIBBS, N. GILBOA, Y. ISHAI. **"Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs"**. In: *CRYPTO*. 2019.

[BBG+20] J. H. BELL, K. A. BONAWITZ, A. GASCÓN, T. LEPOINT, M. RAYKOVA. **"Secure Single-Server Aggregation with (Poly)Logarithmic Overhead"**. In: *Computer and Communications Security (CCS)*. ACM, 2020, pp. 1253–1269.

[BBG19] G. BARUCH, M. BARUCH, Y. GOLDBERG. **"A Little Is Enough: Circumventing Defenses For Distributed Learning"**. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 8632–8642.

[BBL+14] M. BARNI, M. BERNASCHI, R. LAZZERETTI, T. PIGNATA, A. SABELLICO. **"Parallel Implementation of GC-Based MPC Protocols in the Semi-Honest Setting"**. In: *Data Privacy Management and Autonomous Spontaneous Security*. 2014.

[BCD+20] F. BOEMER, R. CAMMAROTA, D. DEMMLER, T. SCHNEIDER, H. YALAME. **"MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference"**. In: *International Conference on Availability, Reliability and Security (ARES)*. Online: https://ia.cr/2020/721. Code: https://github.com/IntelAI/he-transformer. ACM, 2020, 14:1–14:10. CORE Rank B.

[BCE+21] B. BOZDEMIR, S. CANARD, O. ERMIS, H. MÖLLERING, M. ÖNEN, T. SCHNEIDER. **"Privacy-Preserving Density-based Clustering"**. In: *Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 2021, pp. 658–671.

[BCG+19a] E. BOYLE, G. COUTEAU, N. GILBOA, Y. ISHAI, L. KOHL, P. RINDAL, P. SCHOLL. **"Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2019, pp. 291–308.

[BCG+19b] E. BOYLE, G. COUTEAU, N. GILBOA, Y. ISHAI, L. KOHL, P. SCHOLL. **"Efficient Pseudorandom Correlation Generators: Silent OT Extension and More"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 11694. LNCS. Springer, 2019, pp. 489–518.

[BCPS20] M. BYALI, H. CHAUDHARI, A. PATRA, A. SURESH. **"FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning"**. In: *Proc. Priv. Enhancing Technol. (PETS)* 2020.2 (2020), pp. 459–480.

[BDK+18] N. BÜSCHER, D. DEMMLER, S. KATZENBEISSER, D. KRETZMER, T. SCHNEIDER. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *Computer andCommunications Security (CCS)*. ACM, 2018, pp. 847–861.

[BDS+23] F. BOENISCH, A. DZIEDZIC, R. SCHUSTER, A. S. SHAMSABADI, I. SHUMAILOV, N. PAPERNOT. **"When the Curious Abandon Honesty: Federated Learning Is Not Private"**. In: *IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*. IEEE, 2023, pp. 175–199.

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *ACM Transactions on Privacy and Security (TOPS)* 25.2 (2022), 8:1–8:35.

[Bea91] D. BEAVER. **"Efficient Multiparty Protocols Using Circuit Randomization"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 576. LNCS. Springer, 1991, pp. 420–432.

[BEG+19] K. A. BONAWITZ, H. EICHNER, W. GRIESKAMP, D. HUBA, A. INGERMAN, V. IVANOV, C. KIDDON, J. KONEČNÝ, S. MAZZOCCHI, B. MCMAHAN, T. V. OVERVELDT, D. PETROU, D. RAMAGE, J. ROSELANDER. **"Towards Federated Learning at Scale: System Design"**. In: *Proceedings of Machine Learning and Systems (MLSys)*. mlsys.org, 2019.

[Ber] BERKELEY LOGIC SYNTHESIS AND VERIFICATION GROUP. **"ABC: A System for Sequential Synthesis and Verification"**. url =http://www.eecs.berkeley.edu/ alanmi/abc/.

[BGH+22] P. Beerel, M. Georgiou, B. Hamlin, A. J. Malozemoff, P. Nuzzo. **"Towards a Formal Treatment of Logic Locking"**. In: *TCHES* (2022).

[BGIN19] E. Boyle, N. Gilboa, Y. Ishai, A. Nof. **"Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs"**. In: *Computer and Communications Security (CCS)*. ACM, 2019, pp. 869–886.

[BHKR13] M. Bellare, V. T. Hoang, S. Keelveedhi, P. Rogaway. **"Efficient Garbling from a Fixed-key Blockcipher"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2013, pp. 478–492.

[BHS+23] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, H. Yalame. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, pp. 515–533. CORE Rank **A***. Appendix F.

[BIK+17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, K. Seth. **"Practical Secure Aggregation for Privacy-Preserving Machine Learning"**. In: *Computer and Communications Security (CCS)*. ACM, 2017, pp. 1175–1191.

[BK15] N. Buescher, S. Katzenbeisser. **"Faster Secure Computation through Automatic Parallelization"**. In: *USENIX Security*. 2015.

[BL01] A. Beaumont-Smith, C. Lim. **"Parallel Prefix Adder Design"**. In: *IEEE Symposium on Computer Arithmetic*. IEEE, 2001, p. 218.

[BLO16] A. Ben-Efraim, Y. Lindell, E. Omri. **"Optimizing Semi-Honest Secure Multiparty Computation for the Internet"**. In: *Computer and Communications Security (CCS)*. ACM, 2016, pp. 578–590.

[BMGS17] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, J. Stainer. **"Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent"**. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 119–129.

[BMP+24] Y. Ben-Itzhak, H. Möllering, B. Pinkas, T. Schneider, A. Suresh, O. Tkachenko, S. Vargaftik, C. Weinert, H. Yalame, A. Yanai. **"ScionFL: Secure Quantized Aggregation for Federated Learning"**. In: *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. Online: https://ia.cr/2023/652. Code: https://encrypto.de/code/ScionFL. IEEE, 2024.

[BMP13] J. Boyar, P. Matthews, R. Peralta. **"Logic Minimization Techniques With Applications to Cryptology"**. In: *Journal of Cryptology* (2013).

[BMR90] D. Beaver, S. Micali, P. Rogaway. **"The Round Complexity of Secure Protocols"**. In: *Symposium on Theory of Computing (STOC)*. ACM, 1990, pp. 503–513.

[BNO19] A. Ben-Efraim, M. Nielsen, E. Omri. **"Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing"**. In: *Applied Cryptography and Network Security (ACNS)*. Vol. 11464. LNCS. Springer, 2019, pp. 530–549.

[BNP08] A. Ben-David, N. Nisan, B. Pinkas. **"FairplayMP: a system for secure multi-party computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2008, pp. 257–266.

[BP12] J. Boyar, R. Peralta. **"A Small Depth-16 Circuit for the AES S-box"**. In: *IFIP International Information Security Conference*. 2012.

[BSS+24] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, H. Yalame. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

[BTM23] R. Burra, A. Tandon, S. Mittal. **"Empowering SMPC: Bridging the Gap Between Scalability, Memory Efficiency and Privacy in Neural Network Inference"**. In: *arXiv preprint arXiv:2310.10133* (2023).

[BVH+20] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, V. Shmatikov. **"How To Backdoor Federated Learning"**. In: *Artificial Intelligence and Statistics (AISTATS)*. Vol. 108. PMLR, 2020, pp. 2938–2948.

[CB17] H. Corrigan-Gibbs, D. Boneh. **"Prio: Private, Robust, and Scalable Computation of Aggregate Statistics"**. In: *USENIX NSDI*. USENIX Association, 2017, pp. 259–282.

[CCGR97] A. Chaturvedi, J. Carroll, P. Green, J. A. Rotondo. **"A Feature-Based Approach to Market Segmentation via Overlapping K-Centroids Clustering"**. In: *Journal of Marketing Research* 34.3 (1997), pp. 370–377.

[CCKS22] W. Chen, C. A. Choquette-Choo, P. Kairouz, A. T. Suresh. **"The Fundamental Price of Secure Aggregation in Differentially Private Federated Learning"**. In: *International Conference on Machine Learning (ICML)*. Vol. 162. PMLR, 2022, pp. 3056–3089.

[CCPS19] H. Chaudhari, A. Choudhury, A. Patra, A. Suresh. **"ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction"**. In: *Cloud Computing Security Workshop (CCSW@CCS)*. ACM, 2019, pp. 81–92.

[CD16] J. Cong, Y. Ding. "FPGA Technology Mapping". In: *Encyclopedia of Algorithms*. 2016, pp. 773–777.

[CDE+18a] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing. **"SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 10992. LNCS. Springer, 2018, pp. 769–798.

[CDE+18b] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing. **"SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority"**. In: *Advances in Cryptology (CRYPTO)*. 2018.

[CDG+23] S. Carpov, K. Deforth, N. Gama, M. Georgieva, D. Jetchev, J. Katz, I. Leontiadis, M. Mohammadi, A. Sae-Tang, M. Vuille. **"Manticore: Efficient Framework for Scalable Secure Multiparty Computation Protocols"**. In: *JoC* (2023).

[CFLG21] X. Cao, M. Fang, J. Liu, N. Z. Gong. **"FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2021.

[CGOS22] N. Chandran, D. Gupta, S. L. B. Obbattu, A. Shah. **"SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2022, pp. 1361–1378.

[CH85] S. A. Cook, H. J. Hoover. **"A Depth-Universal Circuit"**. In: *SIAM J. Computing* (1985).

[Cho07] Cho, Sungmin and Chatterjee, Satrajit and Mishchenko, Alan and Brayton, Robert. **"Efficient FPGA Mapping Using Priority Cuts"**. In: *Proc. FPGA*. 2007.

[CKP19] J. H. CHEON, D. KIM, J. H. PARK. **"Towards A Practical Cluster Analysis over Encrypted Data"**. In: *Selected Areas in Cryptography (SAC)*. Vol. 11959. Springer, 2019, pp. 227–249.

[CLH+23] L. CHEN, Y. LI, C. HUANG, B. LI, Y. XING, D. TIAN, L. LI, Z. HU, X. NA, Z. LI. **"Milestones in Autonomous Driving and Intelligent Vehicles: Survey of Surveys"**. In: *IEEE Transactions on Intelligent Vehicles* 8.2 (2023), pp. 1046–1056.

[CP22] A. CHOUDHURY, A. PATRA. **"Secure Multi-Party Computation Against Passive Adversaries"**. Springer Nature, 2022.

[DCL+21] Y. DONG, X. CHEN, K. LI, D. WANG, S. ZENG. **"FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority"**. In: *European Symposium on Research in Computer Security (ESORICS)*. Vol. 12972. LNCS. Springer, 2021, pp. 497–518.

[DDK+15] D. DEMMLER, G. DESSOUKY, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER, S. ZEITOUNI. **"Automated Synthesis of Optimized Circuits for Secure Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2015, pp. 1504–1517.

[DG19] N. DRUCKER, S. GUERON. **"Generating a Random String with a Fixed Weight"**. In: *CSCML*. Springer, 2019, pp. 141–155.

[DGK19] N. DRUCKER, S. GUERON, V. KRASNOV. **"Making AES Great Again: The Forthcoming Vectorized AES Instruction"**. In: *International Conference on Information Technology-New Generations (ITNG)*. Springer, 2019, pp. 37–41.

[DGS+23] Y. DISSER, D. GÜNTHER, T. SCHNEIDER, M. STILLGER, A. WIGANDT, H. YALAME. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 14438. LNCS. Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/LUC. Springer, 2023, pp. 3–37. CORE Rank **A**. Appendix G.

[DKS+17] G. DESSOUKY, F. KOUSHANFAR, A. SADEGHI, T. SCHNEIDER, S. ZEITOUNI, M. ZOHNER. **"Pushing the Communication Barrier in Secure Computation using Lookup Tables"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2017.

[DNNR17] I. DAMGÅRD, J. B. NIELSEN, M. NIELSEN, S. RANELLUCCI. **"The TinyTable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited"**. In: *Advances in Cryptology (CRYPTO)*. 2017.

[DPSZ12] I. DAMGÅRD, V. PASTRO, N. P. SMART, S. ZAKARIAS. **"Multiparty Computation from Somewhat Homomorphic Encryption"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 7417. LNCS. Springer, 2012, pp. 643–662.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY–A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[DT13] I. DE, A. TRIPATHY. **"A Secure Two Party Hierarchical Clustering Approach for Vertically Partitioned Data Set with Accuracy Measure"**. In: *Recent Advances in Intelligent Informatics*. Vol. 235. Springer, 2013, pp. 153–162.

[DWL+23] C. DONG, J. WENG, M. LI, J.-N. LIU, Z. LIU, Y. CHENG, S. YU. **"Privacy-Preserving and Byzantine-Robust Federated Learning"**. In: *IEEE TDSC* (2023).

[DZ16] I. DAMGÅRD, R. W. ZAKARIAS. **"Fast Oblivious AES A Dedicated Application of the MiniMac Protocol"**. In: *AFRICACRYPT*. 2016.

[FCJG20] M. FANG, X. CAO, J. JIA, N. GONG. **"Local Model Poisoning Attacks to Byzantine-Robust Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2020, pp. 1605–1622.

[FHK+14] M. FRANZ, A. HOLZER, S. KATZENBEISSER, C. SCHALLHART, H. VEITH. **"CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations"**. In: *International Conference on Compiler Construction*. 2014.

[FMM+21] H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, H. MÖLLERING, T. D. NGUYEN, P. RIEGER, A.-R. SADEGHI, T. SCHNEIDER, H. YALAME, S. ZEITOUNI. **"SAFELearn: Secure Aggregation for private FEderated Learning"**. In: *Deep Learning and Security Workshop (DLS)*. Online: https://ia.cr/2021/386. IEEE, 2021, pp. 56–62.

[GC16] Z. GHEID, Y. CHALLAL. **"Efficient and Privacy-Preserving k-Means Clustering for Big Data Mining"**. In: *IEEE TrustCom/BigDataSE/ISPA*. IEEE, 2016, pp. 791–798.

[GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. **"More Efficient Universal Circuit Constructions"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 10625. LNCS. Springer, 2017, pp. 443–470.

[GKW+20] C. GUO, J. KATZ, X. WANG, C. WENG, Y. YU. **"Better Concrete Security for Half-Gates Garbling (in the Multi-instance Setting)"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 12171. LNCS. Springer, 2020, pp. 793–822.

[GKWY20] C. GUO, J. KATZ, X. WANG, Y. YU. **"Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2020, pp. 825–841.

[GLNP15] S. GUERON, Y. LINDELL, A. NOF, B. PINKAS. **"Fast Garbling of Circuits Under Standard Assumptions"**. In: *Computer and Communications Security (CCS)*. ACM, 2015, pp. 567–578.

[GMS+23] T. GEHLHAR, F. MARX, T. SCHNEIDER, A. SURESH, T. WEHRLE, H. YALAME. **"SafeFL: MPC-friendly framework for Private and Robust Federated Learning"**. In: *Deep Learning Security and Privacy Workshop (DLSP)*. Online: https://ia.cr/2023/555, Code: https://encrypto.de/code/SAFEFL. IEEE, 2023, pp. 69–76.

[GRW18] S. D. GORDON, S. RANELLUCCI, X. WANG. **"Secure Computation with Low Communication from Cross-Checking"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 11274. LNCS. Springer, 2018, pp. 59–85.

[GSCM07] S. GAUCH, M. SPERETTA, A. CHANDRAMOULI, A. MICARELLI. **"User Profiles for Personalized Information Access"**. In: *The Adaptive Web, Methods and Strategies of Web Personalization*. Vol. 4321. LNCS. Springer, 2007, pp. 54–89.

[GSSY24] D. GÜNTHER, J. SCHMIDT, T. SCHNEIDER, H. YALAME. **"FLUENT: A Tool for Efficient Mixed-Protocol Semi-Private Function Evaluation"**. In: *Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2024. CORE Rank **A**.

[Hea24] D. HEATH. **"Efficient Arithmetic in Garbled Circuits"**. In: *EUROCRYPT*. 2024.

[HEKM11] Y. HUANG, D. EVANS, J. KATZ, L. MALKA. **"Faster Secure Two-Party Computation Using Garbled Circuits"**. In: *USENIX Security*. 2011.

[HK19] S. U. HUSSAIN, F. KOUSHANFAR. **"FASE: FPGA Acceleration of Secure Function Evaluation"**. In: *FCCM*. 2019.

[HKN24] D. HEATH, V. KOLESNIKOV, L. K. NG. **"Garbled Circuit Lookup Tables with Logarithmic Number of Ciphertexts"**. In: *EUROCRYPT*. 2024.

[HKS+10] W. HENECKA, S. KÖGL, A. SADEGHI, T. SCHNEIDER, I. WEHRENBERG. **"TASTY: Tool for Automating Secure Two-Party Computations"**. In: *Computer and Communications Security (CCS)*. 2010.

[HKST22] K. HAMACHER, T. KUSSEL, T. SCHNEIDER, O. TKACHENKO. **"PEA: Practical Private Epistasis Analysis Using MPC"**. In: *ESORICS*. Springer, 2022, pp. 320–339.

[HLS+20] C. HE, S. LI, J. SO, M. ZHANG, H. WANG, X. WANG, P. VEPAKOMMA, A. SINGH, H. QIU, L. SHEN, P. ZHAO, Y. KANG, Y. LIU, R. RASKAR, Q. YANG, M. ANNAVARAM, S. AVESTIMEHR. **"FedML: A Research Library and Benchmark for Federated Machine Learning"**. In: *CoRR* abs/2007.13518 (2020).

[HLX+21] M. HAO, H. LI, G. XU, H. CHEN, T. ZHANG. **"Efficient, Private and Robust Federated Learning"**. In: *ACSAC*. 2021.

[HMSG13] N. HUSTED, S. MYERS, A. SHELAT, P. GRUBBS. **"GPU and CPU Parallelization of Honest-but-Curious Secure Two-Party Computation"**. In: *ACSAC*. 2013.

[HMSY21a] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"CONTRIBUTED TALK: SoK: Privacy-Preserving Clustering (Extended Abstract)"**. Privacy in Machine Learning Workshop (PriML@NeurIPS). 2021.

[HMSY21b] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"POSTER: SoK: Privacy-preserving Clustering (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@CCS). 2021.

[HMSY21c] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, pp. 225–248. CORE Rank **A**. Appendix A.

[HRGK18] S. U. HUSSAIN, B. D. ROUHANI, M. GHASEMZADEH, F. KOUSHANFAR. **"MAXelerator: FPGA Accelerator for Privacy Preserving Multiply-Accumulate (MAC) on Cloud Servers"**. In: *DAC*. 2018.

[HRP20] V. HARALAMPIEVA, D. RUECKERT, J. PASSERAT-PALMBACH. **"A Systematic Comparison of Encrypted Machine Learning Solutions for Image Classification"**. In: *Workshop on Privacy-Preserving Machine Learning in Practice (PPMLP)*. ACM, 2020, pp. 55–59.

[HST+21] T. HELDMANN, T. SCHNEIDER, O. TKACHENKO, C. WEINERT, H. YALAME. **"LLVM-based Circuit Compilation for Practical Secure Computation"**. In: *Applied Cryptography and Network Security (ACNS)*. Online: https://ia.cr/2021/797. Code: https://encrypto.de/code/LLVM. Springer, 2021, pp. 99–121. CORE Rank B.

[HTGW18] E. HESAMIFARD, H. TAKABI, M. GHASEMI, R. N. WRIGHT. **"Privacy-Preserving Machine Learning as a Service"**. In: *Proceedings on Privacy Enhancing Technologies (PETS)* 2018.3 (2018), pp. 123–142.

[IKM+13] Y. ISHAI, E. KUSHILEVITZ, S. MELDGAARD, C. ORLANDI, A. PASKIN-CHERNIAVSKY. **"On the Power of Correlated Randomness in Secure Computation"**. In: *TCC*. 2013.

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *Advances in Cryptology (CRYPTO)*. 2003.

[İKS+07] A. İNAN, S. V. KAYA, Y. SAYGIN, E. SAVAŞ, A. A. HINTOĞLU, A. LEVI. **"Privacy Preserving Clustering on Horizontally Partitioned Data"**. In: *Data & Knowledge Engineering* 63.3 (2007), pp. 646–666.

[JA18] A. JÄSCHKE, F. ARMKNECHT. **"Unsupervised Machine Learning on Encrypted Data"**. In: *Selected Areas in Cryptography (SAC)*. Vol. 11349. LNCS. Springer, 2018, pp. 453–478.

[JKM05] S. JHA, L. KRUGER, P. MCDANIEL. **"Privacy preserving clustering"**. In: *European Symposium on Research in Computer Security (ESORICS)*. Vol. 3679. LNCS. Springer, 2005, pp. 397–417.

[JKSS10a] K. JÄRVINEN, V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"Embedded SFE: Offloading Server and Network Using Hardware Tokens"**. In: *FC*. 2010.

[JKSS10b] K. JÄRVINEN, V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs"**. In: *CHES*. 2010.

[JPWU10] G. JAGANNATHAN, K. PILLAIPAKKAMNATT, R. WRIGHT, D. UMANO. **"Communication-Efficient Privacy-Preserving Clustering"**. In: *Transactions on Data Privacy* 3.1 (2010), pp. 1–25.

[JVC18] C. JUVEKAR, V. VAIKUNTANATHAN, A. CHANDRAKASAN. **"GAZELLE: A Low Latency Framework for Secure Neural Network Inference"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 1651–1669.

[JW05] G. JAGANNATHAN, R. N. WRIGHT. **"Privacy-Preserving Distributed K-means Clustering over Arbitrarily Partitioned Data"**. In: *Knowledge Discovery and Data Mining (KDD)*. ACM, 2005, pp. 593–599.

[JXJ+08] D. JIANG, A. XUE, S. JU, W. CHEN, H. MA. **"Privacy-preserving DBSCAN on Horizontally Partitioned Data"**. In: *International Symposium on IT in Medicine and Education*. IEEE, 2008, pp. 1067–1072.

[Kil88] J. KILIAN. **"Founding Cryptography on Oblivious Transfer"**. In: *STOC*. 1988.

[KMSB13] B. KREUTER, B. MOOD, A. SHELAT, K. BUTLER. **"PCF: A Portable Circuit Format for Scalable Two-party Secure Computation"**. In: *USENIX Security*. 2013.

[KMSY21] H. KELLER, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"Balancing Quality and Efficiency in Private Clustering with Affinity Propagation"**. In: *International Conference on Security and Cryptography (SECRYPT)*. Online: https://ia.cr/2021/825. Code: https://encrypto.de/code/ppAffinityPropagation. SciTePress, 2021, pp. 173–184. CORE Rank C.

[KMY+16] J. KONEČNÝ, H. B. MCMAHAN, F. X. YU, P. RICHTÁRIK, A. T. SURESH, D. BACON. **"Federated Learning: Strategies for Improving Communication Efficiency"**. In: *CoRR* abs/1610.05492 (2016).

[KOR+17] M. KELLER, E. ORSINI, D. ROTARU, P. SCHOLL, E. SORIA-VAZQUEZ, S. VIVEK. **"Faster Secure Multi-party Computation of AES and DES Using Lookup Tables"**. In: *ACNS*. 2017.

[KOS16] M. KELLER, E. ORSINI, P. SCHOLL. **"MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer"**. In: *Computer and Communications Security (CCS)*. ACM, 2016, pp. 830–842.

[KPPS21] N. KOTI, M. PANCHOLI, A. PATRA, A. SURESH. **"SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2021, pp. 2651–2668.

[KPPS23] N. KOTI, S. PATIL, A. PATRA, A. SURESH. **"MPClan: Protocol Suite for Privacy-Conscious Computations"**. In: *JoC*. 2023.

[KPR18] M. KELLER, V. PASTRO, D. ROTARU. **"Overdrive: Making SPDZ Great Again"**. In: *EUROCRYPT*. Vol. 10822. LNCS. Springer, 2018, pp. 158–189.

[KPRS22] N. KOTI, A. PATRA, R. RACHURI, A. SURESH. **"Tetrad: Actively Secure 4PC for Secure Training and Inference"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2022.

[KR07] K. A. KUMAR, C. P. RANGAN. **"Privacy Preserving DBSCAN Algorithm for Clustering"**. In: *Advanced Data Mining and Applications*. Vol. 4632. LNCS. Springer, 2007, pp. 57–68.

[KRC+20] N. KUMAR, M. RATHEE, N. CHANDRAN, D. GUPTA, A. RASTOGI, R. SHARMA. **"CrypTFlow: Secure TensorFlow Inference"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2020, pp. 336–353.

[KS08a] V. KOLESNIKOV, T. SCHNEIDER. **"A Practical Universal Circuit Construction and Secure Evaluation of Private Functions"**. In: *Financial Cryptography and Data Security (FC)*. Vol. 5143. LNCS. Springer, 2008, pp. 83–97.

[KS08b] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Applications"**. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 5126. LNCS. Springer, 2008, pp. 486–498.

[KS16] Á. KISS, T. SCHNEIDER. **"Valiant's Universal Circuit Is Practical"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Vol. 9665. LNCS. Springer, 2016, pp. 699–728.

[KSS13] M. KELLER, P. SCHOLL, N. P. SMART. **"An Architecture for Practical Actively Secure MPC with Dishonest Majority"**. In: *Computer and Communications Security (CCS)*. ACM, 2013, pp. 549–560.

[LBY14] D. LIU, E. BERTINO, X. YI. **"Privacy of Outsourced K-means Clustering"**. In: *Asia Conference on Computer and Communications Security (ASIACCS)*. ACM, 2014, pp. 123–134.

[LHR23] R. LAMSAL, A. HARWOOD, M. R. READ. **"Socially Enhanced Situation Awareness from Microblogs Using Artificial Intelligence: A Survey"**. In: *ACM Computing Surveys* 55.4 (2023), 77:1–77:38.

[Lin20] Y. LINDELL. **"Secure Multiparty Computation"**. In: *Communications of the ACM* 64.1 (2020), pp. 86–96.

[LJLA17] J. LIU, M. JUUTI, Y. LU, N. ASOKAN. **"Oblivious Neural Network Predictions via MiniONN Transformations"**. In: *Computer and Communications Security (CCS)*. ACM, 2017, pp. 619–631.

[LLX+21] X. LIU, H. LI, G. XU, Z. CHEN, X. HUANG, R. LU. **"Privacy-Enhanced Federated Learning Against Poisoning Adversaries"**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2021), pp. 1407–1409.

[LMSP21] R. LEHMKUHL, P. MISHRA, A. SRINIVASAN, R. A. POPA. **"MUSE: Secure Inference Resilient to Malicious Clients"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2021, pp. 2201–2218.

[LWB+21] M. LAM, G. WEI, D. BROOKS, V. J. REDDI, M. MITZENMACHER. **"Gradient Disaggregation: Breaking Privacy in Federated Learning by Reconstructing the User Participant Matrix"**. In: *International Conference on Machine Learning (ICML)*. Vol. 139. PMLR, 2021, pp. 5959–5968.

[LWN+15] C. LIU, X. S. WANG, K. NAYAK, Y. HUANG, E. SHI. **"ObliVM: A Programming Framework for Secure Computation"**. In: *IEEE S&P*. 2015.

[LXC+19] L. LI, W. XU, T. CHEN, G. B. GIANNAKIS, Q. LING. **"RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets"**. In: *AAAI*. 2019.

[LXLH13] J. LIU, L. XIONG, J. LUO, J. Z. HUANG. **"Privacy Preserving Distributed DBSCAN Clustering"**. In: 6.1 (2013), pp. 69–85.

[LYF+23] P. LIU, W. YUAN, J. FU, Z. JIANG, H. HAYASHI, G. NEUBIG. **"Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing"**. In: *ACM Computing Surveys* 55.9 (2023), 195:1–195:35.

[LYZ+21] H. LIU, Y. YU, S. ZHAO, J. ZHANG, W. LIU, Z. HU. **"Pushing the Limits of Valiant's Universal Circuits: Simpler, Tighter and More Compact"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 12826. LNCS. Springer, 2021, pp. 365–394.

[Mis92] A. MISHCHENKO. **"Berkeley Logic Interchange Format (BLIF)"**. Tech. rep. 1992.

[MLS+20] P. MISHRA, R. LEHMKUHL, A. SRINIVASAN, W. ZHENG, R. A. POPA. **"DELPHI: A Cryptographic Inference Service for Neural Networks"**. In: *USENIX Security Symposium (USENIX Security)*. USENIX Association, 2020, pp. 2505–2522.

[MMM+22] Z. MA, J. MA, Y. MIAO, Y. LI, R. H. DENG. **"ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning"**. In: *IEEE TIFS* (2022).

[MMR+17] B. MCMAHAN, E. MOORE, D. RAMAGE, S. HAMPSON, B. A. y. ARCAS. **"Communication-Efficient Learning of Deep Networks from Decentralized Data"**. In: *Artificial Intelligence and Statistics (AISTATS)*. Vol. 54. PMLR, 2017, pp. 1273–1282.

[MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. **"Fairplay - Secure Two-Party Computation System"**. In: *USENIX Security*. 2004.

[MÖBC23] M. MANSOURI, M. ÖNEN, W. BEN JABALLAH, M. CONTI. **"SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning"**. In: *Proceedings on Privacy Enhancing Technologies (PETS)* 2023.1 (2023), pp. 140–157.

[MPOT21] X. MENG, D. PAPADOPOULOS, A. OPREA, N. TRIANDOPOULOS. **"Private Hierarchical Clustering and Efficient Approximation"**. In: *Computing Security Workshop (CCSW@CCS)*. ACM, 2021, pp. 3–20.

[MR18] P. MOHASSEL, P. RINDAL. **"ABY^3: A Mixed Protocol Framework for Machine Learning"**. In: *Computer and Communications Security (CCS)*. ACM, 2018, pp. 35–52.

[MRT20] P. MOHASSEL, M. ROSULEK, N. TRIEU. **"Practical Privacy-Preserving K-Means Clustering"**. In: *Proc. Priv. Enhancing Technol. (PETS)* 2020.4 (2020), pp. 414–433.

[MRZ15] P. MOHASSEL, M. ROSULEK, Y. ZHANG. **"Fast and Secure Three-party Computation: Garbled Circuit Approach"**. In: *Computer and Communications Security (CCS)*. ACM, 2015, pp. 591–602.

[MS99] F. MASULLI, A. SCHENONE. **"A Fuzzy Clustering based Segmentation System as Support to Diagnosis in Medical Imaging"**. In: *Artificial Intelligence in Medicine* 16.2 (1999), pp. 129–147.

[MSCS19] L. MELIS, C. SONG, E. D. CRISTOFARO, V. SHMATIKOV. **"Exploiting Unintended Feature Leakage in Collaborative Learning"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2019, pp. 691–706.

[MSS+24] H. MANTEL, J. SCHMIDT, T. SCHNEIDER, M. STILLGER, T. WEISSMANTEL, H. YALAME. **"HyCaMi: High-Level Synthesis for Cache Side Mitigation"**. In: *Design Automation Conference (DAC)*. Online: https://ia.cr/2024/533. ACM, 2024. CORE Rank **A**.

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, pp. 131–145. CORE Rank **A**. Appendix D.

[MZ17] P. MOHASSEL, Y. ZHANG. **"SecureML: A System for Scalable Privacy-Preserving Machine Learning"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2017, pp. 19–38.

[NC23] L. K. NG, S. S. CHOW. **"SoK: Cryptographic Neural-Network Computation"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2023, pp. 497–514.

[NPS99] M. NAOR, B. PINKAS, R. SUMNER. **"Privacy Preserving Auctions and Mechanism Design"**. In: *ACM conference on Electronic commerce*. 1999.

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2021/025. USENIX Association, 2022, pp. 1415–1432. CORE Rank **A***. Appendix B.

[NSH19] M. NASR, R. SHOKRI, A. HOUMANSADR. **"Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2019, pp. 739–753.

[ON20] S. OHATA, K. NUIDA. **"Communication-Efficient (Client-Aided) Secure Two-Party Protocols and Its Application"**. In: *FC*. 2020.

[PAP22] E. POHLE, A. A. AISHAJIANG, B. PRENEEL. **"Poster: Fast Evaluation of S-boxes in MPC"**. In: *NDSS*. 2022.

[PFA22] D. PASQUINI, D. FRANCATI, G. ATENIESE. **"Eluding Secure Aggregation in Federated Learning via Model Inconsistency"**. In: *Computer and Communications Security (CCS)*. 2022.

[PGJ12] S. PATEL, S. GARASIA, D. JINWALA. **"An Efficient Approach for Privacy Preserving Distributed *K*-Means Clustering Based on Shamir's Secret Sharing Scheme"**. In: *Trust Management VI*. Vol. 374. Springer, 2012, pp. 129–141.

[PS20] A. PATRA, A. SURESH. **"BLAZE: Blazing Fast Privacy-Preserving Machine Learning"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020.

[PSSW09] B. PINKAS, T. SCHNEIDER, N. P. SMART, S. C. WILLIAMS. **"Secure Two-Party Computation Is Practical"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 5912. LNCS. Springer, 2009, pp. 250–267.

[PSSY21a] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, pp. 2165–2182. CORE Rank **A***. Appendix E.

[PSSY21b] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation with Applications to Privacy Preserving Machine Learning (Extended Abstract)"**. Privacy-Preserving Machine Learning Workshop (PPML@CRYPTO). 2021.

[PSSY21c] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"CONTRIBUTED TALK: ABY2.0: New Efficient Primitives for STPC with Applications to Privacy in Machine Learning (Extended Abstract)"**. Privacy in Machine Learning Workshop (PriML@NeurIPS). 2021.

[PSSY21d] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"POSTER: ABY2.0: New Efficient Primitives for 2PC with Applications to Privacy Preserving Machine Learning (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@CCS). 2021.

[PSSY22] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation"**. In: *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*. Online: https://ia.cr/2021/1153. IEEE, 2022, pp. 147–157.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-Based PSI with Linear Communication"**. In: *EUROCRYPT*. 2019.

[PZM+24] Q. PANG, J. ZHU, H. MÖLLERING, W. ZHENG, T. SCHNEIDER. **"BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers"**. In: *IEEE S&P*. 2024.

[RBK17] M. S. RAHMAN, A. BASU, S. KIYOMOTO. **"Towards Outsourced Privacy-Preserving Multiparty DBSCAN"**. In: *IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2017, pp. 225–226.

[RBS+22] D. RATHEE, A. BHATTACHARYA, R. SHARMA, D. GUPTA, N. CHANDRAN, A. RASTOGI. **"SecFloat: Accurate Floating-Point meets Secure 2-Party Computation"**. In: *IEEE S&P*. 2022.

[RJHK19] M. S. RIAZI, M. JAVAHERIPI, S. U. HUSSAIN, F. KOUSHANFAR. **"MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation"**. In: *HOST*. 2019.

[RNMS22] P. RIEGER, T. D. NGUYEN, M. MIETTINEN, A.-R. SADEGHI. **"Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection"**. In: *NDSS*. 2022.

[RR21] M. ROSULEK, L. ROY. **"Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits"**. In: *Advances in Cryptology (CRYPTO)*. Vol. 12825. LNCS. Springer, 2021, pp. 94–124.

[RRG+21] D. RATHEE, M. RATHEE, R. K. K. GOLI, D. GUPTA, R. SHARMA, N. CHANDRAN, A. RASTOGI. **"SiRnn: A Math Library for Secure RNN Inference"**. In: *IEEE S&P*. 2021.

[RRK+20a] D. RATHEE, M. RATHEE, N. KUMAR, N. CHANDRAN, D. GUPTA, A. RASTOGI, R. SHARMA. **"CrypTFlow2: Practical 2-Party Secure Inference"**. In: *Computer and Communications Security (CCS)*. ACM, 2020, pp. 325–342.

[RRK+20b] D. RATHEE, M. RATHEE, N. KUMAR, N. CHANDRAN, D. GUPTA, A. RASTOGI, R. SHARMA. **"CrypTFlow2: Practical 2-party Secure Inference"**. In: *Computer and Communications Security (CCS)*. ACM, 2020, pp. 325–342.

[RS20] R. RACHURI, A. SURESH. **"Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020.

[RSC+19] M. S. RIAZI, M. SAMRAGH, H. CHEN, K. LAINE, K. E. LAUTER, F. KOUSHANFAR. **"XONN: XNOR-based Oblivious Deep Neural Network Inference"**. In: *USENIX Security*. 2019.

[RST+22] D. ROTARU, N. P. SMART, T. TANGUY, F. VERCAUTEREN, T. WOOD. **"Actively Secure Setup for SPDZ"**. In: *JoC* 35.1 (2022), p. 5.

[SH21] V. SHEJWALKAR, A. HOUMANSADR. **"Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning"**. In: *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2021.

[SHKR22a] V. SHEJWALKAR, A. HOUMANSADR, P. KAIROUZ, D. RAMAGE. **"Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. IEEE, 2022, pp. 1354–1371.

[SHKR22b] V. SHEJWALKAR, A. HOUMANSADR, P. KAIROUZ, D. RAMAGE. **"Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning"**. In: *IEEE S&P*. 2022.

[SHS+15] E. M. SONGHORI, S. U. HUSSAIN, A.-R. SADEGHI, T. SCHNEIDER, F. KOUSHANFAR. **"TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits"**. In: *IEEE S&P*. 2015.

[SM17] M. SHEIKHALISHAHI, F. MARTINELLI. **"Privacy Preserving Clustering over Horizontal and Vertical Partitioned Data"**. In: *IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 1237–1244.

[SMO07] S. SAMET, A. MIRI, L. OROZCO-BARBOSA. **"Privacy Preserving K-means Clustering in Multi-Party Environment"**. In: *Security and Cryptography (SECRYPT)*. INSTICC Press, 2007, pp. 381–385.

[SPJ19] K. SHAMSI, D. Z. PAN, Y. JIN. **"On the Impossibility of Approximation-Resilient Circuit Locking"**. In: *HOST*. 2019.

[SS08] A. SADEGHI, T. SCHNEIDER. **"Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification"**. In: *ICISC*. Springer, 2008, pp. 336–353.

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), pp. 1407–1409. CORE Rank **A**. Appendix C.

[SWW+22] L. SONG, J. WANG, Z. WANG, X. TU, G. LIN, W. RUAN, H. WU, W. HAN. **"pMPL: A Robust Multi-Party Learning Framework with a Privileged Party"**. In: *Computer and Communications Security (CCS)*. ACM, 2022, pp. 2689–2703.

[SZ13] T. SCHNEIDER, M. ZOHNER. **"GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits"**. In: *Financial Cryptography and Data Security (FC)*. Springer, 2013, pp. 275–292.

[TCBK20] H. C. TANUWIDJAJA, R. CHOI, S. BAEK, K. KIM. **"Privacy-Preserving Deep Learning on Machine Learning as a Service—a Comprehensive Survey"**. In: 8 (2020), pp. 167425–167447.

[Val76] L. G. VALIANT. **"Universal Circuits (Preliminary Report)"**. In: *Symposium on Theory of Computing (STOC)*. ACM, 1976, pp. 196–203.

[VC03] J. VAIDYA, C. CLIFTON. **"Privacy-Preserving K-Means Clustering over Vertically Partitioned Data"**. In: *Knowledge Discovery and Data Mining (KDD)*. ACM, 2003, pp. 206–215.

[WDCC20] E. WANG, J. J. DAVIS, P. Y. K. CHEUNG, G. A. CONSTANTINIDES. **"LUTNet: Learning FPGA Configurations for Highly Efficient Neural Network Inference"**. In: *IEEE Trans. Computers* (2020).

[WGF+22] Y. WEN, J. GEIPING, L. FOWL, M. GOLDBLUM, T. GOLDSTEIN. **"Fishing for User Data in Large-Batch Federated Learning via Gradient Magnification"**. In: *ICML*. 2022.

[WGK13] C. WOLF, J. GLASER, J. KEPLER. **"Yosys – A Free Verilog Synthesis Suite"**. In: *Austrian Workshop on Microelectronics (Austrochip)*. 2013.

[WLW+20] W. WU, J. LIU, H. WANG, J. HAO, M. XIAN. **"Secure and Efficient Outsourced K-Means Clustering using Fully Homomorphic Encryption with Ciphertext Packing Technique"**. In: *IEEE Transactions on Knowledge and Data Engineering* 33.10 (2020), pp. 3424–3437.

[WRK17a] X. WANG, S. RANELLUCCI, J. KATZ. **"Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation"**. In: *Computer and Communications Security (CCS)*. 2017.

[WRK17b] X. WANG, S. RANELLUCCI, J. KATZ. **"Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2017, pp. 21–37.

[WRK17c] X. WANG, S. RANELLUCCI, J. KATZ. **"Global-Scale Secure Multiparty Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2017, pp. 39–56.

[WZL24] J. WU, W. ZHANG, F. LUO. **"On the Security of "LSFL: A Lightweight and Secure Federated Learning Scheme for Edge Computing"**. In: *IEEE TIFS* 19 (2024), pp. 3481–3482.

[XHL+07] W.-j. XU, L.-s. HUANG, Y.-l. LUO, Y.-f. YAO, W. JING. **"Protocols for Privacy-Preserving DBSCAN Clustering"**. In: *International Journal of Security and Its Applications*. Vol. 1. 1. 2007.

[XHZ+23] G. XU, X. HAN, T. ZHANG, H. LI, R. H. DENG. **"SIMC 2.0: Improved Secure ML Inference Against Malicious Clients"**. In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* (2023).

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets (Extended Abstract)"**. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1986, pp. 162–167.

[YCRB18] D. YIN, Y. CHEN, K. RAMCHANDRAN, P. L. BARTLETT. **"Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates"**. In: *International Conference on Machine Learning (ICML)*. Vol. 80. PMLR, 2018, pp. 5636–5645.

[YT19] J. YUAN, Y. TIAN. **"Practical Privacy-Preserving MapReduce Based K-Means Clustering Over Large-Scale Dataset"**. In: *IEEE Transactions on Cloud Computing*. Vol. 7. 2. 2019, pp. 568–579.

[YWZ20] K. YANG, X. WANG, J. ZHANG. **"More Efficient MPC from Improved Triple Generation and Authenticated Garbling"**. In: *Computer and Communications Security (CCS)*. ACM, 2020, pp. 1627–1646.

[ZCJG22] Z. ZHANG, X. CAO, J. JIA, N. Z. GONG. **"FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients"**. In: *KDD*. 2022.

[ZPS+22] Z. ZHANG, A. PANDA, L. SONG, Y. YANG, M. W. MAHONEY, P. MITTAL, K. RAMCHANDRAN, J. GONZALEZ. **"Neurotoxin: Durable Backdoors in Federated Learning"**. In: *International Conference on Machine Learning (ICML)*. Vol. 162. PMLR, 2022, pp. 26429–26446.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates"**. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Vol. 9057. LNCS. Springer, 2015, pp. 220–250.

[ZWM+22] Z. ZHANG, L. WU, C. MA, J. LI, J. WANG, Q. WANG, S. YU. **"LSFL: A Lightweight and Secure Federated Learning Scheme for Edge Computing"**. In: *IEEE TIFS* (2022).

[ZYZL19] S. ZHAO, Y. YU, J. ZHANG, H. LIU. **"Valiant's Universal Circuits Revisited: An Overall Improvement and a Lower Bound"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 11921. LNCS. Springer, 2019, pp. 401–425.

List of Figures

List of Tables

List of Abbreviations

ML	machine learning
MPC	secure multi-party computation
HE	homomorphic encryption
2PC	secure two-party computation
PEFL	privacy-enhanced federated learning
GC	garbled circuit
A-GMW	arithmetic Goldreich-Micali-Wigderson
B-GMW	boolean Goldreich-Micali-Wigderson
HDL	hardware description language
SS	secret sharing
OT	oblivious transfer
AES	advanced encryption standard
AES-NI	AES instruction set
VAES	vector AES
PPML	privacy preserving machine learning
SIMD	single instruction multiple data
LUT	look-up table
PPA	parallel prefix adder
LAN	local area network
WAN	wide area network
UC	universal circuit
PFE	private function evaluation
IP	intellectual property
FHE	fully homomorphic encryption
FPGA	field-programmable gate array
OT	oblivious transfer
3PC	three-party computation
2PC	two-party computation
MLaaS	machine learning as a service
GMW	Goldreich-Micali-Wigderson
PETs	privacy-enhancing technologies
ZK	zero knowledge
DP	differential privacy
SA	secure aggregation
HE	homomorphic encryption

RTT	round trip time
FL	federated learning

List of Own Publications

Peer-reviewed Publications

[BMP+24] Y. Ben-Itzhak, H. Möllering, B. Pinkas, T. Schneider, A. Suresh, O. Tkachenko, S. Vargaftik, C. Weinert, H. Yalame, A. Yanai. **"ScionFL: Secure Quantized Aggregation for Federated Learning"**. In: *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. Online: https://ia.cr/2023/652. Code: https://encrypto.de/code/ScionFL. IEEE, 2024.

[BCD+20a] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, H. Yalame. **"MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference"**. In: *International Conference on Availability, Reliability and Security (ARES)*. Online: https://ia.cr/2020/721. Code: https://github.com/IntelAI/he-transformer. ACM, 2020, 14:1–14:10. CORE Rank B.

[BCD+20b] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, H. Yalame. **"MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference (Extended Abstract)"**. Privacy-Preserving Machine Learning Workshop (PPML@CRYPTO). 2020.

[BCD+20c] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, H. Yalame. **"MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference (Extended Abstract)"**. In: *Privacy-Preserving Machine Learning in Practice Workshop (PPMLP@CCS)*. ACM, 2020, pp. 43–45.

[BCD+20d] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, H. Yalame. **"POSTER: MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@NeurIPS). 2020.

[BHS+23] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, H. Yalame. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, pp. 515–533. CORE Rank **A***. Appendix F.

[BSS+24] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, H. Yalame. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

[BSSY22] A. BRÜGGEMANN, T. SCHNEIDER, A. SURESH, H. YALAME. **"POSTER: Efficient Three-Party Shuffling Using Precomputation"**. In: *ACM Conference on Computer and Communications Security (CCS) Posters/Demos*. ACM, 2022, pp. 3331–3333.

[DGS+23] Y. DISSER, D. GÜNTHER, T. SCHNEIDER, M. STILLGER, A. WIGANDT, H. YALAME. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 14438. LNCS. Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/LUC. Springer, 2023, pp. 3–37. CORE Rank **A**. Appendix G.

[DDK+24] V. DUDDU, A. DAS, N. KHAYATA, H. YALAME, T. SCHNEIDER, N. ASOKAN. **"Attesting Distributional Properties of Training Data for Machine Learning"**. In: *European Symposium on Research in Computer Security (ESORICS)*. Online: https://arxiv.org/abs/2308.09552. Springer, 2024. CORE Rank **A**.

[FMM+21] H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, H. MÖLLERING, T. D. NGUYEN, P. RIEGER, A.-R. SADEGHI, T. SCHNEIDER, H. YALAME, S. ZEITOUNI. **"SAFELearn: Secure Aggregation for private FEderated Learning"**. In: *Deep Learning and Security Workshop (DLS)*. Online: https://ia.cr/2021/386. IEEE, 2021, pp. 56–62.

[GMS+23] T. GEHLHAR, F. MARX, T. SCHNEIDER, A. SURESH, T. WEHRLE, H. YALAME. **"SafeFL: MPC-friendly framework for Private and Robust Federated Learning"**. In: *Deep Learning Security and Privacy Workshop (DLSP)*. Online: https://ia.cr/2023/555, Code: https://encrypto.de/code/SAFEFL. IEEE, 2023, pp. 69–76.

[GSSY24] D. GÜNTHER, J. SCHMIDT, T. SCHNEIDER, H. YALAME. **"FLUENT: A Tool for Efficient Mixed-Protocol Semi-Private Function Evaluation"**. In: *Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2024. CORE Rank **A**.

[HMSY21a] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"CONTRIBUTED TALK: SoK: Privacy-Preserving Clustering (Extended Abstract)"**. Privacy in Machine Learning Workshop (PriML@NeurIPS). 2021.

[HMSY21b] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"POSTER: SoK: Privacy-preserving Clustering (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@CCS). 2021.

[HMSY21c] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, pp. 225–248. CORE Rank **A**. Appendix A.

[HST+21] T. HELDMANN, T. SCHNEIDER, O. TKACHENKO, C. WEINERT, H. YALAME. **"LLVM-based Circuit Compilation for Practical Secure Computation"**. In: *Applied Cryptography and Network Security (ACNS)*. Online: https://ia.cr/2021/797. Code: https://encrypto.de/code/LLVM. Springer, 2021, pp. 99–121. CORE Rank B.

[KMSY21a] H. KELLER, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"Balancing Quality and Efficiency in Private Clustering with Affinity Propagation"**. In: *International Conference on Security and Cryptography (SECRYPT)*. Online: https://ia.cr/2021/825. Code: https://encrypto.de/code/ppAffinityPropagation. SciTePress, 2021, pp. 173–184. CORE Rank C.

[KMSY21b] H. KELLER, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"POSTER: Balancing Quality and Efficiency in Private Clustering with Affinity Propagation (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@CCS). 2021.

[KMSY21c] H. KELLER, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"Privacy-Preserving Clustering (Abstract)"**. In: *32. Kryptotag (crypto day matters)*. Gesellschaft für Informatik e.V. / FG KRYPTO, 2021.

[MSS+24] H. MANTEL, J. SCHMIDT, T. SCHNEIDER, M. STILLGER, T. WEISSMANTEL, H. YALAME. **"HyCaMi: High-Level Synthesis for Cache Side Mitigation"**. In: *Design Automation Conference (DAC)*. Online: https://ia.cr/2024/533. ACM, 2024. CORE Rank **A**.

[MSS+23] F. MARX, T. SCHNEIDER, A. SURESH, T. WEHRLE, C. WEINERT, H. YALAME. **"WW-FL: Secure and Private Large-Scale Federated Learning"**. arXiv:2302.09904. https://arxiv.org/abs/2302.09904. 2023.

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, pp. 131–145. CORE Rank **A**. Appendix D.

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2021/025. USENIX Association, 2022, pp. 1415–1432. CORE Rank **A***. Appendix B.

[PSSY21a] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, pp. 2165–2182. CORE Rank **A***. Appendix E.

[PSSY21b] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation with Applications to Privacy Preserving Machine Learning (Extended Abstract)"**. Privacy-Preserving Machine Learning Workshop (PPML@CRYPTO). 2021.

[PSSY21c] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"CONTRIBUTED TALK: ABY2.0: New Efficient Primitives for STPC with Applications to Privacy in Machine Learning (Extended Abstract)"**. Privacy in Machine Learning Workshop (PriML@NeurIPS). 2021.

[PSSY21d] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"POSTER: ABY2.0: New Efficient Primitives for 2PC with Applications to Privacy Preserving Machine Learning (Extended Abstract)"**. Privacy Preserving Machine Learning Workshop (PPML@CCS). 2021.

[PSSY22] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation"**. In: *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*. Online: https://ia.cr/2021/1153. IEEE, 2022, pp. 147–157.

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), pp. 1407–1409. CORE Rank **A**. Appendix C.

[SYY23a] T. SCHNEIDER, H. YALAME, M. YONLI. **"Griffin: Towards Mixed Multi-Key Homomorphic Encryption"**. In: *International Conference on Security and Cryptography (SECRYPT)*. Online: https://ia.cr/2023/654. SciTePress, 2023, pp. 147–158. CORE Rank C.

[SYY23b] T. SCHNEIDER, H. YALAME, M. YONLI. **"POSTER: Towards Mixed Multi-Key Homomorphic Encryption"**. Annual FHE.org Conference on Fully Homomorphic Encryption (FHE.org) Poster Session. 2023.

In Submission

[MSS+23] F. MARX, T. SCHNEIDER, A. SURESH, T. WEHRLE, C. WEINERT, H. YALAME. **"WW-FL: Secure and Private Large-Scale Federated Learning"**. arXiv:2302.09904. https://arxiv.org/abs/2302.09904. 2023.

Appendices

A SoK: Efficient Privacy-Preserving Clustering (PETs'21)

[HMSY21] A. HEGDE, H. MÖLLERING, T. SCHNEIDER, H. YALAME. **"SoK: Efficient Privacy-Preserving Clustering"**. In: *Proceedings on Privacy Enhancing Technologies (PETs)* 2021.4 (2021). Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/SoK_ppClustering, pp. 225–248. CORE Rank **A**. Appendix A.

https://doi.org/10.2478/popets-2021-0068

Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame

SoK: Efficient Privacy-preserving Clustering

Abstract: Clustering is a popular unsupervised machine learning technique that groups similar input elements into clusters. It is used in many areas ranging from business analysis to health care. In many of these applications, sensitive information is clustered that should not be leaked. Moreover, nowadays it is often required to combine data from multiple sources to increase the quality of the analysis as well as to outsource complex computation to powerful cloud servers. This calls for efficient privacy-preserving clustering. In this work, we systematically analyze the state-of-the-art in privacy-preserving clustering. We implement and benchmark today's four most efficient fully private clustering protocols by Cheon et al. (SAC'19), Meng et al. (ArXiv'19), Mohassel et al. (PETS'20), and Bozdemir et al. (ASIACCS'21) with respect to communication, computation, and clustering quality. We compare them, assess their limitations for a practical use in real-world applications, and conclude with open challenges.

Keywords: Privacy-preserving Protocols, Clustering, Secure Computation

DOI 10.2478/popets-2021-0068
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

In today's world, machine learning (ML) algorithms are widely used to categorize and classify large amounts of data. Applications range from spam filtering over fraud detection, stock market analysis to health diagnostic [1–4]. Moreover, many large IT companies, including Microsoft, Facebook, Google, and Apple, collect massive amounts of data to perform analyses for their commercial benefit [5]. Clustering is a popular unsupervised learning technique and plays a crucial role in data processing and analysis. It divides a set of given input data into subgroups of elements with similar properties.

Cluster analysis is being utilized in various fields with extremely sensitive data such as medical imaging [4] and market research [6], to name a few. Moreover, data protection regulations such as the General Data Protection Regulation (GDPR) in the EU and the Health Insurance Portability and Accountability Act (HIPAA) in the US prohibit companies from sharing sensitive user information. Nevertheless, combining data from different sources, e.g., different hospitals, broadens the database and offers more meaningful, credible, and high-quality clustering results. Additionally, it is often needed to outsource the expensive clustering of large amounts of data to powerful cloud servers. These requirements emphasize the need for privacy-preserving clustering to preserve the privacy of data.

Consequently, a series of efforts have been made to protect the privacy of sensitive input data in clustering through two paradigms for secure computation that can also be combined. The first paradigm leverages homomorphic encryption (HE) [7–9]. HE allows to directly compute functions on encrypted data. The second paradigm uses secure multi-party computation (MPC) [10, 11]. MPC allows mutually distrusting parties to collaboratively compute a joint function over their respective private data. However, these works only cover a few clustering algorithms so far: K-means, K-medoid, Mean-shift, Gaussian Mixture Models Clustering (GMM), Density-Based Spatial Clustering of Applications with Noise (DBSCAN), hierarchical clustering (HC), Affinity Propagation, and Mean-shift. Moreover, we found that only ten works (cf. Tab. 1) provide full privacy protection according to the ideal functionality for privacy-preserving clustering, i.e., they leak nothing beyond the output (cf. §3.1).

Even revealing little and at the first glance minor information during the clustering can have severe consequences for the data privacy of individuals. For example, when using clustering for the segmentation of medical images [4] between two hospitals, revealing the cluster sizes and assignments in each clustering iteration leaks information about how many patients with similar characteristics are input by the other party even before the clustering stabilizes and a final result is reached. Unintended common characteristics between patients might

Aditya Hegde: IIIT-Bangalore, E-mail: aditya.shridhar@iiitb.org (This work was done when the author was intern at the Technical University of Darmstadt)
Helen Möllering, Thomas Schneider, Hossein Yalame: Technical University of Darmstadt, E-mail: lastname@encrypto.cs.tu-darmstadt.de

Algorithm	Paper	PETs	Scenario	Data	Output	Efficiency
K-means	[12, CCS'07]	HE+ASS	2PC	a	final centroids	✗†
	[13, CIC'15]	HE	Outsourcing, 2 Servers	h	final centroids	✗†
	[14, SAC'18]	HE	Outsourcing, 1 Server	$-$	final centroids	✗⋆
	[15, CLOUD'18]	HE	Outsourcing, 2 Servers	$-$	cluster sizes	✗†
	MPC-KMeans [11, PETS'20]	GC	Outsourcing, 2 Servers *or* 2PC	h	final centroids	✓
Mean-shift	HE-Meanshift [9, SAC'19]	HE	Outsourcing, 1 Server	$-$	final centroids	✓
Affinity Propagation	[16, SECRYPT'21]	ASS	Outsourcing or MPC	a	final clusters	✗‡
DBSCAN	[17, S&P'13]	GC	2PC	h	cluster labels, centroids/size possible	✗¶
	ppDBSCAN [18, ASIACCS'21]	GC+ASS	Outsourcing, 2 Servers *or* 2PC	a	cluster labels, centroids/size possible	✓
Hierachical Clustering	PCA/OPT [19, ArXiv'19]	HE+GC	2PC	h	final dendogram	✓

† Computationally expensive due to use of Paillier's HE and no parallelization.
⋆ Costly computation due to use of bit-wise encryption. MPC-KMeans [11] outperforms this scheme by 5000× for 400 data records.
‡ [18] is 194× faster than this scheme for 400 data records.
¶ [18] is 5× faster than this scheme for a dataset size of 500 data records.

Table 1. Fully privacy-preserving clustering protocols (cf. §3.1). HE is homomorphic encryption [7], ASS is arithmetic secret sharing [20], and GC is garbled circuits [21]. v indicates vertically partitioned data, i.e., the data owners hold the values for a subset of parameters from all data records. h indicates horizontally partitioned data, where the data owners hold complete data records with all parameters, a is arbitrarily partitioned data, and " $-$ " indicates the scheme has only one data owner. Schemes that were implemented and benchmarked in §4 are highlighted in gray.

be leaked even though they are only temporarily assigned to one cluster (due to these characteristics which would not have been revealed in the final result). An even more severe privacy breach is demonstrated in [22] where leaking the results of comparison of distances between data records and a threshold can enable to accurately approximate the original data record held by another party. With this, complete patient records could be extracted when clustering medical data. To summarize, it is difficult to concretely determine the effects of leaking intermediate information in advance for all possible constellations. Hence, privacy research should focus on designing efficient private clustering protocols that do not leak anything beyond what can be inferred from the output, i.e., provide full privacy.

Related Work. Privacy-preserving machine learning (PPML) is a hot topic in recent privacy research [23–26]. To provide a better overview over the exploding research field, several surveys have been done. Haralampieva et al. [27] survey existing frameworks in the context of private image classification. An overview about frameworks for private neural network inference is given in [28]. Protocols used for private machine learning training are investigated in [29]. Similarily, Tanuwidjaja et al. [30] summarize existing works on privacy-preserving deep learning and issues when using these schemes as well as possible attacks on private deep learning. Kiss et al. [31] systematically review the state-of-the-art approaches to private decision tree evaluation.

All previous surveys focus on privacy-preserving supervised learning where a training dataset with labelled samples (i.e., known input-output pairs) is used to train a model that can later be used to classify new data records. In contrast, our survey focuses on clustering, a popular *unsupervised* machine learning (ML) technique, which detects unknown patterns in unlabelled data so no "training" of a model is needed. In our work, we systematically survey and evaluate the state-of-the-art in private clustering using secure computation techniques.

An orthogonal line of research uses differential privacy (DP) to protect privacy-preserving machine learning (PPML), including clustering [32–37], against information leakage. Abadi et al. [38] and Shokri et al. [39] provide comprehensive surveys on differentially private deep learning. Generally, the noise added to achieve DP reduces utility whereas secure computation has higher complexity. Hence, DP-based and secure computation-based protocols are not directly comparable and we leave a survey on DP-based clustering for future work.

Our Contributions and Outline. After presenting the preliminaries of privacy-preserving clustering in §2, our Systematization of Knowledge (SoK) paper provides the following core contributions:

– The first comprehensive review and analysis of existing techniques and protocols used for privacy-preserving clustering with respect to security models, privacy limitations, efficiency, and further aspects. We also provide guidelines on how to choose an appropriate privacy-preserving clustering scheme for a specific application (§3).

– An empirical evaluation of the four most efficient and fully private clustering schemes [9, 11, 18, 19], cf. Tab. 1, on a range of criteria, including clustering quality, security and privacy, and runtime/communication overhead (§4). Based on these insights, we provide an analysis of the practicality of the four protocols for real-world

applications based on our results from the benchmarking (§5).

– An implementation of the clustering protocol of [9] and [19] in C++17. Implementations of the remaining two protocols that we also evaluate [11, 18] are publicly available. Our code is available at https://encrypto.de/code/SoK_ppClustering.

2 Preliminaries

2.1 Clustering

Clustering is a well-known unsupervised machine learning (ML) technique, i.e., it deals with detecting unknown patterns in unlabeled data. Concretely, it groups similar input records (*internal homogeneity*) in *clusters* while records belonging to different clusters should be maximally different (*external separation*) [40–42].

Clustering consists of four components: feature selection/normalization, a proximity measure to determine similarity/dissimilarity, the clustering algorithm, and the output assessment [41, 42]. However, most prior works on privacy-preserving clustering mainly focus on a specific clustering algorithm. For example, the proximity measure is typically chosen to enable efficient computation using cryptographic techniques [14, 19]. Furthermore, mostly continuous values are considered while clustering can generally be applied to any kind of variable (i.e., also discrete or nominal values) [42].

Clustering algorithms can be split in two classes: hard and soft (fuzzy) clustering. In hard clustering, each input data record is assigned to exactly one cluster. In soft clustering, data records can be assigned to several clusters with a certain probability. All works on privacy-preserving clustering that we investigated in this work except from [43] have only tackled hard clustering.

Properties of Good Clustering. Records are assigned to the same cluster given they are *similar*. However, (dis)similarity heavily depends on the chosen proximity measure. Additionally, clustering algorithms were designed having specific problems in mind such that they exhibit biases that affect their performance when the assumed conditions are not fulfilled. Therefore, according to Xu and Wunsch [41], no clustering algorithm is universally superior and a good clustering algorithm should be able to cope with: 1) arbitrarily shaped clusters, 2) large datasets, 3) updates with new records without having to cluster old records again, 4) numerical (i.e., discrete and continuous) and nominal variables, and 5) outliers. Furthermore, it should: 6) be insensitive to the order of input records, 7) provide acceptable storage requirements, and 8) minimize the number of input parameters. Finally, it should also be able to handle 9) high-dimensional data records.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
1) Cluster Shapes	−	−	○	+	+	−	−	−
2) Large Datasets	○	−	−	−	−	−	+	○
3) Update Input Data	+	−	−	○	+	+	+	+
4) Nominal Variables	−	+	+	−	+	+	−	−
5) Outliers	−	+	○	−	+	○	+	○
6) Input Order	+	+	+	+	○	+	−	+
7) Storage	+	−	−	+	+	−	+	+
8) # Parameters	−	○	−	○	○	−	○	−
Full privacy	✓	✓	✓	✓	✓	✗	✗	✗

Table 2. Comparison of clustering algorithms with respect to the aspects explained in §2.1: (a) K-means, (b) Affinity Propagation, (c) Single/Complete Linkage HC, (d) Mean-shift, (e) DBSCAN, (f) K-medoid, (g) BIRCH, and (h) GMM. + denotes that the clustering algorithm performs well with respect to the indicated aspect, ○ denotes an average performance, and − indicates that it has some weaknesses. ✓ indicates that a fully privacy-preserving clustering protocol is available and ✗ that it is not available yet.

Clustering Algorithms

In the context of privacy-preserving clustering, four different types of clustering have been studied so far: partitioning-based [8, 11, 14, 16], distribution-based [44, 45], density-based [18, 46, 47], and hierarchical clustering [19, 48–50]. In the following, we summarize these four clustering types and compare the respective algorithms w.r.t. the properties listed before in §2.1. Due to space limitations, we only provide the details of this evaluation for the three algorithms [51–53] for which fully private protocols were proposed (cf. §3.2) and that we benchmark in §4. Details of the other algorithms are given in Appx. A.

Partitioning-based Clustering. Partitioning-based clustering splits the input into K non-overlapping clusters. Typically, an initial random partition is iteratively improved given an objective function [54].

A well-known example is K-means [51]. It has a computational complexity of $\mathcal{O}(NKt)$ and a space complexity of $O(N)$ [40] for dataset size N, K clusters, and t clustering iterations. Furthermore, K-means can only cluster convexly-shaped clusters, cannot to appropriately handle outliers, and requires to pre-determine the number of clusters K [40]. If the initial partitioning,

i.e., the centroid initialization, is done at random, K-means is not deterministic. It may converge to a local optimum [55]. The input order does not affect the clustering result. As the centroids are determined by averaging, K-means is not suitable for nominal variables [56]. New data records typically require only a few additional clustering iterations because they normally do not significantly change the result. Other partitioning-based clustering algorithms that were investigated in the context of privacy-preserving clustering are the closely related K-medoids [57], Kernel K-means [58], Possibilistic C-means [43], as well as Affinity Propagation [16].

Hierarchical Clustering. Hierarchical Clustering (HC) algorithms can be classified into agglomerative and divisive approaches. In agglomerative algorithms, each data record forms an own cluster in the beginning and the clusters are then iteratively merged together based on their proximity. Divisive algorithms follow the opposite approach and start with all elements in one cluster which is then iteratively split up [52, p. 71-72]. HC algorithms output a binary tree/dendrogram[1] where each leaf represents a record and nodes indicate a merge of two similar clusters into one. The root combines all records into a single cluster [40].

As divisive HC exhibits an immense overhead for examining the optimal splits ($2^{N-1}-1$ possibilities [40], where N is the dataset size), mostly agglomerative algorithms have been observed in practice. Traditionally, three merging methods were used: (1) single, (2) complete, and (3) average linkage. Single linkage merges the two clusters with the closest two elements, complete linkage merges the two clusters whose maximally distant pair of elements are closest among all pairs of clusters, and average linkage merges the two clusters that have the smallest average of all pairwise distances of their elements [52, p. 76-77] [59].

Naive HC has computation complexity $\mathcal{O}(N^3)$ and space complexity $\mathcal{O}(N^2)$ [42]. Some HC-based algorithms (e.g., single linkage) cannot detect some cluster shapes. They do not incorporate a notion of noise, but are relatively insensentive to outliers. HC requires to pre-determine the number of clusters K that are obtained by cutting the tree at the respective level [40]. HC needs a restructuring of the tree if new data records are added after the first clustering. Nevertheless, HC can handle any type of variable and the input order does not affect the result.

1 A dendogram is a graph representing a tree structure.

Density-based Clustering. These algorithms use a density-based neighborhood notion such that input records that lay together in a dense area form a cluster. Examples are Mean-shift [53] and DBSCAN [60]. Mean-shift has time complexity $\mathcal{O}(N^2t)$, where N is the dataset size and t is the number of iterations, which makes it inefficient for large datasets. It can handle any cluster shape and flexibly determine the number of clusters K based on the input data. Additionally, the input order does not affect the results. However, the value of the bandwidth h in the Kernel Density Estimator (KDE) used in Mean-shift can significantly affect its performance. A too large h merges distinct clusters while a too small h splits one cluster into multiple smaller groups. The performance also deteriorates for high dimensional data due to the "curse of dimensionality" in the KDE. Similarly, noisy features can hamper the performance [61]. Mean-shift does not incorporate a notion of noise. An update with new records can change the KDE and the local maximas, thus requiring a rerun of the entire algorithm. However, in practice, the new points can be assigned to the cluster containing the nearest mode if the change in the KDE is not significant.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN [62]) specifically recognizes noisy elements and marks them as outliers. It detects arbitrarily shaped clusters and flexibly determines the number of clusters in a dataset based on two input parameters, namely the minimal cluster size and the maximal distance between two clusters. Especially the second parameter can be difficult to determine and DBSCAN cannot correctly handle clusters with significantly different densities [63]. Generally, if appropriate distance measures are chosen, any type of parameter can be clustered. Moreover, the input order does only affect the clustering result in exceptional cases where border elements lay in the range of more than one cluster. If additional data records shall be clustered after a first clustering was finished, their neighbors have to be determined to assess if they can be added to previously created clusters. Otherwise, they may create a new cluster with other outliers, but no completely new clustering is needed. Naive DBSCAN needs $\mathcal{O}(N)$ memory and has computation complexity $\mathcal{O}(N^2)$ [62, 63].

Distribution-based Clustering. Distribution-based clustering algorithms assume that clusters are drawn from an unknown mixture of distributions and aim at approximating the original distributions (i.e., the type and parameters) as well as the number of different distributions (i.e., the number of clusters) [41, 42]. A well-

known example for distribution-based clustering algorithms are Gaussian Mixture Models (GMM) using the Expectation-Maximization (EM) algorithm [64].

Comparison of Clustering Algorithms. Modifications proposed for the clustering algorithms to fix some weaknesses of the original often introduce other problems. Therefore, it is difficult to evaluate them with respect to the general requirements for clustering algorithms (cf. §2.1).

In Tab. 2, we compare the eight baseline clustering algorithms for which privacy-preserving protocols have been proposed with respect to the properties of good clustering algorithms listed in §2.1. We did not include the effect of property 9), i.e., high dimensionality, because it is often not directly linked to the clustering algorithm. Instead, a large number of variables often requires using feature reduction techniques.

2.2 Cryptographic Building Blocks

In the following, we summarize secure computation techniques and respective security models.

Secure Computation. There are two main paradigms for secure computation: Homomorphic encryption (HE) and multi-party computation (MPC). HE [7, 65, 66] enables operations on a set of ciphertexts such that the resulting ciphertext contains the result of a function on the corresponding plaintexts. MPC allows two or more mutually distrusting parties to jointly compute a function on their private inputs. Two well-known generic approaches for MPC are based on garbled circuits (GC) [21] and secret-sharing (SS) [20, 67]. As an example for a SS-based technique, the GMW protocol [20] represents a function as Boolean/Arithmetic circuit and the values are secret-shared using XOR or Arithmetic secret sharing (ASS). Another type of SS is Shamir's secret sharing (SSS) [67].

Security Models. Two main security models have been considered in privacy-preserving clustering: In the *semi-honest/passive security model*, the adversary [68] is assumed to honestly follow the protocol, but tries to learn additional information about the private inputs of other parties. Though this model is weaker than the *malicious* model, that even protects against deviations from the protocol specification, it facilitates practically-efficient applications especially for privacy-preserving machine learning (PPML) [69]. *Full threshold* security means that up to $N-1$ parties can collude without jeopardising privacy while *honest majority* security requires the majority of the parties to not collude.

3 Privacy-preserving Clustering

In this section, we first define privacy-preserving clustering. Then, we categorize and analyse the existing privacy-preserving clustering protocols to conclude which protocols offer good efficiency with strong privacy guarantees. Afterwards, we discuss possible applications and provide indications on how to choose appropriate privacy-preserving clustering schemes for these.

3.1 Functionality and Requirements

In an ideal world with a trusted third party (TTP), all involved parties send their input data to the TTP. The TTP then performs the clustering and returns the output to the parties. The output can vary depending on the application requirements and clustering algorithm. For example, the output can be the cluster centroids or it can be the cluster label for each data record.

We identified the following requirements for privacy-preserving clustering:

Privacy. According to the *ideal functionality* a privacy-preserving clustering protocol must not leak information other than what can be derived from the output of the protocol to be considered as *fully privacy-preserving*. Importantly, this includes that all operations must be obliviously realized and all intermediate results must be kept private.

Efficiency. A privacy-preserving clustering scheme must be efficient in terms of communication and runtime. This means that it must scale well with respect to the dataset size N, the number of clusters K, and the dimensionality d of the input records.

Clustering Quality: A privacy-preserving clustering scheme must offer a good clustering quality of the results independent of a dataset's properties. Specifically, the requirements of good clustering listed in §2.1 should be fulfilled.

Flexibility. A privacy-preserving clustering scheme should ideally be flexibly usable for *outsourcing* [70] and *multi-party computation*. In an outsourcing scenario, one or multiple data owners outsource their data and the computation to untrusted non-colluding parties [70]. Here, the data owners can even be malicious (cf. §2.2). In multi-party computation, several parties interactively compute the clustering on their joint dataset.

3.2 Existing Private Clustering Protocols

In this subsection, we categorize the existing works on privacy-preserving clustering with respect to the underlying plaintext clustering algorithm, security model, scenarios for which protocols where designed, data distributions, used secure computation techniques as well as privacy and efficiency (cf. §3.1). We discuss the strengths and weaknesses of these schemes with respect to these criteria. Tab. 3 contains on overview of all 59 works on privacy-preserving clustering with secure computation techniques that we are currently aware of. It indicates the respective security model, used secure computation techniques, common types of leakages of intermediate values, the type of output, which and how many parties are involved in the protocol, the data partition, and other issues.

Plaintext Clustering Algorithms. Eight clustering algorithms have been investigated in the context of privacy-preserving clustering: K-means (including the two variants Kernel K-means [58] and Possibilistic C-means [43]), K-medoids [57, 71], GMM [44, 45], Mean-shift [9], DBSCAN [22, 46, 47, 72–76], baseline agglomerative HC (e.g., single linkage or complete linkage) [19, 48–50, 77, 78], BIRCH [79, 80], and Affinity Propagation [16, 81]. The vast majority of works focuses on the simple K-means algorithm [8, 11–15, 82–106], which enables an efficient parallelization of computation through packing with homomorphic encryption [8, 88, 95] or amortization through batched oblivious transfers [11]. However, as discussed in §2.1, K-means can be used only for very specific applications where the number of clusters is known in advance and the clusters are convexly shaped. We gave an overview of the strengths and weaknesses of these plaintext clustering algorithms in Tab. 2. Generally, the choice of the plaintext clustering algorithm heavily affects the quality of the clustering result. Some works on privacy-preserving clustering exactly reproduce the original algorithms and hence achieve the same accuracy, e.g., [19, 44, 45, 82]. Others deviate from the original algorithms such as when updating the centroids in K-means due to, e.g., normalization/quantization/specific encodings of the plaintext space [8, 9, 14, 22, 88, 95], adaptations of the original algorithm [14], or approximations [14, 43] which either enhance efficiency or are needed because of the underlying secure computation techniques.

Security Models. All works except for [16, 96, 105] consider only the semi-honest security model (cf. §2.2). A few even do not explicitly define their security model [43, 48, 57, 58, 71, 100, 108]. The semi-honest security model assumes that the adversary correctly follows the protocol while trying to gain additional information. However, this strong assumption is not always realistic. Concretely, the use of protocols that are secure against semi-honest adversaries is only acceptable in specific applications where the participants already generally trust each other but are legally not allowed to share data, e.g., hospitals conducting medical analysis or central banks for financial analytics on country-level. We discuss the requirements and implications of applications on the choice of a privacy-preserving clustering scheme in more detail in §3.3.

Scenarios. Generally, privacy-preserving clustering protocols have been designed for two scenarios: Firstly, multi-party computation (MPC, [16, 44–46, 50, 57, 71, 72, 74–76, 82, 85, 87, 89, 91, 94, 96, 99–101, 103]) with the special case of two-party computation (2PC, [11–13, 19, 22, 48, 49, 73, 74, 77–80, 83, 84, 86, 93]), where two or more data owners jointly perform a secure computation protocol ideally such that nothing beyond the output is leaked to each other (cf. §3.1). Some of these protocols [50, 75, 76, 88, 95, 100, 103] also involve one or more additional (semi-trusted) entities, e.g., represented by servers, that assist in the computation. In contrast, other protocols were designed for the outsourcing scenario where one or more data owners outsource computation (and storage) to external parties who ideally perform the clustering for them without learning anything about the input data [8, 9, 14, 15, 43, 47, 58, 90, 92, 97, 98, 102, 104–106, 108]. As outsourcing aims at using external resources, data owners should not be involved in the execution of the protocol and can go offline, but this is often not fulfilled, e.g., in [43, 47, 97, 98, 104, 105, 108]. Some MPC/2PC protocols can also be used for an outsourcing scenario where the data owners secret share their data among multiple non-colluding parties who then perform the clustering [8, 11, 105]. However, whether a 2PC/MPC clustering protocol is usable for outsourcing heavily depends on its design. This is hindered if data owners are actively involved by computing on plaintext input data, e.g., [22, 44–46, 48, 49, 72–75, 82, 88], or a data owner needs to perform intermediate decryptions, e.g., [86, 91, 95].

Data Partition. The data to be clustered in a private manner can be partitioned in three ways when provided by multiple parties. It is horizontally partitioned when each data owner holds complete (but different) data records [9, 13, 44, 45, 47, 49, 50, 73, 75, 77, 78, 84–86, 91, 92, 94–96, 98, 99, 101–103, 105, 106, 108]. The data is vertically partitioned when data owners hold mutually different parameters of the same data records [44,

Algorithm	Scheme	Privacy	Security	PETs	L1	L2	L3	L4	O1	O2	O3	Interactivity (Scenario)	Data	Other issues
K-means	[82, KDD'03]	✗	◐	HE+blinding	(✗)[1]	✗	✗	✗	✗	✓	✗	all data owners (≥ 3)	v	
	[83, KDD'05]	✗	◐	HE+ASS+GC	✓	✓	✗	✗	✓	✓	✗	2PC	a	wrong division
	[84, ESORICS'05]	✗	◐	HE *or* OPE	✗	✓	✓	✗	✗	✓	✗	2PC	h	
	[12, CCS'07]	✓	◐	HE+ASS	✓	✓	✓	✗	✗	✓	✗	2PC	a	
	[85, SECRYPT'07]	✗	◐	blinding	✗	✓	✗	✗	✓	✓	✗	all data owners	v/h	
	[86, AINAW'07]	✗	◐	HE+ASS+OPE	✓	✗	✗	✗	✓	✓	✗	2PC	h	
	[87, PAIS'08]	✗	◐	ASS	✓	✓	✗	✗	✓	✓	✗	all data owners (≥ 4)	v	
	[88, WIFS'09]	✗	◐	HE	✗	✓	✗	✓	✓	✗	✗	data owners + 1 server	h	
	[89, KAIS'10]	✗	◐	HE+ASS	✓	✓	✗	✗	✓	✗	✗	all data owners	h	
	[90, PAISI'10]	✗	◐	SS	✓	✗	✗	✗	✓	✓	✗	Outsourcing ≥ 3 servers	a	
	[91, ISPA'10]	✗	◐	HE	✓	✓	✗	✗	✗	✓	✗	all data owners	v/h	
	[92, WIFS'11]	✗	◐	HE+GC	✓	✗	✓	✓	✓	✗	✗	Outsourcing, 3 servers	h	
	[93, ISI'11]	✗	◐	HE+ASS	(✗)[1]	✗	✗	✗	✓	✗	✗	2PC	v	
	[94, TM'12]	✗	◐	SSS	✗	✗	✓	✗	✗	✓	✗	all data owners	h	distance calculation unclear
	[95, JIS'13]	✗	◐	HE	✗	✓	✓	✗	✓	✗	✗	data owners + 2 servers	h	
	[96, ICDCIT'13]	✗	●	SSS+ZKP	✗	✗	✓	✗	✗	✓	✗	all data owners	h	
	[97, ASIACCS'14]	✗	◐	HE	✗	✗	✗	✗	✓	✓	✗	outsourcing, 1 data owner + 1 server	–	insecure HE [107]
	[98, MSN'15]	✗	◐	HE	✗	✗	✗	✓	✗	✗	✗	outsourcing, data owners + 1 server	h	insecure HE [107]
	[99, IJNS'15]	✗	◐	HE	✗	✗	✗	✗	✗	✓	✗	all data owners	h	
	[13, CIC'15]	✓	◐	HE	✓	✓	✓	✗	✗	✓	✗	Outsourcing, 2 servers	h	
	[100, ICACCI'16]	✗	N/A	SS	✗	✗	✗	✗	✓	✗	✗	arbitrary number of servers	a	
	[101, ISPA'16]	✗	◐	blinding	✗	✗	✗	✓	✗	✓	✗	all data owners (≥ 3)	h	
	[102, SecComm'17]	✗	◐	HE	✓	✗	✗	✓	✗	✓	✗	outsourcing, ≥ 4 servers	h	
	[103, TII'17]	✗	◐	HE	✗	✗	✗	✗	✗	✗	✗	data owners + 1 server	h	
	[14, SAC'18]	✓	◐	HE	✓	✓	✓	✓	✗	✓	✗	Outsourcing, 1 server	–	
	[15, CLOUD'18]	✓	◐	HE	✓	✓	✓	✗	✗	✓	✗	Outsourcing, 2 servers	–	distance calculation unclear
	[108, CCPE'19]	✗	N/A	HE	✗	✗	✗	✗	✗	✓	✗	Outsourcing, 2 data owners + 1 server	h	insecure HE [107]
	[104, TCC'19]	✗	◐	HE	✓	✗	✗	✓	✓	✗	✗	Outsourcing, 1 data owner + $\geq$ 1server(s)	–	
	[105, Inf. Sci.'20]	✗	◐ (●)[2]	HE+GC	✗	✗	✗	✗	✗	✓	✗	Outsourcing, 2 data owners + 1 server	h	
	[106, SCN'20]	✗	◐	HE+SKC	✓	✗	✗	✓	✗	✓	✗	Outsourcing, 3 servers	h	
	[11, PETS'20]	✓	◐	GC	✓	✓	✓	✗	✗	✓	✗	2PC/Outsourcing	h	
	[8, TKDE'20]	✗	◐	HE	✓	✗[3]	✓	✗	✗	✓	✗	Outsourcing, 2 servers	a	
Kernel K-means	[58, KAIS'16]	✗	N/A	PKC	✓	✗	✗	✗	✓	✗	✗	Outsourcing, 1 server	–	security model
Possibilistic C-means	[43, TBD'17]	✗	N/A	HE	✗	✗	✗	✗	✓	✓	✗	Outsourcing, 1 data owner + 1 server	–	
K-medoids	[57, SMC'07]	✗	N/A	HE+blinding	✓	✗	✗	✓	✗	✗	✗	all data owners	v	exhaustive search
	[71, CCSEIT'12]	✗	N/A	HE+blinding	✓	✗	✗	✓	✗	✗	✗	all data owners	v	exhaustive search
GMM	[45, KAIS'05]	✗	◐	blinding	✓	✓	✗	✗	✓	✗	✗	all data owners	h	
	[44, DCAI'19]	✗	◐	ASS	✓	✓	✗	✗	✓	✗	✗	all data owners (> 2)	v/h	
Affinity Propagation	[81, INCoS'12]	✗	◐	HE + blinding	✓	✓	✗	✓	✓	✗	✗	all data owners	v	
	[16, SECRYPT'21]	✓	◐/●	ASS+GC	✓	✓	✓	✓	✓	✗	✗	all data owners/Outsourcing	a	
Mean-shift	**[9, SAC'19]**	✓	◐	HE	✓	✓	✓	✓	✓	✗	✗	Outsourcing, 1 server	–	
DBSCAN	[72, ISI'06]	✗	◐	blinding	✓	✓	✗	✓	✗	✗	✗	all data owners	v	lack of complete protocol
	[73, ADMA'07]	✗	◐	HE+blinding	✓	✗	✗	✓	✓	✗	✗	2PC	v/h	
	[74, IJSIA'07]	✗	◐	PKC+blinding	✓	✓	✗	✓	✓	✗	✗	all data owners	v	
	[75, ITME'08]	✗	◐	HE+blinding	✓	✗	✗	✓	✓	✗	✗	data owners + 1 server	h	
	[22, TDP'13]	✗	◐	HE+blinding	✓	✗	✗	✓	✓	✗	✗	2PC	a	
	[17, S&P'12]	✓	◐/●[5]	GC	✓	✓	✓	✓	✓	✓	✗	2PC	h	
	[46, SIBCON'17]	✗	◐	HE+PKC	✓	✓	✗	✓	✓	✗	✗	all data owners	v	cluster expansion missing
	[47, PRDC'17]	✗	◐	HE	✓	✗	✗	✓	✗	✗	✗	outsourcing, all data owners + 1 server	h	
	[76, AI'18]	✗	◐	HE	✓	✗	✗	✓	✓	✗	✗	data owners + 1 server	a	uses absolute distance
	[18, ASIACCS'21]	✓	◐	ASS+GC	✓	✓	✓	✓	✓	(✓)[4]	✗	2PC/Outsourcing	a	
HC	[77, SDM'06]	✗	◐	HE+ASS+GC	✓	✓	✗	✓	✗	✓	✗	2PC	h	
	[50, TKDE'07]	✗	◐	blinding *or* SKC	✓	✓	✗	✓	✓	✗	✗	data owners + 1 server	h	SKC not semantically secure
	[49, TDP'10]	✗	◐	HE+GC	✓	✓	✗	✓	✓	✓	✗	2PC	h	
	[48, ISI'14]	✗	N/A	HE	✓	✗	✗	✓	✓	✓	✗	2PC	v	
	[78, ISCC'17]	✗	◐	HE	✓	✓	✗	✓	✗	✗	✓	2PC	v/h	
	[19, ArXiv'19]	✓	◐	HE & GC	✓	✓	✓	✓	✗	✓	✓	2PC	h	
BIRCH	[79, SDM'06]	✗	◐	HE+ASS	✓	✓	✗	✓	✗	✗	✗	2PC	v	
	[80, ADMA'07]	✗	◐	HE+ASS	✓	✓	✗	✓	✗	✗	✗	2PC	a	

[1] Of the parameters hold by the respective data owner.
[2] Assuming max. 1 party deviates from the protocol.
[3] Leaks partial information about cluster sizes.
[4] Not implemented, but possible.
[5] Can be used with any security model of GCs.

Table 3. History overview of privacy-preserving clustering using secure computation techniques. Privacy indicates if fully privacy protection according to the ideal functionality for privacy-preserving clustering (§3.1) is provided (✗: leakage; ✓: no leakage). ◐ is the semi-honest security model, ● is the malicious security model, N/A indicates that no security model was defined. HE is homomorphic encryption, ASS additive secret sharing, SSS Shamir's secret sharing, GC garbled circuits, OPE oblivious polynomial evaluation, PKC public-key cryptography, SKC symmetric-key cryptography, ZKP zero-knowledge proof, blinding is the use of random values for blinding, and other types of secret sharing are summarized by SS. v indicates that the data that shall be clustered is vertically distributed, i.e., the data owners hold the values for a subset of parameters from all data records. h indicates horizontally partitioned data where the data owners hold complete data records with all parameters, and a is arbitrary data partitioning. L1 leaks intermediate centroids, L2 intermediate cluster sizes, L3 other intermediate values (e.g., intermediate cluster assignments or distance comparison results), and L4 the number of clustering iterations. O1 outputs the final cluster labels/assignments, O2 outputs the final centroids, and O3 outputs the final dendogram/tree structure. The schemes with the best privacy guarantees are marked in bold (we do not consider the number of clustering iterations as a severe leakage as it can be easily avoided, cf. §3.2). The efficient and fully private schemes that we implemented and benchmarked in §4 are highlighted in gray.

46, 48, 57, 71–74, 78, 79, 82, 85, 87–89, 91, 93]. An arbitrary partitioning is a mix of both vertical and horizontal data splitting [8, 12, 22, 76, 80, 83, 90, 100]. A realistic data partition depends on the specific application. We discuss this matter in more detail in §3.3.

Used Secure Computation Techniques. Existing privacy-preserving clustering protocols use two main cryptographic techniques. First, there is a range of works that use homomorphic encryption (HE), e.g., [8, 14, 48, 76, 78, 84, 97, 104], but most of them tend to be relatively slow due to the expensive cryptographic operations. Another research direction uses multi-party computation (MPC) techniques like Yao's Garbled Circuits [21], blinding with random numbers, and secret sharing to achieve better efficiency [11, 16, 18, 44, 45, 72, 83, 94]. These schemes tend to have better runtimes, but higher communication than using HE. However, some MPC techniques [10] also have to rely on non-collusion assumptions between (a subset of) the computing parties which can make them more difficult to deploy in real-world applications. Other protocols use a mix of these techniques aiming at combining the strengths of both approaches [12, 19, 46, 77, 79, 103].

Privacy. As discussed in §1, information leakage can cause severe privacy infringement. Ideally, no information beyond what can be extracted from the final output should be derivable (cf. §3.1). However, most of the proposed privacy-preserving clustering schemes leak intermediate values like the intermediate centroids [43, 84, 85, 88, 94, 97, 98, 100, 105, 108], cluster assignments [43, 57, 58, 71, 83, 86, 87, 89–91, 97, 98, 100, 102–106, 108], and/or cluster sizes [8, 57, 71] in each clustering iteration of K-means or K-medoids, thus, failing to provide full privacy protection. Similarly, both private GMM schemes [44, 45] leak the intermediate covariance matrices, means, and probability values for each Gaussian distribution. Many schemes originating from a round-based clustering algorithm such as K-means or GMM leak the number of clustering iterations until convergence, e.g., [12, 13, 15, 43–45, 82, 83, 94, 100]. However, this issue can be avoided by clustering for a fixed number of iterations independent of the input which must be large enough to reach a good clustering result. However, this results in a longer runtime as more iterations are done than normally with a convergence check. Also most DBSCAN-based and HC-based schemes leak information, e.g., distances between data records [46, 50], the comparison results of distances [48, 72, 75, 79, 80], cluster assignments [22, 46, 47, 49, 73, 75, 76, 79, 80], cluster sizes [22, 47, 73, 75, 76], or may even leak concrete input records for specific data constellations [73, 77] to at least one of the involved parties (independent of the party's data ownership). All in all, we only identified ten clustering protocols shown in Tab. 1 that provide fully privacy guarantees (maximally leaking the number of clustering iterations): [9, 11–19].

Efficiency. As stated before, homomorphic encryption-based protocols such as [14, 48, 76, 78, 84, 97, 104] tend to be computationally expensive and, thus, slower than MPC-based schemes, e.g., [11, 18, 44, 45, 72, 94], which require more communication. Due to space limitations, we will focus here on the ten protocols that provide full privacy (cf. §3.1) and compare them in terms of efficiency. Kim and Chang [15] observe an about 2.85× runtime improvement compared to [13] thanks to a more efficient secure comparison. They as well as Bunn and Ostrovsky [12] use the Paillier encryption scheme without any parallelization making it expensive and slow compared to the other more optimised protocols which use, for example, packing or batching of operations [11]. The K-means protocol by Mohassel et al. [11] was experimentally compared to [14] and [18]. It outperforms the K-means protocol by Jäschke et al. [14] by five orders of magnitude on a dataset with 400 elements thanks to an efficient batching and the usage of GC instead of HE. It is also 19× faster on the same dataset than the private DBSCAN protocol of Bozdemir et al. [18], but DBSCAN often achieves significantly better clustering quality [18]. [18] runs about 13 minutes for clustering 500 elements while Zahur and Evans [17] report more than 550 minutes for their private DBSCAN protocol with 480 records. [18] is also 194× faster than the fully-private affinity propagation protocol by Keller et al. [16] thanks to the use of optimized combinations of GC and ASS. No direct comparison between [9, 11, 19] was done so far.

Choice for Benchmarks. To summarize, the MPC-based protocol of Mohassel et al. [11] is the most advanced private K-means scheme w.r.t. privacy and efficiency. Meng et al. [19] and Cheon et al. [9] provide the only schemes that offer fully private single and complete linkage HC/Mean-shift. Bozdemir et al. [18] propose the most efficient fully privacy-preserving DBSCAN protocol. In §4, we focus on these four works by comparing their computation and communication efficiency, security and privacy, and clustering quality.

3.3 Private Clustering Applications

Privacy-preserving clustering can be generally used for two main purposes: to protect sensitive data when out-

sourcing the computation and storage and/or when multiple data owners provide input data to the clustering. For each of these scenarios, we discuss a few example applications in the following to give a guideline on how to choose appropriate private clustering protocols.

Example Applications. Multiple data owners who jointly cluster their combined data is an instance of *multi-party computation* (MPC). In the financial market, clustering is used to automatically detect correlations between securities' stock prices in pair trading, i.e., for investment strategies that leverage discrepancies between typically correlated securities [109, 110]. Additionally, it is used for outlier detection to identify credit card, insurance, or tax frauds and insider trading [3, 111]. In this context, it is typically necessary to cluster data from several sources like competing (investment) banks or insurance companies to detect suspicious behavior [112]. Thereby, the different entities might hold information about the same customer, i.e., they have vertically partitioned data. Furthermore, clustering can be used by companies for enhancing marketing measures, e.g., by market segmentation or personalization of recommendation systems [113, 114]. A larger database increases the quality and reliability of the result but business secrets and customer data must be protected. In such scenarios, a horizontal data partitioning where the companies provide data from different customers is more plausible. Additionally, clustering is also used in medical research and diagnosis [115, 116]. In this context, using data from several sources, e.g., several hospitals, reduces potential bias caused by demographics, ethnicities, or cultures.

MPC. For the aforementioned applications, the parties can always safely trust themselves. Thus, full threshold security (i.e., security against up to $N-1$ collusions, cf. §2.2) as provided by some MPC techniques [117, 118] would be an interesting option. Unfortunately, only Keller et al. [16] and schemes based on a threshold secret sharing (like SSS (cf. §2.2)) with the respective threshold can offer this. Additionally, again only Keller et al. [16] provide full privacy guarantees against malicious adversaries (cf. §2.2). However, if the other partners involved are generally trusted but strictly regulated by data protection laws like HIPAA or GDPR, hindering them from directly sharing data, a 2PC- or MPC-based clustering protocol with honest majority and secure against semi-honest adversaries, e.g., [9, 11, 19], might be sufficient and provides significantly better efficiency than MPC techniques that are secure against full threshold semi-honest or malicious adversaries [119].

Outsourcing. While running "generic" MPC protocols is the most straightforward approach to securely cluster on the joint database of data owners, it suffers from high computation and communication costs and might be practically infeasible for a large number of data owners as MPC protocols often scale quadratically in the number of parties. A more efficient alternative can be to outsource the evaluation.[2] Thereby, the data owners (e.g., competing companies conducting market analyses) might prefer to not rely on a non-collusion assumption needed for MPC-based protocols such as [11] where multiple providers must be found and trusted not to collude. Hence, in such a situation an HE-based outsourcing scheme like [9, 14] might be advantageous. Additionally, in an outsourcing scenario, no data owner should be required to be online or actively involved in the computation. This can only be achieved by some protocols: [11, 13–16, 18, 19, 37, 43, 58, 90, 92, 100, 102, 106].

Privacy vs. Efficiency. Furthermore, there exists a trade-off between data leakage and efficiency. Schemes that can leak complete data records (e.g., [73, 74]) should not be used. When generally trusted parties like hospitals are involved, leaking less critical information like the number of iterations (cf. §3.2) might be acceptable to reduce runtime. However, as pointed out in §1, it is not always possible to fully understand and anticipate the effects of leaking intermediate results. Generally, MPC-based protocols are considered to be faster but require more communication than HE-based protocols. We will give more insight on the efficiency of the four most efficient fully private clustering schemes in §4.

Algorithm Characteristics. Finally, another important aspect for choosing the right private clustering scheme are the input parameters. For instance, K-means, K-medoid, GMM, and HC require the number of clusters as input. This is not an issue when the number of clusters is fixed by the application, e.g., if the goal is to split bank customers' behavior into benign and suspicious. However, a more fine-grained analysis might be needed when different types of malicious behavior can occur that significantly differ from each other (e.g., credit card, tax, or insurance fraud), but it is unclear in advance how many untypical behaviors can occur. Clustering might even be used to detect and differentiate these outliers in the first place. In such a case, a protocol that originates from affinitey propogation, DBSCAN or Mean-shift should be chosen as they will flexibly detect the number of clusters. Furthermore, some algo-

2 A single data owner might of course also outsource clustering.

rithms like K-means and GMM can only detect clusters of convex shapes, while DBSCAN can detect arbitrarily shaped clusters [41]. If new data arrives regularly, it might be beneficial to avoid recomputing the complete clustering by using K-means-, K-medoid-, DBSCAN-, or GMM-based protocols (e.g., [8, 11, 43, 44, 46]). Then, different private clustering schemes give different outputs, e.g., centroids [8, 11] or dendograms [19]. For example, a medical analysis detecting typical characteristics for a specific disease should output centroids as they represent these characteristics. Additionally, centroids also allow to assign new data to the created clusters later on which is not possible with only the cluster labels.

To conclude, the following aspects need to be examined when deciding upon a private clustering protocol: scenario (MPC vs. outsourcing), security/privacy requirements, trust level among the data owners, data distribution and splitting, and the plaintext clustering algorithm's characteristics (e.g., required input parameters that can be anticipated in advance). Based on this information, our extensive summary in Tab. 3 can help to choose an appropriate protocol.

4 Evaluation

In this section, we compare the clustering quality (§4.1), security and privacy (§4.2), and efficiency (§4.3) of the four most efficient fully private clustering protocols [9, 11, 18, 19] identified in §3.2. Details about these protocols are provided in Appx. B.

Software Details. We implemented all four protocols in C++17 and instantiate all cryptographic building blocks with a security level of 128 bits. We instantiate all algorithms with optimal parameters to assess their performance assuming perfect conditions. All our implementations are single threaded for a fair comparison of the efficiency of protocols.

MPC-KMeans [11]. In the remainder of this work, we call the private K-means protocol by Mohassel et al. [11] MPC-KMeans. We use the publicly available implementation[3] from the authors of [11] with default parameter values. Specifically, the statistical security parameter is $\lambda = 40$, the computational security parameter $\kappa = 128$, and the bitlength $\ell = 32$.

HE-Meanshift [9]. We call the private Mean-shift protocol by Cheon et al. [9] HE-Meanshift. The implementation uses the HEAAN library [120] with the same parameters as [9] providing 128-bit security. Specifically, the degree of the polynomial modulus of the plaintext ring N_c is set to 2^{17} and the ciphertext modulus q_L is set to 2^{1480}. Thus, the number of plaintext slots in each ciphertext is 2^{16}. Unless explicitly stated, we set the degree parameter for the kernel $\Gamma = 6$, the MinIdx degree parameter $t = 5$, and the Inv iteration parameter $\zeta = 5$.

PCA/OPT [19]. We call the baseline private HC protocol by Meng et al. [19] PCA (complete linkage) and its extension OPT (single linkage). The implementation uses the ABY framework [10] for Yao's garbled circuits and the libpaillier library [121] for Paillier with identical parameters as [19]. Specifically, the symmetric-key security parameter is $\kappa = 128$ bits and the size of the RSA modulus in Paillier encryption is $\kappa_{\text{pub}} = 2048$ bits. The statistical security parameter is $\lambda = 40$.

ppDBSCAN [18]. We call the privacy-preserving DBSCAN protocol by Bozdemir et al. [18] ppDBSCAN. We use the publicly available C++ implementation[4] provided by the authors which is based on the ABY framework [10]. The computational security parameter is set to $\kappa = 128$ bits and the bitlength $\ell = 32$ bits. The parameter *maxIterations* was set to 4 as also done in [18].

4.1 Clustering Quality

In this section, we evaluate the clustering quality of the four fully private clustering protocols.

Datasets. We use nine datasets from the well-known FCPS [122] and Graves [123] collections designed for benchmarking clustering algorithms. They also include the ground truth separation [124]. In Tab. 4, we summarize four of these datasets for which we present the results of the quality evaluation. The results of our evaluation on the remaining 5 datasets are given in Appx. D.

Metrics for Clustering Quality. We measure the quality of the clustering result using *clustering quality indices*. As no single index is superior [125], we use four well-known indices: Adjusted Rand Index (ARI) [126], Adjusted Mutual Information (AMI) [127], Silhouette Index (SI) [128], and Calinski-Harabasz Index (CHI) [129]. SI and CHI measure the output clusters' separation and compactness while ARI and AMI

3 https://github.com/osu-crypto/secure-kmean-clustering

4 https://encrypto.de/code/ppDBSCAN

Dataset	N	d	K	Property
Hepta	212	3	7	Well-defined clusters
Lsun	400	2	3	Different shapes
Chainlink	1000	3	2	Non-linearly separable clusters
Dense	200	2	2	Different cluster variances

Table 4. Datasets used for evaluating clustering quality, where N is the dataset size, d is the dimension of the data records, and K is the number of clusters.

compare the output clusters to the known ground truth to evaluate the clustering quality [125].

The results for the algorithms with random initialization, MPC-KMeans and HE-Meanshift are averaged over 10 runs. The iterative algorithms MPC-KMeans, HE-Meanshift, KMeans++, and Mean-shift are run for 20 iterations. The number of clusters K for the K-Means and HC protocols, i.e., MPC-KMeans and PCA/OPT, is set to the number of clusters in the ground truth. HE-Meanshift modifies the Mean-shift algorithm to run the mean-shift process on a small and random subset of datapoints, called dusts, to improve efficiency. The number of dusts is set to the largest power of 2 greater than or equal to the number of clusters, to ensure efficiency while maintaining the quality of the clustering.

Original vs. Private Algorithm. We also compare the differences in clustering quality between the private clustering protocols and the original plaintext algorithms to evaluate the *error arising by using privacy preserving techniques*. PCA, OPT and ppDBSCAN are identical to plaintext HC with complete and single linkage and DBSCAN respectively which is why we do not include the results for their plaintext implementations here. The underlying computations in MPC-KMeans are identical to the standard K-means protocol except for differences in the initialization of the centroids. We evaluate this effect using the plaintext KMeans++ [130] algorithm (a variant of K-Means with an improved cluster initialization where the centroids are initialized with data records far apart from each other). HE-Meanshift, in contrast, introduces several modifications to the standard Mean-shift algorithm [53] to make the computation HE friendly, e.g., using a polynomial kernel and adopting dust-sampling for efficient mode-seeking. We compare the clustering quality of HE-Meanshift to a plaintext implementation of Mean-shift to evaluate the combined effect of the changes.

Fig. 1 summarises the results of our evaluation of the clustering quality with the four quality indices.

Hepta Dataset. All algorithms achieve a relatively good clustering quality. PCA, OPT and ppDBSCAN output exactly the ground truth and achieve the best scores on the four indices. MPC-KMeans has a slightly worse clustering quality than the HC algorithms. HE-Meanshift achieves significantly lower scores and its *high standard deviation* in comparison to KMeans++ and Mean-shift indicates that the initialization of dusts has a significant impact on the clustering quality.

Lsun Dataset. OPT and ppDBSCAN output exactly the ground truth and, thus, achieve the maximal scores for the ARI and AMI. While the output of PCA significantly differs from the ground truth, it is noteworthy that it achieves similar scores as OPT and ppDBSCAN on the SI and CHI that measure internal cluster properties, i.e., separation and compactness. HE-Meanshift again shows large standard deviations due to the random initialization. The poor clustering quality achieved by MPC-KMeans and KMeans++ on the ARI and AMI is due to the non-convexly shaped clusters in the ground truth where K-means does not work well. The best scores achieved by HE-Meanshift are significantly higher than their plaintext counterparts, possibly due to favourable (random) initialization for the non-convexly shaped clusters in some of the runs.

Chainlink Dataset. OPT and ppDBSCAN have the same output as the groundtruth and achieve the highest ARI and AMI scores though the presence of non-linearly separable clusters in the dataset leads to lower SI and CHI scores. The remaining algorithms perform poorly. A poor clustering quality of MPC-KMeans and KMeans++ is expected since K-Means does not work well on non-linearly separable clusters.

Dense Dataset. MPC-KMeans, HE-Meanshift, ppDBSCAN, KMeans++, and Mean-shift achieve good scores on all indices while the HC protocols, i.e., PCA and OPT, have significantly lower scores in comparison. Intuitively, the poor clustering quality of HC-based protocols can be attributed to the large variance in one of the clusters of the Dense dataset. This can cause incorrect merging of clusters since clusters are merged based on their proximity, which can be large when the variance of the cluster is high.

Conclusion. ppDBSCAN consistently achieves the highest scores and is able handle different shapes, non-linear clusters, and high cluster variance well. PCA and OPT achieve a relatively good clustering quality on three out of four datasets, but they (completely) fail on the Dense dataset. The K-Means and Mean-shift protocols have comparable clustering quality that heavily varies between different datasets. The K-means-based protocols can only cluster very specific datasets that do not contain non-convexly shaped and non linearly-separable clusters.

HE-Meanshift tends to have large standard deviations which indicate a strong dependency on dust initialization. However, the highest score achieved by HE-Meanshift is comparable to that of plaintext Mean-shift which indicates that the modifications introduced for its HE-friendly computation do not decrease accuracy. In contrast, MPC-KMeans has a small standard deviation and achieves a similar clustering quality to KMeans++, which shows that the randomness used for centroid initialization has a smaller impact on final output.

4.2 Security & Privacy

In this section, we discuss the security and privacy of the four clustering protocols.

Security Model w.r.t Scenario. All four works are in the static semi-honest security model i.e., the adversary can corrupt some of the parties at the onset of the computation and correctly follows the protocol description, but attempts to learn information about the private inputs of the honest parties.

MPC-KMeans, PCA/OPT, and ppDBSCAN consider the outsourced two-party computation setting where multiple data owners secret share their input among two non-colluding servers to privately cluster the dataset. In contrast, in HE-Meanshift, a *single* data owner outsources its computation to a *single* server.

Informally, a protocol is said to be secure if anything that can be computed by a party participating in the protocol can also be derived from the input and output of this party. This is formalized by using a *simulator* which generates a view that is indistinguishable from a real protocol execution given the party's input and output [132]. MPC-KMeans [11] and PCA/OPT [19] provide such a formal proof of security.

The security of HE-Meanshift [9] follows directly from the security of the used CKKS encryption scheme since only the input and final output are sent. We note that the recent attack on the CKKS scheme by Li and Micciancio [133] does not affect the security of HE-Meanshift, as discussed by Cheon et al. [134]. Specifically, the attack requires access to a decryption oracle which is not available to the server in the outsourced single-server computation setting.

Similarly, the security of ppDBSCAN [18] follows directly from the security of the employed secure two-party computation techniques, specifically GC and ASS (cf. §2.2), as no intermediate values are opened and the conversions are provably secure [10].

Leakage from Outputs. Provable security of the protocols ensures that the computation does not leak anything more than what is revealed in an ideal world where a trusted third party obtains the inputs, computes the clustering functionality and returns the output. However, the information leaked from the clustering *output* is not captured in the security definition and we discuss this in the following.

HE-Meanshift outputs the cluster labels for every record in the dataset. However, this is not a privacy concern since the protocol is intended to be used in the outsourced single-server computation setting where the entire dataset is known to the client.

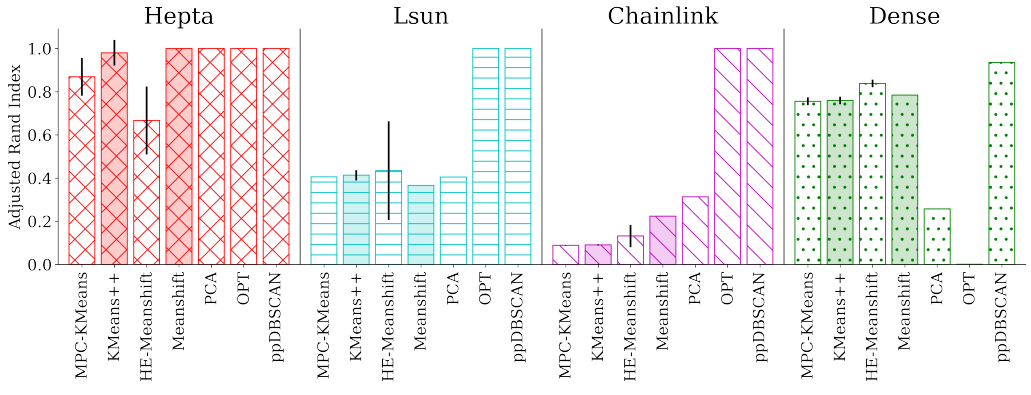
MPC-KMeans and ppDBSCAN can be adapted to output either the cluster centroids or cluster labels. MPC-KMeans also outputs the number of iterations for the clustering to converge which is related to the distribution of the underlying dataset. We have already discussed how to avoid this leakage in §3.2.

The PCA/OPT algorithms output a *point-agnostic dendrogram* in addition to the cluster centroids. The point-agnostic dendrogram is intended to be a privacy-preserving variant of the dendrogram output by a plaintext HC algorithm since the latter provides the complete merging history which leaks information in a setting with multiple data owners. The point-agnostic dendrogram is computed by first applying a random and private permutation on the input records to fuzz the merging history and by retaining the metadata of only sufficiently large clusters. Intuitively, this allows obtaining useful metadata akin to the plaintext computation while still preserving privacy. However, it is unclear how to formalize/measure the information leakage from the protocol output.

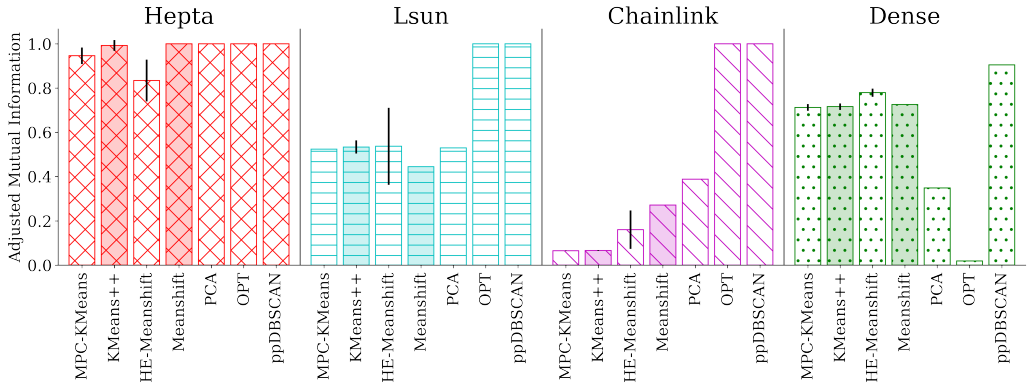
4.3 Efficiency

Asymptotic Analysis. First, we compare the asymptotic runtime, communication, and round complexity of the four investigated private clustering protocols and depict the results in Tab. 5. Asymptotically, MPC-KMeans is the most efficient with respect to communication and runtime in terms of dataset size N, input records' dimension d and number of clusters K.

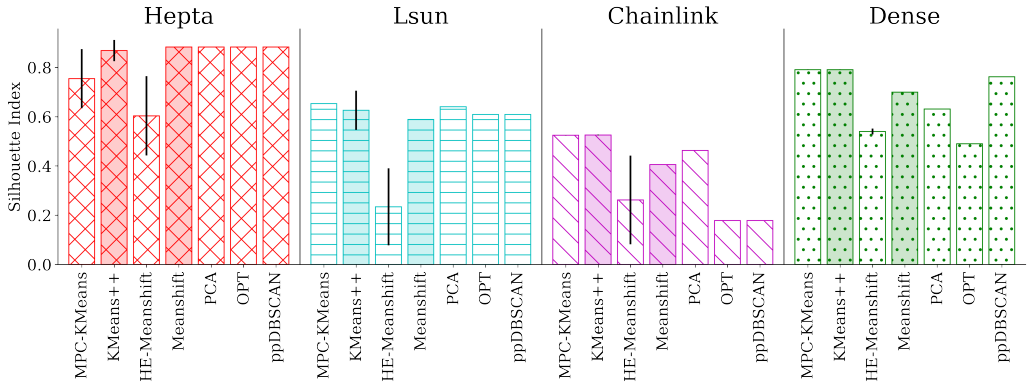
Hardware Details. All experiments are run on two machines (one for each party) each equipped with a 2.8 GHz Intel Core i9-7960X processor running Linux, 32 vCPUs and 128 GB RAM. We consider two network settings. The LAN setting has bandwidth 1 Gbps and



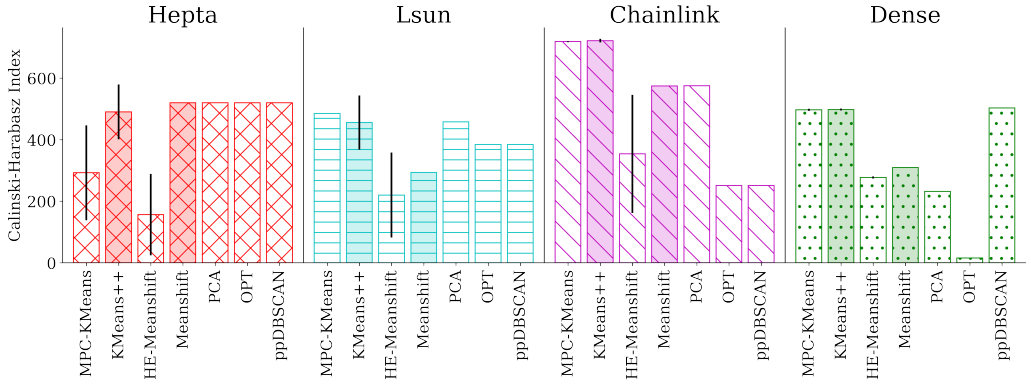
(a) Adjusted Rand Index (ARI)



(b) Adjusted Mutual Information (AMI)



(c) Silhouette Index (SI)



(d) Calinski-Harabasz Index (CHI)

Fig. 1. Clustering quality evaluation of the fully-private clustering protocols MPC-KMeans [11], HE-Meanshift [9], PCA/OPT [19], and ppDBSCAN [18] evaluated on datasets Hepta (red), Lsun (blue), Chainling (purple), and Dense (green). As comparison to the plaintext clustering algorithms (shaded bars), we also include KMeans++ and Mean-shift from Python Scikit-learn [131]. Each subfigure (a-d) corresponds to a different clustering quality index: ARI, AMI, SI, and CHI (cf. §4.1) and larger values indicate better clustering quality. The values are averaged over 10 runs and the error bar shows the standard deviation.

Protocol	Runtime	Communication	Rounds
MPC-KMeans [11]	$\Theta(NK(d+\ell)t)$	$\Theta\left(NK(d\ell^2+\ell\kappa)t\right)$	$\Theta(\lceil\log K\rceil t)$
HE-Meanshift [9]	$\Theta\left((NK_d d^2 t)/(N_c \log d)\right)$	$\Theta(NdK_d\kappa)$	2
PCA [19]	$\Theta(N^3\lambda)$	$\Theta(N^3\lambda\kappa)$	$\Theta(N^2)$
OPT [19]	$\Theta(N^2(\lambda+d))$	$\Theta(N^2(\lambda\kappa+\kappa_{\text{pub}}))$	$\Theta(N^2)$
ppDBSCAN [18]	$\Theta(N^2(N+d))$	$\Theta(N^2\ell\kappa)$	$\mathcal{O}(N^3)$

Table 5. Asymptotic runtime, communication, and round complexity of the private clustering protocols MPC-KMeans [11], HE-Meanshift [9], PCA/OPT [19], and ppDBSCAN [18]. N is the dataset size, d is the dimension, ℓ is the bitlength of the data records, K is the number of clusters, K_d is the number of dusts used in HE-Meanshift, $\kappa = 128$ is the computational security parameter, $\lambda = 40$ is the statistical security parameter, N_c is the number of plaintext slots in CKKS, $\kappa_{\text{pub}} = 2048$ is the size of the RSA modulus in Paillier encryption [65].

RTT 1 ms while the WAN setting has bandwidth 100 Mbps and RTT 100 ms.

Benchmarks. We evaluate the efficiency of the four privacy-preserving clustering protocols and their scalability to large datasets, datasets with many clusters, and multi-dimensional data. We generate synthetic datasets with $N \in \{50, 100, 150, 200, 250\}$ data points of dimension $d \in \{2, 8\}$ and bitlength $\ell = 32$, and number of clusters $K \in \{2, 5\}$. We run the protocols on all possible combinations of the above described parameters. The iterative protocols MPC-KMeans and HE-Meanshift are run for 5 iterations to reach a comparable clustering quality and enabling a fair comparison of their efficiency.

To analyze the scalability of the protocols to large multi-dimensional datasets, we generate a synthetic collection of *large datasets* with parameters $N \in \{2^{13}, 2^{14}, 2^{15}, 2^{16}\}$, $d \in \{1, 2, 4, 8, 16\}$ and $K \in \{2, 5, 10, 15, 20\}$. The memory consumption of MPC-KMeans was too large (greater than 128 GB) to benchmark on datasets where $N \cdot d > 2^{19}$. Similarly, the memory consumption of PCA/OPT [19] and ppDBSCAN [18] was too large even for the smallest dataset with $N = 2^{13}, d = 1$, and $K = 2$ due to the usage of the memory intensive ABY framework [10]. We thus exclude these protocols from our benchmarks on the large datasets. Fig. 4 in Appx. C presents the memory consumption of our implementations of the protocols for a small and large dataset.

Communication. We plot the communication costs in the bottom rows of Fig. 2 (small datasets) and Fig. 5 in Appx. C (large datasets).

The communication cost for HE-Meanshift is identical across different small datasets (Fig. 2) because the entire dataset can be encrypted in one ciphertext. This inefficient packing of the dataset leads to HE-Meanshift's communication being 2× higher than that of MPC-KMeans on average for small datasets. However, for large datasets (Fig. 5), HE-Meanshift's communication cost is up to 11.5× lower than that of MPC-KMeans on average due to optimal packing. The communication cost of PCA is 6× higher than that of ppDBSCAN on average while the communication of ppDBSCAN is 2× higher than that of OPT on average. While the communication cost increases linearly in the input records' dimension d for HE-Meanshift, d does not have a significant effect on the communication of MPC-KMeans since the communication during the assignment of input records to clusters is independent of d. This phase has the highest communication complexity and dominates the overall communication cost. In Fig. 5(f) HE-Meanshift's communication for $K = 10$ and $K = 15$ is identical as both use the same number of dusts $K_d = 16$ and hence send the same number of ciphertexts. The communication costs of PCA/OPT are independent of the input records' dimension d while the communication costs of PCA/OPT and ppDBSCAN are independent of the number of clusters K.

Runtimes. We plot the LAN runtimes in the top row of Fig. 2 (small datasets) and Fig. 5 in Appx. C (large datases), and the WAN runtimes for small datasets in Fig. 3 in Appx. C, all averaged over 10 runs.

MPC-KMeans has the lowest runtime and is up to 700× faster than HE-Meanshift on small datasets and 9.5× faster than HE-Meanshift on large datasets over LAN. Thus, HE-Meanshift scales well to large, multi-dimensional datasets due to lower communication costs, but is not suitable for small datasets. As expected, the runtime of MPC-KMeans in Fig. 5(b) is marginally affected by increasing the dimension of the input records d since its assignment of the input records (which is the bottleneck of the protocol) to the clusters is independent of d. On the other hand, the runtime of HE-Meanshift is linear in d since it directly affects the number of ciphertexts and the efficiency of the bootstrapping operation.

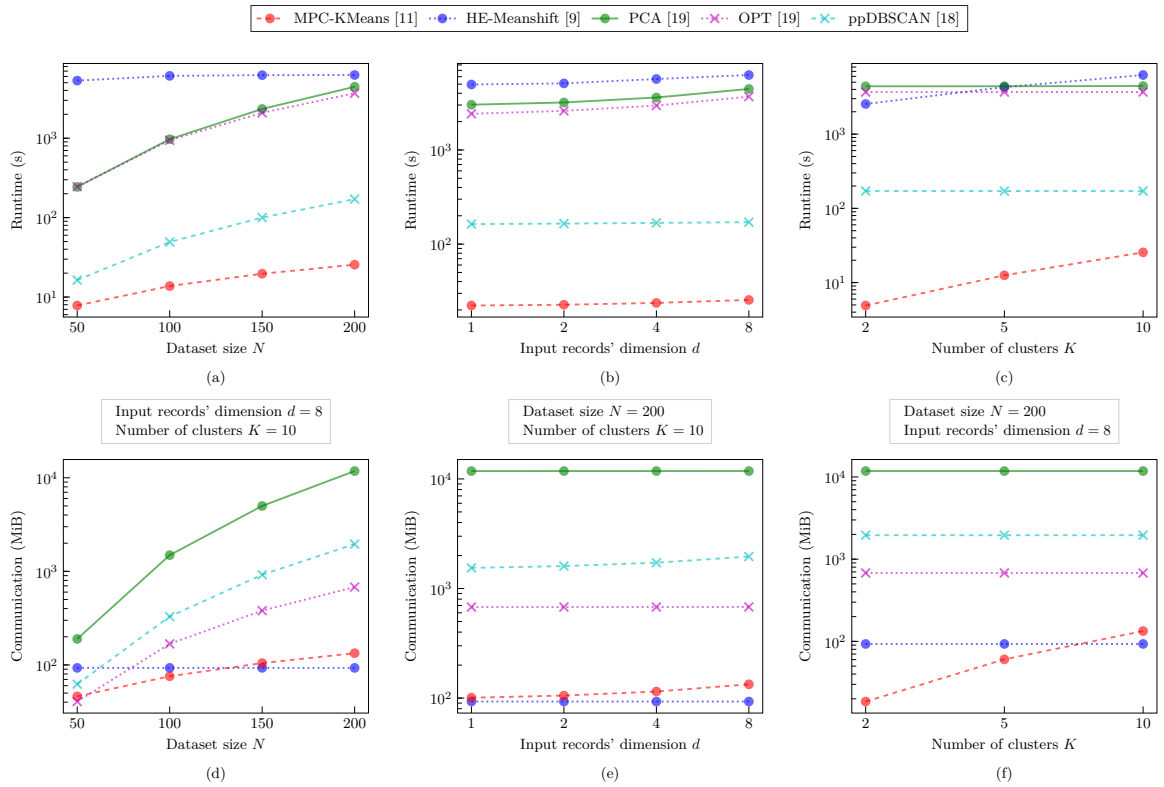


Fig. 2. LAN runtimes in seconds (top row) and **communication** in MiB (bottom row) of the fully-private clustering protocols MPC-KMeans [11], HE-Meanshift [9], PCA/OPT [19], and ppDBSCAN [18] for varying dataset size N, input records' dimension d, number of clusters K, and bitlength $\ell = 32$. In (a) and (d) $d = 8$ and $K = 10$, in (b) and (e) $N = 200$ and $K = 10$, and in (c) and (f) $N = 200$ and $d = 8$.

ppDBSCAN's average runtime is 14× higher over LAN and 2.5× higher over WAN than that of MPC-KMeans. However, since ppDBSCAN's runtime is independent of the number of clusters, this gap in runtime diminishes with the increase in number of clusters. OPT and PCA have similar runtimes wich are 15× higher than that of ppDBSCAN on average over LAN, even though OPT has less communication. Moreover, OPT has the highest runtime on average over WAN. One possible reason for this might be higher concrete computation costs for OPT which is not captured by the asymptotic complexity but highlighted due to benchmarking on small datasets.

5 Real-world Application and Open Challenges

In this section, we discuss the challenges that need to be solved to make privacy-preserving clustering practical for real-world applications.

Parameters. In a privacy-preserving setting, it is typically impossible to perform a preliminary analysis of the data since it is distributed among multiple data owners and not available to a single party. However, to set parameters like the number of clusters K for K-Means (i.e., in MPC-KMeans [11]) and HC (i.e., in PCA/OPT [19]), the distance parameter ϵ for ppDBSCAN [18], or the number of dusts for HE-Meanshift [9], insights about the dataset are often needed to achieve high clustering quality. We also observed in our experimental evaluation that the degree of the kernel and the value of the MinIdx parameter in HE-Meanshift has significant impact on the clustering quality by amplifying the distances between data records. In specific cases, some parameters like the number of clusters K can be given by the application (cf. §3.3). But this is not the case for less intuitive parameters like the initial distribution specifications of GMM or the neighborhood radius in DBSCAN. Only ppDBSCAN [18], out of the four protocols that we benchmarked, can determine these parameters when they are not fixed by the application and inputs are provided by more than one party.

Secure Clustering Quality Evaluation. The issue of dataset-dependent parameters is amplified by the lack of secure clustering quality evaluation techniques. Specifically, plaintext clustering algorithms are often simply run several times with a range of different parameter values and the best output is selected based on the score of a clustering quality index. This is not possible in privacy-preserving clustering. Firstly, as in plaintext clustering, no ground truth is known, so metrics like ARI or AMI that compare to the ground truth cannot be used. Secondly, as the private clustering result is typically split among the data owners or only consists of the centroids, the clusters' compactness and the separation between different clusters cannot be measured without additional secure computation. Thus, also internal indices like SI and CHI cannot be used easily. The inherent overhead of secure computation makes it expensive to perform multiple runs with different parameter values.

Clustering Quality and Efficiency. Clustering algorithms that make minimal assumptions about the shape of clusters and are robust to outliers are especially important in the case of privacy-preserving clustering, because it is impossible to analyze the dataset or remove noisy records before clustering. None of the privacy-preserving clustering protocols we consider investigate soft clustering (cf. §2.1) and only ppDBSCAN has the notion of noise. The K-means-based protocols, i.e., MPC-KMeans, are very sensitive to outliers. Additionally, as shown by our quality evaluation in §4.1 and as discussed in §2.1, K-means only succeeds on clustering convexly shaped clusters which is not the case for all datasets. HE-Meanshift's clustering quality also strongly fluctuates depending on the dataset's properties (cf. §4.1). This is especially problematic for privacy-preserving clustering where the dataset distribution is often not known in advance. In contrast, hierarchical clustering like PCA/OPT is less sensitive to noise and more flexible with respect to the data distribution, i.e., the cluster shapes. However, as shown in §4.3, PCA/OPT cannot be run on large datasets while MPC-KMeans and HE-Meanshift scale significantly better to large datasets. Our evaluation shows that ppDBSCAN performs well on different types of datasets while also having lower runtimes than other protocols (except MPC-KMeans).

Recommendations. Among the protocols we evaluate, MPC-KMeans seems to be the most efficient alternative when clustering large multi-dimensional datasets. HE-Meanshift might be a better choice when a *single* resource-constrained data-owner outsources clustering to a more powerful server over a high-latency and low-bandwidth network. For smaller datasets, ppDBSCAN seems to be the best option which performs well on a variety of dataset types and also achieves low runtimes. However, choosing the input parameter ϵ that determines the maximum distance between two data records to be considered as neighbors requires domain expertise and partial information about the dataset. This can be avoided by using MPC-KMeans which only requires setting the more intuitive number of clusters K which in some cases is also given by the application (cf. §3.3).

Open Challenges. To summarize, for practical application, privacy-preserving clustering protocols must be (1) efficient in terms of runtime and communication, (2) memory efficient, (3) only have parameters that are mostly independent of the input data, (4) insensitive to noise, and (5) flexible to cluster data of any distribution with high quality. Unfortunately, none of the state-of-the-art works can fulfill all these requirements simultaneously. Additionally, there is the need for a secure clustering quality evaluation to assess the quality of a clustering result run in a privacy-preserving manner. Finally, privacy-research has not tackled privacy-preserving soft clustering.

6 Conclusion

In this work, we systematically surveyed and analyzed the state-of-the-art in privacy-preserving clustering. We benchmarked and compared four efficient protocols [9, 11, 18, 19] that securely realize four different clustering algorithms, with respect to clustering quality, communication, and runtime to investigate their practicality for real-world applications. Finally, we discussed open challenges to make privacy-preserving clustering practical.

ACKNOWLEDGEMENTS

We thank Oliver Schick for his support with implementing PCA/OPT [19]. 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) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the BMBF and the HMWK within ATHENE.

References

[1] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu, "On Network-level Clusters for Spam Detection." in *NDSS*, 2010.

[2] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and Structural Biotechnology Journal*, 2015.

[3] M. Ahmed, A. N. Mahmood, and M. R. Islam, "A Survey of Anomaly Detection Techniques in Financial Domain," in *Future Generation Computer Systems*, 2016.

[4] F. Masulli and A. Schenone, "A fuzzy clustering based segmentation system as support to diagnosis in medical imaging," *Artificial Intelligence in Medicine*, 1999.

[5] S. Gauch, M. Speretta, A. Chandramouli, and A. Micarelli, "User profiles for personalized information access," in *The adaptive web*, 2007.

[6] A. Chaturvedi, J. D. Carroll, P. E. Green, and J. A. Rotondo, "A feature-based approach to market segmentation via overlapping k-centroids clustering," *Journal of Marketing Research*, 1997.

[7] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford university Stanford, 2009.

[8] W. Wu, J. Liu, H. Wang, J. Hao, and M. Xian, "Secure and efficient outsourced K-means clustering using fully homomorphic encryption with ciphertext packing technique," in *TDKE*, 2020.

[9] J. H. Cheon, D. Kim, and J. H. Park, "Towards a practical cluster analysis over encrypted data," in *SAC*, 2019.

[10] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *NDSS*, 2015.

[11] P. Mohassel, M. Rosulek, and N. Trieu, "Practical privacy-preserving K-means clustering," in *PETS*, 2020.

[12] P. Bunn and R. Ostrovsky, "Secure two-party K-means clustering," in *CCS*, 2007.

[13] F.-Y. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, "Privacy-preserving and outsourced multi-user K-means clustering," in *CIC*, 2015.

[14] A. Jäschke and F. Armknecht, "Unsupervised Machine Learning on Encrypted Data," in *SAC*, 2018.

[15] H. Kim and J. Chang, "A privacy-preserving k-means clustering algorithm using secure comparison protocol and density-based center point selection," in *International Conference on Cloud Computing*, 2018.

[16] H. Keller, H. Möllering, T. Schneider, and H. Yalame, "Balancing quality and efficiency in private clustering with affinity propagation," in *SECRYPT*, 2021.

[17] S. Zahur and D. Evans, "Circuit structures for improving efficiency of security and privacy tools," in *IEEE S&P*, 2013.

[18] B. Bozdemir, S. Canard, O. Ermis, H. Möllering, M. Önen, and T. Schneider, "Privacy-preserving density-based clustering," in *ASIACCS*, 2021.

[19] X. Meng, D. Papadopoulos, A. Oprea, and N. Triandopoulos, "Private two-party cluster analysis made formal & scalable," *arXiv:1904.04475v2*, 2019.

[20] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[21] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[22] J. Liu, L. Xiong, J. Luo, and J. Z. Huang, "Privacy preserving distributed DBSCAN clustering," in *Transactions on Data Privacy*, 2013.

[23] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *IEEE S&P*, 2020.

[24] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*, 2020.

[25] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security*, 2020.

[26] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *USENIX Security*, 2021.

[27] V. Haralampieva, D. Rueckert, and J. Passerat-Palmbach, "A systematic comparison of encrypted machine learning solutions for image classification," in *PPMLP*, 2020.

[28] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *ARES*, 2020.

[29] L. Song, H. Wu, W. Ruan, and W. Han, "SoK: Training machine learning models over multiple sources with privacy preservation," in *arXiv:2012.03386*, 2020.

[30] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service—a comprehensive survey," in *IEEE Access*, 2020.

[31] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, "SoK: modular and efficient private decision tree evaluation," in *PETS*, 2019.

[32] U. Stemmer, "Locally private K-means clustering," in *ACM-SIAM Symposium on Discrete Algorithms*, 2020.

[33] L. Ni, C. Li, X. Wang, H. Jiang, and J. Yu, "DP-MCDBSCAN: Differential privacy preserving multi-core DBSCAN clustering for network user data," in *IEEE Access*, 2018.

[34] M.-F. Balcan, T. Dick, Y. Liang, W. Mou, and H. Zhang, "Differentially private clustering in high-dimensional Euclidean spaces," in *ICML*, 2017.

[35] D. Su, J. Cao, N. Li, E. Bertino, M. Lyu, and H. Jin, "Differentially private K-means clustering and a hybrid approach to private optimization," in *TOPS*, 2017.

[36] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin, "Differentially private K-means clustering," in *Data and Application Security and Privacy*, 2016.

[37] W. Wu and H. Huang, "A DP-DBSCAN clustering algorithm based on differential privacy preserving," in *Computer Engineering and Science*, 2015.

[38] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *CCS*, 2016.

[39] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS*, 2015.

[40] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," in *Annals of Data Science*, 2015.

[41] R. Xu and D. Wunsch, "Survey of clustering algorithms," in *TNN*, 2005.

[42] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," in *ACM Computing Surveys*, 1999.

[43] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "PPHOPCM: privacy-preserving high-order possibilistic c-means algorithm for big data clustering with cloud computing," *IEEE Transactions on Big Data*, 2017.

[44] M. Hamidi, M. Sheikhalishahi, and F. Martinelli, "Privacy preserving Expectation Maximization (EM) clustering construction," in *DCAI*, 2019.

[45] X. Lin, C. Clifton, and M. Zhu, "Privacy-preserving clustering with distributed EM mixture modeling," in *Knowledge and Information Systems*, 2005.

[46] I. V. Anikin and R. M. Gazimov, "Privacy preserving DBSCAN clustering algorithm for vertically partitioned data in distributed systems," in *International Siberian Conference on Control and Communications*, 2017.

[47] M. S. Rahman, A. Basu, and S. Kiyomoto, "Towards outsourced privacy-preserving multiparty DBSCAN," in *PRDC*, 2017.

[48] I. De and A. Tripathy, "A secure two party hierarchical clustering approach for vertically partitioned data set with accuracy measure," in *Recent Advances in Intelligent Informatics*, 2014.

[49] G. Jagannathan, K. Pillaipakkamnatt, R. Wright, and D. Umano, "Communication-efficient privacy-preserving clustering," in *Transactions on Data Privacy*, 2010.

[50] A. İnan, S. V. Kaya, Y. Saygın, E. Savaş, A. A. Hintoğlu, and A. Levi, "Privacy preserving clustering on horizontally partitioned data," in *TDKE*, 2007.

[51] H. Steinhaus, "Sur la division des corp materiels en parties," in *Bulletin L'Académie Polonaise des Science*, 1956.

[52] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, "Cluster analysis," in *Wiley*, 2011.

[53] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," in *TIT*, 1975.

[54] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, "A distribution-based clustering algorithm for mining in large spatial databases," in *ICDE*, 1998.

[55] J. M. Pena, J. A. Lozano, and P. Larranaga, "An empirical comparison of four initialization methods for the K-means algorithm," in *Pattern Recognition Letters*, 1999.

[56] Zhexue Huang and M. K. Ng, "A fuzzy k-modes algorithm for clustering categorical data," in *TFS*, 1999.

[57] J. Zhan, "Privacy preserving K-medoids clustering," in *SMC*, 2007.

[58] K.-P. Lin, "Privacy-preserving kernel K-means clustering outsourcing with random transformation," *Knowledge and Information Systems*, 2016.

[59] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, 1988.

[60] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in Large spatial databases with noise." in *SIGKDD*, 1996.

[61] Y. Ren, C. Domeniconi, G. Zhang, and G. Yu, "A weighted adaptive mean shift clustering algorithm."

[62] M. Ester, "Density-based clustering," in *Encyclopedia of Database Systems*. Springer, 2009.

[63] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," in *ACM SIGMOD*, 1999.

[64] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," in *Journal of the Royal Statistical Society*, 1977.

[65] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[66] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017.

[67] A. Shamir, "How to share a secret," in *Communication of the ACM*, 1979.

[68] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[69] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE S&P*, 2017.

[70] S. Kamara and M. Raykova, "Secure outsourced computation in a multi-tenant cloud," in *IBM Workshop on Cryptography and Security in Clouds*, 2011.

[71] S. K. Dash, D. P. Mishra, R. Mishra, and S. Dash, "Privacy preserving K-medoids clustering: An approach towards securing data in mobile cloud architecture," in *Conference on Computational Science, Engineering and Information Technology*, 2012.

[72] A. Amirbekyan and V. Estivill-Castro, "Privacy preserving DBSCAN for vertically partitioned data," in *Intelligence and Security Informatics*, 2006.

[73] K. A. Kumar and C. P. Rangan, "Privacy preserving DBSCAN algorithm for clustering," in *Advanced Data Mining and Applications*, 2007.

[74] W.-j. Xu, L.-s. Huang, Y.-l. Luo, Y.-f. Yao, and W. Jing, "Protocols for privacy-preserving DBSCAN clustering," in *International Journal of Security and Its Applications*, 2007.

[75] D. Jiang, A. Xue, S. Ju, W. Chen, and H. Ma, "Privacy-preserving DBSCAN on horizontally partitioned data," in *International Symposium on IT in Medicine and Education*, 2008.

[76] N. Almutairi, F. Coenen, and K. Dures, "Secure third party data clustering using ϕ data: Multi-user order preserving encryption and super secure chain distance matrices," in *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2018.

[77] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, "A new privacy-preserving distributed K-clustering algorithm," in *SDM*, 2006.

[78] M. Sheikhalishahi and F. Martinelli, "Privacy preserving clustering over horizontal and vertical partitioned data," in *Symposium on Computers and Communications*, 2017.

[79] P. K. Prasad and C. P. Rangan, "Privacy preserving birch algorithm for clustering over vertically partitioned databases," in *Workshop on Secure Data Management*, 2006.

[80] K. Prasad and P. Rangan, "Privacy preserving birch algorithm for clustering over arbitrarily partitioned databases," *ADMA*, 2007.

[81] X. Zhu, M. Liu, and M. Xie, "Privacy-preserving affinity propagation clustering over vertically partitioned data," in *International Conference on Intelligent Networking and Collaborative Systems*, 2012.

[82] J. Vaidya and C. Clifton, "Privacy-preserving K-means clustering over vertically partitioned data," in *SIGKDD*, 2003.

[83] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed K-means clustering over arbitrarily partitioned data," in *SIGKDD*, 2005.

[84] S. Jha, L. Kruger, and P. McDaniel, "Privacy preserving clustering," in *ESORICS*, 2005.

[85] S. Samet, A. Miri, and L. Orozco-Barbosa, "Privacy preserving K-means clustering in multi-party environment," in *SECRYPT*, 2007.

[86] C. Su, F. Bao, J. Zhou, T. Takagi, and K. Sakurai, "Privacy-preserving two-party K-means clustering via secure approximation," in *AINA*, 2007.

[87] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaş, and A. Levi, "Distributed privacy preserving K-means clustering with additive secret sharing," in *International Workshop on Privacy and Anonymity in Information Society*, 2008.

[88] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving user clustering in a social network," in *Information Forensics and Security*, 2009.

[89] J. Sakuma and S. Kobayashi, "Large-scale k-means clustering with user-centric privacy-preservation," in *Knowledge and Information Systems*, 2010.

[90] M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. V. Jawahar, "Efficient privacy preserving K-means clustering," in *Pacific-Asia Workshop on Intelligence and Security Informatics*, 2010.

[91] T.-K. Yu, D. Lee, S.-M. Chang, and J. Zhan, "Multi-party K-means clustering with privacy consideration," in *ISPA*, 2010.

[92] M. Beye, Z. Erkin, and R. L. Lagendijk, "Efficient privacy preserving K-means clustering in a three-party setting," in *Information Forensics and Security*, 2011.

[93] Z. Lin and J. W. Jaromczyk, "Privacy preserving two-party K-means clustering over vertically partitioned dataset," in *ISI*, 2011.

[94] S. Patel, S. Garasia, and D. Jinwala, "An efficient approach for privacy preserving distributed K-means clustering based on shamir's secret sharing scheme," in *Trust Management VI*, 2012.

[95] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Privacy-preserving distributed clustering," in *EURASIP Journal on Information Security*, 2013.

[96] S. Patel, V. Patel, and D. Jinwala, "Privacy preserving distributed K-means clustering in malicious model using zero knowledge proof," in *Distributed Computing and Internet Technology*, 2013.

[97] D. Liu, E. Bertino, and X. Yi, "Privacy of outsourced K-means clustering," in *ASIACCS*, 2014.

[98] X. Liu, Z. L. Jiang, S. M. Yiu, X. Wang, C. Tan, Y. Li, Z. Liu, Y. Jin, and J. Fang, "Outsourcing two-party privacy preserving K-means clustering protocol in wireless sensor networks," in *MSN*, 2015.

[99] S. J. Patel, D. Punjani, and D. C. Jinwala, "An efficient approach for privacy preserving distributed clustering in semi-honest model using elliptic curve cryptography," *International Journal of Network Security*, 2015.

[100] V. Baby and N. S. Chandra, "Distributed threshold K-means clustering for privacy preserving data mining," in *ICACCI*, 2016.

[101] Z. Gheid and Y. Challal, "Efficient and privacy-preserving K-means clustering for big data mining," in *IEEE TrustCom/BigDataSE/ISPA*, 2016.

[102] H. Rong, H. Wang, J. Liu, J. Hao, and M. Xian, "Outsourced k-means clustering over encrypted data under multiple keys in spark framework," in *Security and Privacy in Communication Networks*, 2017.

[103] K. Xing, C. Hu, J. Yu, X. Cheng, and F. Zhang, "Mutual privacy preserving K-means clustering in social participatory sensing," in *TII*, 2017.

[104] J. Yuan and Y. Tian, "Practical privacy-preserving MapReduce based K-means clustering over large-ccale dataset," in *TCM*, 2019.

[105] Z. L. Jiang, N. Guo, Y. Jin, J. Lv, Y. Wu, Z. Liu, J. Fang, S. Yiu, and X. Wang, "Efficient two-party privacy-preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing," *Information Sciences*, 2020.

[106] Y. Zou, Z. Zhao, S. Shi, L. Wang, Y. Peng, Y. Ping, and B. Wang, "Highly secure privacy-preserving outsourced k-means clustering under multiple keys in cloud computing," in *Security and Communication Networks*, 2020.

[107] Y. Wang, "Notes on two fully homomorphic encryption schemes without bootstrapping." Cryptology ePrint Archive, Report 2015/519.

[108] Y. Cai and C. Tang, "Privacy of outsourced two-party k-means clustering," *Concurrency and Computation: Practice and Experience*, 2019.

[109] S. M. Sarmento and N. Horta, "Enhancing a pairs trading strategy with the application of machine learning," in *Expert Systems with Applications*, 2020.

[110] R. Adusumilli, "DBSCAN Clustering for Trading," 2020, https://towardsdatascience.com/dbscan-clustering-for-trading-4c48e5ebffc8.

[111] S. Panigrahi, A. Kundu, S. Sural, and A. K. Majumdar, "Credit card fraud detection: A fusion approach using dempster–shafer theory and bayesian learning," in *Information Fusion*, 2009.

[112] A. Sangers, M. van Heesch, T. Attema, T. Veugen, M. Wiggerman, J. Veldsink, O. Bloemen, and D. Worm, "Secure Multiparty PageRank Algorithm for Collaborative Fraud Detection," in *FC*, 2019.

[113] A. Chaturvedi, J. Carroll, P. Green, and J. A. Rotondo, "A feature-based approach to market segmentation via overlapping k-centroids clustering," *Journal of Marketing Research*, 1997.

[114] Y. S. Cho, S. C. Moon, S. C. Noh, and K. H. Ryu, "Implementation of personalized recommendation system using k-means clustering of item category based on rfm," in *ICMIT*, 2012.

[115] Q. Guo, X. Lu, Y. Gao, J. Zhang, B. Yan, D. Su, A. Song, X. Zhao, and G. Wang, "Cluster Analysis: A New Approach for Identification of Underlying Risk Factors for Coronary Artery Disease in Essential Hypertensive Patients," in *Scientific Reports*, 2017.

[116] F. Masulli and A. Schenone, "A fuzzy clustering based segmentation system as support to diagnosis in medical imaging," *Artificial Intelligence in Medicine*, 1999.

[117] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "MOTION - A framework for mixed-protocol multi-party computation," Cryptology ePrint Archive, Report 2020/1137.
[118] M. Keller, "MP-SPDZ: a versatile framework for multi-party computation," in *CCS*, 2020.
[119] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *PETS*, 2020.
[120] "HEAAN," https://github.com/snucrypto/HEAAN, 2020.
[121] "Paillier library," http://acsc.cs.utexas.edu/libpaillier, 2010.
[122] A. Ultsch, "Clustering with SOM," in *Workshop on Self-Organizing Maps*, 2005.
[123] D. Graves and W. Pedrycz, "Kernel-based fuzzy clustering and fuzzy clustering: A comparative experimental study," in *Fuzzy Sets and Systems*, 2010.
[124] M. Gagolewski, "Benchmark suite for clustering algorithms version 1," 2020, https://github.com/gagolews/clustering_benchmarks_v1.
[125] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, 2013.
[126] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, 1985.
[127] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, 2010.
[128] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, 1987.
[129] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," in *Communications in Statistics-theory and Methods*, 1974.
[130] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," *ACM-SIAM Symposium on Discrete Algorithms*, 2007.
[131] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *JMLR*, 2011.
[132] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, 2017.
[133] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," Cryptology ePrint Archive, Report 2020/1533.
[134] J. H. Cheon, S. Hong, and D. Kim, "Remark on the security of CKKS scheme in practice," Cryptology ePrint Archive, Report 2020/1581.
[135] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, 2007.
[136] X. Liu, M. Yin, J. Luo, and W. Chen, "An improved affinity propagation clustering algorithm for large-scale data sets," in *International Conference on Natural Computation*, 2013.
[137] F. Shang, L. Jiao, J. Shi, F. Wang, and M. Gong, "Fast affinity propagation clustering: A multilevel approach," *Pattern Recognition*, 2012.
[138] D. Dueck, *Affinity propagation: clustering data by passing messages*, 2009.
[139] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, 2014.
[140] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," *ACM SIGMOD*, 1996.
[141] C. E. Rasmussen *et al.*, "The infinite Gaussian mixture model," in *NIPS*, 1999.
[142] X. He, D. Cai, Y. Shao, H. Bao, and J. Han, "Laplacian regularized gaussian mixture model for data clustering," *IEEE Transactions on Knowledge and Data Engineering*, 2010.
[143] J. P. Patist, W. Kowalczyk, and E. Marchiori, "Maintaining gaussian mixture models of data streams under block evolution," in *International Conference on Computational Science*, 2006.
[144] R. C. Pinto and P. M. Engel, "A fast incremental gaussian mixture model," *PloS one*, 2015.
[145] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *EUROCRYPT*, 2018.
[146] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical method for comparison on homomorphically encrypted numbers," in *ASIACRYPT*, 2019.

A Discussion of Strengths and Weaknesses of Additional Clustering Algorithms

Affinity Propagation. Affinity Propagation [135] is a deterministic partitioning-based clustering algorithm that has a computational complexity of $\mathcal{O}(N^2t)$ and space complexity of $\mathcal{O}(N^2)$, where N is the dataset size and t the number of clustering iterations [136]. It flexibly determines the required number of clusters based on the input data such that outliers that do not fit into any cluster form a a cluster on their own [135]. The clustering result is independent of the input order of the data records, as Affinity Propagation always iterates through the complete dataset in each iteration. Affinity Propagation requires the input of a *preference* value for each input record that indicates its likelihood to be chosen as exemplar (similar to a centroid in K-means) of a cluster. If the preference values are not well chosen, it can lead to suboptimal clustering results [137]. If all records are equally likely the preference values are set to the same value for all records, e.g., the median or minimum of the distances [135]. The respective distance measure can be freely chosen, thus, also any variable type could be clustered [138]. On the downside, Affinity Propagation can only detect spherical clusters [139] and a re-clustering is needed if new data records are added to the input dataset after it has already been clustered, as this changes the responsibility and availability matrices.

BIRCH. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is a divisive HC algorithm [140]. Its computation and space complexity is linear in the dataset size [40]. Generally, BIRCH is relatively insensitive to noisy elements as it allows to remove elements in sparse regions [40, 140]. Because

each input record is processed incrementally and inserted into the subtree representing the assigned closest cluster, BIRCH can handle new data records well, but is affected by the input order [40, 140]. Moreover, it can only detect convexly shaped clusters with records with metric attributes [40, 140]. Additionally, BIRCH requires to input a threshold for the maximal cluster size and the branching factor of the tree. If the number of cluster K is not given, all sub-clusters in the tree are returned.

GMM. Gaussian Mixture Models (GMM) Clustering is a distribution-based clustering algorithm that uses the Expectation-Maximization (EM) algorithm [64, 141]. GMM has a computational complexity of $\mathcal{O}(NKd^3t)$, where N is the dataset size, K is the number of cluster, d is the data dimension, and t the number of clustering iterations and its space complexity is also linear in N [142–144]. The assumption of a Gaussian distribution of cluster elements restricts the type of the variables to real numbers. GMM fails when clusters have specific constellations, e.g., when one cluster is surrounded by another one, or if the clusters are not convexly shaped. It takes the number of clusters K as input and is relatively sensitive to the selection of the initial parameters of the cluster distributions [41]. Although GMM does not explicitly acknowledge the notion of noise, its result is relatively insensitive to outliers [40, 54], but Keller et al. [16] demonstrate that outliers can still cause significant misclassifcations and incorrect merges between different clusters. As it process the whole data set in each iteration, GMM is not affected by the input order. A few new data records can be clustered by GMM with only a few additional iterations.

B Summaries of Fully-Private Clustering Protocols

MPC-KMeans [11]. Mohassel et al. [11] propose a secure two-party K-means (cf. §2.1) protocol in the semi-honest security model using the ABY framework [10] for secure two-party computation. We call this protocol MPC-KMeans in the following. MPC-KMeans can also be used in an outsourcing scenario [70] where multiple data owners outsource the clustering to two non-colluding servers. The authors also propose a multi-party variant where parties first locally run the plaintext K-means on their local datasets and then proceed to securely compute the joint clustering result based on the previously determined local centroids of all parties.

In the two-party protocol, each data owner runs the plaintext K-means algorithm on its local datasets to compute $\frac{K}{2}$ local clusters. The centroids of these clusters are then secret-shared and used to initialize the centroids for the clustering over the combined dataset.

MPC-KMeans' building blocks are optimized for two computational settings: the amortized setting where the same function is evaluated multiple times on different inputs and the adaptive setting where the inputs to multiple evaluations of the function depend on the output of previous evaluations. Intuitively, the updates of the centroids are non-adaptive in one clustering iteration but adaptive across several iterations. Therefore, the authors introduce efficient protocols for secure multiplication and the calculation of the squared Euclidean distance in the adaptive amortized setting. They also propose an efficient protocol for computing the index of the minimum element in a list of t values using a recursive tree evaluation of a customized Garbled Circuit. This increases the number of rounds by $\lceil \log t \rceil$, but it reduces the communication costs by a factor of 2. MPC-KMeans terminates when the difference between new and old centroids is less than a predefined threshold.

The authors use the squared Euclidean distance. They also benchmark the Manhattan distance ($\max_{i \in [1,d]} |x_i - y_i|$, where d is the dimension) and Chessboard distance ($\sum_{i=1}^{d} |x_i - y_i|$), but show that computing squared Euclidean distance in the adaptive amortized setting is faster than the other two distances. The computation is done on fixed point numbers using the truncation method of [69] where each party locally truncates its share with an error of at most one bit in the least significant bit of the fractional part. The authors show that truncation has a negligible impact on the accuracy of clustering.

HE-Meanshift [9]. Cheon et al. [9] propose a HE-friendly variant of the Mean-shift clustering algorithm (cf. §2.1) in the semi-honest security model using the fully homomorphic encryption (FHE) scheme CKKS [66, 145]. We call this protocol HE-Meanshift in the following. The protocol is designed for the outsourced computation setting where a single, possibly resource-constrained, data owner securely outsources the computation to a server.

CKKS computes on real numbers, but it supports only addition and multiplication. Thus, HE-Meanshift replaces the non-polynomial operations in Mean-shift by polynomial operations. The gradient ascent algorithm used in Mean-shift for mode-seeking requires computing the derivative of the kernel. The authors of [9] propose the HE-friendly polynomial kernel G_{he} shown in Eq. 1. Given the degree parameter $\Gamma \in \mathbb{N}$, the derivative of G_{he} can be computed with a constant multiplicative factor using $\Gamma + 1$ multiplications and 2 subtraction operations.

$$G_{\text{he}}(\boldsymbol{x}, \boldsymbol{y}) = (1 - \|\boldsymbol{x} - \boldsymbol{y}\|^2)^{2^\Gamma + 1}. \tag{1}$$

HE-Meanshift uses a fixed bandwidth parameter $h = 1$ in the kernel density estimator (KDE) used to compute the density function in Mean-shift. h is a smoothing parameter for the density function and it is the only parameter for the classical Mean-shift algorithm. Although h is fixed in HE-Meanshift, the Γ parameter still offers some flexibility by amplifying the distance between the data points.

Due to the computation overhead of FHE, the authors adopt a random sampling strategy called *dust sampling* which involves sampling K_d points called dusts from the dataset to reduce the $\mathcal{O}(N^2)$ complexity of the original Mean-shift algorithm. HE-Meanshift then performs the mode-seeking on the dusts instead on all points in the dataset. Recall that modes are points corresponding to local maxima in the KDE and represent areas of high density. In Mean-shift, each point is mapped to the cluster containing the closest mode and the number of clusters is equal to the number of distinct modes. Thus, HE-Meanshift can use a relatively low value for K_d that is at least equal to the number of clusters K to compute all clusters in the dataset. This not only improves the efficiency of the mode-seeking but also reduces the costs for bootstrapping, which is now proportional to K_d. However, setting K_d requires prior information about the

number of clusters in the dataset in contrast to the plaintext Mean-shift where only a value for h is needed.

After a predefined number of iterations, each point in the input dataset is assigned a cluster label based on the final value of the dusts. Since two or more sampled dusts might converge to the same mode and hence the same cluster, HE-Meanshift uses a secure PointLabeling algorithm to robustly assign records to clusters. The Inv and MaxIdx protocols of [146] are used for division and comparison.

HE-Meanshift does not require communication except from sending and receiving the data to/from the untrusted processing party. It is usable for a single data owner outsourcing the clustering. However, the protocol is not usable for most outsourcing scenarios where multiple data owners cluster their joint data since the encrypted output can be decrypted only by a single data owner.

PCA/OPT [19]. Meng et al. [19] introduce two-party privacy-preserving hierarchical clustering (HC) protocols with single and complete linkage (cf. §2.1) in the semi-honest security model using additively homomorphic encryption [65] and Yao's Garbled Circuits [21]. In contrast to the protocols discussed before in §3, they do not return the resulting clusters/its indices, but a dendrogram (cf. §2.1) indicating the clustering's merging history and metadata containing statistical information about each merge like the new cluster's size and a representative element/centroid. To limit information leakage through this returned metadata, the protocols output only metadata of sufficiently large merges or of the final clusters.

In the baseline protocol, called *PCA*, the two parties calculate the pairwise squared Euclidean distances between the clusters using the additively homomorphic property of Paillier encryption [65]. Then, both parties get access to the plaintext values of the distance matrix blinded with random values such that they can collaboratively cluster the input elements and update the merging dendrogram leveraging two GC-protocols that determine the minimum/maximum distance.

The authors introduce an extension of *PCA* called *OPT* that reduces HC's computation complexity of $O(N^3)$, where N is the dataset size, with single linkage by leveraging the symmetry of the minimum distance. This accelerates the search for the next pair of clusters that have to be merged by a factor of N.

PCA can also be extended to the outsourcing scenario [70] where an arbitrary number of data owners secret share their input data among two non-colluding servers that run the privacy-preserving clustering. However, an extension to more than two servers is not straightforward due to the usage of GCs.

ppDBSCAN [18]. Bozdemir et al. [18] propose a privacy-preserving DBSCAN [60] protocol in the semi-honest security setting using the ABY framework [10]. We call this protocol ppDBSCAN in the following. ppDBSCAN can either be used as secure two-party computation protocol or in an outsourcing scenario with two computing parties, e.g., servers, and an arbitrary number of data owners. The authors point out that the post-processing can be adapted to provide an arbitrary output, e.g., cluster labels, cluster sizes, etc. By assessing the needed recursive depth of the neighborhood exploration ppDBSCAN's complexity can be reduced to a low cubic complexity (from normal cubic complexity). All computations are done on integers.

Initially, the data owners arithmetically share their input records among the two non-colluding parties (which are potentially represented by themselves). Then, the pair-wise squared Euclidean distances are computed between all data records in Arithmetic Sharing [20] to assess which elements have sufficiently many neighbors (i.e., lie in a dense area) to form a cluster. The results are stored as binary values to enhance the the efficiency of the clustering process mostly done with GC [21]. Additionally, the distance computation and the cluster expansion is also parallelized with SIMD operations.

C Additional Benchmarking Results

Fig. 3 summarizes the WAN runtimes of the fully private clustering protocols on small datasets. Fig. 4 depicts the memory consumption of the fully private clustering protocols for a small and large dataset. Fig. 5 summarizes the runtime of HE-Meanshift and MPC-KMeans on large datasets over LAN network.

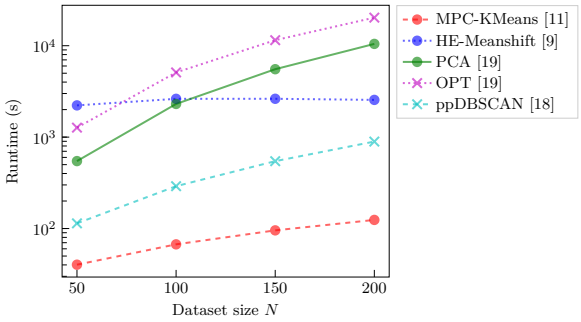


Fig. 3. WAN runtime in seconds of the private clustering protocols MPC-KMeans [11], HE-Meanshift [9], PCA/OPT [19], and ppDBSCAN [18] for varying dataset size N, K=2 clusters, dimension $d = 8$, and bitlength $\ell = 32$.

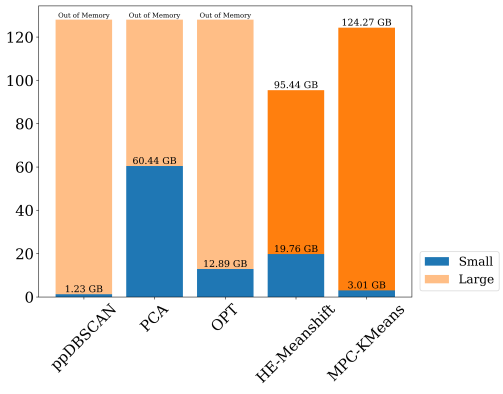


Fig. 4. Memory consumption in GB of the privacy-preserving clustering protocols ppDBSCAN [18], PCA/OPT [19], HE-Meanshift [9], and MPC-KMeans [11] for a small ($N = 200, d = 8, K = 10$) and large ($N = 65536, d = 4, K = 20$) dataset.

D Additional Clustering Quality Evaluation

We compare the clustering quality on nine widely used datasets: Hepta (Tab. 6), Lsun (Tab. 7), Target (Tab. 8), Wingnut (Tab. 9), Tetra (Tab. 10), Chainlink (Tab. 11), and EngyTime (Tab. 12) from [122]; and Dense (Tab. 13) and ZigZag Noisy (Tab. 14) from [123]. Each dataset has different characteristics such as different cluster shapes or different densities.

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.883	519.937
MPC-KMeans	0.869	0.946	0.754	292.539
KMeans++	1.0	1.0	0.883	519.937
HE-Meanshift	0.667	0.834	0.603	156.518
Mean-shift	1.0	1.0	0.883	519.937
ppDBSCAN	1.0	1.0	0.609	384.439
OPT	1.0	1.0	0.883	519.937
PCA	1.0	1.0	0.883	519.937

Table 6. Hepta

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.609	384.439
MPC-KMeans	0.405	0.524	0.653	485.003
KMeans++	0.405	0.524	0.653	485.003
HE-Meanshift	0.434	0.537	0.234	220.118
Mean-shift	0.366	0.445	0.589	293.479
ppDBSCAN	1.0	1.0	0.609	384.439
OPT	1.0	1.0	0.609	384.439
PCA	0.405	0.529	0.641	458.157

Table 7. Lsun

Algorithm	ARI	AMI	SI	CHI
Ground Truth 1	-	-	0.260	27.869
Ground Truth 2	-	-	0.249	0.494
MPC-KMeans	0.534	0.615	0.678	709.651
KMeans++	0.611	0.639	0.742	738.291
HE-Meanshift	0.215	0.311	0.383	101.586
Mean-shift	0.626	0.645	0.766	590.057
ppDBSCAN	1.0	1.0	0.249	0.494
OPT	1.0	1.0	0.249	0.494
PCA	0.207	0.377	0.506	90.502

Table 8. Dataset Target

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.630	1061.016
MPC-KMeans	0.417	0.326	0.567	805.066
KMeans++	0.425	0.334	0.570	815.492
HE-Meanshift	0.475	0.451	0.373	611.832
Mean-shift	0.638	0.538	0.621	1027.873
ppDBSCAN	1.0	1.0	0.630	1061.016
OPT	1.0	1.0	0.630	1061.016
PCA	1.0	1.0	0.630	1061.016

Table 9. Dataset Wingnut

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.726	418.391
MPC-KMeans	0.961	0.975	0.689	390.920
KMeans++	1.0	1.0	0.726	418.391
HE-Meanshift	0.518	0.587	0.124	109.916
Mean-shift	1.0	1.0	0.726	418.391
ppDBSCAN	0.94	0.94	0.694	391.819
OPT	0.000	0.000	-0.436	1.472
PCA	0.987	0.982	0.718	409.221

Table 10. Dataset Tetra

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.179	250.865
MPC-KMeans	0.088	0.065	0.525	718.934
KMeans++	0.087	0.064	0.525	718.788
HE-Meanshift	0.132	0.160	0.262	353.929
Mean-shift	0.223	0.272	0.405	574.488
ppDBSCAN	1.0	1.0	0.179	250.865
OPT	1.0	1.0	0.179	250.865
PCA	0.313	0.388	0.463	575.488

Table 11. Chainlink

Algorithm	ARI	AMI	SI	CHI
Ground Truth 1	-	-	0.557	2921.700
Ground Truth 2	-	-	0.577	3075.082
MPC-KMeans	0.844	0.783	0.578	3158.931
KMeans++	0.843	0.783	0.578	3158.931
HE-Meanshift	0.801	0.720	0.198	1681.223
Mean-shift	0.833	0.769	0.578	3176.809
ppDBSCAN	0.612	0.493	-0.416	115.717
OPT	0.000	0.000	0.479	8.044
PCA	0.042	0.150	0.472	1318.72

Table 12. Dataset EngyTime

Algorithm	ARI	AMI	SI	CHI
Ground Truth	-	-	0.740	433.656
MPC-KMeans	0.756	0.713	0.790	497.182
KMeans++	0.768	0.723	0.790	499.284
HE-Meanshift	0.838	0.779	0.540	277.155
Mean-shift	0.784	0.725	0.699	309.728
ppDBSCAN	0.935	0.904	0.762	503.083
OPT	0.000	0.019	0.490	15.626
PCA	0.257	0.348	0.631	231.817

Table 13. Dense

Algorithm	ARI	AMI	SI	CHI
Ground Truth 1	-	-	-0.050	30.858
Ground Truth 2	-	-	0.500	317.869
MPC-KMeans	0.497	0.636	0.489	321.627
KMeans++	0.519	0.655	0.540	362.227
HE-Meanshift	0.498	0.648	0.538	368.285
Mean-shift	0.542	0.699	0.510	351.971
ppDBSCAN	1.0	1.0	-0.050	30.858
OPT	1.0	1.0	-0.040	30.858
PCA	0.521	0.672	0.504	371.251

Table 14. Dataset ZigZag Noisy

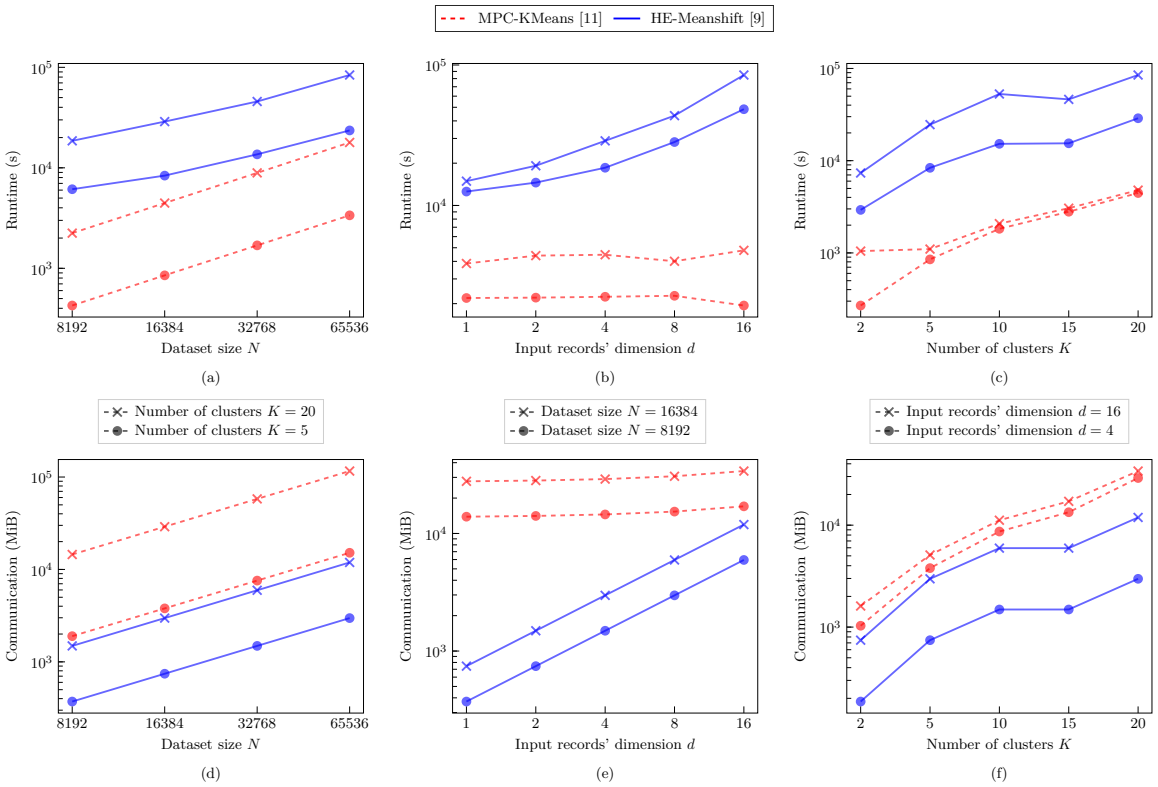


Fig. 5. LAN runtimes in seconds (top row) and **communication** in MiB (bottom row) of the fully-private clustering protocols MPC-KMeans [11] and HE-Meanshift [9] for varying dataset size N, input records' dimension d, number of clusters K, and bitlength $\ell = 32$. In (a) and (d) is $d = 4$, in (b) and (e) $K = 20$, and in (c) and (f) $N = 16384$.

B FLAME: Taming Backdoors in Federated Learning (USENIX Security'22)

[NRC+22] T. D. NGUYEN, P. RIEGER, H. CHEN, H. YALAME, H. MÖLLERING, H. FEREIDOONI, S. MARCHAL, M. MIETTINEN, A. MIRHOSEINI, S. ZEITOUNI, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER. **"FLAME: Taming Backdoors in Federated Learning"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2021/025. USENIX Association, 2022, pp. 1415–1432. CORE Rank **A***. Appendix B.

https://encrypto.de/papers/NRCYMFMMMZKSS22.pdf

FLAME: Taming Backdoors in Federated Learning

Thien Duc Nguyen[1], Phillip Rieger[1], Huili Chen[2], Hossein Yalame[1], Helen Möllering[1], Hossein Fereidooni[1], Samuel Marchal[3], Markus Miettinen[1], Azalia Mirhoseini[4], Shaza Zeitouni[1], Farinaz Koushanfar[2], Ahmad-Reza Sadeghi[1], and Thomas Schneider [1]
[1]*Technical University of Darmstadt, Germany*[*]; [2]*University of California San Diego, USA*; [3]*Aalto University and F-Secure, Finland*; [4]*Google, USA*

Abstract

Federated Learning (FL) is a collaborative machine learning approach allowing participants to jointly train a model without having to share their private, potentially sensitive local datasets with others. Despite its benefits, FL is vulnerable to so-called *backdoor attacks*, in which an adversary injects manipulated model updates into the federated model aggregation process so that the resulting model will provide targeted false predictions for specific adversary-chosen inputs. Proposed defenses against backdoor attacks based on detecting and filtering out malicious model updates consider only very specific and limited attacker models, whereas defenses based on differential privacy-inspired noise injection significantly deteriorate the benign performance of the aggregated model. To address these deficiencies, we introduce FLAME, a defense framework that estimates the sufficient amount of noise to be injected to ensure the elimination of backdoors. To minimize the required amount of noise, FLAME uses a model clustering and weight clipping approach. This ensures that FLAME can maintain the benign performance of the aggregated model while effectively eliminating adversarial backdoors. Our evaluation of FLAME on several datasets stemming from application areas including image classification, word prediction, and IoT intrusion detection demonstrates that FLAME removes backdoors effectively with a negligible impact on the benign performance of the models.

1 Introduction

Federated learning (FL) is an emerging collaborative machine learning trend with many applications, such as next word prediction for mobile keyboards [39], medical imaging [49], and intrusion detection for IoT [44] to name a few. In FL, clients locally train models based on local training data and then provide these model updates to a central aggregator who combines them into a global model. The global model is then propagated back to the clients for the next training iteration.

FL promises efficiency and scalability as the training is distributed among many clients and executed in parallel. In particular, FL improves privacy by enabling clients to keep their training data locally [38]. Despite its benefits, FL has been shown to be vulnerable to so-called *poisoning attacks* where the adversary manipulates the local models of a subset of clients participating in the federation so that the malicious updates get aggregated into the global model. Untargeted poisoning attacks merely aim at deteriorating the performance of the global model and can be defeated by validating the performance of uploaded models [12]. In this paper, we therefore focus on the more challenging problem of *backdoor attacks* [7, 45, 57, 59], i.e., targeted poisoning attacks in which the adversary seeks to stealthily manipulate the resulting global model in a way that attacker-controlled inputs result in incorrect predictions chosen by the adversary.

Deficiencies of existing defenses. Existing defenses against backdoor attacks can be roughly divided into two categories: The first one comprises anomaly detection-based approaches [4,9,22,51] for identifying and removing potentially poisoned model updates. However, these solutions are effective only under very specific adversary models, as they make detailed assumptions about the attack strategy of the adversary and/or the underlying distribution of the benign or adversarial datasets. If these very specific assumptions do not hold, the defenses may fail. The second category is inspired by differential privacy (DP) techniques [7,56], where individual weights[1] of model updates are clipped to a maximum threshold and random noise is added to the weights for diluting/reducing the impact of potentially poisoned model updates on the aggregated global model. In contrast to the first category, DP techniques [7,56] are applicable in a generic adversary model without specific assumptions about adversarial behavior and data distributions and are effective in eliminating the impact of malicious model updates. However, straightforward application of DP approaches severely deteriorates the benign

[*]Emails: {ducthien.nguyen, ahmad.sadeghi}@trust.tu-darmstadt.de

[1]Parameters of neural network models typically consist of 'weights' and 'biases'. For the purposes of this paper, however, these parameters can be treated identically and we will refer to them as 'weights' for brevity.

performance of the aggregated model because the amount of noise required to ensure effective elimination of backdoors also results in significant modifications of individual weights of benign model updates [7, 57].

In this paper, we develop a resilient defense against backdoors by combining the benefits of both defense types without suffering from the limitations (narrow attacker model, assumptions about data distributions) and drawbacks (loss of benign performance) of existing approaches. To this end, we introduce an approach in which detection of anomalous model updates and tuned clipping of weights are combined to minimize the amount of noise needed for backdoor removal of the aggregated model while preserving its benign performance.

Our Goals and Contributions. We present FLAME, a resilient aggregation framework for FL that eliminates the impact of backdoor attacks while maintaining the benign performance of the aggregated model. This is achieved by three modules: DP-based noising of model updates to remove backdoor contributions, automated model clustering approach to identify and eliminate potentially poisoned model updates, and model weight clipping before aggregation to limit the impact of malicious model updates on the aggregation result. The last two modules can significantly reduce the amount of random noise required by DP noising for backdoor elimination. In particular, our contributions are as follows:

- We present FLAME, a defense framework against backdoor attacks in FL that is capable of eliminating backdoors without impacting the benign performance of the aggregated model. Contrary to earlier backdoor defenses, FLAME is applicable in a *generic* adversary model, i.e., it does not rely on strong assumptions about the attack strategy of the adversary, nor about the underlying data distributions of benign and adversarial datasets (§4.1).
- We show that the amount of required Gaussian noise can be radically reduced by: a) applying our clustering approach to remove potentially malicious model updates and b) clipping the weights of local models at a proper level to constrain the impact of individual (especially malicious) models on the aggregated model. (§4.3)
- We provide a noise boundary proof for the amount of Gaussian noise required by noise injection (inspired by DP) to eliminate backdoor contributions (§5).
- We extensively evaluate our defense framework on real-world datasets from three very different application areas. We show that FLAME reduces the amount of required noise so that the benign performance of the aggregated model does not degrade significantly, providing a crucial advantage over state-of-the-art defenses using straightforward injection of DP-based noise (§7).

As an orthogonal aspect, we also consider how the privacy of model updates against an honest-but-curious aggregator can be preserved and develop a secure multi-party computation approach that can preserve the privacy of individual model updates while realizing our backdoor defense approach (§8).

2 Background and Problem Setting

2.1 Federated Learning

Federated Learning [38, 50] is a concept for distributed machine learning that links n clients and an aggregator to collaboratively build a global model G. In a training iteration $t \in \{1,\ldots,T\}$, each client $i \in \{1,\ldots,n\}$ locally trains a local model W_i with p parameters (indicating both weights and biases) $w_i^1,\ldots,w_i^p$ based on the previous global model G_{t-1} using its local data D_i and sends it to the aggregator which aggregates the received models W_i into the global model G_t.

Several aggregation mechanisms have been proposed recently: 1) *Federated Averaging* (FedAvg) [38], 2) *Krum* [9], 3) *Adaptive Federated Averaging* [42], and 4) *Trimmed mean or median* [60]. Although we evaluate FLAME's effectiveness on several aggregation mechanisms in §7.1, we generally focus on FedAvg in this work as it is commonly applied in FL [21, 28, 39, 44, 47, 50, 54] and related work on backdoor attacks [7, 22, 51, 57, 59]. In FedAvg, the global model is updated by averaging the weighted models as follows: $G_t = \Sigma_{i=1}^{n} s_i \times W_i / s$, where $s_i = \|D_i\|, s = \Sigma_{i=1}^{n} s_i$. However, in practice, a malicious client might provide falsified information about its dataset size (i.e., a large number) to amplify the relative weight of its updates [57]. Previous works often employed equal weights ($s_i = 1/n$) for the contributions of all clients [7, 51, 59]. We adopt this approach in this paper, i.e., we set $G_t = \Sigma_{i=1}^{n} W_i / n$. Further, other state-of-the-art aggregation rules, e.g., *Krum* [9], *Adaptive Federated Averaging* [42], and *Trimmed mean or median* [60] also do not consider the sizes of local training datasets by design.

2.2 Backdoor Attacks on Federated Learning

In backdoor attacks, the adversary $\mathcal{A}$ manipulates the local models W_i of k compromised clients to obtain poisoned models W_i' that are then aggregated into the global model G_t and thus affect its properties. In particular, $\mathcal{A}$ wants the poisoned model G_t' to behave normally on all inputs except for specific attacker-chosen inputs $x \in I_{\mathcal{A}}$ (where $I_{\mathcal{A}}$ denotes the so-called *trigger set*) for which attacker-chosen (incorrect) predictions should be output. Figure 1 shows common techniques used in FL backdoor attacks, including 1) data poisoning, e.g., [45, 51, 59], where $\mathcal{A}$ manipulates training datasets of models, and 2) model poisoning, e.g., [7, 57] where $\mathcal{A}$ manipulates the training process or the trained models themselves. Next, we will briefly discuss these attack techniques.

Data Poisoning. In this attack, $\mathcal{A}$ adds manipulated data $D^{\mathcal{A}}$ to the training datasets of compromised clients i by flipping data labels, e.g., by changing the labels of a street sign database so that pictures showing a 30 km/h speed limit are labeled as 80 km/h [51], or, by adding triggers into data samples (e.g., a specific pixel pattern added to images [59]) in combination with label flipping. We denote the fraction

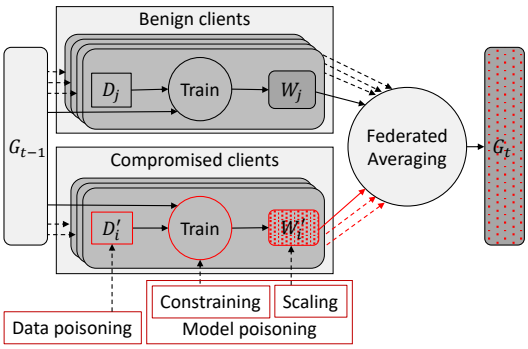


Figure 1: An overview of backdoor attacks.

of injected poisoned data $D_i^{\mathcal{A}}$ in the overall poisoned training dataset D'_i of client i as *Poisoned Data Rate (PDR)*, i.e., $PDR_i = |D_i^{\mathcal{A}}|/|D'_i|$.

Model Poisoning. This attack technique requires that $\mathcal{A}$ can fully control a number of clients. $\mathcal{A}$ poisons the training datasets of these clients and manipulates how they execute the training process by modifying parameters and scaling the resulting model update to maximize the attack impact while evading the aggregator's anomaly detector [7, 57]. This is done by (1) scaling up the weights of malicious model updates to maximize attack impact (e.g., *model-replacement* attack [7], or, *projected gradient descent (PGD) attack with model replacement* [57]), or, scaling down model updates to make them harder to detect (e.g., *train-and-scale* [7]) and (2) constraining the training process itself to minimize the deviation of malicious models from benign models to evade anomaly detection (e.g., *constrain-and-scale* attack [7]).

2.3 Adversary Goals and Capabilities

The goals of the adversary are two-fold:

Impact: The adversary $\mathcal{A}$ aims to manipulate the global model G so that the modified model G' provides incorrect predictions $f(G',x) = c' \neq f(G,x)$ for any inputs $x \in I_{\mathcal{A}}$, where $I_{\mathcal{A}}$ is the so-called *trigger set* consisting of specific attacker-chosen inputs and c' denotes the incorrect prediction chosen by the adversary.

Stealthiness: To make the poisoned model G' hard to detect by aggregator A, it should closely mimic the behavior of G on all other inputs not in $I_{\mathcal{A}}$, i.e.:

$$f(G',x) = \begin{cases} c' \neq f(G,x) & \forall x \in I_{\mathcal{A}} \\ f(G,x) & \forall x \notin I_{\mathcal{A}} \end{cases} \quad (1)$$

Additionally, to make poisoned models as indistinguishable as possible from benign models, the distance (e.g., euclidean) between a poisoned model W' and a benign model W must be smaller than a threshold η denoting the distinction capability of the anomaly detector of aggregator A, i.e., $dist(W,W') < \eta$. The adversary can estimate this distance by comparing the local malicious model to the global model or to a local model trained on benign data.

Adversarial Capabilities. In this paper, we make no specific assumptions about the adversary's behavior. We assume that the adversary $\mathcal{A}$ has full control over $k < \frac{n}{2}$ clients and their training data, processes, and parameters [7, 59]. We denote the fraction of compromised clients as Poisoned Model Rate $PMR = \frac{k}{n}$. Furthermore, $\mathcal{A}$ has full knowledge of the aggregator's operations, including potentially applied backdoor defenses. However, $\mathcal{A}$ has no control over any processes executed at the aggregator nor over the honest clients.

2.4 Preliminaries

HDBSCAN [11] is a density-based clustering algorithm that uses the distance of data points in n-dimensional space to group data points that are located near each other together into a cluster. Hereby the number of clusters is determined dynamically. Data points that do not fit to any cluster are considered outliers. However, while HDBSCAN's predecessor DBSCAN [19] uses a predefined maximal distance to determine whether two points belong to the same cluster, HDBSCAN determines this maximal distance for each cluster independently, based on the density of points. Thus, in HDBSCAN, neither the maximal distance nor the total number of clusters need to be predefined.

Differential Privacy (DP). DP is a privacy technique that aims to ensure that the outputs do not reveal individual data records of participants. DP is formally defined as follows:

Definition 1 ((ε,δ)-differential privacy)**.** *A randomized algorithm $\mathcal{M}$ is (ε,δ)-differentially private if for any datasets D_1 and D_2 that differ on a single element, and any subset of outputs $S \in Range(\mathcal{M})$, the following inequality holds:*

$$Pr[\mathcal{M}(D_1) \in S] \leq e^{\varepsilon} \cdot Pr[\mathcal{M}(D_2) \in S] + \delta.$$

Here, ε denotes the privacy bound and δ denotes the probability of breaking this bound [18]. Smaller values of ε and δ indicate stronger privacy. A commonly used approach to enforce differential privacy is adding random Gaussian noise $N(0,\sigma^2)$ to the output of the algorithm [3, 18].

3 Problem Setting and Objectives

Backdoor Characterization. Following common practice in FL-related papers (e.g., [7, 12, 22]), we represent Neural Networks (NNs) using their weight vectors, in which the extraction of weights is done identically for all models by flattening/serializing the weight/bias matrices in a predetermined order. Figure 2 shows an abstract two-dimensional representation of the weight vectors of local models compared to the global model G_{t-1} of the preceding aggregation round. Each model W_i can be characterized with two factors: *direction (angle)* and *magnitude (length)* of its weight vector $(w^1, w^2, \ldots, w^p)$. The angle between two updates W_i and W_j can be measured, e.g., by using the cosine distance metric c_{ij} as defined in (2) while their magnitude difference is measured by the L_2-norm e_{ij} as defined in (3).

$$c_{ij} = 1 - \frac{W_i W_j}{\|W_i\| \|W_j\|} = 1 - \frac{\sum_{k=1}^{p} w_i^k w_j^k}{\sqrt{\sum_{k=1}^{p} (w_i^k)^2} \sqrt{\sum_{k=1}^{p} (w_j^k)^2}} \quad (2)$$

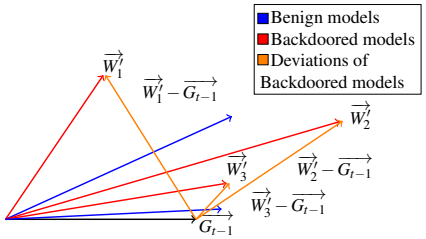


Figure 2: Weight vectors of benign and backdoored models.

$$e_{ij} = \|W_i - W_j\| = \sqrt{\sum_{k=1}^{p} (w_i^k - w_j^k)^2} \tag{3}$$

Benign and backdoored local models are shown in blue and red colors and are labeled with W_i or W'_i, respectively. Note that the benign models are typically not identical due to the potentially partially non-iid nature of their training data.

The impact of the adversarial goal (injection of a backdoor) causes a deviation in the model parameters that manifests itself as a difference in the direction and/or magnitude of the backdoored model's weight vector in comparison to benign models, e.g., the deviations among local models and to the global model G_{t-1} of the previous aggregation round. Since the adversary has full control over the training process of compromised clients, he can fully control these distances, e.g., by changing the direction (in the case of W'_1) or magnitude (in the case of W'_2) of the backdoored models' weight vectors.

Figure 2 also shows three kinds of backdoored models resulting from different types of backdoor attacks. The first type W'_1 has a similar weight vector, but a *large angular deviation* from the majority of local models and the global model. This is because such models are trained to obtain high accuracy on the backdoor task, which can be achieved by using a large poisoned data rate (*PDR*) or a large number of local training epochs (cf. Distributed Backdoor Attack (DBA) [59]). The second backdoor type W'_2 has a small angular deviation but a *large magnitude* to amplify the impact of the attack. Such models can be crafted by the adversary by *scaling up* the model weights to boost its effect on the global model (cf. *Model-replacement* attack in [7]). The third backdoor type W'_3 has a similar weight vector as benign models, the angular difference and the magnitude are not substantially different compared to benign models and, thus less distinguishable from benign models. Such *stealthy* backdoored models can be crafted by the adversary by carefully constraining the training process or scaling down the poisoned model's weights (cf. *Constrain-and-scale* attack [7] or FLIoT attack [45]).

Defense Objectives. A generic defense that can effectively mitigate backdoor attacks in the FL setting needs to fulfill the following objectives: (i) *Effectiveness*: To prevent the adversary from achieving its attack goals, the impact of backdoored model updates must be eliminated so that the aggregated global model does not demonstrate backdoor behavior. (ii) *Performance*: Benign performance of the global model must be preserved to maintain its utility. (iii) *Independence from data distributions and attack strategies*: The defense method must be applicable to generic adversary models, i.e., it must not require prior knowledge about the backdoor attack method, or make assumptions about specific data distributions of local clients, e.g., whether the data are iid or non-iid.

4 FLAME Overview and Design

We present the high-level idea of FLAME and the associated design challenges to fulfill the objectives identified in §3.

4.1 High-level Idea

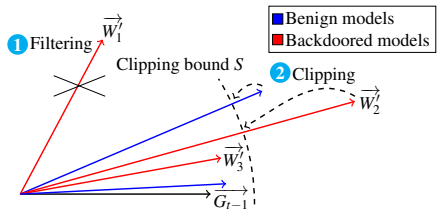
Motivation. Earlier works (e.g., Sun *et al.* [56]) use differential privacy-inspired noising of the aggregated model for eliminating backdoors. They determine the sufficient amount of noise to be used empirically. In the FL setting this is, however, challenging, as one cannot in general assume the aggregator to have access to training data, in particular to poisoned datasets. What is therefore needed is a generic method for determining how much noise is sufficient to remove backdoors effectively. On the other hand, the more noise is injected into the model, the more its benign performance will be impacted.

FLAME Overview. FLAME estimates the noise level required for backdoor removal in the FL setting without extensive empirical evaluation and having access to training data (this noise bound is formally proven in §5). In addition, to effectively limit the amount of required noise, FLAME uses a novel clustering-based approach to identify and remove adversarial model updates with high impact and applies a dynamic weight-clipping approach to limit the impact of models that the adversary has scaled up to boost their performance. As discussed in §3, one cannot guarantee that all backdoored models can be detected since the adversary can fully control both the angular and magnitude deviation to make the models arbitrarily hard to detect. Our clustering approach therefore aims to remove models with high attack impact (having larger angular deviation) rather than all malicious models. Fig. 3 illustrates the high-level idea of FLAME consisting of the above three components: filtering, clipping, and noising. We emphasize, however, that each of these components needs to be applied with great care, since, a naïve combination of noising with clustering and clipping leads to poor results as it easily fails to mitigate the backdoor and/or deteriorates the benign performance of the model, as we show in §C. We detail the design of each component and its use in the FLAME defense approach in §4.3.

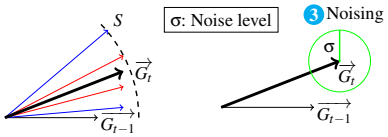
4.2 Design Challenges

To realize the high-level idea presented above, we need to solve the following technical challenges.

C_1- Filtering out backdoored models with large angular deviations in dynamic scenarios. As discussed in §3, the weight vector of a well-trained backdoored model, W', has a *higher angular difference* in comparison to weight vectors of benign models W. FLAME deploys a clustering approach to



(a) Applying clustering and clipping.



(b) Applying noising

Figure 3: High-level idea of FLAME defense.

identify such poisoned models and *remove* them from FL aggregation (detailed in §4.3.1). The effect of clustering-based filtering is shown in Fig. 3a where model W_1' is removed from the aggregated model as it does not align with the directions of benign models. In contrast to existing clustering-based defenses, we need an approach that can also work in a *dynamic attack setting*, i.e., the number of injected backdoors is unknown and may vary between training rounds. To this end, we make a key observation: clustering approaches using a fixed number of clusters $n_{cluster}$ for identifying malicious models are inherently vulnerable to attacks with varying numbers of backdoors[2] $n_{backdoor}$. This is because the adversary can likely cause at least one backdoor model to be clustered together with benign models due to the pigeonhole principle by simultaneously injecting $n_{backdoor} \geq n_{cluster}$ backdoors. We seek to solve this challenge by employing a clustering solution that dynamically determines the clusters for model updates, thereby allowing it to adapt to dynamic attacks.

C_2-Limiting the impact of scaled-up backdoors. To limit the impact of backdoored models that the adversary artificially scales up to boost the attack (e.g., W_2' in Fig. 2), the weight vectors of models with high magnitudes can be clipped [56]. The effect of clipping is shown in Fig. 3a where the weight vectors of all models with a magnitude beyond the clipping bound S (in particular, backdoored model W_2') are clipped to S by scaling down the weight vectors. The resulting clipped weight vectors are shown on the left side of Fig. 3b. The challenge here is how to select a proper clipping bound without empirically evaluating its impact on the training datasets (which are not available in the FL setting). If the applied clipping bound is too large, an adversary can boost its model W' by scaling its weights up to the clipping bound, thereby maximizing the backdoor impact on the aggregated global model G. However, if the applied clipping bound is too small, a large fraction of benign model updates W will be clipped, thereby leading to performance deterioration of the aggregated global model G on the main task. We tackle this challenge in §4.3.2, where we show how to select a clipping bound that can not be influenced by the adversary and that effectively limits the impact of scaled-up backdoored models.

C_3-Selecting suitable noise level for backdoor elimination. As mentioned in §4.1, FLAME uses model noising that applies Gaussian noise with noise level σ to mitigate the adversarial impact of backdoored models (e.g., W_3' in Fig. 2). Similar to the clipping bound, however, also here the noise level σ must be carefully selected, as it has a direct impact on the effectiveness of the defense and the model's benign performance. If it is too low, the aggregated model might retain backdoor behavior after model noising, rendering the defense ineffective, while excessive noise will degrade the utility of the aggregated model. To address this challenge, we develop an approach for reliably estimating a sufficient but minimal bound for the applied noise in §5.

4.3 FLAME Design

As discussed in §4.1, our defense consists of three main components: filtering, clipping, and noising. Figure 4 shows these components and the workflow of FLAME during training round t. Algorithm 1 outlines the procedure of FLAME. In the rest of this section, we detail the design of these components to resolve the challenges in §4.2.

Algorithm 1 FLAME

Input: n, G_0, T ▷ n is the number of clients, G_0 is the initial global model, T is the number of training iterations
Output: G_T^* ▷ G_T^* is the updated global model after T iterations
for each training iteration t in $[1,T]$ **do**
  for each client i in $[1,n]$ **do**
    $W_i \leftarrow \text{CLIENTUPDATE}(G_{t-1}^*)$ ▷ The aggregator sends G_{t-1}^* to Client i who trains G_{t-1}^* using its data D_i locally to achieve local modal W_i and sends W_i back to the aggregator.
  $(c_{11},\ldots,c_{nn}) \leftarrow \text{COSINEDISTANCE}(W_1,\ldots,W_n)$ ▷ $\forall i,j \in (1,\ldots,n)$, c_{ij} is the cosine distance between W_i and W_j
  $(b_1,\ldots,b_L) \leftarrow \text{CLUSTERING}(c_{11},\ldots,c_{nn})$ ▷ L is the number of admitted models, b_l is the index of the l^{th} model
  $(e_1,\ldots,e_n) \leftarrow \text{EUCLIDEANDISTANCES}(G_{t-1}^*,(W_1,\ldots,W_n))$ ▷ e_i is the Euclidean distance between G_{t-1}^* and W_i
  $S_t \leftarrow \text{MEDIAN}(e_1,\ldots,e_n)$ ▷ S_t is the adaptive clipping bound at round t
  for each client l in $[1,L]$ **do**
    $W_{b_l}^c \leftarrow G_{t-1} + (W_{b_l} - G_{t-1}) \cdot \text{MIN}(1,\gamma)$ ▷ Where γ $(= S_t/e_{b_l})$ is the clipping parameter, $W_{b_l}^c$ is the admitted model after clipped by the adaptive clipping bound S_t
  $G_t \leftarrow \sum_{l=1}^{L} W_{b_l}^c / L$ ▷ Aggregating, G_t is the plain global model before adding noise
  $\sigma \leftarrow \lambda \cdot S_t$ where $\lambda = \frac{1}{\varepsilon} \cdot \sqrt{2ln\frac{1.25}{\delta}}$ ▷ Adaptive noising level
  $G_t^* \leftarrow G_t + N(0,\sigma^2)$ ▷ Adaptive noising

4.3.1 Dynamic *Model Filtering*

The *Model Filtering* component of FLAME utilizes a *dynamic clustering* technique based on HDBSCAN [11] that

[2] We consider two backdoors to be independent if they use different triggers.

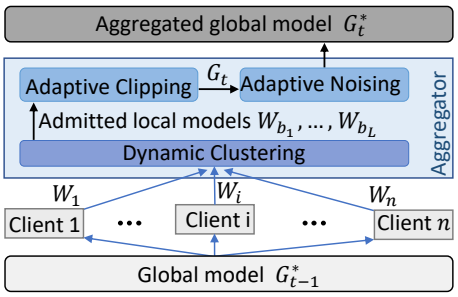


Figure 4: Illustration of FLAME's workflow in round t.

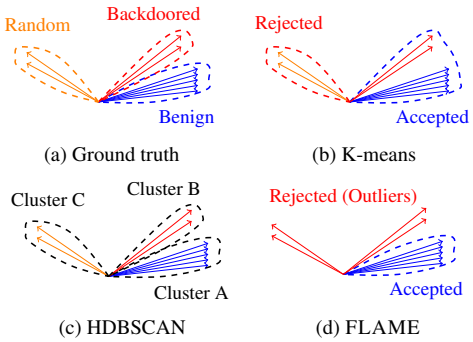


Figure 5: Comparison of clustering quality for (a) ground truth, (b) using K-means with 2 clusters as in Auror [51], (c) straightforward applied HDBSCAN and (d) our approach as in FLAME.

identifies poisoned models with high angular deviations from the majority of updates (e.g., W_1' in Fig. 3a). Existing clustering-based defenses [9, 51] identify potentially malicious model updates by clustering them into two groups where the smaller group is always considered malicious and thus removed. However, if no malicious models are present in the aggregation, this approach may lead to many models being incorrectly removed and thus a reduced accuracy of the aggregated model. These approaches also do not protect against attacks in which adversary $\mathcal{A}$ simultaneously injects multiple backdoors by using different groups of clients to inject different backdoors. If the number of clusters is fixed, there is the risk that poisoned and benign models end up in the same cluster, in particular, if models with different backdoors differ significantly. Consequently, existing model clustering methods do not adequately address challenge C_1 (§4.2). Fig. 5 shows the behavior of different clustering methods on a set of model updates' weight vectors. Fig. 5a shows the ground truth of an attack scenario where $\mathcal{A}$ uses two groups of clients: one group is used to inject a backdoor, whereas the other group provides random models with the goal of fooling clustering-based defenses. Fig. 5b shows how in this setting, K-means (as used in Auror [51]) fails to successfully separate benign and poisoned models as all poisoned models end up in the same cluster with the benign models.

To overcome the limitations of existing defenses, we design our clustering solution and ensure that: (i) it is able to handle dynamic attack scenarios where multiple backdoors are injected simultaneously, and (ii) it minimizes false positives of poisoned model identification. In contrast to existing approaches that try to place poisoned models into one cluster, our approach considers each poisoned model individually as an outlier, so that it can gracefully handle multiple simultaneous backdoors and thus address challenge C_1.

FLAME uses pairwise cosine distances to measure the angular differences between all model updates and applies the HDBSCAN clustering algorithm [11]. The advantage here is that cosine distances are not affected even if the adversary scales up model updates to boost their impact as this does not change the angle between the updates' weight vectors. Since the HDBSCAN algorithm clusters the models based on their density of the cosine distance distribution and *dynamically determines the required number of clusters*, we leverage it for our dynamic clustering approach. We describe HDBSCAN and how we apply it in detail in §E. In particular, HDBSCAN labels models as outliers if they do not fit into any cluster. This allows FLAME to effectively handle multiple poisoned models with different backdoors by labeling them as outliers. To realize this, we set the minimum cluster size to be at least 50% of the clients, i.e., $\frac{n}{2}+1$, so that the resulting cluster will contain the majority of updates (which we assume to be benign, cf. §2.3). All remaining (potentially poisoned) models are marked as *outliers*. This behavior is depicted in Fig. 5d where all the models from Clusters B and C from Fig. 5c are considered as outliers. Hence, to the best of our knowledge, our approach is the first FL backdoor defense that is able to gracefully handle also dynamic attacks in which the number of injected backdoors may vary. The clustering step is shown in lines 6-7 of Alg. 1 where L models are retained after clustering.

4.3.2 Adaptive Clipping and Noising

As discussed in §4.2 (challenges C_2 and C_3), determining a proper clipping bound and noise level for model weight clipping and noising is not straightforward. We present our new approach for selecting an effective clipping bound and reliably estimating a sufficient noise level that can effectively eliminate backdoors while preserving the performance of the main task. Furthermore, our defense approach is resilient to adversaries that dynamically adapt their attacks.

Adaptive Clipping. Fig. 6 shows the variation of the average L_2-norms of model updates of benign clients in three different datasets (cf. §6) over subsequent training rounds. We can observe that the *L_2-norms of benign model updates become smaller in later training rounds*. To effectively remove backdoors while minimizing the impact on benign updates, the clipping bound S needs to be dynamically adapted to this decreasing trend of the L_2-norm. Recall that clipping is performed after clustering by scaling down model weights so that the L_2-norm of the scaled model becomes smaller or equal to the clipping threshold. We describe how FLAME determines a proper scaling factor for each model update W_i in t^{th} training round as follows: Given the index set $(b_1, \ldots b_L)$ of the models admitted by the clustering method (line 7 of

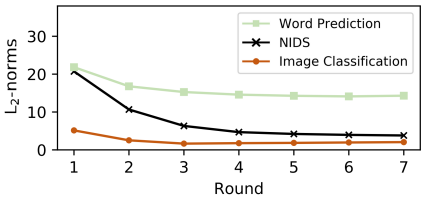


Figure 6: L_2-norms of model updates depending on the number of training rounds for different datasets.

Alg. 1), the aggregator first computes the clipping bound S_t as the median of the L_2-norms of all n model updates: $S_t = \text{MEDIAN}(e_1, \ldots, e_n)$. It should be noted that for determining the clipping bound, the rejected models are also considered to ensure that even if benign models were filtered, the computed median S_t is still determined based on benign values. However, after determining the clipping bound, only the admitted models $W_1, \ldots, W_L$ are considered for later processing. The scaling factor for the l^{th} admitted model is computed as $\gamma = \frac{S_t}{e_{b_l}}$ where e_{b_l} is the L_2-norm of the model update W_{b_l}. Clipping scales down model updates as follows: $W^c_{b_l} = G_{t-1} + (W_{b_l} - G_{t-1}) \cdot \text{MIN}(1, \gamma)$ (detailed in line 8-11 of Alg. 1) where the multiplication is computed coordinate-wise. It is worth noting that weighting contributions (i.e., adjusting scaling factor) based on client data sizes is insecure. As we point out in §2.1, the reported dataset sizes by clients cannot be trusted, i.e., the adversary can lie about their dataset sizes to maximize attack impact [57]. Hence, we follow common practice in literature and weight the contributions of all clients equally regardless of their dataset size [7, 9, 12, 59]. By using the median as the clipping bound S_t, we ensure that S_t is always in the range of the L_2-norms between benign models and the global model since we assume that more than 50% of clients are benign (cf. §2.3). We evaluate the effectiveness of the clipping approach in §B.2.

Adaptive Noising. It has been shown that by adding noise to a model's weights, the impact of outlier samples can be effectively mitigated [17]. Noise can also be added to poisoned samples (special cases of outliers) used in backdoor injection. The more noise is added to the model during the training process, the less responsive the model will be to the poisoned samples. Thus, increasing model robustness against backdoors. Eliminating backdoors utilizing noise addition is conceptually the same in a centralized or federated setting (e.g., [7, 17]): In both cases, noise is added to the model weights to smooth out the effect of poisoned data (cf. Eq. 5). The challenge is to determine as small a noise level as possible to eliminate backdoors and at the same time not deteriorate the benign performance of the model. As we discuss in detail in §5.1, the amount of noise is determined by estimating the sensitivity based on the differences (distances) among local models, which can be done without access to training data. We then add Gaussian noise to the global model G_t to yield a noised global model G^*_t as follows: $G^*_t = G_t + N(0, \sigma^2)$, see Lines 13-14 of Alg. 1 for more details. This ensures that backdoor contributions are effectively eliminated from the aggregated model. In particular, we show in §5.1 how the noise-based backdoor elimination technique can be transferred from a centralized to a federated setting by analysing the relationship between aggregated Gaussian noise applied to the global model and individual noising of each local model.

5 Security Analysis

5.1 Noise Boundary Proof of FLAME

In this section, we provide a proof to corroborate that FLAME can neutralize backdoors in the FL setting by applying strategical noising with bound analysis on the noise level. We first formulate the noise boundary guarantee of FLAME in Theorem 1. Subsequently, we explain related parameters and prove how the noise level bound for σ can be estimated. This is done by generalizing theoretical results from previous works [17, 18] to the FL setting. Then, we show how the filtering and clipping component of FLAME helps to effectively reduce the noise level bound in Theorem 2. We provide a formal proof for linear models and extend the proof to DNNs using empirical evaluation. This is because providing formal proof for DP-based backdoor security for DNN models is still an open research problem even for centralized settings.

Theorem 1. *A (ε, δ)-differentially private model with parameters G and clipping bound S_t is backdoor-free if random Gaussian noise is added to the model parameters yielding a noised version G^* of the model: $G^* \leftarrow G + N(0, \sigma^2_G)$ where the noise scale σ_G is determined by the clipping bound S_t and a noise level factor λ: $\sigma_G \leftarrow \lambda \cdot S_t$ and $\lambda = \frac{1}{\varepsilon} \cdot \sqrt{2ln\frac{1.25}{\delta}}$.*

We explore the key observation that an ML model with a sufficient level of differential privacy is *backdoor-free*. With this new definition of backdoor-free models in the DP domain, the main challenge to defeat backdoors in the FL setting is to decide a proper noise scale for the global model without knowledge of the training datasets. Furthermore, we need to minimize the amount of noise added to the global model to preserve its performance on the main task. None of the prior DP-based FL backdoor defense techniques provide a solution to the noise determination problem [56]. For the first time, FLAME presents an approach to *estimate* the proper noise scale that ensures the global model is backdoor-free. The noise boundary proof in Theorem 1 consists of two steps:
Step 1 (S1). By introducing the data hiding property of DP (Def. 1) and its *implication as the theoretical guarantee for backdoor-free* models. We also discuss function sensitivity (Def. 2) which is an important factor for selection of the DP parameters (ε, δ).
Step 2 (S2). We show how FLAME *generalizes backdoor elimination from centralized setting to federated setting* with theoretical analysis of the noise boundary (Eq. 5 and 6). FLAME *is the first FL defense against backdoors that provides noise level proof with bounded backdoor effectiveness.*

(S1) DP foundations and re-interpretation as Backdoor-free. As discussed in §2.4, by definition, DP makes the difference between data points indistinguishable. FLAME leverages this property of DP for backdoor elimination. In particular, we can consider D_1 and D_2 in Def. 1 as the benign and backdoored dataset. The inequality of DP suggests that algorithm $\mathcal{M}$ has a high probability of producing the same outputs on the benign and the poisoned dataset, meaning that the backdoor is eliminated. The noise level σ is determined based on the DP parameters (ε, δ) and the *sensitivity* of the function f defined below:

Definition 2 (Sensitivity)**.** *Given the function $f : \mathcal{D} \to \mathbb{R}^d$ where $\mathcal{D}$ is the data domain and d is the dimension of the function output, the sensitivity of the function f is defined as:*

$$\Delta = \max_{D_1,\, D_2 \in \mathcal{D}} ||f(D_1) - f(D_2)||_2, \tag{4}$$

where D_1 and D_2 differs on a single element $||D_1 - D_2||_1 = 1$.

As shown in Lemma 1 [18], this definition can be extended to datasets differing by more than one element, i.e., can be generalized to the DP in the multiple-point-difference setting.

(S2) Generalizing backdoor resilience from centralized to federated setting (FLAME). In the centralized setting, the defender has access to the model to be protected, the benign dataset, and the outlier (backdoored) samples. As such, he can estimate the sensitivity Δ for (ε, δ)-DP. When applying Gaussian noise with the noise scale $\sigma = \frac{\Delta}{\varepsilon}\sqrt{2ln\frac{1.25}{\delta}}$, the defender can enforce a lower bound on the prediction loss of the model on the backdoored samples for backdoor elimination [28]. However, this robustness rationale *cannot* be directly transferred from the centralized setting to the FL setting since the defender in the federated scenario (i.e., aggregator) only has access to received model updates, but not the datasets to estimate the sensitivity Δ for the global model.

FLAME extends DP-based noising for backdoor elimination to the federated setting based on the following observation: if one can ensure that all aggregated models are benign (i.e., backdoor-free), then it is obvious that the aggregated global model will also be backdoor-free. This intuition can be formally proven if the FL aggregation rule is Byzantine-tolerant. To ensure that any backdoor potentially present in the model is eliminated and the aggregated model is benign, a sufficient DP noise level is added to individual local models. However, since the local models are independent, adding noise to each local model is mathematically equivalent to the case where aggregated noise is added to the global model. This is conceptually equivalent to the conventional centralized setting, for which it has been formally shown that DP noise can eliminate backdoors [17]. In the following, we therefore show that adding DP noise to local models is equivalent to adding 'aggregated' DP noise to the global model.

We write the standard deviation of noise for the local models in the form $\sigma_i \leftarrow \frac{\alpha_i \cdot e_i}{\varepsilon} \cdot \sqrt{2ln\frac{1.25}{\delta}}$ where $\alpha_i = \frac{\Delta_i}{e_i}$, Δ_i and e_i is the sensitivity and the L_2 norm of the model W_i, respectively. Mathematically, the FL system with FLAME has:

$$\begin{aligned} G^* &= \frac{1}{n}\Sigma_{i=1}^{n} W_i^* = \frac{1}{n}[\,\Sigma_{i=1}^{n}\, W_i + N(0,\sigma_i^2)] \\ &= \frac{1}{n}\,\Sigma_{i=1}^{2} W_i + \frac{1}{n}\Sigma_{i=1}^{n} N(0,\sigma_i^2) \\ &= \frac{1}{n}\,\Sigma_{i=1}^{2} W_i + N(0, \frac{1}{n}\Sigma_{i=1}^{n}\sigma_i^2) \\ &= G + N(0,\sigma_G^2) \end{aligned} \tag{5}$$

in which W_i^* are local models and G^* the global model after adding noise $N(0,\sigma_i^2)$. Equation 5 represents the fact that adding DP noise to each local model (i.e., $W_i + N(0,\sigma_i^2)$) is equivalent to adding an 'aggregated' DP noise on the global model (i.e., $G + N(0,\sigma_G^2)$). More specifically, this equivalent Gaussian noise on the global model is the sum of Gaussian noise applied on each local model with a scaling factor $N_G = \frac{1}{n}\Sigma_{i=1}^{n} N_i$. Here, N_G and N_i are random variables with distribution $N(0,\sigma_G^2)$ and $N(0,\sigma_i^2)$, respectively. As such, we can compute the equivalent noise scale for the global model:

$$\begin{aligned} \sigma_G^2 &= \frac{1}{n^2}\Sigma_{i=1}^{n}\sigma_i^2 = (\frac{1}{\varepsilon}\sqrt{2ln\frac{1.25}{\delta}})^2 \cdot \frac{1}{n^2}\Sigma_{i=1}^{n}\Delta_i^2 \\ &= (\frac{1}{\varepsilon}\sqrt{2ln\frac{1.25}{\delta}})^2 \cdot \frac{1}{n^2}\Sigma_{i=1}^{n}\alpha_i^2 e_i^2. \end{aligned} \tag{6}$$

Equation 6 describes the relation between the DP noise added on FLAME's global model and the DP noise added on each local model. This noise scale relation in Eq. 6 together with the transformation in Eq. 5 enable FLAME to provide guaranteed security for the global model against backdoors, thereby addressing Challenge C_3 .

In Alg. 1, we use the median of Euclidean distances e_i as the upper bound S_t to clip the admitted local models (line 9-11). We hypothesize that the sensitivity of a model W_i is positively correlated with its weight magnitude $|W_i|$ (see Theorem 2 for details). In the case of linear models, the sensitivity Δ has a *linear* relation with the model weight $|\overrightarrow{w}|$ (see Eq. 8). Therefore, we use the following approximation:

$$\frac{1}{n^2}\Sigma_{i=1}^{n}\alpha_i^2 e_i^2 = \frac{1}{n^2}\Sigma_{i=1}^{n}\Delta_i^2 \approx S_t^2,$$

where S_t is the weight clipping bound. Having substituted the above approximation into Eq. 6, we can compute the noise scale of DP that FLAME deploys on the global model N_G:

$$\sigma_G \approx \frac{S_t}{\varepsilon}\sqrt{2ln\frac{1.25}{\delta}} \tag{7}$$

This concludes the proof of Theorem 1. □

FLAME's adaptive noising step applies the Gaussian noise with the noise scale computed in Eq. 7 on the global model for backdoor elimination as shown in Alg. 1, line 13-14. Note that FLAME's noising scheme is *adaptive* since the clipping bound S_t is obtained dynamically in each t^{th} epoch.

Next, we present Theorem 2 and justify how FLAME design reduces the derived noise level with *step 3 (S3)* below.

(S3) Clustering and clipping components in FLAME help to reduce the DP noise boundary. Recall that FLAME protects the FL system against backdoor attacks using three steps: clustering, clipping, and adding DP noise. The overall workflow of FLAME is shown in Fig. 4. If multiple backdoors exist in the FL system, the first two steps (clustering and clipping) can remove a subset of backdoors as shown in Fig. 3a. Note that the remaining backdoors are *'closer'* to the benign model updates in terms of both magnitude and direction. This gives us the *intuition* that removing the remaining backdoors by adding DP noise becomes easier (i.e., the noise scale σ_G is smaller) after the first two steps of FLAME.

We can see from Theorem 1 that the Gaussian noise scale σ required for backdoor resilience increases with the sensitivity of each local model Δ_i. We describe two characteristics of the model parameter W, i.e., direction and magnitude in §4. We discuss how these two factors impact the sensitivity of the model defined in Eq. 4 below.

Theorem 2. *Backdoor models with large angular deviation from benign ones, or with large parameter magnitudes have high sensitivity values* Δ.

Proving DP-based backdoor security for DNN models is still an open problem, even in the centralized setting. We, therefore, adopt a common approach in literature (e.g., [17]) by providing theoretical proof for linear models and validating it for DNNs empirically.

Proof: for a linear model f where the function output is determined by the inner product of model weight vector $\overrightarrow{w}$ and the data vector $\overrightarrow{x}$, we have

$$f(w;\, x) = \overrightarrow{w} \cdot \overrightarrow{x} = |w| \cdot |x| \cdot cos\theta, \tag{8}$$

where $\theta = < \overrightarrow{w}, \overrightarrow{x} >$ is the angle between two vectors. In this case, it is straightforward to see that if the backdoor attack changes the parameter magnitude $|w|$ or the direction θ of the model f, the resulting poisoned model f' has a large sensitivity value based on the definition in Eq. 4. □

This analysis suggests that backdoor models with large angular deviations or with large weight magnitudes have a high sensitivity value Δ. Recall that FLAME deploys dynamic clustering (§4.3.1) to remove poisoned models with large cosine distances, and employs adaptive clipping (§4.3.2) to remove poisoned models with large magnitudes. Therefore, the sensitivity of the remaining backdoor models is lower compared to the one before applying these two steps. As a result, FLAME *can use a small Gaussian noise to eliminate the remaining backdoors after applying clustering and clipping, which is beneficial for preserving the main task accuracy.*

We empirically show how the noise scale for backdoor elimination changes after applying each step of FLAME. Particularly, we measure the *smallest* Gaussian noise scale σ required to defeat *all backdoors* (i.e., $BA = 0\%$) in three settings: i) No defense components applied (which is equivalent to the previous DP-based defense [7, 18]); ii) After applying dynamic clustering; iii) After applying both dynamic clustering and adaptive clipping (which is the setting of FLAME).

Table 1: Effect of clustering and clipping in FLAME on minimal Gaussian noise level σ for backdoor elimination in the NIDS scenario, in terms of Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*).

σ	Only Noising		After Clustering		After Clustering & Clipping	
	BA	MA	BA	MA	BA	MA
0.01	100.0%	100.0%	0.0%	80.5%	0.0%	100.0%
0.08	3.5%	66.7%	0.0%	66.7%	0.0%	100.0%
0.10	0.0%	54.2%	0.0%	66.1%	0.0%	87.6%

We conduct this comparison experiment on the IoT-Traffic dataset (cf. §6). For each communication round, 100 clients are selected where $k = 40$ are adversaries. We remove the backdoor by adding Gaussian noise $N(0, \sigma^2)$ to the aggregated model. Table 1 summarizes the evaluation results in the above three settings. We can observe from the comparison results that the noise scale required to eliminate backdoors decreases after individual deployment of clustering and clipping. This corroborates the correctness of Theorem 2.

5.2 Attack and Data Distribution Assumption

In FLAME, we do not make specific assumptions about the attack and data distribution compared to the existing clustering-based defenses. Let $X = (X_1, \ldots, X_b)$ be a set of distributions of benign models $(W_1, \ldots, W_{n-k})$ where $b \leq n-k$. The deviation in X is caused by the diversity of the data. Let $X' = (X'_1, \ldots, X'_a)$ be a set of distributions of poisoned models $(W'_1, \ldots, W'_k)$ where $a \leq k$. The deviation in X' is caused by the diversity of the benign data and backdoors (e.g., poisoned data or model crafting). Existing works assume that $X'_i \approx X'_j \quad (\forall i, j : 1 \leq i, j \leq a)$ (see e.g., [22] or $X' \neq X$ [9,51]). However, this assumption does not hold in many situations because (i) there can be one or multiple attackers injecting multiple backdoors [7], or (ii) the adversary can inject one or several random (honeypot) models having a distribution X'_r that is significantly different from $X \cup (X' \setminus X'_r)$, and (iii) the adversary can control how much the backdoored models deviate from benign ones as discussed in §3. Therefore, approaches that purely divide models into two groups, e.g., K-means [51] will incorrectly classify models having distribution X'_r into the malicious group and all remaining models (having distributions drawn from $(X \cup (X' \setminus X'_r))$ into the benign group. As a result, all backdoored models having distributions drawn from $(X' \setminus X'_r)$ are classified as benign, as demonstrated in Fig. 5b. In contrast, FLAME does not rely on such specific assumptions (the adversary can arbitrarily choose X'). If the distribution X'_i of a poisoned model is similar to benign distributions in X, FLAME will falsely classify X'_i as being. But if the distribution X'_j of a poisoned model is different from the distributions in X, FLAME will identify X'_j as an outlier and classify the associated model as malicious. To identify deviating and thus potentially malicious models, FLAME leverages the HDBSCAN algorithm to identify regions of high density in the model space. Any models that are not located in the dense regions will be categorized as out-

liers, as shown in Fig. 5d. As discussed in §3, FLAME aims to remove models with distributions X'_j that have a higher attack impact compared to models with distribution X'_i. It is worth noting, however, that the impact of such remaining backdoored models will be eliminated by the noising component as shown in §5.1

Striking a balance between accuracy and security: Clustering and DP-based approaches affect model accuracy as discussed in §4.2 (Challenges C_2 and C_3). In particular, an approach that aims to maximize the number of filtered malicious models may lead to many false positives, i.e., many benign models being filtered out. Moreover, applying a very low clipping bound or a very high level of injected noise will degrade model accuracy. To address these problems, FLAME is configured so that the clustering component removes only models with high attack impact rather than all malicious models, i.e., it aims to remove the first backdoor type W'_1 as shown in Fig. 3. In addition, FLAME carefully estimates the clipping bound and noise level to ensure backdoor elimination while preserving model performance. As discussed in §4.3.2, the L_2-norms of model updates depend on the number of training rounds, dataset types, and type of backdoors. Consequently, the clipping threshold and noise level should be adapted to L_2-norms. We therefore apply the median of the L_2-norms of model updates as the clipping bound S_t (cf. Lines 9-11 of Alg. 1). This ensures that S_t is always computed between a benign local model and the global model since we assume that more than 50% of clients are benign (cf. §2.3). Further, estimating noise level based on S_t (cf. Lines 13-14 of Alg. 1) also provides a noise boundary that ensures that the global model is resilient against backdoors as discussed in §5.1. Moreover, our comparison of potential values for S_t presented in §B.2 and §B.3 shows that the chosen clipping bound and noise level provide the best balance between accuracy and security, i.e., FLAME eliminates backdoor while retaining the global model's performance on the main task.

6 Experimental Setup

We conduct all the experiments using the PyTorch deep learning framework [2] and use the source code provided by Bagdasaryan *et al.* [7], Xie *et al.* [59] and Wang *et al.* [57] to implement the attacks. We reimplemented existing defenses to compare them with FLAME.

Datasets and Learning Configurations. Following recent research on poisoning attacks on FL, we evaluate FLAME in three typical application scenarios: word prediction [35, 38–40], image classification [13, 49, 50], and an IoT intrusion detection [44, 47, 48, 54] as summarized in Tab. 2. Verification of the effectiveness of FLAME against state-of-the-art attacks in comparison to existing defenses (cf. Tab. 3 and Tab. 4) are conducted on these three datasets in the mentioned application scenarios. Experiments for evaluating specific performance aspects of FLAME are performed on the IoT dataset as it represents a very diverse and real-world setting with clear security implications.

Table 2: Datasets used in our evaluations.

Application	Datasets	#Records	Model	#params
WP	Reddit	20.6M	LSTM	∼20M
NIDS	IoT-Traffic	65.6M	GRU	∼507k
IC	CIFAR-10	60k	ResNet-18 Light	∼2.7M
	MNIST	70k	CNN	∼431k
	Tiny-ImageNet	120k	ResNet-18	∼11M

Evaluation Metrics. We consider a set of metrics for evaluating the effectiveness of backdoor attack and defense techniques as follows: *BA - Backdoor Accuracy* indicates the accuracy of the model in the backdoor task, i.e., it is the fraction of the trigger set for which the model provides the wrong outputs as chosen by the adversary. The adversary aims to maximize *BA*, while an effective defense prevents the adversary from increasing it. *MA - Main Task Accuracy* indicates the accuracy of a model in its main (benign) task. It denotes the fraction of benign inputs for which the system provides correct predictions. The adversary aims at minimizing the effect on *MA* to reduce the chance of being detected. The defense system should not negatively impact *MA*. *TPR - True Positive Rate* indicates how well the defense identifies poisoned models, i.e., the ratio of the number of models correctly classified as poisoned (True Positives - TP) to the total number of models being classified as poisoned: $TPR = \frac{TP}{TP+FP}$, where FP is False Positives indicating the number of benign clients that are wrongly classified as malicious. *TNR - True Negative Rate* indicates the ratio of the number of models correctly classified as benign (True Negatives - TN) to the total number of benign models: $TNR = \frac{TN}{TN+FN}$, where FN is False Negatives indicating the number of malicious clients that are wrongly classified as benign.

7 Experimental Results

In this section, we evaluate FLAME against backdoor attacks in the literature (§7.1) and demonstrate that our defense mechanism is resilient to adaptive attacks (§7.2). In addition, we show the effectiveness of each of FLAME's components in §B and FLAME overhead in §D. Finally, we evaluate the impact of the number of clients (§7.3) as well as the degree of non-IID data (§7.4).

7.1 Preventing Backdoor Attacks

Effectiveness of FLAME. We evaluate FLAME against the state-of-the-art backdoor attacks called *constrain-and-scale* [7], DBA [59], PGD and Edge-Case [57] and an untargeted poisoning attack [20] (cf. §F) using the same attack settings as in the original works with multiple datasets. The results are shown in Tab. 3. FLAME completely mitigates the *constrain-and-scale* attack ($BA = 0\%$) for all datasets. Moreover, our defense does not affect the Main Task Accuracy (*MA*) of the system as MA reduces by less than 0.4% in all experiments. The DBA attack as well as the Edge-Case attack [57] are also successfully mitigated ($BA = 3.2\%/4.0\%$). Further, FLAME is also effective against PGD attacks ($BA =$

Table 3: Effectiveness of FLAME against state-of-the-art attacks for the respective dataset, in terms of Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*). All metric values are reported as percentages.

Attack	Dataset	No Defense		FLAME	
		BA	*MA*	*BA*	*MA*
Constrain-and-scale [7]	Reddit	100	22.6	0	22.3
	CIFAR-10	81.9	89.8	0	91.9
	IoT-Traffic	100.0	100.0	0	99.8
DBA [59]	CIFAR-10	93.8	57.4	3.2	76.2
Edge-Case [57]	CIFAR-10	42.8	84.3	4.0	79.3
PGD [57]	CIFAR-10	56.1	68.8	0.5	65.1
Untargeted Poisoning [20]	CIFAR-10	-	46.72	-	91.31

Table 4: Effectiveness of FLAME in comparison to state-of-the-art defenses for the *constrain-and-scale* attack on three datasets, in terms of Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*). All values are percentages.

Defenses	Reddit		CIFAR-10		IoT-Traffic	
	BA	*MA*	*BA*	*MA*	*BA*	*MA*
Benign Setting	-	22.7	-	92.2	-	100.0
No defense	100.0	22.6	81.9	89.8	100.0	100.0
Krum [9]	100.0	9.6	100.0	56.7	100.0	84.0
FoolsGold [22]	**0.0**	**22.5**	100.0	52.3	100.0	99.2
Auror [51]	100.0	**22.5**	100.0	26.1	100.0	96.6
AFA [42]	100.0	22.4	**0.0**	91.7	100.0	87.4
DP [18]	14.0	18.9	**0.0**	78.9	14.8	82.3
Median [60]	**0.0**	22.0	**0.0**	50.1	0.0	87.7
FLAME	**0.0**	22.3	**0.0**	**91.9**	**0.0**	**99.8**

0.5 %). It should be noted that suggesting words is a quite challenging task, causing the MA even without attack to be only 22.7%, aligned with previous work [7].

We extend our evaluation to various backdoors on three datasets. For NIDS, we evaluate 13 different backdoors (Mirai malware attacks) and 24 device types (78 IoT devices). The results show that FLAME is able to mitigate all backdoor attacks completely while achieving a high MA=99.8%. We evaluate 5 different word backdoors for WP, and 90 different image backdoors for IC, which change the output of a whole class to another class. In all cases, FLAME successfully mitigates the attack while still preserving the *MA*.

Comparison to existing defenses. We compare FLAME to existing defenses: Krum [9], FoolsGold [22], Auror [51], Adaptive Federated Averaging (AFA) [42], Median [60] and a generalized differential privacy (DP) approach [7, 40]. Tab. 4 shows that FLAME is effective for all 3 datasets, while previous works either fail to mitigate backdoors or reduce the main task accuracy. Krum, FoolsGold, Auror, and AFA do not effectively remove poisoned models and *BA* often remains at 100%. Also, some defenses make the attack even more successful than without defense. Since they remove many benign updates (cf. §B) but fail to remove a sufficient number of poisoned updates, these defenses increase the *PMR* and, therefore, also the impact of the attack. Some defenses, e.g., Krum [9], Auror [51] or AFA [42] are not able to handle non-iid data scenarios like Reddit. In contrast, FoolsGold is only effective on the Reddit dataset ($TPR = 100\%$) because it works well on highly non-independent and identically distributed (non-IID) data (cf. §9). Similarly, AFA only mitigates backdoors on the CIFAR-10 dataset since the data are highly IID (each client is assigned a random set of images) such that the benign models share similar distances to the global model (cf. §9). Additionally, the model's *MA* is negatively impacted. The DP-based defense is effective, but it significantly reduces *MA*. For example, it performs best on the CIFAR-10 dataset with $BA = 0$, but *MA* decreases to 78.9% while FLAME increases *MA* to 91.9% which is close to the benign setting (no attacks), where $MA = 92.2\%$.

Effectiveness of FLAME's Components. Further, we have also conducted an extensive evaluation of the effectiveness of each of FLAME's components. Due to space limitations, we would like to refer to §B for the details.

7.2 Resilience to Adaptive Attacks

Given sufficient knowledge about FLAME, an adversary may seek to use adaptive attacks to bypass the defense components. In this section, we analyze such attack scenarios and strategies including *changing the injection strategy*, *model alignment*, and *model obfuscation*.

Changing the Injection Strategy. The adversary $\mathcal{A}$ may attempt to inject several backdoors simultaneously to execute different attacks on the system in parallel or to circumvent the clustering defense (cf. §2.2). FLAME is also effective against such attacks (cf. Fig. 5). To further investigate the resilience of FLAME against such attacks, we conduct two experiments: 1) assigning different backdoors to malicious clients and 2) letting each malicious client inject several backdoors. To ensure that each backdoor is injected by a sufficient number of clients, we increased the PMR for this experiment. We conducted these experiments with $n = 100$ clients of which $k = 40$ are malicious on the IoT-Traffic dataset with each type of Mirai attack representing a backdoor. First, we evaluate FLAME for $0, 1, 2, 4$, and 8 backdoors, meaning that the number of malicious clients for each backdoor is $0, 40, 20, 10$, and 5. Our experimental results show that our approach is effective in mitigating the attacks as $BA = 0\% \pm 0.0\%$ in all cases, with $TPR = 95.2\% \pm 0.0\%$, and $TNR = 100.0\% \pm 0.0\%$. For the second experiment, 4 backdoors are injected by each of the 40 malicious clients. Also, in this case, the results show that FLAME can completely mitigate the backdoors.

Model Alignment. Using the same attack parameter values, i.e., *PDR* (cf. §2.2), for all malicious clients can result in high distances between benign and poisoned models. Those high distances can be illustrated as a gap between poisoned and benign models, s.t. the clustering can separate them. Therefore, a sophisticated adversary can generate models that bridge the gap between them such that they are merged to the same cluster in our clustering. We evaluate this attack on the IoT-Traffic dataset for $k = 80$ malicious clients and $n = 200$ clients in total. To remove the gap, each malicious client is assigned a random amount of malicious data, i.e., a random *PDR* ranging from 5% to 20%. As Tab. 5 shows, when we apply model

Table 5: Resilience to model alignment attacks in terms of Backdoor Accuracy (*BA*), Main Task Accuracy (*MA*), True Positive Rate (*TPR*), True Negative Rate (*TNR*) in percent.

	BA	*MA*	*TPR*	*TNR*
Model Filtering	100.0	91.98	0.0	33.04
FLAME	**0.0**	**100.0**	**5.68**	**33.33**

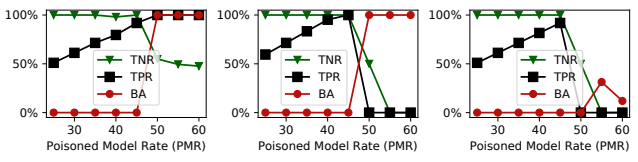
filtering only, our clustering component cannot identify the malicious clients well ($TPR = 0\%$), resulting in $BA = 100\%$. However, when we apply FLAME, although *TPR* remains low (5.68%) FLAME still mitigates the attack successfully (*BA* reduces from 100% to 0%). This can be explained by the fact that when the adversary $\mathcal{A}$ tunes malicious updates to be close to the benign ones, the attack's impact is reduced and consequently averaged out by our noising component.

Model Obfuscation. $\mathcal{A}$ can add noise to the poisoned models to make them difficult to detect. However, our evaluation of such an attack on the IoT-Traffic dataset shows that this strategy is not effective. We evaluate different noise levels to determine a suitable standard deviation for the noise. Thereby, we observe that a noise level of 0.034 causes the models' cosine distances in clustering to change without significantly impacting *BA*. However, FLAME can still efficiently defend this attack: *BA* remains at 0% and *MA* at 100%.

7.3 Effect of Number of Clients

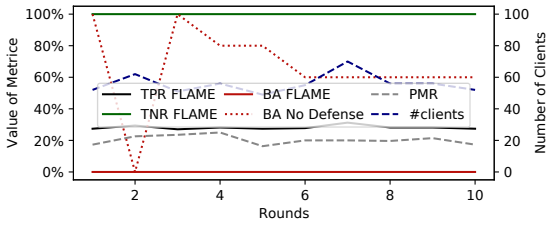
Impact of Number of Malicious Clients. We assume that the number of benign clients is more than half of all clients (cf. §2.2) and our clustering is only expected to be successful when $PMR = \frac{k}{n} < 50\%$ (cf. §4.3.1). We evaluate FLAME for different *PMR* values. Figure 7 shows how *BA*, *TPR*, and *TNR* change in the IC, NIDS, and WP applications for *PMR* values from 25% to 60%. It shows that FLAME is only effective if $PMR < 50\%$ so that only benign clients are admitted to the model aggregation ($TNR = 100\%$) and thus $BA = 0\%$. However, if $PMR > 50\%$, FLAME fails to mitigate the attack because the majority of poisoned models will be included resulting in low *TNR*. Interestingly, FLAME accepted all models for $PMR = 50\%$ ($TPR = 0\%$ and $TNR = 100\%$). For the IC application, since the IC data are non-IID, poisoned models are not similar. Therefore, some poisoned models were excluded from the cluster resulting in a high *TPR* even for *PMRs* higher than 50%. However, the majority of poisoned models were selected resulting in the drop in the *TNR*.

Varying number of clients in different training rounds.

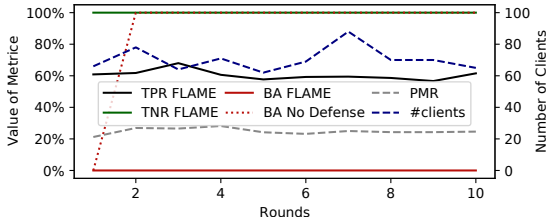


(a) IC (b) NIDS (c) WP

Figure 7: Impact of the poisoned model rate $PMR = \frac{k}{n}$ on the evaluation metrics. *PMR* is the fraction of malicious clients k per total clients n.



(a) Image Classification



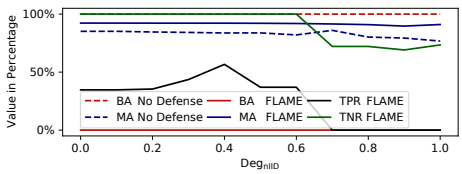
(b) Network Intrusion Detection

Figure 8: Impact of the number of clients on FLAME

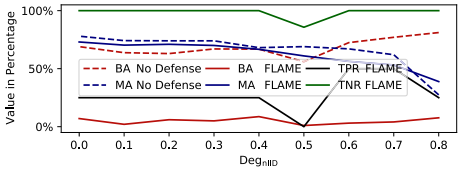
In general, FLAME is a round-independent defense, i.e., it does not use information from previous rounds such as which clients were excluded in which rounds. Therefore, FLAME will not be affected if the number of clients or number of malicious clients varies as long as the majority of clients remain benign. To demonstrate this, we simulate realistic scenarios in which clients can join and drop out dynamically. We conducted an experiment where during each round, the total number of available clients is randomly selected. As the result, the number of malicious clients will also be random. In this experiment, we used a population of 100 clients in total, out of which 25 are malicious. In each round, a random number (from 60 to 90) of clients are selected, so that the fraction of malicious clients (*PMR*) varies in each round. Figure 8 shows the experimental results. One can see that the proportion of malicious clients (*PMR*) does not affect the effectiveness of FLAME, i.e., the backdoor is completely removed ($BA = 0\%$) in every round. Since all poisoned models are detected, their negative effect on the aggregated model is removed. Therefore, the MA with FLAME is better than the one without defense, and is almost always 100 % aligned with the results in Tab. 4.

7.4 Impact of the Degree of non-IID Data

Since clustering is based on measuring differences between benign and malicious updates, the distribution of data among clients might affect our defense. We conduct two experiments for both *Constrain-and-scale* and Edge-Case PGD on the CIFAR-10 dataset. For Reddit and IoT datasets, changing the degree of non-IID data is not meaningful since the data have a natural distribution as every client obtains data from different Reddit users or traffic chunks from different IoT devices. Following previous works [20,57], we vary the degree of non-IID data Deg_{nIID} by changing the fraction of images belonging to a specific class assigned to clients. In particular, we divide the clients into 10 groups corresponding to the 10 classes of



(a) *Constrain-and-scale*



(b) Edge-Case PGD

Figure 9: Impact of degree of non-IID data on FLAME for *constrain-and-scale* using the Deg_{nIID} and for the Edge-Case PGD attack using the α parameter of the Dirichlet distribution.

CIFAR-10. The clients of each group are assigned a fixed fraction of $\text{Deg}_{\text{nIID}}$of the images from its designated image class, while the rest of the images will be assigned to it at random. Consequently, the data distribution is random, i.e., completely IID if $\text{Deg}_{\text{nIID}} = 0\%$ (all images are randomly assigned) and completely non-IID if $\text{Deg}_{\text{nIID}} = 100\%$ (a client only gets images from its designated class).

Figure 9a shows the evaluation results for the *constrain-and-scale* attacks. Although FLAME does not detect the poisoned models for very non-IID scenarios, it still mitigates the attack as the BA remains 0% for all values of Deg_{nIID}. For low Deg_{nIID}, FLAME effectively identifies the poisoned models ($TNR = 100\%$) and the MA remains on almost the same level as without defense. As shown in Fig. 9b, FLAME also mitigates the Edge-Case PGD attack effectively for all α values of the Dirichlet distribution and the MA also stays on the same level as without defense. However, since not all poisoned models are detected, a higher σ is determined dynamically to mitigate the *constrain-and-scale* backdoor, resulting in a slightly reduced MA for $\text{Deg}_{\text{nIID}} \geq 0.7$ (MA is 91.9% for $\text{Deg}_{\text{nIID}} = 0.6$, and is reduced to 91.0% for $\text{Deg}_{\text{nIID}} = 1.0$). Note that Fig. 9 shows the evaluation results in a training round t where the global model G_t is close to convergence [7], thus even though the TNR decreases with a large value of Deg_{nIID}, the drop of MA with FLAME is not substantial.

8 Privacy-preserving Federated Learning

A number of attacks on FL have been proposed that aim to infer from parameters of a model the presence of a specific training sample in the training dataset (*membership inference attacks*) [41,46,52], properties of training samples (*property inference attacks*) [23,41], try to assess the proportion of samples of a specific class in the data (*distribution estimation attacks*) [58]. Inference attacks by the aggregator $\mathcal{A}^s$ are significantly stronger, as $\mathcal{A}^s$ has access to the local models [43] and can also link gained information to a specific user, while the global model anonymizes the individual contributions.

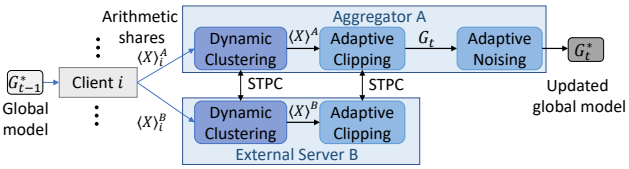


Figure 10: Overview of private FLAME in round t using Secure-Two-Party Computation (STPC).

Therefore, enhanced privacy protection for FL is needed that prohibits access to the local model updates.

Adversary Model (privacy). In this adversary type, $\mathcal{A}^s$ attempts to infer sensitive information about clients' data D_i from their model updates W_i [23,41,46,52] by maximizing the information $\phi_i = \text{INFER}(W_i)$ that $\mathcal{A}^s$ gains about the data D_i of client i by inferring from its corresponding model W_i.

Deficiencies of existing defenses. Generally, there are two approaches to protect the privacy of clients' data: differential privacy (DP; [18]) and cryptographic techniques such as homomorphic encryption [24] or multi-party computation [14]. DP is a statistical approach that can be efficiently implemented, but it can only offer high privacy protection at the cost of a significant loss in accuracy due to the noise added to the models [6,61]. In contrast, cryptographic techniques provide strong privacy guarantees as well as high accuracy at the cost of reduced efficiency.

Private FLAME. To securely implement FLAME using STPC, we use an optimized combination of three prominent STPC techniques as implemented with state-of-the-art optimizations in the ABY framework [14]. Fig. 10 shows an overview of private FLAME. It involves n clients and two non-colluding servers, called aggregator A and external server B. Each client $i \in \{1,...,n\}$ splits the parameters of its local update W_i into two Arithmetic shares $\langle X \rangle_i^A$ and $\langle X \rangle_i^B$, such that $W_i = \langle X \rangle_i^A + \langle X \rangle_i^B$, and sends $\langle X \rangle_i^A$ to A and $\langle X \rangle_i^B$ to B. A and B then privately compute the new global model via STPC. We co-design the distance calculation, clustering, adaptive clipping, and aggregation of FLAME (cf. Alg. 1) of FLAME as efficient STPC protocols.To further improve performance, we approximate HDBSCAN with the simpler DBSCAN [10] to avoid the construction of the minimal spanning tree in HDBSCAN as it is very expensive to realize with STPC. See §G for more details on private FLAME evaluation of its accuracy and performance.

9 Related Work

In general, existing backdoor defenses can roughly be divided into two main categories. The first one aims to distinguish malicious updates and benign updates by 1) clustering model updates [9,15,22,29,33,34,51], 2) changing aggregation rules [25,60], and 3) using root dataset [4]. The second category is based on differential privacy techniques [7,56]. Next, we will discuss these points in detail.

Clustering model updates. Several backdoor defenses, such as Krum [9], AFA [42], and Auror [51], aim at separat-

ing benign and malicious model updates. However, they only work under specific assumptions about the underlying data distributions, e.g., Auror and Krum assume that data of benign clients are iid. In contrast, FoolsGold and AFA [42] assume that benign data are non-iid. In addition, FoolsGold assumes that manipulated data are iid. As a result, these defenses are only effective under specific circumstances (cf. §7.1) and cannot handle the simultaneous injection of multiple backdoors (cf. §4.3.1). Moreover, such defenses cannot detect stealthy attacks, e.g., where the adversary constrains their poisoned updates within benign update distribution such as *Constrain-and-scale* attacks [7]. In contrast, FLAME does not make any assumption about the data distribution, clipping, and noising components can also mitigate stealthy attacks, and FLAME can defend against injection of multiple backdoors (cf. §4.3.1).

Changing aggregation rules. Instead of using FedAvg [38], Yin *et al.* [60] and Guerraoui *et al.* [25] propose using the median parameters from all local models as the global model parameters, i.e., $G_t = \text{MEDIAN}(W_1^t, \ldots, W_n^t)$. However, the adversary can bypass it by injecting stealthy models like W_3^t (cf. Fig. 2), in which the parameters of poisoned model will be selected to be incorporated into the global model. Further, our evaluation in §7.1 shows that Median also reduces the performance of the model significantly.

Using root data. Although FLTrust [12] can defend against byzantine clients (with arbitrary behavior) and detect poisoning attacks including backdoors, it requires the aggregator to have access to a benign root dataset. Baffle [4] utilizes clients using their own data to evaluate the performance of the aggregated model to detect backdoors. However, this approach has two limitations, e.g., (i) the backdoor triggers are only known to the attacker, i.e., one cannot ensure that the benign clients would have such trigger data to activate the backdoor, and (ii) Baffle does not work in a non-IID data scenario with a small number of clients as clients cannot distinguish deficits in model performance due to the backdoor from lack of data.

Differential Privacy-based approaches. Clipping and noising are known techniques to achieve differential privacy (DP) [18]. However, directly applying these techniques to defend against backdoor attacks is not effective because they significantly decrease the Main Task Accuracy (§7.1) [7]. FLAME tackles this by i) identifying and filtering out potential poisoned models that have a high attack impact (cf. §4.3.1), and ii) eliminating the residual poison with an appropriate adaptive clipping bound and noise level, such that the Main Task Accuracy is retained (cf. §4.3.2).

10 Conclusion

In this paper, we introduce FLAME, a resilient aggregation framework for FL that eliminates the impact of backdoor attacks while maintaining the performance of the aggregated model on the main task. We propose a method to approximate the amount of noise that needs to be injected into the global model to neutralize backdoors. Furthermore, in combination with our dynamic clustering and adaptive clipping, FLAME can significantly reduce the noise scale for backdoor removal and thus preserve the benign performance of the global model. In addition, we design, implement, and benchmark efficient secure two-party computation protocols for FLAME to ensure the privacy of clients' training data and to impede inference attacks on client updates.

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG) SFB-1119 CROSSING/236615297, the European Research Council (ERC, grant No. 850990 PSOTI), the EU H2020 project SPATIAL (grant No. 101021808), GRK 2050 Privacy & Trust/251805230, HMWK within ATHENE project, NSF-TrustHub (grant No. 1649423), SRC-Auto (2019-AU-2899), Huawei OpenS3 Lab, and Intel Private AI Collaborative Research Center. We thank the anonymous reviewers and the shepherd, Neil Gong, for constructive reviews and comments.

References

[1] Reddit dataset, 2017. https://bigquery.cloud.google.com/dataset/fh-bigquery:reddit_comments.

[2] Pytorch, 2019. https://pytorch.org.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *CCS*. ACM, 2016.

[4] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. BaFFLe: Backdoor Detection via Feedback-based Federated Learning. In *ICDCS*, 2021.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *USENIX Security*, 2017.

[6] Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving Deep Learning via Additively Homomorphic Encryption. In *TIFS*, 2017.

[7] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How To Backdoor Federated Learning. In *AISTATS*, 2020.

[8] Moran Baruch, Gilad Baruch, and Yoav Goldberg. A Little Is Enough: Circumventing Defenses For Distributed Learning. In *NIPS*, 2019.

[9] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *NIPS*, 2017.

[10] Beyza Bozdemir, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider. Privacy-preserving density-based clustering. In *ASIACCS*, 2021.

[11] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2013.

[12] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *NDSS*, 2021.

[13] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *USENIX Operating Systems Design and Implementation*, 2014.

[14] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015.

[15] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization. In *ICML*, 2019.

[16] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine Learning DDoS Detection for Consumer Internet of Things Devices. In *arXiv preprint:1804.04159*, 2018.

[17] Min Du, Ruoxi Jia, and Dawn Song. Robust anomaly detection and backdoor attack detection via differential privacy. In *ICLR*, 2020.

[18] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. In *Foundations and Trends in Theoretical Computer Science*, 2014.

[19] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, 1996.

[20] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *USENIX Security*, 2020.

[21] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: secure aggregation for private federated learning. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 56–62. IEEE, 2021.

[22] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, 2020. originally published as arxiv:1808.04866.

[23] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations. In *CCS*, 2018.

[24] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009.

[25] Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *ICML*. PMLR, 2018.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[27] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and Analysis of Hajime, a Peer-to-Peer IoT Botnet. In *NDSS*, 2019.

[28] Li Huang, Yifeng Yin, Zeng Fu, Shifa Zhang, Hao Deng, and Dianbo Liu. LoAdaBoost: Loss-Based AdaBoost Federated Machine Learning on medical Data. In *arXiv preprint:1811.12629*, 2018.

[29] Youssef Khazbak, Tianxiang Tan, and Guohong Cao. Mlguard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning. In *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020.

[30] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and Other Botnets. In *IEEE Computer*, 2017.

[31] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[32] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[33] Suyi Li, Yong Cheng, Yang Liu, Wei Wang, and Tianjian Chen. Abnormal client behavior detection in federated learning. *arXiv preprint arXiv:1910.09933*, 2019.

[34] Suyi Li, Yong Cheng, Wei Wang, Yang Liu, and Tianjian Chen. Learning to detect malicious clients for robust federated learning. *arXiv preprint arXiv:2002.00211*, 2020.

[35] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J. Dally. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. In *ICLR*, 2018.

[36] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*. IEEE, 2017.

[37] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205, 2017.

[38] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, 2017.

[39] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative Machine Learning without Centralized Training Data. Google AI, 2017.

[40] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning Differentially Private Language Models Without Losing Accuracy. In *ICLR*, 2018.

[41] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *IEEE S&P*, 2019.

[42] Luis Muñoz-González, Kenneth T. Co, and Emil C. Lupu. Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging. In *arXiv preprint:1909.05125*, 2019.

[43] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*, 2019.

[44] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. DÏoT: A Federated Self-learning Anomaly Detection System for IoT. In *ICDCS*, 2019.

[45] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning Attacks on Federated Learning-Based IoT Intrusion Detection System. In *Workshop on Decentralized IoT Systems and Security*, 2020.

[46] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. In *NDSS*, 2018.

[47] Jianji Ren, Haichao Wang, Tingting Hou, Shuai Zheng, and Chaosheng Tang. Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things. In *IEEE Access*, 2019.

[48] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Merouane Debbah. Federated Learning for Ultra-Reliable Low-Latency V2V Communications. In *GLOBCOM*, 2018.

[49] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Federated Learning for Medical Imaging. In *Intel AI*, 2018.

[50] Micah Sheller, Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation. In *Brain Lesion Workshop*, 2018.

[51] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems. In *ACSAC*, 2016.

[52] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership Inference Attacks Against Machine Learning Models. In *IEEE S&P*, 2017.

[53] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. In *TMC*, 2018.

[54] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated Multi-Task Learning. In *NIPS*, 2017.

[55] Saleh Soltan, Prateek Mittal, and Vincent Poor. BlackIoT: IoT Botnet of High Wattage Devices Can Disrupt the Power Grid. In *USENIX Security*, 2018.

[56] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[57] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*, 2020.

[58] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. Eavesdrop the Composition Proportion of Training Labels in Federated Learning. In *arXiv preprint:1910.06044*, 2019.

[59] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. DBA: Distributed Backdoor Attacks against Federated Learning. In *ICLR*, 2020.

[60] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *ICML*. PMLR, 2018.

[61] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *USENIX ATC*, 2020.

A Datasets and Learning Configurations

Word Prediction (WP). We use the Reddit dataset of November 2017 [1] with the same settings as state-of-the-art works [7, 38, 40] for comparability. In particular, each user in the dataset with at least 150 posts and not more than 500 posts is considered as a client. This results in 80 000 clients' datasets with sizes between 298 and 32 660 words.

The model consists of two LSTM layers and a linear output layer [7, 38]. To be comparable to the attack setting in [7], we evaluate FLAME on five different backdoors, each with a different trigger sentence corresponding to a chosen output.

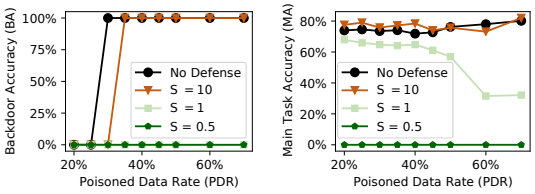
Image Classification (IC). For image classification, we use mainly the CIFAR-10 dataset [31], a standard benchmark dataset for image classification, in particular for FL [38] and backdoor attacks [7, 8, 42]. It consists of 60 000 images of 10 different classes. The adversary aims at changing the predicted label of one class of images to another class of images. We use a lightweight version of the ResNet18 model [26] with 4 convolutional layers with max-pooling and batch normalization [7]. The experimental setup consists of 100 clients and uses a PMR of 20%. In addition to the CIFAR-10 dataset, we also evaluate FLAME's effectiveness on two further datasets for image classification. The *MNIST* dataset consists of 70 000 handwritten digits [32]. The learning task is to classify images to identify digits. The adversary poisons the model by mislabeling labels of digit images before using it for training [51]. We use a convolutional neural network (CNN) with 431000 parameters. The *Tiny-ImageNet* [3] consists of 200 classes and each class has 500 training images, 50 validation images, and 50 test images. We used ResNet18 [26] model.

Network Intrusion Detection System (NIDS). We test backdoor attacks on IoT anomaly-based intrusion detection systems that often represent critical security applications [5, 16, 27, 30, 44, 45, 55]. Here, the adversary aims at causing incorrect classification of anomalous traffic patterns, e.g., generated by IoT malware, as benign patterns. Based on the FL anomaly detection system DÏoT [44], we use three datasets called DIoT-Benign, DIoT-Attack, and UNSW-Benign [44, 53] from real-world home and office deployments (four homes and two offices located in Germany and Australia). DIoT-Attack contains the traffic of 5 anomalously behaving IoT devices, infected by the Mirai malware [44]. Moreover, we collected a fourth IoT dataset containing communication data from 24 typical IoT devices (including IP cameras and power plugs) in three different smart home settings and an office setting. Following [44], we extracted

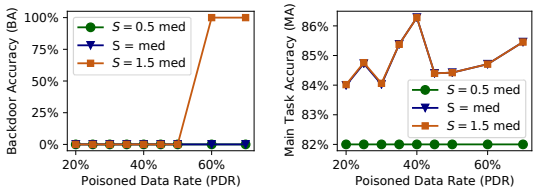
[3] https://tiny-imagenet.herokuapp.com

Table 6: Effectiveness of the clustering component, in terms of True Positive Rate (*TPR*) and True Negative Rate (*TNR*), of FLAME in comparison to existing defenses for the *constrain-and-scale* attack on three datasets. All values are in percentage and the best results of the defenses are marked in bold.

Defenses	Reddit		CIFAR-10		IoT-Traffic	
	TPR	TNR	TPR	TNR	TPR	TNR
Krum	9.1	0.0	8.2	0.0	24.2	0.0
FoolsGold	**100.0**	**100.0**	0.0	90.0	32.7	84.4
Auror	0.0	90.0	0.0	90.0	0.0	70.2
AFA	0.0	88.9	**100.0**	**100.0**	4.5	69.2
FLAME	22.2	**100.0**	23.8	86.2	**59.5**	**100.0**



(a) BA for static bound (b) MA for static bound



(c) BA for dynamic bound (d) MA for dynamic bound

Figure 11: Effectiveness of FLAME's clipping bound in terms of Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*). *S* is the clipping bound and *med* the L_2-norm median.

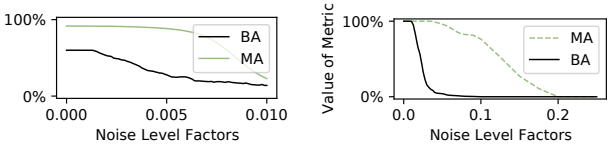
device-type-specific datasets capturing the devices' communication behavior. We simulate the FL setup by splitting each device type's dataset among several clients (from 20 to 200). Each client has a training dataset corresponding to three hours of traffic measurements containing samples of roughly 2 000-3 000 communication packets. The learning model consists of 2 GRU layers and a fully connected layer.

B Effectiveness of FLAME's Components

B.1 Effectiveness of the Clustering Component

We show the results for the clustering component in Tab. 6. As shown there, our filtering achieves $TNR = 100\%$ for the Reddit and IoT-Traffic datasets, i.e., FLAME only selects benign models in this attack setting. Recall that the goal of clustering is to filter out the poisoned models with high attack impact, i.e., not necessarily all poisoned models (cf. §4.1). This allows FLAME to defend backdoor attacks effectively, even if not all poisoned models are filtered. For example, although for the CIFAR-10 dataset in Tab. 6 the TNR is not 100 % (86.2%), the attack is still mitigated by the noising component, such that the BA is 0 % (cf. Tab. 4).

B.2 Effectiveness of Clipping



(a) Image Classification (b) Network Intrusion Detection

Figure 12: Impact of different noise level factors on the Backdoor Accuracy (*BA*) and Main Task Accuracy (*MA*).

Fig. 11 demonstrates the effectiveness of FLAME's dynamic clipping where S is the median of L_2-norms compared to a static clipping bound [7] and different choices for a dynamic clipping boundary (i.e., median, half of median, median multiplied by 1.5). The experiments are conducted for the IoT-Traffic dataset, which is non-iid. Fig. 11a and Fig. 11b show that a small static bound $S = 0.5$ is effective to mitigate the attack ($BA = 0\%$), but *MA* drops to 0% rendering the model useless. Moreover, a higher static bound like $S = 10$ is ineffective as $BA = 100\%$ if the Poisoned Data Rate (PDR) $\geq 35\%$. In contrast, FLAME's dynamic clipping threshold performs significantly better as *BA* consistently remains at 0% while *MA* remains high (cf. Fig. 11c and Fig. 11d).

B.3 Effectiveness of Adding Noise

Fig. 12 shows the impact of adding noise to the intermediate global models with respect to different noise level factors λ to determine the standard deviation of the noise σ dynamically based on the median L_2-norm of the updates S_t as $\sigma = \lambda S_t$. As it can be seen, increasing λ reduces the *BA*, but it also negatively impacts the performance of the model in the main task (*MA*). Therefore, the noise level must be dynamically tuned and combined with the other defense components to optimize the overall success of the defense. The noise level factor is determined by $\lambda = \frac{1}{\varepsilon}\sqrt{2ln\frac{1.25}{\delta}}$ for (ε, δ)-DP. We use standard DP parameters and set $\varepsilon = 3705$ for IC, $\varepsilon = 395$ for the NIDS and $\varepsilon = 4191$ for the NLP scenario. Accordingly, $\lambda = 0.001$ for IC and NLP, and $\lambda = 0.01$ for the NIDS scenario. The DP budget is dependent on the considered dataset scenario. It is determined based on the median of the dataset sizes of the clients and the size of the model used. It can thus be empirically determined by the aggregator. Analogous to determining the clipping boundary S (cf. 4.3.2), using the median ensures that the used dataset size is within the range of benign values.

C Naïve Combination

Furthermore, we test a naïve combination of the defense components by stacking clipping and adding noise (using a fixed clipping bound of 1.0 and a standard deviation of 0.01 as in [7]) on top of a clustering component using K-means. However, this naïve approach still allows a *BA* of 51.9% and a *MA* of 60.24%, compared to a *BA* of 0.0% and a *MA* of 89.87% of FLAME in the same scenario for the CIFAR-10 dataset. Based on our evaluations in §7.1, it becomes apparent that

FLAME's dynamic nature goes beyond previously proposed defenses that consist of static baseline ideas, which FLAME significantly optimizes, extends, and automates to offer a comprehensive dynamic and private defense against sophisticated backdoor attacks.

D Overhead of FLAME

We evaluated FLAME for 6 different device types from the IoT dataset. In this experiment, only benign clients participated and the model was randomly initialized. The highest observed overhead was 4 additional rounds. In average, 1.67 additional training rounds were needed to achieve at least 99% of the *MA* that was achieved without applying the defense, i.e., FLAME does not prevent the model from converging.

E HDBSCAN

HDBSCAN [11] is a density-based clustering technique that classifies data samples in different clusters without predefined the maximum distance and the number of clusters. In the following, we describe HDBSCAN in detail, following the implementation of McInnes *et al.* [36, 37]. However, we focus on the behavior of HDBSCAN for the parameters that FLAME uses, i.e., when `min_cluster_size`=$N/2+1$ and `min_samples=1`, e.g., because of the choice for `min_cluster_size` we skip parts that deal with multiple clusters. HDBSCAN first uses the given distances to build a minimal spanning tree (MST), where the vertices represent the individual data points and the edges are weighted by the distances between the respective points. Then it uses the MST to build a binary tree where the leaf nodes represent the vertices of the MST and the non-leaf nodes represent the edges of the MST. For this, first, all vertices are considered as separate trees (of size 1). For this, first, all vertices are considered as separate trees (of size 1) and then, starting from the edge with the lowest weight, iteratively the trees are merged by creating a non-leaf-node for each edge of the MST and set the (previously not connected) subtrees containing the endpoints of the edge as children for the new node (represented by calling the function `make_binary_tree`. In the next step, HDBSCAN collects all nodes of the binary tree as candidates, that cover at least $N/2+1$ data points. Since only non-leaf nodes fulfill the requirement of covering at least $N/2+1$ data points, each cluster candidate is based on a node, representing an edge in the MST. It uses the weight of the edge and the number of covered points to calculate a so-called stability value. Then HDBSCAN uses the stability value to determine the cluster candidate with the most homogeneous density and returns this candidate as majority cluster. Finally, it assigns the cluster label to all data points inside this cluster and labels all points outside of this cluster as noise.

F Effectiveness of FLAME against untargeted poisoning attacks

Another attack type related to backdooring is *untargeted poisoning* [8, 9, 20]. Unlike backdoor attacks that aim to incorporate specific backdoor functionalities, untargeted poisoning aims at rendering the model unusable. The adversary uses crafted local models with low Main Task Accuracy to damage the global model G. Fang at el. [20] propose such an attack bypassing state-of-the-art defenses. Although we do not focus on untargeted poisoning, our approach intuitively defends it since, in principle, this attack also trade-offs attack impact against stealthiness. To evaluate the effectiveness of FLAME against this attack, we test the Krum-based attack proposed by [20] on FLAME. Since [20]'s evaluation uses image datasets, we evaluate FLAME's resilience against it with CIFAR-10. The evaluation results show that although the attack significantly damages the model by reducing *MA* from 92.16% to 46.72%, FLAME can successfully defend against it and *MA* remains at 91.31%.

G Performance of Private FLAME

For our implementation, we use the STPC framework ABY [14] which implements the three sharing types, including state-of-the-art optimizations and flexible conversions and the open-source privacy-preserving DBSCAN by Bozdemir et al. [10]. All STPC results are averaged over 10 experiments and run on two separate servers with Intel Core i9-7960X CPUs with 2.8 GHz and 128 GB RAM connected over a 10 Gbit/s LAN with 0.2 ms RTT.

Approximating HDBSCAN by DBSCAN. We measure the effect of approximating HDBSCAN by DBSCAN including the binary search for the neighborhood parameter ε. The results show that our approximation has a negligible loss of accuracy. For some applications, the approximation even performs slightly better than the standard FLAME, e.g., for CIFAR-10, private FLAME correctly filters all poisoned models, while standard FLAME accepts a small number ($TNR = 86.2\%$), which is still sufficient to achieve $BA = 0.0\%$.

Runtime of Private FLAME. We evaluate the runtime in seconds per training iteration of the cosine distance, Euclidean distance + clipping + model aggregation, and clustering steps of Alg. 1 in standard (without STPC) and in private FLAME (with STPC). The results show that private FLAME causes a significant overhead on the runtime by a factor of up to three orders of magnitude compared to the standard (non-private) FLAME. However, even if we consider the largest model (Reddit) with $K = 100$ clients, we have a total server-side runtime of 22 081.65 seconds ($\approx$ 6 hours) for a training iteration with STPC. Such runtime overhead would be acceptable to maintain privacy, especially since mobile phones, which would be a typical type of clients in FL [38], are not always available and connected so that there will be delays in synchronizing clients' model updates in FL. These delays can then also be used to run STPC. Furthermore, achieving provable privacy by using STPC may even motivate more clients to contribute to FL in the first place and provide more data.

C Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries" (IEEE TIFS'23)

[SSY23] T. SCHNEIDER, A. SURESH, H. YALAME. **"Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries""**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 18 (2023), pp. 1407–1409. CORE Rank **A**. Appendix C.

https://doi.org/10.1109/TIFS.2023.3238544

Comments on "Privacy-Enhanced Federated Learning Against Poisoning Adversaries"

Thomas Schneider, Ajith Suresh, and Hossein Yalame

***Abstract*— Liu et al. (2021) recently proposed a privacy-enhanced framework named PEFL to efficiently detect poisoning behaviours in Federated Learning (FL) using homomorphic encryption. In this article, we show that PEFL does not preserve privacy. In particular, we illustrate that PEFL reveals the entire gradient vector of all users in clear to one of the participating entities, thereby violating privacy. Furthermore, we clearly show that an immediate fix for this issue is still insufficient to achieve privacy by pointing out multiple flaws in the proposed system.**

***Index Terms*— Federated learning (FL), homomorphic encryption, poisoning and inference attacks, data privacy.**

I. Introduction

Federated Learning (FL) is a new distributed machine learning approach that allows multiple entities to jointly train a model without sharing their private and sensitive local datasets with others. In FL, clients locally train models using their local training data, then send model updates to a central aggregator, which merges them into a global model. FL is used in a variety of applications such as word prediction for mobile keyboards in GBoard [1] and medical imaging [2]. Despite its benefits, FL has been shown to be susceptible to model poisoning [3] and inference attacks [4]. In model poisoning attacks, an adversary injects poisoned model updates by corrupting a subset of clients, with which the adversary can compromise the user's data privacy as well as the FL model's integrity [5], [6]. The recent work of Liu et al. [7] proposed a privacy-enhanced framework called PEFL to detect poisoning behaviors in FL. PEFL aims to prevent malicious users from inferring memberships by uploading malicious gradients and semi-honest servers from invading users' privacy. Furthermore, PEFL claims to be the first effort to detect poisoning behaviors in FL while using ciphertext and uses homomorphic encryption (HE) as the underlying technology.

In this article, we have a closer look at the PEFL system of [7] and identify multiple privacy vulnerabilities. In particular, we show that each of the three main protocols in PEFL – SecMed, SecPear, and SecAgg, reveals significant information about the user's gradients to one of the computing servers thereby compromising privacy. Furthermore, we demonstrate that combining information from the protocols enables a computation server to learn the gradient vectors of all users in clear, thereby breaking the PEFL system.

Manuscript received 25 March 2022; revised 30 December 2022; accepted 3 January 2023. Date of publication 20 January 2023; date of current version 2 February 2023. This work was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Program through Privacy-Preserving Services On The Internet (PSOTI) under Grant 850990, in part by the Deutsche Forschungsgemeinschaft (DFG) under Grant SFB1119 CROSSING and Grant 236615297, and in part by GRK2050 Privacy and Trust under Grant 251805230. *(Corresponding author: Hossein Yalame.)*

The authors are with the Cryptography and Privacy Engineering Group (ENCRYPTO), Technical University of Darmstadt, 64289 Darmstadt, Germany (e-mail: schneider@encrypto.cs.tu-darmstadt.de; suresh@encrypto.cs.tu-darmstadt.de; yalame@encrypto.cs.tu-darmstadt.de).

Digital Object Identifier 10.1109/TIFS.2023.3238544

II. Liu et al.'s Protocols Are Not Private

In this section, we revisit the Privacy Enhanced Federated Learning (PEFL) system in [7], but with our notations for clarity. We begin with an overview of PEFL's four entities:

- *Key Generation Center (KGC):* Trusted entity managing public and private keys (pk, sk) for HE.
- *Data Owners (U_x):* Each data owner U_x, for $x \in [\mathsf{m}]$, locally trains the local model on their data and computes the gradient vector $\vec{\mathbf{g}}_x = \{g_x^1, \ldots, g_x^{\mathsf{n}}\}$. Here, m denotes the total number of users in the system and n denotes the dimension of the gradient vector.
- *Service Provider (SP):* SP receives all gradients submitted by data owners and aggregates them (usually by averaging) to produce an optimized global model.
- *Cloud Platform (CP):* CP assists SP in the computations and operates on a pay-per-use basis.

The threat model assumes that SP and CP are both semi-honest, whereas data owners can be maliciously corrupt. Furthermore, the four entities mentioned above are non-colluding.

We now examine the PEFL system in-depth, focusing on the amount of information visible to each entity. More specifically, we are interested in how much information the cloud platform (CP) learns from the protocol execution.

A. Calculation of Gradients

The protocol begins with each user U_x training the model locally and obtaining the corresponding gradient vector $\vec{\mathbf{g}}_x = \{g_x^1, \ldots, g_x^{\mathsf{n}}\}$. User U_x then encrypts and sends the gradient vector to SP using CP's public key pk_c. As shown in (1), the gradient vectors corresponding to all users can be viewed as a matrix $\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}}$.

$$\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}} = \begin{array}{c} \\ U_1 \\ U_2 \\ \vdots \\ U_{\mathsf{m}} \end{array} \begin{array}{c} \begin{array}{ccccccc} c_1 & c_2 & \cdots & c_i & \cdots & c_{\mathsf{n}-1} & c_{\mathsf{n}} \end{array} \\ \begin{pmatrix} g_1^1 & g_1^2 & \cdots & g_1^i & \cdots & g_1^{\mathsf{n}-1} & g_1^{\mathsf{n}} \\ g_2^1 & g_2^2 & \cdots & g_2^i & \cdots & g_2^{\mathsf{n}-1} & g_2^{\mathsf{n}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ g_{\mathsf{m}}^1 & g_{\mathsf{m}}^2 & \cdots & g_{\mathsf{m}}^i & \cdots & g_{\mathsf{m}}^{\mathsf{n}-1} & g_{\mathsf{m}}^{\mathsf{n}} \end{pmatrix} \end{array}. \tag{1}$$

B. Median Computation Using SecMed

The SP and CP use the SecMed algorithm (cf. Figure 4 in [7]) to compute the median value for each of the n coordinates. SP sends $g_j^i + r_i$ to CP for each user U_j corresponding to a coordinate c_i, where r_i denotes a random pad sampled by SP for each coordinate c_i (but the same for all users). The CP decrypts and computes the medians based on these padded values. The CP then encrypts the medians before sending them to the SP. Finally, the SP removes the pad r_i to achieve the desired results $\vec{\mathbf{g}}_y$ by utilizing the underlying encryption scheme's homomorphic property.

1) Leakage: The view of CP while executing SecMed is consolidated in the matrix $\vec{\mathbf{V}}_{\mathsf{SecMed}}$:

$$\vec{\mathbf{V}}_{\mathsf{SecMed}} = \begin{array}{c} \\ U_1 \\ \vdots \\ U_{\mathsf{m}} \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & \cdots & c_i & \cdots & c_{\mathsf{n}} \end{array} \\ \begin{pmatrix} g_1^1 + r_1 & \cdots & g_1^i + r_i & \cdots & g_1^{\mathsf{n}} + r_{\mathsf{n}} \\ & & \vdots & & \\ g_{\mathsf{m}}^1 + r_1 & \cdots & g_{\mathsf{m}}^i + r_i & \cdots & g_{\mathsf{m}}^{\mathsf{n}} + r_{\mathsf{n}} \end{pmatrix} \end{array}. \quad (2)$$

We observe that for each coordinate c_i, CP learns a "shifted" distribution of gradients in clear across all users. This is clearly a violation of privacy [5], [6] because it leaks a lot more information to CP and thus does not meet the design goal of 'Privacy' claimed in [7]. The main source of the leakage is that SP uses *same random pad r_i for all users* with respect to a coordinate c_i. While the aforementioned leakage could be prevented by using different random pads across users, we emphasize that the use of the same pad is unavoidable for the SecMed algorithm to remain correct. In detail, the median of g_j^i values is calculated by first computing the median of $g_j^i + r_i$, then removing r_i from the result. This requires that the same r_i value be associated with each g_j^i value, or the computation's correctness will be violated.

C. Computing Pearson Correlation Coefficient Using SecPear

Once the coordinate-wise medians are computed, the next step in PEFL is to calculate the Pearson correlation coefficient $\rho_{x,y}$ between the coordinate-wise medians $\vec{\mathbf{g}}_y$ and the gradient of the user U_x. This is achieved via the SecPear protocol (cf. Figure 5 in [7]) where SP communicates $\vec{\mathbf{g}}_x \cdot p_x$ and $\vec{\mathbf{g}}_y \cdot p_y$ to CP. The view of CP in SecPear with respect to $\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}}$ is $\vec{\mathbf{V}}_{\mathsf{SecPear}}$:

$$\vec{\mathbf{V}}_{\mathsf{SecPear}} = \begin{array}{c} \\ U_1 \\ \vdots \\ U_{\mathsf{m}} \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & \cdots & c_i & \cdots & c_{\mathsf{n}} \end{array} \\ \begin{pmatrix} g_1^1 \cdot p_1 & \cdots & g_1^i \cdot p_1 & \cdots & g_1^{\mathsf{n}} \cdot p_1 \\ & & \vdots & & \\ g_{\mathsf{m}}^1 \cdot p_{\mathsf{m}} & \cdots & g_{\mathsf{m}}^i \cdot p_{\mathsf{m}} & \cdots & g_{\mathsf{m}}^{\mathsf{n}} \cdot p_{\mathsf{m}} \end{pmatrix} \end{array}. \quad (3)$$

1) Leakage: Similar to the problem with SecMed above, here CP learns the correlation between each coordinate in the gradient vector $\vec{\mathbf{g}}_x$. SP uses the *same random pad p_x for all coordinates*, which causes leakage. However, using different pads for the coordinates does not address the issue since the use of the same pad is required for the SecPear algorithm to remain correct (cf. Proposition 1 in [7]). More elaborately, for $d_x = \vec{\mathbf{g}}_x \cdot p_x$ and $d_y = \vec{\mathbf{g}}_y \cdot p_y$, computation of $\rho_{x,y}$ involves computing the covariance $Cov(d_x, d_y)$ and the standard deviations $\sigma(d_x)$ and $\sigma(d_y)$. As shown in Proposition 1 in [7], the correctness of $\rho_{x,y}$ relies on the following observations:

$$Cov(d_x, d_y) = p_x p_y \cdot Cov(\vec{\mathbf{g}}_x, \vec{\mathbf{g}}_y),$$
$$\sigma(d_x) = p_x \cdot \sigma(\vec{\mathbf{g}}_x)\ ,\ \sigma(d_y) = p_y \cdot \sigma(\vec{\mathbf{g}}_y).$$

If different pads are used for the coordinates, the above relations do not hold, and hence $\rho_{d_x,d_y} = \frac{Cov(d_x,d_y)}{\sigma(d_x)\sigma(d_y)} \neq \rho_{x,y}$.

D. Aggregating the Gradients Using SecAgg

SecAgg, the final stage in PEFL, aggregates the gradients after scaling them with a factor based on the Pearson correlation coefficient calculated in SecPear. SP communicates $\vec{\mathbf{g}}_x + s_x$ to CP for this purpose, as shown in $\vec{\mathbf{V}}_{\mathsf{SecAgg}}$:

$$\vec{\mathbf{V}}_{\mathsf{SecAgg}} = \begin{array}{c} \\ U_1 \\ \vdots \\ U_{\mathsf{m}} \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & \cdots & c_i & \cdots & c_{\mathsf{n}} \end{array} \\ \begin{pmatrix} g_1^1 + s_1 & \cdots & g_1^i + s_1 & \cdots & g_1^{\mathsf{n}} + s_1 \\ & & \vdots & & \\ g_{\mathsf{m}}^1 + s_{\mathsf{m}} & \cdots & g_{\mathsf{m}}^i + s_{\mathsf{m}} & \cdots & g_{\mathsf{m}}^{\mathsf{n}} + s_{\mathsf{m}} \end{pmatrix} \end{array}. \quad (4)$$

1) Leakage: Again, similar to SecMed, $\vec{\mathbf{V}}_{\mathsf{SecAgg}}$ reveals a "shifted" distribution of each user's gradient values across all the coordinates to CP. When combining information from $\vec{\mathbf{V}}_{\mathsf{SecPear}}$ (3) and $\vec{\mathbf{V}}_{\mathsf{SecAgg}}$ (4), a more significant leakage occurs. Consider the gradient at coordinates i, j for user U_x. From $\vec{\mathbf{V}}_{\mathsf{SecPear}}$, CP learns $a_1 = g_x^i \cdot p_x$ and $a_2 = g_x^j \cdot p_x = (g_x^i \cdot \delta_x^{ij}) \cdot p_x$ where $\delta_x^{ij} = g_x^j / g_x^i$. Similarly, CP learns $b_1 = g_x^i + s_x$ and $b_2 = g_x^j + s_x = (g_x^i + \Delta_{ij}^x) + s_x$ from $\vec{\mathbf{V}}_{\mathsf{SecAgg}}$, where $\Delta_{ij}^x = g_x^j - g_x^i$. Given that CP can compute both δ_x^{ij} and Δ_{ij}^x in clear, CP learns g_x^i and g_x^j by solving the equations. For instance, $a_2 = (g_x^i + \Delta_{ij}^x) \cdot p_x = a_1 + \Delta_{ij}^x \cdot p_x$ reveals p_x. Using this method, CP learns the entire gradient matrix $\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}}$, thereby breaching the PEFL system's privacy.

E. Practical and Probabilistic Attacks

Another practical attack on PEFL would be to allow CP to register as an honest user $U_{\mathsf{m}+1}$ in the PEFL system and submit its gradients. This action does not violate the semi-honest assumption of CP in the PEFL threat model and may represent scenarios in which CP has some side channel information about some user gradients. Knowing $\vec{\mathbf{g}}_{\mathsf{m}+1}$, CP learns r_i for all $i \in [\mathsf{n}]$ and hence the gradient matrix $\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}}$ given in (1) in clear from (2) and thereby breaking the privacy of the entire PEFL system.

Considering the matrices $\vec{\mathbf{V}}_{\mathsf{SecMed}}$ and $\vec{\mathbf{V}}_{\mathsf{SecPear}}$ together, we note that it is sufficient for CP to be aware of just one gradient, say g_x^i, to compromise the system's privacy. In particular, g_x^i will allow CP to learn the random pad r_i corresponding to the i-th coordinate c_i in $\vec{\mathbf{V}}_{\mathsf{SecMed}}$, revealing the gradients of all users at that coordinate. Similarly, knowing g_x^i allows CP to learn the random pad p_x corresponding to user U_x in $\vec{\mathbf{V}}_{\mathsf{SecPear}}$ and reveals the gradient vector $\vec{\mathbf{g}}_x$ to CP in clear. CP will now learn the entire gradient matrix $\vec{\mathbf{G}}_{\mathsf{m}\times\mathsf{n}}$ by combining the information from $\vec{\mathbf{V}}_{\mathsf{SecMed}}$ and $\vec{\mathbf{V}}_{\mathsf{SecPear}}$.

Finally, we note that CP can launch probabilistic attacks by looking for similar values in the matrices $\vec{\mathbf{V}}_{\mathsf{SecMed}}$ and $\vec{\mathbf{V}}_{\mathsf{SecPear}}$ and attempting to correlate the random pads. This is possible in PEFL because CP is aware of the correlation between different rows of $\vec{\mathbf{V}}_{\mathsf{SecMed}}$ as well as columns of $\vec{\mathbf{V}}_{\mathsf{SecPear}}$.

REFERENCES

[1] B. McMahan and D. Ramage, *Federated Learning: Collaborative Machine Learning Without Centralized Training Data*. Mountain View, CA, USA: Google AI, 2017.

[2] M. Sheller, A. Reina, B. Edwards, and J. Martin, *Federated Learning for Medical Imaging*. Santa Clara, CA, USA: Intel AI, 2018.

[3] T. D. Nguyen et al., "FLAME: Taming backdoors in federated learning," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 1415–1432.

[4] H. Fereidooni et al., "SAFELearn: Secure aggregation for private federated learning," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2021, pp. 56–62.

[5] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 603–618.

[6] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients—How easy is it to break privacy in federated learning?" in *Proc. NeurIPS*, 2020, pp. 16937–16947.

[7] X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, "Privacy-enhanced federated learning against poisoning adversaries," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4574–4588, 2021.

D VASA: Vector AES Instructions for Security Applications (ACSAC'21)

[MSY21] J.-P. MÜNCH, T. SCHNEIDER, H. YALAME. **"VASA: Vector AES Instructions for Security Applications"**. In: *Annual Computer Security Applications Conference (ACSAC)*. Online: https://ia.cr/2021/1493. Code: https://encrypto.de/code/VASA. ACM, 2021, pp. 131–145. CORE Rank **A**. Appendix D.

https://doi.org/10.1145/3485832.3485897

VASA: Vector AES Instructions for Security Applications

Jean-Pierre Münch
TU Darmstadt
Darmstadt, Germany
jean-pierre.muench@posteo.de

Thomas Schneider
TU Darmstadt
Darmstadt, Germany
schneider@encrypto.cs.tu-darmstadt.de

Hossein Yalame
TU Darmstadt
Darmstadt, Germany
yalame@encrypto.cs.tu-darmstadt.de

ABSTRACT

Due to standardization, AES is today's most widely used block cipher. Its security is well-studied and hardware acceleration is available on a variety of platforms. Following the success of the Intel AES New Instructions (AES-NI), support for Vectorized AES (VAES) has been added in 2018 and already shown to be useful to accelerate many implementations of AES-based algorithms where the order of AES evaluations is fixed a priori.

In our work, we focus on using VAES to accelerate the computation in secure multi-party computation protocols and applications. For some MPC building blocks, such as OT extension, the AES operations are independent and known a priori and hence can be easily parallelized, similar to the original paper on VAES by Drucker et al. (ITNG'19). We evaluate the performance impact of using VAES in the AES-CTR implementations used in Microsoft CrypTFlow2, and the EMP-OT library which we accelerate by up to 24%.

The more complex case that we study for the first time in our paper are dependent AES calls that are not fixed yet in advance and hence cannot be parallelized manually. This is the case for garbling schemes. To get optimal efficiency from the hardware, enough independent calls need to be combined for each batch of AES executions. We identify such batches using a deferred execution technique paired with early execution to reduce non-locality issues and more static techniques using circuit depth and explicit gate independence. We present a performance and a modularity-focused technique to compute the AES operations efficiently while also immediately using the results and preparing the inputs. Using these techniques, we achieve a performance improvement via VAES of up to 244% for the ABY framework and of up to 28% for the EMP-AGMPC framework. By implementing several garbling schemes from the literature using VAES acceleration, we obtain a 171% better performance for ABY.

KEYWORDS

privacy preserving machine learning, secure multi-party computation, VAES.

ACM Reference Format:
Jean-Pierre Münch, Thomas Schneider, and Hossein Yalame. 2021. VASA: Vector AES Instructions for Security Applications. In *Annual Computer Security Applications Conference (ACSAC '21), December 6–10, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3485832.3485897

ACSAC '21, December 6–10, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8579-4/21/12…$15.00
https://doi.org/10.1145/3485832.3485897

1 INTRODUCTION

The primitive of choice for encryption and similar tasks is AES. It is used for communication encryption [71, 83], disk storage encryption [21, 34], and database encryption [72] among other applications. To improve the performance and resource utilization of this important primitive, the AES-NI extension to the x86 instruction set was introduced [5, 56] with common implementations computing AES-128 with ~ 1.3 cycles/byte on one core [2].

History of VAES. Further improving on this, Intel has developed support for vector AES (VAES) instructions [33] and shipped it starting with their Ice Lake microarchitecture [35]. These VAES instructions compute a single round of AES on different blocks, using multiple different round keys [33, 56]. The original paper of [33] already discussed the importance of batching data to the vector AES-NI instructions and microarchitectural properties of these instructions. The authors demonstrated how to apply VAES to several modes of operations of block ciphers such as AES-CTR, AES-CBC, AES-GCM, and AES-GCM-SIV with up to 4× performance improvements. This increased throughput of AES in standard modes of operation can yield direct performance improvements for applications such as storage encryption, disk encryption, database encryption, or secure channels (TLS) [21, 34, 72, 83, 86] and VAES is already included in the popular OpenSSL library [89]. Subsequently, Drucker and Gueron showed how to use VAES to accelerate Pseudo-Random Functions (PRFs) and Pseudo-Random Generators (PRGs) [30]. Multiple NIST Post Quantum Cryptography Project candidates use Deterministic Random Bit Generators (DRBGs) for which the implementation of Drucker and Siri achieves up to 4× performance improvement using VAES [29]. The contribution of this VAES-accelerated DRBG was evaluated for the post-quantum secure multivariate-polynomial signature scheme Rainbow [26] in [31], and for the key encapsulation mechanism BIKE [6] in [32].

Our Motivation. What is common to all these applications of VAES studied before is that the algorithm is fixed beforehand, and hence the parallelization can be done manually. For maximum throughput with VAES, the main challenge is to batch enough independent AES calls together for the AES hardware units to be constantly busy and not idle when processing blocks.

However, finding a good batching becomes much more challenging when the algorithm and hence sequence of AES operations is not fixed in advance. Some AES operations can depend on the output of others but some do not and many small memory-abstracted library invocations are expensive. This batching problem and its solutions are not unique to AES on x86-64 using VAES (which is

Table 1: Summary of our performance improvements. New Batched AES-NI indicates whether the implementation received an *additional* batching AES-NI implementation. VAES indicates whether the performance improvement includes VAES.

Framework	New Batched AES-NI	VAES	Max. Total Improv.
ABY (Ref) [13, 25, 97]	✓	✓	244%
ABY (Custom) [25, 40, 41, 97]	✗	✓	171%
EMP-OT [90]	✗	✓	30%
EMP-AGMPC [90, 92]	✓	✓	24%
CrypTFlow2 [82]	✗	✓	52%

our focus). It can be generalized to all non-trivial implementations of cryptographic primitives which includes pipelined AES implementations on ARM [7], bitsliced AES implementations [17, 63] as well as more unusual techniques like instance-vectorized hash functions. A natural area where such complex dependencies occur is Secure Multi-party Computation (MPC), especially garbled circuits [10, 40, 67, 85, 95, 97], which is why we use them for assessing the performance impact for VAES. More concretely, with garbled circuits, typically binary circuits using primarily AND and XOR gates are evaluated with XOR gates only requiring XOR operations [67], whereas AND gates do require AES operations to be and sending ciphertexts. These garbled circuits can then be used for high performance secure two-party computation, interactive zero-knowledge proofs of arbitrary statements [44, 60, 97], and other applications.

MPC allows to securely compute a public function on private input data provided by multiple parties and hence is an interactive way for computing under encryption. Since several years, a multitude of companies, including Alibaba, Bosch, NTT, and Unbound among many others in the MPC Alliance [4], are working on MPC technology. We study the ABY framework [25] for passively secure two-party computation and the EMP-AGMPC [90, 92] framework for actively secure multi-party computation. As we are manually changing the implementation of these schemes without changing the protocols, we substantially increase the deployability of these frameworks and dependent works as well as providing guidance to how similar effects can be achieved for similar frameworks.

Privacy-preserving machine-learning (PPML) is a popular application of MPC. Here, general machine-learning techniques are run on private data while also protecting the model parameters. The private output is the inference or training result [39]. PPML has become a hot topic in recent years and gained the attention of major software, service and hardware vendors, e.g., Facebook [66], Google [16], Intel [15], and Microsoft [82], all of whom are working on increasing its practicality. Applications of PPML include private healthcare-based inference, e.g., to predict illnesses [22, 69, 84], private healthcare model training to acquire models without having to reveal patient data [1], and private clustering to partition data according with common features [73]. In particular, in this work, we discuss private ML inference in the state-of-the-art framework Microsoft CrypTFlow2 [82] where one party holds a pre-trained model and the other a data item to be classified and then the protocol allows classification using the model without the two parties revealing their private inputs. We improve CrypTFlow2 [82] using VAES. As our focus lies on manual implementation improvements, we substantially increase such PPML applications' deployability without sacrificing compatibility or security.

Our Contributions. Our main contributions are as follows:

- We expand the focus of VAES from microarchitectural issues where the order of AES operations is fixed a priori, to protocol and implementation design where the sequence of AES operations is not known in advance. For this, we introduce automatic batch identification and computation techniques for efficient use of AES in complex security applications.
- We report the first performance measurements for VAES in the area of Multi-Party Computation (MPC) and show performance improvements for the MPC frameworks ABY, EMP-OT and EMP-AGMPC, as well as the PPML framework CrypTFlow2. Our improvements are summarized in Table 1.
- We provide our implementations for re-use by others and as guidance for future implementation efforts at https://encrypto.de/code/VASA.

Outline. The rest of this paper is organized as follows: We start with providing the necessary background on the investigated types of MPC and the hardware acceleration of AES in x86 processors (§ 2). Next, we provide context to our work with related work (§ 3). Following that, we describe our computational framework for efficient batch identification and computation and how we applied it (§ 4). Next, we evaluate and discuss the performance of the applications (§ 5). Finally, we conclude and provide possible future research directions (§ 6).

2 BACKGROUND

In this section, we provide a brief background on secure multi-party computation and how AES is computed using AES-NI and VAES on x86-based processors.

2.1 AES Computation

There are two instruction set extensions on x86 for providing functionality relating to the computation of AES: the AES new instructions (AES-NI) and the vector AES instructions (VAES) [5, 33, 56]. For the encryption direction, the key instructions from these extensions are `AESENC` and `AESENCLAST` which compute a single AES round and the last AES round, respectively. The difference between AES-NI and VAES is the instructions' width and how many blocks and round keys they work with: AES-NI is restricted to one and VAES also allows two or four. Thus, one can compute AES-128 by chaining an XOR operation with nine `AESENC` and one `AESENCLAST` using a pre-expanded key. The key expansion itself can also take advantage of the `AESENCLAST` instruction and is most efficiently

done using the technique of Gueron et al. [40]. As most modern x86 processors providing the AES extensions are pipelined, the data dependency between the AES instructions can lead to pipeline stalls if not filled otherwise. This is the reason why multiple independent AES calls are batched together, allowing interleaved execution of the instructions, i.e., starting execution of the second round of all batched AES calls before starting execution of the third round of any one of them.

This leads to optimal, minimal sizes for batches of AES calls which depend on the microarchitecture involved as they need to hide the latency of the instructions using the throughput and the width of the instructions. A summary of these performance characteristics using the data of [38] for modern x86 processor architectures is provided in Table 2. The performance characteristics of 128-bit AES instructions have remained the same for all successors of AMD's Zen architecture so far. Also the performance characteristics of the `AESENC` and `AESENCLAST` instructions are identical.

Table 2: AES-NI and VAES instruction latencies, throughput [38], and resulting minimal batch size for optimal efficiency. Width 128 bits corresponds to AES-NI and other values are VAES. Cycles per instruction is abbreviated as "cyc/instr".

Architecture	Width [bits]	Latency [cycles]	Throughput [cyc/instr]	Minimal Batch Size
Intel Haswell	128	7	1	7
Intel Skylake	128	4	1	4
Intel IceLake	128	3	0.5	6
	256	3	0.5	12
	512	3	1	12
AMD Zen	128	4	0.5	8
AMD Zen3	256	4	0.5	16

2.2 Secure Multi-Party Computation

The goal of secure multi-party computation (MPC) is to compute arbitrary functions among multiple parties on private inputs only known to one party each [12, 14, 78, 94, 95]. Most relevant for this work are protocols for oblivious transfer (OT), garbled circuits (GC), and privacy-preserving machine-learning (PPML).

Oblivious Transfer (OT). In oblivious transfer, one party (the receiver) inputs a choice bit and the other (the sender) supplies two messages. The receiver then learns only the message corresponding to the choice bit. The computation of OT protocols typically uses a small number of invocations of a public-key-based OT protocol [23, 74] to extend to a larger number of OTs using symmetric cryptography [8, 9, 57]. The primary bottleneck of these OT extension protocols are the communication time, the computation of a bit matrix transposition, and the computation of encryption operations using AES [8]. Common variants of the above OT functionality which allow to decrease communication are random OT (R-OT) where the sender gets two random strings and the receiver gets one of them depending on the choice bit, and correlated OT (C-OT) where the sender can input a correlation that the returned strings have to satisfy. Additionally, there has been a line of research looking to further minimize the communication needed for C-OT using a learning parity with noise (LPN) assumption [18, 19, 93]. These pseudo-random correlation generators (PCGs), like FERRET [93], reduce communication at the expense of computation, and increased complexity where a large matrix-vector product with randomized entries is computed.

Garbled Circuits (GC). Secure computation of general functions is typically performed using a circuit-oriented representation of that function. Garbled circuits (GCs) are one approach for this, originally proposed for two parties [95] and later generalized multiple parties [12]. In GC, the key invariant is that each wire's value is represented by two random keys which represent the zero and one bits. The garbling party knows both wire keys and the evaluating party only ever learns one key for each wire. For each gate a garbled table is generated forming the garbled circuit, to allow translation of a given pair of gate-input-wire keys to the output wire key corresponding to the correct output bit. The evaluator obtains the keys corresponding to the circuit input wires via OT. Early constructions [12, 75, 95] used garbled tables that could effectively be generated in parallel due to a lack of data dependencies. However, more modern schemes like free-XOR [67], HalfGates [97], or PRF-based garbling [40] require a topologically ordered processing of gates in exchange for requiring only two ciphertexts instead of three per AND gate, and XOR gates require no communication in free-XOR [67] or one ciphertext in PRF-garbling [40]. As these schemes require at least four applications of a cryptographic function on some counter or gate identifier as well as the gate input keys to generate the tables, most implementations use AES with a fixed key [13, 42] though instantiations with variable keys were also proposed in [40, 41]. Yao's garbled circuits protocol described above initially provides security against passive adversaries [70] and there have been extensions in research to security against active adversaries [51, 76, 77, 91, 92] that can arbitrarily deviate from the protocol specification. The latest of these schemes [91, 92] uses the free-XOR optimization [67] and parties jointly compute authenticated versions of the garbled tables so that a malicious garbler does not know the actual tables nor can tamper with them while a malicious evaluator only sees random-looking ciphertexts.

AES vs. LowMC. With free-XOR [67] and the S-box of [17], a Boolean circuit for AES consists of 5 210 AND gates [47]. Starting with LowMC [3], several dedicated MPC-friendly block ciphers have been designed that minimize the number of AND gates (or also multiplicative depth) over AES [3, 27, 28, 61]. Due to their smaller and/or shallower circuits, such *MPC-friendly block ciphers improve the function that is evaluated via MPC*, e.g., to privately evaluate a block cipher, called Oblivious Pseudo-Random Function (OPRF) [81], which has several applications like private set intersection for unbalanced set sizes in private contact discovery [62, 65]. However, the *MPC protocols themselves are still implemented with AES* (e.g., garbling schemes, OT extension, or PRFs). The reason for that is the superb performance of hardware acceleration of AES in today's CPUs which are highly optimized ASICs that require only ~ 1.3 cycles/byte on one core using AES-NI [2]. In our paper, we

show how the efficiency of such implementations of MPC protocols can be further improved by using VAES.

Privacy-Preserving Machine-Learning (PPML). The goal of PPML is to apply machine-learning techniques while preserving the privacy of the data and models [37, 39, 45, 64]. While this application can include training and inference [39], we focus on inference, in particular on inference for neural networks as done in Microsoft CrypTFlow2 [82]. This involves computing the linear and non-linear stages using optimized protocols for the client's private data input and the server's private model input, only yielding the result to the client. We note that the practicality of PPML has improved drastically over time to the point where now accurate, full-sized neural network inference is possible in a privacy-preserving setting even on moderately powerful hardware [82].

3 RELATED WORK

In this section, we discuss how our work relates to previous work. In particular, we discuss the relation to previous protocol-level and implementation-level improvements.

3.1 Protocol-Level Improvements

One primary direction for research in the past has been to improve the protocols themselves, e.g., by reducing the amount of communication or the number of invocations to computationally expensive primitives [10, 43, 67, 75, 85, 91, 97]. In addition, some works handle the circuit generation for MPC protocols from specifications in a high-level language by using industry-grade hardware synthesis tools and tweaking them for logic synthesis [24, 46, 79, 87]. Our work is largely orthogonal to these approaches as we focus on improving the implementations and the frameworks used for them. However, there are advances in protocol design which significantly complicate efficient implementation, e.g., the requirement for gates in circuits to be processed in topological order [40, 67, 97]. There have been prior works that modified the protocol and increased communication to allow for more efficient computation [55], but we do not follow their approach and maintain protocol compatibility. This focus on implementation improvements for relatively low-level building blocks allows protocol compatible performance improvements for the discussed protocols and those building on top of it. Such works include Cerebro [98], TinyGarble2 [52], and CrypTFlow2 [82] all of which build on EMP [90] and can thus profit from our improvements of EMP.

3.2 Implementation-Level Improvements

Another major direction has been improving the implementation of the protocols. This has seen four sub-directions: Improving the performance of individual operations, improving the parallelization of the implementation, improving the memory behavior, and using dedicated hardware to accelerate computationally expensive steps.

Operations. In OT extension, bit matrix transposition is one of the most computationally expensive operations [8]. Previous optimizations of this operation have been using an asymptotically optimal transposition algorithm [36], or 128-bit vector registers [90]. We improve on the latter through the use of wider AVX512 vector registers instead. Beyond this, OT extension has been a major application of fixed-key AES [13] on which we improve through the use of VAES instead of AES-NI for the implementation. Furthermore, there have been efforts to increase the performance of individual operations in GC, e.g., improving the implementation performance of the individual garbling and evaluation operations for individual gates [13, 40]. We improve upon these prior works by considering multiple gates of the same type at once. A natural question is, whether a library like OpenSSL can be used for implementing AES operations. This is an appropriate solution if only large batches of AES calls occur and these are well-supported by OpenSSL. However, this would not allow the use of VAES which is currently not used by OpenSSL, and it would bring significant overhead for smaller batches due to the memory abstraction needed.

Parallelization. Previous work to parallelize the evaluation of garbled circuits has seen coarse- and fine-grained approaches [11, 20, 50, 55]. Coarse-grained approaches [11, 20] are typically used to have multiple threads compute different parts of the same garbled circuit and are largely orthogonal to our in-thread optimizations of the computation strategy. Alternatively, they may have traded communication, e.g., not using free-XOR, for added parallelism to exploiting using dedicated hardware like graphics processing units or Intel Quick Assist Technology [55]. The more fine-grained approaches [11, 20] have primarily focused on using a layering technique, as we also discuss, however, intending to outsource the work to different threads instead of exploit the high instruction-level parallelism that modern processors provide. Additionally, previous work has suggested splitting the garbling and the evaluating roles with a suitable sub-division of circuits [20] or overlapping the computation with the garbling and evaluation operations [50], both of which are orthogonal to what we do.

Memory Behavior. A smaller line of previous research has explored the limitation of memory use for GC [48, 52, 68, 87, 96]. Their motivation for this was two-fold in allowing the computation of large circuits not fitting into most memory configurations and improving locality for caches through smaller code and data. We note that the techniques to only partially load circuits into memory are orthogonal to ours, requiring at most invoking early execution occasionally. We also consider cache locality important. However, our focus is more on the actual computation and the first-level cache as opposed to keeping the data in a cache at all.

Hardware-Acceleration. There has been a line of research using field-programmable gate-arrays (FPGAs) to accelerate garbled circuit operations [53, 54, 58, 59, 88]. Our work is independent of and alternative to the main contributions of these prior works. However, the scheduling discussed for FASE [53] is similar for hardware to what we do for identifying batches, though their techniques are focused on the specific dedicated hardware architecture they build, making it unsuitable for our software-oriented approach.

4 OUR FRAMEWORK

The first step in our manually implemented techniques to apply VAES is the identification of batches of independent AES calls for small-scale batch processing (§ 4.1). The second step is to process the AES operations (§ 4.2). Finally, we show how we used these techniques with the ABY [25], EMP-OT [90], CrypTFlow2 [82], and EMP-AGMPC [90, 92] frameworks (§ 4.3).

4.1 Batch Identification

For identifying batches, we use two approaches: *dynamic batching* and *static batching*. *Dynamic batching* primarily uses runtime information for minimally invasive batching. *Static batching* provides reusable batching information from preprocessing but requires more substantial changes to the code.

4.1.1 Dynamic Batching. The core idea behind dynamic batching is to defer execution of operations until they are actually needed and to compute all pending operations when *one* is needed. In processing circuits, the application of this works by modifying the main processing loop iterating over all gates and adding AES-based AND gates to a queue and processing all queued AND gates as a batch once any one of them is referenced as an input dependency. An example of when the processing is invoked is provided in Fig. 1. Implementing this technique requires potentially a few hours of manual effort to identify the core processing loop, to implement the deferred execution identification, and to identify relevant modifications and extensions which we briefly discuss next.

Correctness Extensions. The basic technique works well if there is one type of non-free gates requiring AES operations. However, some schemes have AND and XOR operations requiring AES operations using a shared gate index counter to uniquely produce values per-gate. For these, new design space choices manifest, in particular, whether it is possible and desirable to separate the domains of the counters or to track the gate identifiers as well and not just the minimal information for computing the gate. Additionally, one can imagine that it is possible to not maintain separate queues for the different gate types but rather join them into a shared one which complicates the gate processing at the potential of gained performance through more AES calls being potentially batched together to reach minimum optimal batch size even in complex circuits. Furthermore, we note that dynamic batching can be combined with the approach of having a variable number of cryptographic gates associated with an administrative gate, in which case it is beneficial to track the number of actual gate tasks associated with each administrative gate and keep a global count to allow the batch processing algorithm to choose appropriate sub-batches. Both of these extensions each require a few hours of effort for the architectural changes.

Optimizations. The basic batching techniques have further optimizations. First, the use of this batching can inadvertently lead to significant gaps in time between visiting and enqueuing a gate and processing it, meaning it might be pushed out of registers or lower-level caches. To avoid such unloads, one should consider to regularly empty the queue by processing the stored tasks even if more tasks could still be added without violating correctness. This holds especially true if any given processed sub-batch only processes a small number of gates, e.g., $b = 4$, and the queue has reached a size that is a multiple of b. Additionally, one can consider to only partially process the stored tasks in the queue using a multiple of the preferred processing width to potentially allow more gates to be directly enqueued without triggering processing at an undesirable length. When the basic technique encounters an AND gate referencing a queued AND gate, it will always trigger the computation of all queued AND gates. Another optimization in this scenario is to check whether the referenced AND gate is early enough in the queue which is guaranteed to have been processed once the processing reaches the current AND gate and then enqueueing the current AND gate without triggering processing. The implementation effort for these optimizations potentially requires a few hours of effort on top of the basic queue implementation.

4.1.2 Static Batching. A different approach than the dynamic technique is to preprocess the circuit to gain more holistic information on batching opportunities. These techniques can be paired with dynamic batching techniques for further improved efficiency. The three techniques we discuss are layering (identifying layers of dependencies), SIMD (grouping multiple guaranteed independent gates into one administrative one), and a more generic smart arrangement.

Layering. Layering techniques assign a gate to how many non-free gates lie between it and the original input. Non-free gates on equal layers are then necessarily independent and each layer can be seen as a batch of AES calls to be computed. An example of associated layers is provided in the right graph of Fig. 1. Layering can be done in addition to dynamic batching which can potentially identify independent tasks across layers, e.g., if the first gate of the second layer references the first gate of the first layer and early evaluation or peephole optimizations allow such batches. The effort to add layering support to an implementation varies significantly with the architecture and can range from a few hours for adding, computing and using the attribute to significantly more if a more complex processing strategy than a sequential loop is used.

SIMD. Single-instruction multiple-data (SIMD) gates are explicitly specified administrative gates that represent the same gate being applied to multiple input wires in parallel. They present natural opportunities for batches and even allow batching techniques in more complex gate scheduling scenarios where other techniques are not applicable. The cost to this is either the identification of such SIMD tasks or the need for the execution of a circuit several time as a batch as well as the need for explicit program-level representation. Similarly to layering, the implementation cost for SIMD gates varies with the architecture and can quickly take a dozen or more hours. As all gate processing methods need to be SIMD-aware, gates must be extracted and collected and SIMD gates must be specified or detected in a given circuit description.

Smart Arrangement. This technique is more general and provides heuristics for circuit generators and manually optimized building blocks of gates. For example, circuit generators should output circuits that allow circuit-internal SIMD gate operations and prefer

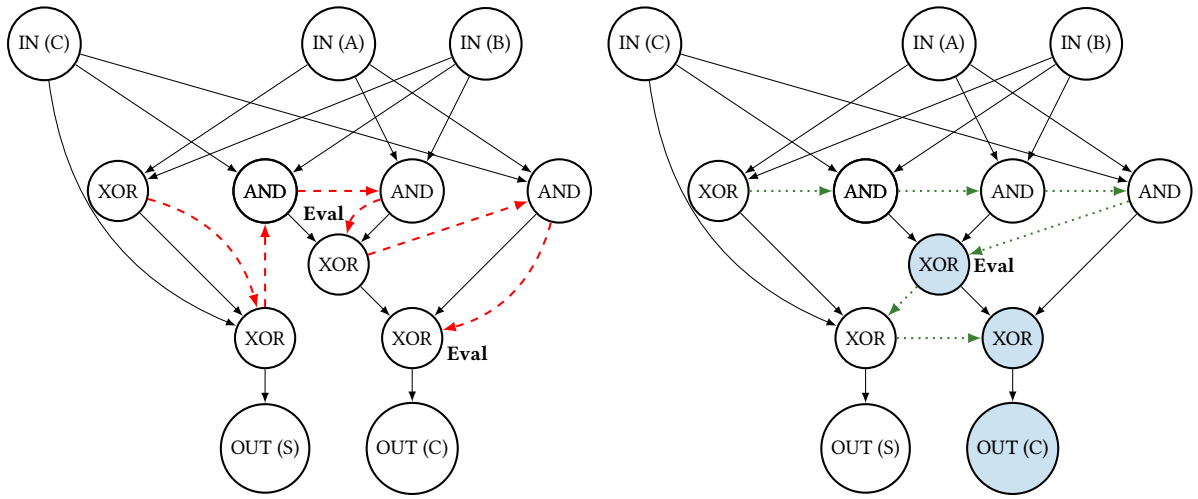


Figure 1: A simple 1-bit adder with different manually chosen gate orderings as an illustrative example for the freedom of topological ordering. Solid black arrows denote data dependencies, red dashed arrows denote one possible sub-optimal ordering in the left graph and green dotted arrows show a preferable ordering in the right graph. The "Eval" marks denote places where dynamic batching with free XORs would trigger processing of the queued fresh AND gates. All unfilled nodes are on the first layer in the right figure and all light-blue-filled nodes are on the second layer. A layer is defined to be the set of all nodes with the same amount of non-free (AND) gates between them and the input on the critical path.

larger layers over smaller layers. An example of such improved gate arrangement is provided from the left to the right graph in Fig. 1. Additionally, locality has to be considered when generating circuits, i.e., usage of wires must stay close to where they are generated as not to push the wire values out of caches, while maintaining enough distance to allow batching on current and more instruction-level parallel future architectures.

4.2 Batch Computation

After one has identified a batch of independent AES calls, they need to be computed. For this, we have used two techniques: register-oriented computation, which focuses on performance and simplicity to the compiler, and memory-oriented computation, which focuses on modularity.

4.2.1 Register-Oriented Computation. Our primary technique for processing batches describes the task computations as low-level as possible without resorting to assembly. By using vector register types and constant-sized loops we give the compiler as many opportunities for optimization as possible while still allowing the conciseness of high-level code. Concretely, we have identified five steps executed continuously in a loop for all tasks.

1) Fill the appropriate lanes of the vector values with the task-specific data, both non-vector computable and loaded data, e.g., the lane 0 (the lowest 128 bit of the value) of all three virtual registers are assigned to gate 0, whereas lane 1 of all three is assigned to gate 1 and hold the input wire keys and a processed garbled table value. 2) Perform vectorizable operations on the input data, e.g., deriving computed inputs from loaded inputs with a global offset. 3) Perform the AES operations on the prepared inputs and keys with a sufficiently large batch size. 4) Execute vectorizable post-processing on the results and potentially other input values, e.g., XORing pairs of AES outputs as required by the scheme. 5) Do the remaining post-processing and scatter the data back to memory, e.g., handle operations that cannot be vectorized and where data needs to be extracted from the vectors first. Then, write the values back to the memory location where they are expected.

The cost of such a low-level approach is, of course, that not just the AES code needs to be re-written to satisfy the types of each used architecture and extension but also the immediately surrounding code leading to significant code duplication. An example implementation for HalfGate's [97] AND evaluation with fixed keys [13] and VAES is given in Listing 1 in Appendix A. Depending on the familiarity of the developer with the available platform instructions, their invocation, and the availability of validation methods, this register-oriented technique can be implemented within a few hours per optimized functionality.

4.2.2 Memory-Oriented Computation. Our memory-oriented technique addresses the code duplication concerns of the register-oriented one but can result in less performance. In particular, it only requires that a core primitive for this technique, e.g., electronic codebook mode, is implemented in an architecture-specific way. This core primitive is then used with a memory abstraction wherever needed while ensuring a sufficiently large number of AES calls for every invocation. The main loop for this only consists of three steps: 1) perform the data loading and preprocessing, 2) let the optimized library perform the operations, and 3) read the results using the memory abstraction and post-processing and store them. The pre-processing and post-processing steps for this approach can use platform-independent instructions lowering code duplication

Table 3: Overview of improved frameworks, used batch identification methods, and batch computation strategies used.

	Batch Identification (§ 4.1)		
Framework (§ 4.3)	Dynamic (§ 4.1.1)	Static (§ 4.1.2)	Computation (§ 4.2)
ABY [25] (§ 4.3.1)	Non-Free-XOR + SIMD	SIMD	Register-Oriented (§ 4.2.1)
EMP-OT [90] (§ 4.3.2)	—	—	Memory-Oriented (§ 4.2.2)
EMP-AGMPC [90, 92] (§ 4.3.3)	Regular-Early-Execution	—	Memory-Oriented (§ 4.2.2)
CrypTFlow2 [82] (§ 4.3.4)	—	—	Memory-Oriented (§ 4.2.2)

for handling a batch of gates at a time. However, this technique has performance overhead if implemented this way as implementing counter-mode can be significantly slower than with a dedicated implementation as the compiler might generate general-purpose 64-bit store instructions and adds from the abstract code. In contrast, a direct use of 64-bit vector additions might be significantly faster. An example implementation for EMP-AG2PC's [90, 91] AND evaluation with fixed keys [13] and VAES is given in Listing 2 in Appendix A. As this technique favors engineering efficiency over runtime efficiency, the required effort for its implementation is generally a few hours if a pre-existing implementation can be adapted and some form of batch identification has already been implemented.

4.3 Frameworks

To measure the performance impact of batching, VAES, and the above techniques we have applied them to the MPC frameworks and libraries ABY [25], EMP-OT [90], and EMP-AGMPC [90], and the PPML framework Microsoft CrypTFlow2 [82]. We will now briefly discuss our changes to each framework and library and provide an overview in Table 3.

4.3.1 ABY. We chose to use ABY [25] as it is a flexible, optimized framework for mixed-protocol secure two-party computation. For our modifications, we targeted the GC subcomponent of ABY which uses HalfGates garbling [97] with a fixed AES key [13] and invokes OpenSSL individually for every single AES operation used. We changed this fixed-key AES garbling, which we call "PRP" based on the public random permutation assumption used, to use a register-oriented computation. We furthermore added to ABY support for two more instantiations of the encryption functions in the HalfGates [97] garbling scheme: CIRC [97] is based on a circular security assumption and uses the wire keys as AES keys. MI [41] provides better multi-instance security and uses the wire key as the data input and the gate index as the AES key starting from a random offset. We note that these three schemes "PRP" / "CIRC" / "MI" need 0 / 4 / 2 computations of the AES key schedule to garble an AND gate respectively. Garbled circuit evaluation requires 0 / 2 / 2 key schedules per AND gate respectively. Neither the evaluation nor the garbling of XOR gates requires communication or AES operations with HalfGates.

Furthermore, we added an implementation of the PRF-based garbling scheme of Gueron et al. [40] which is secure in the standard model. It uses 8 AES operations with 4 keys for garbling an AND gate, 2 uniquely keyed operations for evaluating an AND, 3 uniquely keyed AES operations for XOR garbling, and 1-2 uniquely keyed AES operations for XOR evaluation. We identify batches using dynamic batching with support for SIMD gates and with support for two queues with shared indices for the PRF-based scheme. For all these four schemes, we implemented two register-oriented backends each for the batch processing: one using AES-NI and 128-bit operations, and another one using VAES and AVX512.

4.3.2 EMP-OT. We chose EMP-OT [90] because it is a state-of-the-art implementation for oblivious transfer and it is the underlying OT library for the two frameworks in § 4.3.3 and § 4.3.4 and other recent works [52]. We modified the main OT protocol implementations [8, 9, 57] by replacing the AES-NI based ECB and pseudo-random generator (PRG) implementations in the referenced EMP-Tool library [90] with VAES and widened the batch size from 8 to 16. Additionally, we widened the bit matrix transposition algorithm to use 512-bit AVX512 operations instead of 128-bit SSE operations. Finally, we changed the LPN-based FERRET OT [93] implementation to use VAES instead of AES-NI for selecting the matrix-vector multiplication entries.

4.3.3 EMP-AGMPC. The EMP-AGMPC [90, 92] framework provides a low-communication actively secure garbling scheme. For the implementation, we used a memory-oriented computation strategy mirroring the modular design of the EMP toolkit that strongly encourages modularity. We used basic dynamic batching with early execution for the online and preprocessing phases' circuit processing. In the corresponding EMP-OT library [90] which implements the actively secure OT extension of [9], we instantiate the PRG using VAES.

4.3.4 CrypTFlow2. Microsoft CrypTFlow2 [82] is a state-of-the-art framework for general PPML neural network inference. The implementation uses a sub-part of EMP-OT [90] for OT operations. We extended the modular implementation of CrypTFlow2 with VAES-based implementations for: 1) the 128-bit and 256-bit PRGs, 2) the AES-NI based ECB, and 3) the circular-secure correlation robust function in the garbling scheme of Gueron et al. [40].

5 EVALUATION

This section presents the benchmarking platform and the performance results we achieved for the frameworks from § 4.3.

5.1 Evaluation Platform

For all measurements, we use an Apple Macbook Pro with an Intel Core i7-1068NG7, 2x16GB of dual rank Samsung LPDDR4-3733 RAM (K4UCE3Q4AA-MGCL). It runs Arch Linux using the Linux 5.9.13.arch1-1 kernel along with GCC 10.2.0 and Clang 11.0.0 which were used for compiling the code. For comparative AES-NI measurements we use the same machine.

5.2 ABY

For ABY (cf. § 4.3.1), we ran the benchmarks with both parties locally using a single sample per triple of circuit, scheme and implementation backend (reference, AES-NI, and VAES). For each measurement, the garbling times are taken from the logs of the party running the garbling operation and the data-input-dependent online time from the other party running the evaluation which are executed after each other in ABY. This is done to capture the pure computation time for garbling and evaluation. For the evaluation, we use circuits of AES (with 65× parallel SIMD), SHA-1 (with 512-bit input and 63× parallel SIMD), and for circuit-based private set intersection (PSI) the sort-compare-shuffle (SCS) circuit (1024 elements of 32-bits) [49], and circuit phasing (1024 elements per side of 32-bit, 3 hash functions, $\varepsilon = 1.2$, stash of size 1) [80]. For the summary in Table 4, we computed the geometric mean over the performance results of the four above circuits. The detailed measurements are given in Table 8 in Appendix B. The binaries were produced by GCC. We note a range of performance improvements from the use of batched execution of 67 - 161% and an additional 17 - 171% from the use of VAES. In particular, we observe better performance improvements from VAES for garbling schemes needing more cryptographic operations per gate, e.g., circularly secure computation (CIRC) benefits more than public-random permutation based computation (PRP) (cf. § 4.3.1).

Discussion. We make two key observations for the ABY benchmarks in Table 4: First, using batch sizes larger than one increases the throughput, as can be seen from the runtime decrease of the baseline reference (by 80-130%). Second, the use of VAES does increase performance further, more so in scenarios where more AES operations are done per gate, i.e., with the schemes not using fixed AES keys with HalfGates [13, 97]. Additionally, an investigation using a profiler showed a high miss-speculation rate for the AES-NI code using regular "if" branches with the condition depending on an unpredictable label bit. Therefore, the use of masking facilitated by AVX512 is a secondary factor contributing to performance as it does not invoke speculative execution miss-predicting the branch with 50% probability. Finally, we note the odd behavior that multi-instance secure computation (MI) is significantly slower than circular-secure computation (CIRC) for AES-NI during the evaluation even though they should be tied given that they perform similar AES operations. Concerning the impact of VAES beyond improving speculative execution behavior, we see performance increases of 27% (garbling) and 36% (evaluation) for fixed-key AES because the AES processing makes up only a somewhat small amount of processing time. The HalfGates variable-keyed schemes see a 47% (MI garbling), 43% (CIRC evaluation), and 57% (CIRC garbling) performance increase. PRF-based garbling schemes see the largest increase with 51% (garbling) and 75% (evaluation) due to a large amount of AES operations necessary, given that each AND gate garbling requires 8 AES operations, each AND evaluation 2, each XOR garbling 3, and each XOR evaluation at least 1.

5.3 EMP-OT

For oblivious transfers, we evaluated EMP-OT [90] (cf. § 4.3.2). We ran it single-threaded with 100 million OT operations computed on localhost. For the one-time base OT operations, that use public-key crypto, the default number of OT operations was used, and times were excluded from the throughput results. As base OT protocols, we use the protocol of Naor and Pinkas [74] for passive security assumptions and SimplestOT [23] for active security, except for FERRET OT [93] which uses its own base OT protocol. The library uses fixed-key AES for its PRG [13], the optimized version of [8] of the protocol by Ishai et al. [57] for passive security, and the variant by Asharov et al. [9] for active security.

In addition, we also measured the performance of FERRET-OT [93] as it is a protocol with very little communication after the initial base OTs. EMP-OT was compiled with Clang. The results are shown in Table 5. We note the range of performance improvements of 14.8 - 30.1% from the use of VAES. We also observe that the performance increase is particularly high for random OTs (R-OTs) which can be attributed to a lower amount of system interaction due to the reduced amount of communication for R-OTs.

Table 4: Geometric means of the run-times in milliseconds of ABY [25] for the evaluation of AES, SHA-1, SCS-PSI, and Phasing-PSI with the detailed parameters as described in § 5.2. "Ref" indicates the reference ABY implementation, AES-NI and VAES indicate batched implementations. Garbling scheme names are as introduced in § 4.3. Improv% shows the performance improvement of VAES over AES-NI.

		Garbling Scheme			
Operation	Impl.	PRP	MI	CIRC	PRF
Garbling	Ref [25]	110.6	—	—	—
	AES-NI	47.1	61.0	72.1	197.4
	VAES	37.0	41.3	46.0	130.3
	Improv%	**27.2%**	**47.5%**	**56.7%**	**51.5%**
Evaluation	Ref [25]	56.5	—	—	—
	AES-NI	31.1	59.8	41.3	103.3
	VAES	22.9	29.4	28.9	59.0
	Improv%	**36.1%**	**103.5%**	**43.0%**	**75.0%**

Discussion. From the OT performance data in Table 5, we see that AVX512 and VAES notably improve performance, by 20 - 30% for the EMP libraries' traditional OT implementation, which use VAES for the PRG and AVX512 for bit transposition. Additionally, we observe mild performance improvements of 16.6% for the FERRET protocols, mainly using AES to generate the random matrices in the core matrix-vector multiplication.

Table 5: Run-times in seconds of 10 million OTs for EMP-OT [90] before "Ref" and after implementation of VAES support. The functionalities are general OT (OT), Correlated OT (C-OT), and Random OT (R-OT). Improv% shows the performance improvement of VAES over AES-NI. Higher throughput is better.

			OT Functionality		
Security	Library	Impl	OT	C-OT	R-OT
Passive	EMP-OT IKNP [8, 57]	Ref [8, 57, 90]	0.35	0.20	0.33
		VAES	0.28	0.16	0.25
		Improv%	**20.0%**	**20.0%**	**24.2%**
	EMP-OT FERRET [93]	Ref [90, 93]	1.33	1.14	1.32
		VAES	1.13	0.99	1.09
		Improv%	**15.0%**	**10.4%**	**17.4%**
Active	EMP-OT ALSZ [9]	Ref [9, 90]	0.39	0.24	0.38
		VAES	0.32	0.19	0.29
		Improv%	**17.9%**	**20.8%**	**23.7%**
	EMP-OT FERRET [93]	Ref [90, 93]	1.38	1.2	1.37
		VAES	1.21	1.04	1.16
		Improv%	**12.3%**	**13.3%**	**15.3%**
	+ Random Choice	Ref [90, 93]	—	0.94	—
		VAES	—	0.80	—
		Improv%	**—**	**14.8%**	**—**

Table 6: Run-times in milliseconds for the evaluation of various parts of SHA256 in EMP-AGMPC [90, 92] (§ 5.4). The computation backend ("Comp. Backend") indicates the implementation strategy used. The evaluated parts are the one-time setup, the function-independent preprocessing, the function-dependent preprocessing, and the input-dependent online phase. The values in parenthesis show the performance improvement in percent over the reference. Lower run-times are better.

	Operation			
Comp. Backend	Setup	Function-Independent	Function-Dependent	Online
Ref [90, 92]	45.0	564.5	247.0	7.0
VAES	45.9 (**−2.1%**)	580.7 (**−2.8%**)	250.6 (**−1.4%**)	6.7 (**5.0%**)
Batched + VAES	45.4 (**−0.9%**)	453.0 (**24.6%**)	250.7 (**−1.5%**)	7.0 (**0.7%**)

Table 7: Geometric mean of run-times in seconds for CrypTFlow2 [82] inference (§ 5.5) using the SqueezeNetImgNet, SqueezeNetCIFAR, ResNet50, and DenseNet121 networks. Ring32-OT denotes the 32-bit ring-based implementation using OT. "Ref" indicates the reference implementation using AES-NI and VAES indicates our version using VAES. Improv% shows the performance improvement of VAES over AES-NI. Lower run-times are better.

		Sub-Operation						
Type	Impl	Convolution	Truncation	ReLU	MatrixMultiplication	BatchNormalization	MaxPool	Total
Ring32-OT	Ref [82]	96.5	30.7	9.6	94.0	15.6	3.7	126.8
	VAES	97.0	21.0	6.8	94.5	13.5	2.5	119.1
	Improv%	**−0.5%**	**46.5%**	**40.4%**	**−0.5%**	**15.9%**	**47.1%**	**6.5%**

5.4 EMP-AGMPC

For EMP-AGMPC [90, 92] (cf. § 4.3.3), we ran SHA256 with three parties on localhost with binaries compiled with Clang. The runs were performed 11 times and then averaged. After the initial measurements, we decided to benchmark with batching applied and while using only a VAES-enabled library implementation of AES-ECB, the PRG, and the OT functionalities. The resulting performance numbers are shown in Table 6. In this table, the computation backend indicates the implementation strategy used, with the numbers in parenthesis being the performance improvements over the previous row.

Here, VAES allow to improve performance by up to 28%. The most substantial performance improvement is in the function-independent pre-processing phase. During that phase, the code uses additional garbling and evaluation techniques to prepare for the following phases based on the number of gates of the MPC function to be computed.

Discussion. The AGMPC performance data (in Table 6) shows substantial performance differences. The performance increase from VAES in the online phase stems from the OT used with the extra batching moving values out of registers again due to the gap between successive accesses. The most notable improvement is the 25% performance increase through batching in the function-independent preprocessing phase combined with VAES. This is because the garbling operations used in that phase benefit sufficiently from the batching, and there are not too many XORs sparsing out the AND gates and their memory.

5.5 CrypTFlow2

As CrypTFlow2 [82] (cf. § 4.3.4) uses EMP-OT internally, it is a natural target to investigate how the internal improvements benefit the overall performance of a more end-to-end application. As benchmarks we run inference for the SqueezeCIFAR, ResNet50, DenseNet121, and SqueezeNetImgNet networks. Each of these networks has its dedicated driver executable as usual for this application, was compiled using GCC and run via localhost with both parties on the same machine, to focus on the computational. The default settings used did utilize multiple load-intensive threads for both the client and the server, but had no noticeable impact on performance consistency.

A summary of the results using the geometric mean is given in Table 7 and the details are shown in Table 9 in Appendix B. Times below 1 second were omitted from the table.

Discussion. Table 7 shows that the VAES-based speed-up for the OT-based Ring32 implementation is 6.5% in total. The non-linear layers have particularly contributed to this improvement, with both the ReLU and MaxPool layers improving by over 40%. In particular, we observe no performance changes for the linear convolution and matrix multiplication steps for the Ring32 implementation. This is because these are primarily bound by the speed of the operating system interaction. We can also conceive that the performance improvement for the Ring32 implementation does stem from the relatively short focus on VAES during the operations.

6 CONCLUSION AND FUTURE WORK

In this work, we have shown how AES-NI and VAES can be used to speed up MPC protocols and applications, in particular for the case where operations are not known a priori.

Summary. We started with discussing how dynamic batching and its extensions and optimizations use deferred execution to provide better batches of AES calls to the hardware units. Next, we have discussed how more explicit measures in the code like SIMD gates and layering find batches of tasks with more invasive code modifications. Furthermore, we have discussed how to compute the batched calls using abstract pre- and post-processing and platform-specific AES computation in our memory-oriented computation strategy. Our alternative register-oriented strategy accepted code duplication for a low-level register value oriented code description that the compiler and the processor can execute well more easily. Following that, we applied these techniques to ABY [13, 25, 97], EMP-OT [90], EMP-AGMPC [90, 92], and Microsoft CrypTFlow2 [82]. For ABY we implemented additional garbled circuit variants [40, 41, 97] for comparison. We then evaluated the performance impact of the use of VAES and batching techniques. In ABY, these batching techniques have significantly increased performance without changing the hardware requirements. The use of VAES has yielded further significant performance improvements in ABY, EMP-OT, Microsoft CrypTFlow2, and some parts of EMP-AGMPC.

Future Work. Our research can be extended in multiple directions.

Improved Modelization. The techniques presented in § 4.1 and § 4.2 could be further improved. A more theoretical modelization and a more detailed analysis of the interaction with cache effects could yield valuable insights for future implementations.

Merging Register- and Memory-oriented Computation. Our computation techniques from § 4.2 require to make a manual choice between low code duplication, high performance, and clarity to the compiler. Further research could find techniques to automatically achieve low code duplication, high performance and clarity. For this, techniques from programming language and compiler research might be useful.

Further Applications in MPC. VAES and the other AVX512 extensions can be used to improve performance in further applications in MPC such as the most recent garbling schemes [10, 43, 85] that reduce communication (which is the main bottleneck in MPC) at the cost of more computation.

AVAILABILITY

The open source code of our changed VAES implementations is freely available under the permissive Apache license at https://encrypto.de/code/VASA.

ACKNOWLEDGMENTS

We sincerely thank Nir Drucker as well as Shay Gueron for contacting us with very helpful comments and pointers to the history of VAES which helped us to substantially improve our paper. Shay Gueron was the inventor of the concept, perceived usages and motivation, architecture, and microarchitectural implementation for vectorized AES in Intel processors when he was with Intel.

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) — SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE.

REFERENCES

[1] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. 2019. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *CCS*.

[2] Kahraman Akdemir, Martin Dixon, Wajdi Feghali, Patrick Fay, Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, and Ronen Zohar. 2010. Breakthrough AES performance with intel AES new instructions. *White paper* (2010). https://www.intel.ua/content/dam/www/public/us/en/documents/white-papers/aes-breakthrough-performance-paper.pdf.

[3] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *EUROCRYPT*.

[4] MPC Alliance. 2020. *MPC Alliance*. https://www.mpcalliance.org/

[5] AMD. 2021. *AMD64 Architecture Programmer's Manual: Volumes 1-5*. Advanced Micro Devices, Inc. https://www.amd.com/system/files/TechDocs/40332.pdf.

[6] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, and Carlos Aguilar Melchor. 2017. BIKE: Bit Flipping Key Encapsulation. (2017).

[7] Arm. 2021. *Arm Architecture Reference Manual*. Arm Limited. https://documentation-service.arm.com/static/60119835773bb020e3de6fee?token=.

[8] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*.

[9] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2015. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *EUROCRYPT*.

[10] Tomer Ashur, Efrat Cohen, Carmit Hazay, and Avishay Yanai. 2021. A New Framework for Garbled Circuits. Cryptology ePrint Archive. https://eprint.iacr.org/2021/739.

[11] Mauro Barni, Massimo Bernaschi, Riccardo Lazzeretti, Tommaso Pignata, and Alessandro Sabellico. 2014. Parallel Implementation of GC-Based MPC Protocols in the Semi-Honest Setting. In *Data Privacy Management and Autonomous Spontaneous Security*.

[12] D. Beaver, S. Micali, and P. Rogaway. 1990. The Round Complexity of Secure Protocols. In *STOC*.

[13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient Garbling from a Fixed-key Blockcipher. In *S&P*.

[14] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. 2016. Optimizing Semi-honest Secure Multiparty Computation for the Internet. In *CCS*.

[15] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: A Mixed-protocol Machine Learning Framework for Private Inference. In *ARES*.

[16] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-preserving Machine Learning. In *CCS*.

[17] Joan Boyar, Philip Matthews, and René Peralta. 2013. Logic Minimization Techniques with Applications to Cryptology. *Journal of Cryptology* 26 (2013).

[18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *CCS*.

[19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. In *CRYPTO*.

[20] Niklas Buescher and Stefan Katzenbeisser. 2015. Faster Secure Computation through Automatic Parallelization. In *USENIX Security*.

[21] Tom Caputi. 2016. ZFS-Native Encryption. OpenZFS Developer Summit.

[22] Sergiu Carpov, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. 2021. GenoPPML – A Framework for Genomic Privacy-preserving Machine Learning. Cryptology ePrint Archive. https://eprint.iacr.org/2021/733.

[23] Tung Chou and Claudio Orlandi. 2015. The Simplest Protocol for Oblivious Transfer. In *LATINCRYPT*.

[24] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. 2015. Automated Synthesis of Optimized Circuits for Secure Computation. In *CCS*.

[25] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY–A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[26] Jintai Ding and Dieter Schmidt. 2005. Rainbow, a New Multivariable Polynomial Signature Scheme. In *ACNS*.

[27] Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. 2019. Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC. In *EUROCRYPT*.

[28] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. 2018. Rasta: A Cipher with Low ANDdepth and few ANDs per Bit. In *CRYPTO*.

[29] Nir Drucker and Shay Gueron. 2019. CTR DRBG with Vector AES NI. Code: https://github.com/aws-samples/ctr-drbg-with-vector-aes-ni.

[30] Nir Drucker and Shay Gueron. 2019. Generating a Random String with a Fixed Weight. In *International Symposium on Cyber Security Cryptography and Machine Learning (CSCML)*. Springer.

[31] Nir Drucker and Shay Gueron. 2021. Speed Up Over the Rainbow. In *International Conference on Information Technology-New Generations (ITNG)*. Springer. Online: https://ia.cr/2020/408.

[32] Nir Drucker, Shay Gueron, and Dusan Kostic. 2020. QC-MDPC Decoders with Several Shades of Gray. In *PQCrypto*.

[33] Nir Drucker, Shay Gueron, and Vlad Krasnov. 2019. Making AES Great Again: The Forthcoming Vectorized AES Instruction. In *16. International Conference on Information Technology-New Generations (ITNG)*. Springer. Online: https://ia.cr/2018/392.

[34] Morris Dworkin. 2010. *Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices*. National Institute for Standards and Technology. https://csrc.nist.gov/publications/detail/sp/800-38e/final

[35] Intel. 2018. Intel architecture instruction set extensions programming reference. https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf.

[36] J.O. Eklundh. 1972. A Fast Computer Method for Matrix Transposing. In *IEEE Transactions on Computers*.

[37] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad Reza Sadeghi, Thomas Schneider, Hossein Yalame, and Shaza Zeitouni. 2021. SAFELearn: Secure Aggregation for private FEderated Learning. In *DLS*.

[38] Agner Fog. 2021. Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs. https://www.agner.org/optimize/instruction_tables.pdf.

[39] Thore Graepel, Kristin Lauter, and Michael Naehrig. 2013. ML Confidential: Machine Learning on Encrypted Data. In *ICISC*.

[40] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. 2015. Fast Garbling of Circuits Under Standard Assumptions. In *CCS*.

[41] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. 2020. Better Concrete Security for Half-Gates Garbling (in the Multi-instance Setting). In *CRYPTO*.

[42] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. 2020. Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers. In *IEEE S&P*.

[43] David Heath and Vladimir Kolesnikov. 2020. Stacked Garbling. In *CRYPTO*.

[44] David Heath and Vladimir Kolesnikov. 2020. Stacked Garbling for Disjunctive Zero-Knowledge Proofs. In *EUROCRYPT*.

[45] Aditya Hegde, Helen Möllering, Thomas Schneider, and Hossein Yalame. 2021. SoK: Efficient Privacy-preserving Clustering. *PETS* (2021).

[46] Tim Heldmann, Thomas Schneider, Oleksandr Tkachenko, Christian Weinert, and Hossein Yalame. 2021. LLVM-Based Circuit Compilation for Practical Secure Computation. In *ACNS*.

[47] Wilko Henecka and Thomas Schneider. 2013. Faster Secure Two-Party Computation with Less Memory. In *ASIACCS*.

[48] Wilko Henecka and Thomas Schneider. 2013. Faster Secure Two-Party Computation with Less Memory. In *CCS*.

[49] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS*.

[50] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security*.

[51] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. 2014. Amortizing Garbled Circuits. In *CRYPTO*.

[52] Siam Hussain, Baiyu Li, Farinaz Koushanfar, and Rosario Cammarota. 2020. TinyGarble2: Smart, Efficient, and Scalable Yao's Garble Circuit. In *ACM Workshop on Privacy-Preserving Machine Learning in Practice (PPMLP)*.

[53] Siam U. Hussain and Farinaz Koushanfar. 2019. FASE: FPGA Acceleration of Secure Function Evaluation. In *FCCM*.

[54] Siam U. Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2018. MAXelerator: FPGA Accelerator for Privacy Preserving Multiply-Accumulate (MAC) on Cloud Servers. In *DAC*.

[55] Nathaniel Husted, Steven Myers, Abhi Shelat, and Paul Grubbs. 2013. GPU and CPU Parallelization of Honest-but-Curious Secure Two-Party Computation. In *ACSAC*.

[56] Intel. 2021. Intel 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf.

[57] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO*.

[58] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2010. Embedded SFE: Offloading Server and Network Using Hardware Tokens. In *FC*.

[59] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2010. Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs. In *CHES*.

[60] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. 2013. Zero-Knowledge Using Garbled Circuits: How to Prove Non-Algebraic Statements Efficiently. In *CCS*.

[61] Daniel Kales, Léo Perrin, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. 2017. Improvements to the Linear Operations of LowMC: A Faster Picnic. Cryptology ePrint Archive. https://ia.cr/2017/1148.
[62] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile Private Contact Discovery at Scale. In *USENIX Security*.
[63] Emilia Käsper and Peter Schwabe. 2009. Faster and Timing-Attack Resistant AES-GCM. In *CHES*.
[64] Hannah Keller, Helen Möllering, Thomas Schneider, and Hossein Yalame. 2021. Balancing Quality and Efficiency in Private Clustering with Affinity Propagation. In *SECRYPT*.
[65] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. 2017. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *PETS* (2017).
[66] B. Knott, S. Venkataraman, A.Y. Hannun, S. Sengupta, M. Ibrahim, and L.J.P. van der Maaten. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *arXiv 2109.00984*.
[67] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*.
[68] Ben Kreuter, Benjamin Mood, Abhi Shelat, and Kevin Butler. 2013. PCF: A Portable Circuit Format for Scalable Two-party Secure Computation. In *USENIX Security*.
[69] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow: Secure TensorFlow Inference. In *IEEE S&P*.
[70] Yehuda Lindell and Benny Pinkas. 2008. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology* (2008).
[71] Chris M. Lonvick and Tatu Ylonen. 2006. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253. https://rfc-editor.org/rfc/rfc4253.txt
[72] Microsoft and Contributors. 2019. *Transparent Data Encryption (TDE)*. https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-ver15
[73] Payman Mohassel, Mike Rosulek, and Ni Trieu. 2020. Practical Privacy-preserving K-means Clustering. *PETS* (2020).
[74] Moni Naor and Benny Pinkas. 2001. Efficient Oblivious Transfer Protocols. In *Society for Industrial and Applied Mathematics*.
[75] Moni Naor, Benny Pinkas, and Reuban Sumner. 1999. Privacy Preserving Auctions and Mechanism Design. In *ACM conference on Electronic commerce*.
[76] Jesper Buus Nielsen and Claudio Orlandi. 2009. LEGO for Two-Party Secure Computation. In *TCC*.
[77] Jesper Buus Nielsen and Claudio Orlandi. 2016. Cross and Clean: Amortized Garbled Circuits with Constant Overhead. In *TCC*.
[78] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*.
[79] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. SynCirc: Efficient Synthesis of Depth-OptimizedCircuits for Secure Computation. In *HOST*.
[80] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*.
[81] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. 2009. Secure Two-Party Computation Is Practical. In *ASIACRYPT*.
[82] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*.
[83] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://rfc-editor.org/rfc/rfc8446.txt
[84] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-Based Oblivious Deep Neural Network Inference. In *USENIX Security*.
[85] Mike Rosulek and Lawrence Roy. 2021. Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits. In *CRYPTO*.
[86] Palash Sarkar. 2008. A General Mixing Strategy for the ECB-Mix-ECB Mode of Operation. In *Information Processing Letters*. https://www.sciencedirect.com/science/article/pii/S0020019008002652
[87] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. 2015. TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits. In *IEEE S&P*.
[88] Ebrahim M. Songhori, Shaza Zeitouni, Ghada Dessouky, Thomas Schneider, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2016. GarbledCPU: A MIPS Processor for Secure Computation in Hardware. In *DAC*.
[89] The OpenSSL Project. 2003. OpenSSL: The Open Source toolkit for SSL/TLS. www.openssl.org.
[90] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty Computation Toolkit. https://github.com/emp-toolkit.
[91] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In *CCS*.
[92] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. In *CCS*.
[93] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast Extension for Correlated OT with Small Communication. In *CCS*.
[94] Andrew C. Yao. 1982. Protocols for secure computations. In *FOCS*.
[95] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets. In *FOCS*.
[96] Samee Zahur and David Evans. 2015. Obliv-C: A Language for Extensible Data-Oblivious Computation. Cryptology ePrint Archive. https://eprint.iacr.org/2015/1153.
[97] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*.
[98] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. 2021. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *USENIX Security*.

A EXAMPLE CODE FOR OUR IMPLEMENTATION

We present example code for the register-oriented batch computation strategy from § 4.2.1 in Listing 1 and for the memory-oriented one from § 4.2.2 in Listing 2.

Listing 1: Register-oriented implementation of HalfGates's evaluation [13, 97] using fixed-key VAES and AVX512F.

```
void halfgates_eval_vaes(uint8_t* expanded_key, Gate* gates, uint8_t* tables, size_t num_gates, uint64_t tableCounter) {
constexpr size_t width = 8;
constexpr size_t num_regs = (width + 3) / 4;
constexpr size_t used_lanes = std::min(size_t(4), width);
constexpr size_t offset = used_lanes * 2;
// do the leftovers with another call with width==1
__m512i ONE = _mm512_set_epi64(0, 1, 0, 1, 0, 1, 0, 1);
__m512i FULL_OFFSET = _mm512_set_epi64(0, offset, 0, offset, 0, offset, 0, offset);
__m512i counter = _mm512_set_epi64(0, (tableCounter + 3) * 2, 0, (tableCounter + 2) * 2, 0, (tableCounter + 1) * 2, 0, (tableCounter + 0) * 2);
__m512i keys[11];
//omitted loading round keys with _mm512_broadcasti32x4_epi32
for(size_t p = 0; p < num_gates; p += width) {
  __m512i leftData[num_regs], rightData[num_regs], leftKeys[num_regs], rightKeys[num_regs], finalMask[num_regs];
  __m128i* targetGateKey[width];
  for(size_t w = 0; w < num_regs; ++w) {
    for(size_t l = 0; l < used_lanes; ++l) {
      // Gate fetching
      Gate* currentGate = gates[p + w * used_lanes + l];
      // match the left* operations with right* ones
      Gate* leftParentId = currentGate->leftParent;
      uint8_t* leftParentKey = leftParent->evalKey;
      __m128i leftParentKeyLocal = _mm_loadu_si128((__m128i*)(leftParentKey));
      leftKeys[w] = _mm512_inserti32x4(leftKeys[w], leftParentKeyLocal, l);
      targetGateKey[used_lanes * w + l] = currentGate->evalKey;
      uint8_t lpbit = leftParentKey[15] & 0x01;
      uint8_t lpbit11 = (lpbit << 1) | lpbit;
      uint8_t rpbit = rightParentKey[15] & 0x01;
      uint8_t rpbit11 = (rpbit << 1) | rpbit;
      __m128i finalMaskLocal = _mm_maskz_loadu_epi64(lpbit11, (__m128i*)tables);
      tables += 16;
      __m128i rightTable = _mm_loadu_si128((__m128i*)(tables));
      const __m128i rightMaskUpdate = _mm_xor_si128(rightTable, leftParentKeyLocal);
      finalMaskLocal = _mm_mask_xor_epi64(finalMaskLocal, rpbit11, finalMaskLocal, rightMaskUpdate);
      tables += 16;
      finalMask[w] = _mm512_inserti32x4(finalMask[w], finalMaskLocal, l);
    }
    // Vector processing
    leftData[w] = counter;
    rightData[w] = _mm512_add_epi64(counter, ONE);
```

```
    counter = _mm512_add_epi64(counter, FULL_OFFSET);
    // match the left* operations with right* ones
    // use a combiner, e.g., left shift + XOR
    leftData[w] = mix_keys(leftKeys[w],leftData[w]);
    leftKeys[w] = leftData[w];
    leftData[w]=_mm512_xor_si512(leftData[w],keys[0]);
  }
  for(size_t r = 1; r < 10; ++r) {
    for(size_t w = 0; w < num_regs; ++w) {
      leftData[w] = _mm512_aesenc_epi128(leftData[w],
          keys[r]);
      rightData[w] = _mm512_aesenc_epi128(rightData[w],
          keys[r]);
    }
  }
  for(size_t w = 0; w < num_regs; ++w) {
      // match the left* operations with right* ones
    leftData[w] = _mm512_aesenclast_epi128(leftData[w],
        keys[10]);
    leftData[w] = _mm512_xor_si512(leftData[w], leftKeys[
        w]);
    // Vector HalfGates post-processing
    leftData[w] = _mm512_xor_si512(leftData[w], rightData
        [w]);
    leftData[w] = _mm512_xor_si512(leftData[w], finalMask
        [w]);
    for(size_t l = 0; l < used_lanes; ++l) {
      // scattering
      __m128i extracted = _mm512_extracti32x4_epi32(
          leftData[w], l);
      _mm_storeu_si128((__m128i*)(targetGateKey[
          used_lanes * w + l]), extracted);
        }
      }
  }
}
```

Listing 2: Memory-oriented implementation of the batched AND evaluation for actively secure garbled circuits [90, 91].

```
// ONLINE_BATCH_SIZE is an upper bound
void EvaluateANDGates(uint8_t* mask_input,int indices[
    ONLINE_BATCH_SIZE], size_t num_gates, int& ands) {
int mask_indices[ONLINE_BATCH_SIZE];
block lefts[ONLINE_BATCH_SIZE],rights[ONLINE_BATCH_SIZE];
block H[ONLINE_BATCH_SIZE][2];
for (size_t ii = 0; ii < num_gates; ++ii) {
  // preprocessing
  int i = indices[ii];
  int index = 2 * mask_input[cf->gates[4 * i]] +
      mask_input[cf->gates[4 * i + 1]];
  mask_indices[ii] = index;
  lefts[ii] = labels[exec_times][cf->gates[4 * i]];
  rights[ii] = labels[exec_times][cf->gates[4 * i + 1]];
}
// AES processing
Hash(H, lefts, rights, indices, mask_indices, num_gates);
for (size_t ii = 0; ii < num_gates; ++ii) {
  // postprocessing
  int i = indices[ii];
  int index = 2 * mask_input[cf->gates[4 * i]] +
      mask_input[cf->gates[4 * i + 1]];
  GT[exec_times][ands][index][0] = GT[exec_times][ands][
      index][0] ^ H[ii][0];
  GT[exec_times][ands][index][1] = GT[exec_times][ands][
      index][1] ^ H[ii][1];
  block ttt = GTK[exec_times][ands][index] ^ fpre->Delta;
  ttt = ttt & MASK;
  GTK[exec_times][ands][index] = GTK[exec_times][ands][
      index] & MASK;
  GT[exec_times][ands][index][0] = GT[exec_times][ands][
      index][0] & MASK;
  if (cmpBlock(&GT[exec_times][ands][index][0], &GTK[
      exec_times][ands][index], 1))
    mask_input[cf->gates[4 * i + 2]] = false;
  else if (cmpBlock(&GT[exec_times][ands][index][0], &ttt
      , 1))
    mask_input[cf->gates[4 * i + 2]] = true;
  else
    cout << ands << "no match GT!" << endl;
  mask_input[cf->gates[4 * i + 2]] = logic_xor(mask_input
      [cf->gates[4 * i + 2]], getLSB(GTM[exec_times][
      ands][index]));
  labels[exec_times][cf->gates[4 * i + 2]] = GT[
      exec_times][ands][index][1] ^ GTM[exec_times][ands
      ][index];
  ands++;
}
```

B DETAILED MEASUREMENTS

We present the detailed performance measurements for ABY (cf. § 5.2) in Table 8 from which the summary in Table 4 was computed. Additionally, we present the detailed performance measurements for CrypTFlow2 (cf. § 5.5) in Table 9 from which the summary in Table 7 was computed.

Table 8: Run-times in milliseconds of ABY [25] for the evaluation of AES, SHA-1, SCS-PSI, and Phasing-PSI with the detailed parameters as described in § 5.2. "Ref" indicates the reference ABY implementation, AES-NI indicates the batched one using AES-NI and VAES the one using VAES. Improv% shows the performance improvement of VAES over AES-NI based PRGs and ECB implementations. Garbling scheme names are as introduced in § 4.3. Lower run-times are better.

			Garbling Scheme			
Operation	Circuit		PRP	MI	CIRC	PRF
Garbling	AES	Ref [25]	47.3	—	—	—
		AES-NI	20.5	27.6	31.3	98.5
		VAES	16.6	19.0	20.8	66.2
		Improv%	**23.4%**	**45.4%**	**50.4%**	**48.6%**
	SHA1	Ref [25]	236.7	—	—	—
		AES-NI	95.4	118.6	145.7	576.2
		VAES	69.8	79.3	87.9	378.3
		Improv%	**36.6%**	**49.6%**	**65.8%**	**52.3%**
	SCS-PSI	Ref [25]	153.0	—	—	—
		AES-NI	75.3	98.9	112.3	288.1
		VAES	63.9	74.2	79.7	192.7
		Improv%	**17.8%**	**33.3%**	**40.9%**	**49.5%**
	PSI-Phasing	Ref [25]	87.3	—	—	—
		AES-NI	33.4	42.6	52.7	92.9
		VAES	25.3	26.1	30.7	59.6
		Improv%	**31.8%**	**63.2%**	**71.6%**	**55.8%**
Evaluation	AES	Ref [25]	23.0	—	—	—
		AES-NI	12.5	23.1	15.6	47.9
		VAES	8.6	11.7	10.2	25.1
		Improv%	**45.0%**	**97.1%**	**53.4%**	**91.1%**
	SHA1	Ref [25]	108.8	—	—	—
		AES-NI	56.0	139.7	80.9	261.7
		VAES	38.2	51.5	52.3	151.3
		Improv%	**46.5%**	**171.5%**	**54.7%**	**73.0%**
	SCS-PSI	Ref [25]	76.2	—	—	—
		AES-NI	41.9	92.5	57.3	135.7
		VAES	33.1	43.5	41.9	78.0
		Improv%	**26.5%**	**112.7%**	**36.8%**	**74.1%**
	PSI-Phasing	Ref [25]	53.2	—	—	—
		AES-NI	31.9	42.7	40.1	66.9
		VAES	25.0	28.4	31.1	41.0
		Improv%	**27.5%**	**50.5%**	**28.7%**	**62.9%**

Table 9: Run-times in seconds for CrypTFlow2 [82] (§ 5.5) inference using the SqueezeNetImgNet (SqzImg), SqueezeNetCIFAR (SqzCIFAR), ResNet50, and DenseNet121 networks. Ring32-OT denotes the 32-bit ring-based implementation using OT. Ref indicates the reference implementation (using AES-NI) and VAES indicates the version using VAES. Improv% shows the performance improvement of VAES over AES-NI. Lower run-times are better.

			Sub-Operation						
Type	Network	Impl	Convolution	Truncation	ReLU	MatMul	BatchNormalization	MaxPool	Total
Ring32-OT	SqzImg	Ref [82]	28.1	—	4.0	27.2	—	4.7	39.0
		VAES	28.0	—	2.9	26.9	—	3.1	35.6
		Improv%	**0.6%**	**—**	**36.7%**	**0.9%**	**—**	**53.0%**	**9.6%**
	SqzCIFAR	Ref [82]	28.0	—	4.0	27.0	—	4.4	38.5
		VAES	28.2	—	2.9	27.2	—	3.2	35.8
		Improv%	**−0.8%**	**—**	**38.9%**	**−0.9%**	**—**	**37.1%**	**7.5%**
	ResNet	Ref [82]	439.7	30.8	18.7	436.1	12.7	3.2	513.3
		VAES	448.2	20.9	12.7	444.5	11.2	2.1	503.1
		Improv%	**−1.9%**	**47.5%**	**46.5%**	**−1.9%**	**13.2%**	**52.1%**	**2.0%**
	DenseNet	Ref [82]	250.1	30.6	28.6	244.3	19.2	2.7	335.6
		VAES	250.0	21.1	20.5	243.9	16.2	1.9	313.8
		Improv%	**0.1%**	**45.5%**	**39.5%**	**0.2%**	**18.6%**	**46.6%**	**6.9%**

E ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation (USENIX Security'21)

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *USENIX Security Symposium (USENIX Security)*. Online: https://ia.cr/2020/1225. USENIX Association, 2021, pp. 2165–2182. CORE Rank **A***. Appendix E.

https://encrypto.de/papers/PSSY21.pdf

ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation

Arpita Patra
Indian Institute of Science

Thomas Schneider
TU Darmstadt

Ajith Suresh
Indian Institute of Science

Hossein Yalame
TU Darmstadt

Abstract

Secure Multi-party Computation (MPC) allows a set of mutually distrusting parties to jointly evaluate a function on their private inputs while maintaining input privacy. In this work, we improve semi-honest secure two-party computation (2PC) over rings, with a focus on the efficiency of the online phase.

We propose an efficient mixed-protocol framework, outperforming the state-of-the-art 2PC framework of ABY. Moreover, we extend our techniques to multi-input multiplication gates without inflating the online communication, i.e., it remains independent of the fan-in. Along the way, we construct efficient protocols for several primitives such as scalar product, matrix multiplication, comparison, maxpool, and equality testing. The online communication of our scalar product is two ring elements *irrespective* of the vector dimension, which is a feature achieved for the first time in the 2PC literature.

The practicality of our new set of protocols is showcased with four applications: i) AES S-box, ii) Circuit-based Private Set Intersection, iii) Biometric Matching, and iv) Privacy-preserving Machine Learning (PPML). Most notably, for PPML, we implement and benchmark training and inference of Logistic Regression and Neural Networks over LAN and WAN networks. For training, we improve online runtime (both for LAN and WAN) over SecureML (Mohassel et al., IEEE S&P'17) in the range 1.5×–6.1×, while for inference, the improvements are in the range of 2.5×–754.3×.

1 Introduction

Secure Multi-Party Computation (MPC) [13, 45, 98] allows n mutually distrusting parties to jointly compute a function on their private inputs. The computation guarantees i) privacy– no set of t corrupt parties can learn more information than the output, and ii) correctness– corrupt parties cannot force others to accept a wrong output. Due to its immense potential, MPC can be used for solving real-life applications such as privacy-preserving auctions [77] and remote diagnostics [23], secure genome analysis [14, 96], and recently in the domain of privacy-preserving machine learning (PPML) [16, 27, 30, 31, 52, 57, 67, 75, 84, 91, 101].

MPC protocols can be broadly classified into low-latency [28, 46, 74, 81, 97] and high-throughput [4, 27, 30, 31, 67, 84] categories. The low-latency protocols are built using Yao's garbled circuits (GC) [10, 66, 97–99] and result in constant-round solutions. Secret-sharing (SS) based solutions have been used for high-throughput protocols, but require a number of communication rounds linear in the multiplicative depth of the circuit. However, less communication than GC-based protocols facilitates several instances of SS-based protocols to be executed in parallel, leading to high throughput. The characteristics of the categories mentioned above put forth the need for a mixed-protocol framework [31, 39, 73, 75, 92], where the protocol is split into blocks and each block is executed in one of the following three worlds: i) Arithmetic, ii) Boolean, and iii) Yao. While the arithmetic world performs operations on ℓ-bit rings (or fields), both boolean and Yao world perform operations on bits. Also, arithmetic and boolean worlds operate using an SS-based approach while the Yao world uses a GC-based approach.

To achieve practical runtimes, several works [12, 26, 27, 30, 31, 38, 61, 67, 91] considered the paradigm of having an *input-independent* setup phase where the parties generate a lot of correlated randomness (e.g., Beaver multiplication triples [8]) which are then used in the *input-dependent* online phase to enable a very fast computation on the parties' inputs. Moreover, the benchmarking results of [94] and the works of [17, 34, 35, 37, 39] have showcased the efficiency improvements of protocols compared to rings over their field counterparts. The 32/64-bit computations done in standard CPUs, emulating ring operations, allow for very simple and efficient implementations.

In this work, we focus on the specific problem of secure two-party computation (2PC) [38, 39] with mixed protocols over rings. Our aim is to minimize the online communication and rounds keeping high throughput as our end-goal.

1.1 Our Contributions

We propose an efficient mixed-protocol framework for secure 2PC over an ℓ-bit ring. Our protocols are secure against a *semi-honest* adversary and use an *input-independent* setup. We build several building blocks with the focus on online efficiency. Our contributions can be summed up as follows:

2PC (§3). We propose an efficient 2PC protocol over ℓ-bit rings, requiring a communication of just 2 ring elements per multiplication in the online phase. Our construction relies on Beaver's circuit randomization technique [8] (§3.1.1), but uses a different perspective of the technique. Moreover, our protocol helps in realising efficient primitives as will be shown in §5. We believe that our new perspective can bring several further optimizations where Beaver's randomization technique is currently being used.

Protocol	Ref.	Setup	Online	
		Comm [bits]	Comm [bits]	Rounds
MULT $z = ab$	[39]	$2\ell(\kappa+\ell)$	4ℓ	**1**
	[12]	$2\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
	[78]	$2\ell(\kappa+\ell)$	4ℓ	**1**
	ABY2.0	$2\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
MULT3 $z = abc$	[39]	$4\ell(\kappa+\ell)$	8ℓ	2
	[12]	$4\ell(\kappa+\ell)$	4ℓ	2
	[78]	$8\ell(\kappa+\ell)$	6ℓ	**1**
	ABY2.0	$8\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**
MULT4 $z = abcd$	[39]	$6\ell(\kappa+\ell)$	12ℓ	2
	[12]	$6\ell(\kappa+\ell)$	6ℓ	2
	[78]	$22\ell(\kappa+\ell)$	8ℓ	**1**
	ABY2.0	$22\ell(\kappa+\ell)$	$\mathbf{2\ell}$	**1**

Table 1: Comparison of ABY2.0 and existing works for 2PC protocols. Best values for the online phase are marked in bold.

Tab. 1 shows our improvement over previous works. For 2-input multiplication, we achieve the same complexity as [12], but using a completely different approach. Moreover, for an N-input multiplication gate, our solution has a *constant* cost of 2 ring elements and one round of interaction. This is a massive improvement over [78], where they require communication of $2N$ ring elements. Round complexity wise, the naive method of multiplying N elements by taking two at a time requires $\log_2(N)$ online rounds and overall communication of $4(N-1)$ ring elements for [39] and $2(N-1)$ for [12].

Mixed Protocol Conversions (§4). The mixed world conversions, that enable easy transition between Arithmetic (**A**), Boolean (**B**) and Yao (**Y**) sharing, are now celebrated in the literature [3, 26, 57, 75, 91] due to their potential in building practically-efficient protocols. We propose a new set of conversions that outperform the state-of-the-art conversions of ABY [39] in the online phase. Our solution reduces the number of online rounds of ABY from 2 to 1 for most of the conversions. We achieve this because, in contrast to ABY, we forgo OTs in the online phase of our conversions.

Tab. 2 provides the concrete costs for the mixed protocol conversions. The conversion from sharing type S to sharing type D is denoted as $S2D$, where $S, D \in \{\mathbf{A}, \mathbf{B}, \mathbf{Y}\}$. For the setup phase, we use correlated OTs (cOT) [5] which incur a communication of $\ell + \kappa$ bits per cOT on ℓ-bit strings, where κ is the computational security parameter. It is evident from Tab. 2 that for all except the Y2B conversion, our conversions outperform ABYs' in the online phase.

Conv.	Ref.	Setup	Online	
		Comm [bits]	Comm [bits]	Rounds
Y2B	ABY [39]	0	**0**	**0**
	ABY2.0	ℓ	ℓ	1
B2Y	ABY [39]	$2\ell\kappa$	$\ell\kappa+\ell$	2
	ABY2.0	$2\ell\kappa$	$\boldsymbol{\ell\kappa}$	**1**
A2Y	ABY [39]	$4\ell\kappa$	$2\ell\kappa+\ell$	2
	ABY2.0	$4\ell\kappa$	$\boldsymbol{\ell\kappa}$	**1**
Y2A	ABY [39]	$2\ell\kappa$	$(\ell^2+3\ell)/2$	2
	ABY2.0	$3\ell\kappa+2\ell$	$\boldsymbol{\ell}$	**1**
A2B	ABY [39]	$4\ell\kappa$	$2\ell\kappa+\ell$	**2**
	ABY2.0	$4\ell\kappa+\ell$	$\boldsymbol{\ell\kappa+\ell}$	**2**
B2A	ABY [39]	$\ell\kappa$	$(\ell^2+\ell)/2$	2
	ABY2.0	$\ell\kappa+\ell^2$	$\mathbf{2\ell}$	**1**

Table 2: Comparison of ABY2.0 and ABY for the conversions. The values are reported for ℓ-bit values. Best values for the online phase are marked in bold.

Building Blocks (§5). We propose efficient constructions for widely-used building blocks that include Scalar Product, Depth-Optimized Circuits, Matrix Multiplication, Comparison, Non-linear Activation functions, and Maxpool. The highlights include:

– *Scalar Product (§5.1):* Our new protocol incurs an online communication that is *independent* of the vector dimension n. This feature is achieved for the first time in the 2PC literature. Concretely, we require communication of just 2 ring elements as opposed to 4n elements of [39]. Since scalar product forms an essential building block for most of the widely used ML algorithms [27, 30, 31, 56, 73, 75, 91] such as Linear Regression, Logistic Regression, and Clustering, our solution substantially improves the performance of their secure 2PC implementations by several orders of magnitude.

– *Matrix Multiplication (§5.2):* Matrix multiplication is the fundamental building block in most ML algorithms. For instance, the linear layer in a Neural Network (NN) as well as the convolution operation in a Convolutional Neural Network [95] can be viewed as an instance of matrix multiplication. We extend the 2PC multiplication protocol to support vector operations and provide an efficient matrix multiplication protocol.

– *Depth Optimized Circuits (§5.3):* The Parallel Prefix Adder (PPA) [7, 47] used in the recent PPML literature [73] incurs a multiplicative depth of $\log_2(\ell)$ since it uses two-input AND gates only. We propose round efficient PPA constructions using a combination of two, three, and four input AND gates.

For a 64-bit ring, our solution has 2× fewer rounds and also less online communication compared to the PPA used in [73].

– *Comparison (§5.4):* Our new protocol for checking less than relation improves the online communication of the comparison protocol of [78] by 6× and reduces the number of online rounds from 4 to 3.

– *Maximum of three elements (§5.7):* Our new protocol improves the online communication of [78] by 14× while reducing the online rounds from 5 to 4.

– *Equality Test (§5.10):* Our new protocol for checking the equality of two ℓ-bit values, improves the online rounds of [87] from $\log_2(\ell)$ to $\log_4(\ell)$.

Applications (§6). The practicality of our constructions are showcased in these four popular applications:

– *AES S-box (§6.2):* Using our protocol for 3-input multiplication, we obtain an S-box with an AND-depth of 3 instead of 4 before. This improves the online round complexity of AES by factor 1.33×.

– *Circuit-based PSI (§6.3):* Using our efficient equality testing protocol, we improve the online communication of the state-of-the-art circuit-based PSI [87] by 2.35× and the online round complexity by 1.3×.

– *Biometric Matching (§6.4):* We propose a round-optimized as well as a communication-optimized solution for computing the minimum Euclidean distance, which forms the core for biometric matching. For the round-optimized variant, we improve over ABY [39] by 2.2× in communication and 1.6× in rounds in the online phase. Similarly, for the communication-optimized variant, we improve over [78] by 20.8× in communication and 1.3× in rounds.

– *Privacy-Preserving Machine Learning (§6.5):* Here we implement the training and inference of Logistic Regression and Neural Networks in a LAN and a WAN setting and benchmarked over datasets with various feature sizes.

Algorithm	Ref.	LAN		WAN	
		TP (x10^4)	Improvem.	TP (x10^4)	Improvem.
Logistic Regression	[75]	1,344.4	**31.5×**	4.0	**9.9×**
	ABY2.0	**42,372.4**		**39.9**	
Neural Networks	[75]	43.0	**716.0**×	0.1	**710.7**×
	ABY2.0	**30,797.0**		**92.39**	

Table 3: Comparison of the online throughput (TP) of ABY2.0 and SecureML [75] for inference on the MNIST [70] dataset.

For training, we obtain online runtime improvements over SecureML [75] in the range 2.7×–6.1× for LAN and 1.5×–2.8× for WAN. For inference, we used *throughput* as one metric to capture the effect of runtime and communication utilization in a single shot. Our improvement for inference ranges from 7.9×–754.3× for LAN, while it ranges from 2.5×–753.2× for WAN. Tab. 3 provides the concrete details for inference over the MNIST [70] dataset.

1.2 Related Work

Here, we provide a concise summary of related work. More details on the preliminaries are given in §A.

Secret Sharing (SS). The works of [38,61] proposed efficient SS-based solutions for the dishonest majority setting over fields, which was then extended to the ring setting in [33]. The solution involves the generation of Beaver multiplication triples [8] in the setup phase and evaluation of the circuit (multiplication gates) in the online phase using the generated triples. For the 2PC case, the aforementioned approach requires two public reconstructions among the parties per multiplication gate in the online phase. In contrast, we require only one public reconstruction among the parties. Later, works like [59, 60, 79] focused on improving the setup cost using techniques like Oblivious Transfer (OT) and Homomorphic Encryption (HE). [12] improved the number of public reconstructions required in the online phase from two to one using a function-dependent preprocessing, but requires additional communication of four ring elements in the preprocessing phase.

Multi-Input Multiplication. In the boolean setting, [40] extended two-input AND gates to the general N-input case using lookup tables. Recently, [78] extended the multiplication from two-input to arbitrary input using Beaver triple extension with a focus on minimizing the online rounds. However, the online communication of [78] scale with the fan-in of the multiplication gates as opposed to ours, where we achieve an online communication of 2 ring elements.

Mixed-Protocol Conversions. Mixed 2PC protocols that combine GC-based and SS-based approaches benefit from their respective advantages and were used in many privacy-preserving applications such as face recognition [49], fingerprint recognition [24], biometric matching [39], and machine learning [57, 73, 75, 91]. The first mixed-protocol framework for MPC was TASTY [49, 65], which combined garbled circuits with homomorphic encryption. ABY [39] then proposed an efficient framework in the semi-honest model combining state-of-the-art 2PC approaches based on Arithmetic sharing, Boolean sharing, and GCs. The work of [92] shows conversions between MPC based on arithmetic secret sharing and garbled circuits with malicious security. Later, the ABY framework was extended to the three and four party honest-majority setting by [31, 73]. HyCC [26] provides a compiler to automatically partition a function (specified in ANSI C) into sub-functions such that each sub-function is evaluated with either Arithmetic sharing, Boolean sharing or Garbled Circuits (GC).

2 Preliminaries

Here, we describe our security model and the parameters and notations used. More details along with a brief overview of the state-of-the-art 2PC protocols are given in §A.

Semi-honest Security Model. In this work, we consider a semi-honest (aka passive) adversary [32, 53, 100], who is "honest-but-curious". The adversary is guaranteed to follow the protocol steps but will try to learn additional information from the messages that he has seen during the protocol execution. Though not the strongest model, this model forms the first step towards achieving protocols with stronger security guarantees [6, 29, 68, 71]. Also, the setting facilitates practically-efficient protocols with higher performance especially for PPML applications [30, 75, 91]. In practical scenarios where the computation is outsourced to a set of servers, the reputation of the servers forces them to behave semi-honestly. Moreover, in many application scenarios, semi-honest behaviour can be enforced by attestation using tools like Intel SGX or ARM TrustZone. We refer the reader to [44] for details on the model.

Parameters and Notation. In our framework, we have two parties $\mathcal{P} = \{P_0, P_1\}$ who are connected by a bidirectional synchronous channel (eg. instantiated via TLS over TCP/IP). Our protocols are designed to work over an ℓ-bit ring denoted by $\mathbb{Z}_{2^\ell}$. κ denotes the computational security parameter. In our implementation, we use $\ell = 64$ and $\kappa = 128$.

For two vectors $\vec{a}, \vec{b}$ of length n, the scalar dot product is denoted by $\vec{a} \odot \vec{b} = \sum_{j=1}^{\mathsf{n}} \mathsf{a}_j \mathsf{b}_j$. Here a_j and b_j denote the j^{th} elements of vectors $\vec{a}$ and $\vec{b}$ respectively. For a bit $\mathsf{u} \in \{0,1\}$, $\overline{\mathsf{u}}$ denotes the complement value $1 \oplus \mathsf{u}$. For two matrices $\mathbf{A}, \mathbf{B}$, matrix multiplication is denoted by $\mathbf{A} \circ \mathbf{B}$. Table 4 depicts notation that we use throughout the paper.

P_0, P_1	Parties performing secure computation
$\mathbb{Z}_{2^\ell}$	Ring of size ℓ bits; $\ell = 64$ in this work
κ	Symmetric security parameter; $\kappa = 128$ in this work
a_j	j-th element of vector $\vec{a}$
$\vec{a} \odot \vec{b}$	Scalar dot product between two vectors $\vec{a}$ and $\vec{b}$
$\mathbf{A} \circ \mathbf{B}$	Multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$
$[\mathsf{v}]_i$	$[\cdot]$-sharing of $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ held by P_i s.t. $\mathsf{v} = [\mathsf{v}]_0 + [\mathsf{v}]_1$
$\langle \mathsf{v} \rangle_i = ([\delta_\mathsf{v}]_i, \Delta_\mathsf{v})$	$\langle \cdot \rangle$-sharing of $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ held by P_i s.t. $\mathsf{v} = \Delta_\mathsf{v} - [\delta_\mathsf{v}]_1 - [\delta_\mathsf{v}]_0$
$t \in \{A, B, Y\}$	Type of sharing: **A**rithmetic, **B**oolean, or **Y**ao
$x^s = s2t(x^t)$	Sharing conversion from source s to target t
OT	Oblivious Transfer
HE	Homomorphic Encryption
cOT_ℓ^n	n instances of Correlated OT on ℓ-bit strings
MSB/LSB	Most / Least Significant Bit
FPA	Fixed-point Arithmetic
SED	Squared Euclidean Distance

Table 4: Notations used throughout this paper.

Our protocols are cast into an *input-independent* setup phase and an *input-dependent* online phase. To enable parties to non-interactively sample a random value, parties perform a one-time key-setup that establishes random keys among them for a pseudo-random function (PRF) which can be instantiated, for instance, using AES in counter mode. Towards this, each party P_i for $i \in \{0,1\}$ samples a random key $K_i \in_R \{0,1\}^\kappa$ and sends it to the other party. The shared key is now defined as $K = K_0 + K_1$.

For applications such as machine learning where the inputs are decimal numbers, we use the Fixed-Point Arithmetic (FPA) representation [27, 30, 31, 73, 75] to embed the value in the underlying ring. Decimal value is treated as an ℓ-bit integer in signed 2's complement representation. The most significant bit (MSB) represents the sign while the least significant x bits represent the fractional part. For our implementation, we use $\ell = 64$ and $x = 13$.

3 2PC in Arithmetic, Boolean & Yao World

The contribution of this section is our new 2PC over ring $\mathbb{Z}_{2^\ell}$. This construction gives us a new 2PC in the arithmetic world and in the Boolean world. The latter is easily derived by having $\ell = 1$. The 2PC in Yao's world is borrowed from ABY [39]. Below, we start with our new 2PC over $\mathbb{Z}_{2^\ell}$. We describe the secret-sharing semantics, the sharing and reconstruction protocols, and the multiplication protocols (both for setup and online phase) with various fan-ins. Our final 2PC for any functionality represented over an arithmetic circuit over $\mathbb{Z}_{2^\ell}$ can be obtained by running the following steps in sequence: (a) sharing all the inputs via the sharing protocols, (b) gate by gate evaluation (using linearity of our secret sharing and the multiplication protocols) and (c) output reconstruction via the reconstruction protocol.

3.1 2PC in Arithmetic World

We provide the details for our 2PC scheme here. Before going into the details, we present a high-level overview of our scheme and a side-by-side comparison with the well-known Beaver's circuit randomization technique [8]. Our protocol, inspired by the 3PC protocol of ASTRA [30], achieves a communication similar to [12]. The highlight of our protocol is its effectiveness towards efficient realisations for multiple input multiplication gates and dot product operations as will be explained in §3.1.4 and §5.1 later.

3.1.1 High-level Overview of Our 2PC over Ring

Consider two parties P_0, P_1 with values a, b additively shared among them who want to compute a multiplication gate with output $\mathsf{c} = \mathsf{a} \cdot \mathsf{b}$.

Beaver's Technique [8] on Gate Inputs (cf. left of Fig. 1). In 2PC, there has been a lot of works [38, 39, 57, 61, 91] that use Beaver's [8] circuit randomization technique to compute the product $\mathsf{a} \cdot \mathsf{b}$. In this technique (cf. left side of Fig. 1), the inputs of the multiplication gate are randomized first and the corresponding correlated randomness is generated independently (preferably in a setup phase). In detail, parties interactively generate an additive sharing of the multiplication triple $(\delta_\mathsf{a}, \delta_\mathsf{b}, \delta_\mathsf{ab})$ with $\delta_\mathsf{ab} = \delta_\mathsf{a}\delta_\mathsf{b}$ during the setup phase before the actual inputs are known. Now, we can write

$$\begin{aligned}\mathsf{a} \cdot \mathsf{b} &= ((\mathsf{a} + \delta_\mathsf{a}) - \delta_\mathsf{a})((\mathsf{b} + \delta_\mathsf{b}) - \delta_\mathsf{b}) \\ &= (\mathsf{a} + \delta_\mathsf{a})(\mathsf{b} + \delta_\mathsf{b}) - (\mathsf{a} + \delta_\mathsf{a})\delta_\mathsf{b} - (\mathsf{b} + \delta_\mathsf{b})\delta_\mathsf{a} + \delta_\mathsf{ab}.\end{aligned}$$

Let $\Delta_a = (a + \delta_a)$ and $\Delta_b = (b + \delta_b)$ be the randomized versions of the input values of a multiplication gate. Then, during the online phase, parties locally compute an additive sharing of Δ_a using additive shares of a and δ_a. Similarly, an additive sharing of Δ_b is computed. This is followed by the parties mutually exchanging the shares of Δ_a and Δ_b to enable public reconstruction of Δ_a and Δ_b. Then using the above equation, parties can locally compute a sharing of $a \cdot b$. Note that this method requires communicating 4 elements per multiplication (2 elements per reconstruction). We observe that the communication is required for enabling parties to obtain the value of Δ_a and Δ_b in clear.

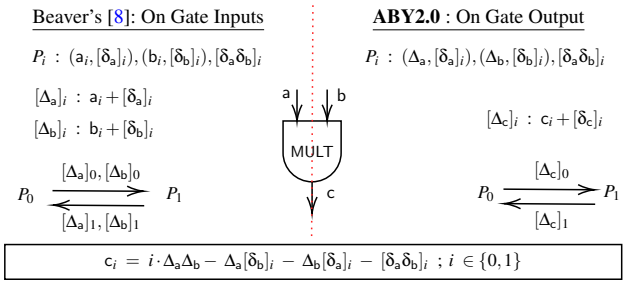


Figure 1: High level overview of Beaver's [8] and **ABY2.0**

Our Technique on Gate Outputs (cf. right of Fig. 1). With this insight, we modify the sharing semantics so that the parties are ensured to have the Δ value as a part of their share, corresponding to every wire value (including the inputs of a multiplication gate). As a result, the reconstructions of Δ_a and Δ_b are no longer required. This may give the wrong impression that no communication is required for evaluating a multiplication gate. It is true that now the parties can locally evaluate the additive sharing of $c = a \cdot b$. But in order to proceed further, a sharing for c according to the new sharing semantics needs to be generated. This requires both parties to obtain Δ_c in clear. Hence, the parties locally compute an additive sharing of Δ_c using the shares of c computed earlier and mutually exchange their shares to reconstruct Δ_c.

Our technique, in summary, shifts the need of reconstruction (which alone causes communication for a multiplication gate) from per input wire to the *output* wire alone for a multiplication gate. For a traditional 2-input multiplication gate, we reduce the number of reconstructions (each involves sending 2 elements) from 2 to 1. As a result, we improve communication by a factor of $2\times$. The impact is much higher for an N-input multiplication gate (cf. §3.1.4) and a scalar product of two N-dimensional vectors (cf. §5.1). For scalar product, Beaver's circuit re-randomization required $2N$ reconstructions, whereas our techniques need a *single* one, offering a gain of $2N\times$. Our constructions can be generalized to the n-party scenario (which is out of scope for this work) and bring a significant pay-off, as the cost per reconstruction depends linearly on the number of parties.

3.1.2 Sharing Semantics

$[\cdot]$-sharing. A value $v \in \mathbb{Z}_{2^\ell}$ is said to be $[\cdot]$-shared among $\mathcal{P}$, if party P_i for $i \in \{0,1\}$ holds $[v]_i$ such that $v = [v]_0 + [v]_1$.

$\langle\cdot\rangle$-sharing. A value $v \in \mathbb{Z}_{2^\ell}$ is said to be $\langle\cdot\rangle$-shared among $\mathcal{P}$, if there exist values $\delta_v, \Delta_v \in \mathbb{Z}_{2^\ell}$ such that i) δ_v is $[\cdot]$-shared among P_0, P_1, ii) $\Delta_v = v + \delta_v$, and iii) Δ_v is known to both P_0, P_1 in clear. We denote the shares of individual parties as $\langle v\rangle_i = ([\delta_v]_i, \Delta_v)$ for $i \in \{0,1\}$.

We use $\delta_{v_1 \ldots v_n}$ to represent the product $\delta_{v_1}\delta_{v_2}\cdots\delta_{v_n}$. Similarly, $\Delta_{v_1 \ldots v_n}$ represents $\Delta_{v_1}\Delta_{v_2}\cdots\Delta_{v_n}$.

3.1.3 Protocols

Sharing Protocol. Protocol SHARE enables party P_i for $i \in \{0,1\}$ to generate a $\langle\cdot\rangle$-sharing of its input value v. During the setup, P_i samples random $[\delta_v]_i$ while the parties together sample $[\delta_v]_{1-i}$ so that P_i will get to know $\delta_v = [\delta_v]_0 + [\delta_v]_1$ in clear. During the online phase, P_i computes $\Delta_v = v + \delta_v$ and sends it to P_{1-i}.

Reconstruction Protocol. To reconstruct value v given $\langle v\rangle$, protocol REC proceeds as follows: parties mutually exchange their missing $[\cdot]$-share of δ_v and locally compute $v = \Delta_v - [\delta_v]_0 - [\delta_v]_1$.

Linear Operations. Our sharing scheme is linear in the sense that given $\langle a\rangle, \langle b\rangle$ and public constants c_1, c_2, parties can locally compute $\langle y\rangle = c_1 \cdot \langle a\rangle + c_2 \cdot \langle b\rangle$. For this, P_i for $i \in \{0,1\}$ locally sets $\Delta_y = c_1 \cdot \Delta_a + c_2 \cdot \Delta_b$ and $[\delta_y]_i = c_1 \cdot [\delta_a]_i + c_2 \cdot [\delta_b]_i$.

Protocol MULT$(\langle a\rangle, \langle b\rangle)$

Setup:

- P_i for $i \in \{0,1\}$ samples random $[\delta_y]_i \in_R \mathbb{Z}_{2^\ell}$.
- Parties execute setupMULT$([\delta_a], [\delta_b])$ to generate $[\delta_{ab}]$.

Online:

- P_i for $i \in \{0,1\}$ locally computes and sends to P_{1-i} $[\Delta_y]_i = i \cdot \Delta_{ab} - \Delta_a[\delta_b]_i - \Delta_b[\delta_a]_i + [\delta_{ab}]_i + [\delta_y]_i$.
- P_i for $i \in \{0,1\}$ locally sets $\Delta_y = [\Delta_y]_0 + [\Delta_y]_1$.

Figure 2: Multiplication Protocol

Multiplication Protocol. Given the $\langle\cdot\rangle$-sharing of a, b, the goal of protocol MULT (cf. Fig. 2) is to generate $\langle y\rangle$ where $y = ab$. For correctness to hold, we will need

$$\begin{aligned}\Delta_y = y + \delta_y &= ab + \delta_y = (\Delta_a - \delta_a)(\Delta_b - \delta_b) + \delta_y \\ &= \Delta_a\Delta_b - \Delta_a\delta_b - \Delta_b\delta_a + \delta_a\delta_b + \delta_y.\end{aligned}$$

Since the δ-values are not available in clear to any of P_0, P_1, they cannot compute the value Δ_y on their own. But if we enable the parties obtain a $[\cdot]$-sharing of $\delta_{ab} = \delta_a\delta_b$, then each of them can compute a $[\cdot]$-sharing of Δ_y which they can mutually exchange to obtain Δ_y in clear. So the problem of multiplication reduces to generating $[\delta_{ab}]$ given $[\delta_a]$ and

$[\delta_b]$. We use protocol setupMULT to accomplish this task, the details of which is provided later in this subsection. We note that Turbospeedz [12] achieves same online cost as that of ours, but with a more expensive preprocessing. We provide more details in §A.3.

To summarize, during the setup phase, parties first locally sample the $[\cdot]$-shares for δ_y. In parallel, parties execute the setupMULT protocol on $[\delta_a]$ and $[\delta_b]$ to obtain $[\delta_{ab}]$. During the online phase, the parties locally compute $[\Delta_y]$ and subsequently reconstruct Δ_y.

We now provide the details for instantiating setupMULT using two of the well-known primitives: i) Oblivious Transfer (OT) as used in [39, 59] and ii) Homomorphic Encryption (HE) as used in [38,49,90]. These two approaches have been rallied against each other in terms of practical efficiency in the past and fair competition is still going on. In our work, we make only black-box access to these primitives, and hence an improvement in any of them will have a direct impact on the efficiency of the setup phase of our protocols.

Note that $\delta_{ab} = ([\delta_a]_0 + [\delta_a]_1)([\delta_b]_0 + [\delta_b]_1) = [\delta_a]_0[\delta_b]_0 + [\delta_a]_0[\delta_b]_1 + [\delta_a]_1[\delta_b]_0 + [\delta_a]_1[\delta_b]_1$. Here P_i for $i \in \{0,1\}$ can locally compute $[\delta_a]_i[\delta_b]_i$ and hence the problem reduces to computing $[\delta_a]_0[\delta_b]_1$ and $[\delta_a]_1[\delta_b]_0$.

OT based setupMULT**.** In our OT-based approach, we use Correlated OTs (cOT) [5] where the sender inputs a correlation function $f(\cdot)$ to cOT and obtains (m_0, m_1), where m_0 is a random element and $m_1 = f(m_0)$. We use cOT^n_ℓ to represent n parallel instances of 1-out-of-2 Correlated OTs on ℓ bit input strings.

To compute $[([\delta_a]_0[\delta_b]_1)]$, the parties execute cOT^ℓ_ℓ with P_0 being the sender and P_1 being the receiver. For the j-th instance of cOT where $j \in \{0,\ldots,\ell-1\}$, P_0 inputs the correlation $f_j(x) = x + 2^j[\delta_a]_0$ and obtains $(m_{j,0} = r_j, m_{j,1} = r_j + 2^j[\delta_a]_0)$. P_1 inputs choice bit b_j as the j-th bit of $[\delta_b]_1$ and obtains m_{j,b_j} as output. Now the $[\cdot]$-shares are defined as $[([\delta_a]_0[\delta_b]_1)]_0 = \sum_{j=0}^{\ell-1}(-r_j)$ and $[([\delta_a]_0[\delta_b]_1)]_1 = \sum_{j=0}^{\ell-1} m_{j,b_j}$. Computation of $[([\delta_a]_1[\delta_b]_0)]$ proceeds similarly with the role of the parties reversed.

HE-based setupMULT**.** In a HE based solution, P_0, using his public key pk_0, encrypts its messages $[\delta_a]_0, [\delta_b]_0$ in independent ciphertexts and sends the ciphertexts to P_1. In parallel, P_1 computes the ciphertexts corresponding to $[\delta_a]_1, [\delta_b]_1$ and a random element $r \in_R \mathbb{Z}_{2^\ell}$ using pk_0. Upon receiving the ciphertexts from P_0, P_1 computes the ciphertext corresponding to $\mathsf{v} = [\delta_a]_0[\delta_b]_1 + [\delta_a]_1[\delta_b]_0 - r$ using the homomorphic property of the underlying HE. P_1 then sends encryption of v to P_0 who then decrypts it using his secret key sk_0. Note that (v, r) forms an additive sharing of the desired value: $[\delta_a]_0[\delta_b]_1 + [\delta_a]_1[\delta_b]_0 = \mathsf{v} + r$.

A more detailed description for instantiating setupMULT using OT and HE is provided in the full version [83].

3.1.4 Multi-Input Multiplication Gates

3-Input Multiplication Gate. We show how to compute a 3-input multiplication gate (MULT3) with three inputs $\mathsf{a}, \mathsf{b}, \mathsf{c}$ with each input being $\langle\cdot\rangle$-shared. Similar to 2-input multiplication, we can write

$$\begin{aligned}\Delta_y &= \mathsf{abc} + \delta_y = (\Delta_a - \delta_a)(\Delta_b - \delta_b)(\Delta_c - \delta_c) + \delta_y \\ &= \Delta_{abc} - \Delta_{ab}\delta_c - \Delta_{bc}\delta_a - \Delta_{ac}\delta_b + \Delta_a\delta_{bc} + \Delta_b\delta_{ac} \\ &\quad + \Delta_c\delta_{ab} - \delta_{abc} + \delta_y.\end{aligned}$$

Here we need to generate the $[\cdot]$-sharing of four terms, namely $\delta_{ab}, \delta_{bc}, \delta_{ac}$ and δ_{abc} which is done by protocol setupMULT3. The protocol can be instantiated using either OT or HE in a similar fashion to that of setupMULT and the details are provided in the full version [83].

Multi-Input Multiplication Gate. We can extend our method to handle a 4-input multiplication (MULT4) gate and in the most general case, an N-input multiplication gate (MULTN) for any positive constant N, without inflating the online communication which remains just 2 ring elements independent of the fan-in of the gate. In contrast, the previous solution [78] requires an online communication of $2N$ ring elements for an N-input multiplication gate. Note that our improved online communication comes at the cost of an expensive setup and hence to maintain balance, we use $N \in \{3,4\}$ in our applications. We provide more details of [78] along with a comparison to our protocol in §A.3.

A more detailed description of MULT3, MULT4 and MULTN is given in the full version [83].

3.2 2PC in Boolean World

All the protocols mentioned above work over a Boolean ring ($\mathbb{Z}_{2^1}$) as well. This can be achieved by replacing additions (or subtractions) with XORs and multiplications with ANDs.

Negation Protocol. Given the **B**-sharing of a bit u as $\langle \mathsf{u}\rangle^{\mathbf{B}} = ([\delta_u], \Delta_u)$, the goal of a NOT protocol is to generate the boolean sharing of $\bar{\mathsf{u}}$. This can be done locally by setting $\Delta_{\bar{u}} = 1 \oplus \Delta_u$ and $[\delta_{\bar{u}}] = [\delta_u]$.

3.3 2PC in Yao World

For the Yao world, we follow the sharing semantics introduced by ABY [39]. For a wire u with value $\mathsf{v} \in \{0,1\}$, party P_0 acts as the garbler with the zero-key on the wire (K^0_u) being its share, while P_1 acts as the evaluator with the actual key (K^v_u) as its share. More formally, $\langle \mathsf{v}\rangle_0 = \mathsf{K}^0_u$ and $\langle \mathsf{v}\rangle_1 = \mathsf{K}^v_u$.

We use the free-XOR technique [66] in the garbling scheme, which enables the XOR gates to be evaluated without any communication. Here, the one-key for a wire is defined as a fixed offset from the zero-key as $\mathsf{K}^1_u = \mathsf{K}^0_u \oplus R$ with the least significant bit (LSB) of value R being set to 1 to enable

point-and-permute [10]. The value R is chosen by P_0 and is fixed across all the wires in the circuit.

To generate a $\langle\cdot\rangle$-sharing of a bit v, protocol $\mathsf{SHARE}(P_i,\mathsf{v})$ proceeds as follows: P_0 chooses a random zero-key $\mathsf{K}_\mathsf{u}^0 \in_R \{0,1\}^\kappa$ and sets $\mathsf{K}_\mathsf{u}^1 = \mathsf{K}_\mathsf{u}^0 \oplus R$, where κ denotes the computational security parameter. If $P_i = P_0$, P_0 sends $\mathsf{K}_\mathsf{u}^\mathsf{v}$ to P_1. For the case when $P_i = P_1$, parties engage in a cOT_κ^1 with P_0 being the sender and P_1 being the receiver. Here P_0 inputs the correlation function $f_R(x) = x \oplus R$ and obtains $(\mathsf{K}_\mathsf{u}^0, \mathsf{K}_\mathsf{u}^1 = \mathsf{K}_\mathsf{u}^0 \oplus R)$ while P_1 inputs v as choice bit and receives $\mathsf{K}_\mathsf{u}^\mathsf{v}$ as the output.

To generate a $\langle\cdot\rangle$-sharing of an ℓ-bit value v, parties execute the $\mathsf{SHARE}()$ protocol on each of its bits ($\mathsf{v}[j]$ for $j \in \{0,\ell-1\}$) in parallel. For a value $\mathsf{v} \in \mathbb{Z}_{2^\ell}$, we abuse the notation slightly and use $\langle\mathsf{v}\rangle$ to denote the $\langle\cdot\rangle$-sharing corresponding to each bit of v. We refer readers to ABY [39] for a formal description of the two-party Yao world and the operations.

4 Mixed Protocol Conversions

In this section, we show techniques to convert the shared values among the three protocols, namely– Arithmetic, Boolean, and Yao. We use the superscripts $\{\mathbf{A},\mathbf{B},\mathbf{Y}\}$ to distinguish the sharing and the respective protocols in the Arithmetic, Boolean, and Yao world respectively.

4.1 Standard Conversions

Here we detail the conversions amongst the three protocols. While most of the conversions of ABY [39] demand the execution of OT in the online phase, our protocols invoke OT in the setup phase only. This makes the online phase of the conversions– (a) free of any cryptographic operations and (b) run for just 1 round as opposed to 2 rounds for OT in ABY (cf. Tab. 2), except the Arithmetic to Boolean conversion.

Y2B. Given the $\langle\cdot\rangle^\mathbf{Y}$-sharing of a bit $\mathsf{u} \in \{0,1\}$, the goal is to generate its equivalent Boolean sharing. As observed in ABY, since the last bit of the zero and one key are distinct, XORing the LSB of K_u^0 and $\mathsf{K}_\mathsf{u}^\mathsf{u}$ results in the underlying bit u. Hence, each P_i for $i \in \{0,1\}$ Boolean-shares the LSB of their respective shares $\langle\mathsf{u}\rangle_i^\mathbf{Y}$ followed by locally XORing the shares to obtain the desired result. We note that P_0 can perform $\mathsf{SHARE}^\mathbf{B}(P_0,\mathsf{LSB}(\mathsf{K}_\mathsf{u}^0))$ already in the setup phase.

B2Y. To convert $\langle\mathsf{u}\rangle^\mathbf{B}$ to its equivalent $\langle\cdot\rangle^\mathbf{Y}$-sharing, P_i for $i \in \{0,1\}$ first locally sets $\mathsf{u}_i = (1-i)\cdot\Delta_\mathsf{u} \oplus [\delta_\mathsf{u}]_i$. It is easy to verify that $\mathsf{u} = \mathsf{u}_0 \oplus \mathsf{u}_1$. This is followed by party P_i generating $\langle\mathsf{u}_i\rangle^\mathbf{Y}$ by executing the $\mathsf{SHARE}^\mathbf{Y}(P_i,\mathsf{u}_i)$ protocol as described in §3.3. Given $\langle\mathsf{u}_0\rangle^\mathbf{Y}, \langle\mathsf{u}_1\rangle^\mathbf{Y}$, the parties can locally compute $\langle\mathsf{u}\rangle^\mathbf{Y} = \langle\mathsf{u}_0\rangle^\mathbf{Y} \oplus \langle\mathsf{u}_1\rangle^\mathbf{Y}$ using the free-XOR technique [66]. In our solution, we observe that parties can generate $\langle\mathsf{u}_1\rangle^\mathbf{Y}$ in the setup phase, with u_1 available in the setup phase itself. This observation allows us to shift the OT run to the setup phase, as opposed to ABY [39].

A2Y. The conversion from $\langle\mathsf{v}\rangle^\mathbf{A}$ to its equivalent $\langle\cdot\rangle^\mathbf{Y}$-sharing proceeds similar to that of the B2Y conversion. Party P_i for $i \in \{0,1\}$ locally sets $\mathsf{v}_i = (1-i)\cdot\Delta_\mathsf{v} - [\delta_\mathsf{v}]_i$ so that $\mathsf{v} = \mathsf{v}_0 + \mathsf{v}_1$. During the setup phase, P_0 garbles a two-input adder circuit which computes $\mathsf{y} = \mathsf{x}_0 + \mathsf{x}_1$, given the inputs $\mathsf{x}_0, \mathsf{x}_1 \in \mathbb{Z}_{2^\ell}$. The garbled circuit is then sent to P_1. In parallel, parties execute $\mathsf{SHARE}^\mathbf{Y}(P_1,\mathsf{v}_1)$ to generate $\langle\mathsf{v}_1\rangle^\mathbf{Y}$. During the online phase, parties execute $\mathsf{SHARE}^\mathbf{Y}(P_0,\mathsf{v}_0)$ to generate $\langle\mathsf{v}_0\rangle^\mathbf{Y}$. This is followed by P_1 locally evaluating the garbled adder circuit to generate $\langle\mathsf{v}\rangle^\mathbf{Y}$ which is our desired result. The adder circuit consists of ℓ AND gates [20]. Using the half-gates technique [99], this has setup communication of $2\ell\kappa$ bits.

Y2A. To convert $\langle\mathsf{v}\rangle^\mathbf{Y}$ to $\langle\mathsf{v}\rangle^\mathbf{A}$, parties proceed similarly to ABY [39] as follows: During the setup phase, P_0 samples a random value $r \in_R \mathbb{Z}_{2^\ell}$ and executes $\mathsf{SHARE}^\mathbf{Y}(P_0,r)$ and $\mathsf{SHARE}^\mathbf{A}(P_0,r)$ to generate $\langle r\rangle^\mathbf{Y}$ and $\langle r\rangle^\mathbf{A}$ respectively. In parallel, P_0 garbles an Adder circuit and sends the garbled circuit along with the decoding information to P_1. During the online phase, P_1 evaluates the garbled circuit with inputs $\langle\mathsf{v}\rangle^\mathbf{Y}$ and $\langle r\rangle^\mathbf{Y}$ to generate $\langle\mathsf{v}+r\rangle^\mathbf{Y}$. Using the decoding information, P_1 obtains the value $(\mathsf{v}+r)$ in clear followed by executing $\mathsf{SHARE}^\mathbf{A}(P_1,\mathsf{v}+r)$ to generate $\langle\mathsf{v}+r\rangle^\mathbf{A}$. Parties then locally compute $\langle\mathsf{v}\rangle^\mathbf{A} = \langle\mathsf{v}+r\rangle^\mathbf{A} - \langle r\rangle^\mathbf{A}$.

A2B. To convert an arithmetic share $\langle\mathsf{v}\rangle^\mathbf{A}$ to its equivalent Boolean share, parties use a Boolean Adder circuit similar to that of the A2Y conversion. Here, party P_i for $i \in \{0,1\}$ locally sets $\mathsf{v}_i = (1-i)\cdot\Delta_\mathsf{v} - [\delta_\mathsf{v}]_i$ followed by executing $\mathsf{SHARE}^\mathbf{B}(P_i,\mathsf{v}_i)$ to generate $\langle\mathsf{v}_i\rangle^\mathbf{B}$. Parties then evaluate the circuit using the 2PC protocol as described in §3. As mentioned in ABY [39] and ABY3 [73], the adder circuit can either be instantiated in its size-optimized [20] or depth-optimized variant (Parallel-prefix Adder [69]) and both these methods result in a non-constant (dependent on ℓ) number of rounds. A constant-round solution is to use $\mathsf{Y2B}(\mathsf{A2Y}(\langle\mathsf{v}\rangle^\mathbf{A}))$.

Bit2A. Here the goal is to generate the arithmetic sharing of a bit $\mathsf{v} \in \{0,1\}$, given its Boolean sharing $\langle\mathsf{v}\rangle^\mathbf{B}$. Let v^a denote the value of bit v when viewed over an ℓ-bit ring. Then for $\mathsf{v} = \mathsf{v}_0 \oplus \mathsf{v}_1$, we can write $\mathsf{v}^\mathsf{a} = \mathsf{v}_0^\mathsf{a} + \mathsf{v}_1^\mathsf{a} - 2\mathsf{v}_0^\mathsf{a}\mathsf{v}_1^\mathsf{a}$. We make use of this observation in the rest of the paper several times. Note that $\mathsf{v}^\mathsf{a} = (\Delta_\mathsf{v} \oplus \delta_\mathsf{v})^\mathsf{a} = \Delta_\mathsf{v}^\mathsf{a} + \delta_\mathsf{v}^\mathsf{a} - 2\Delta_\mathsf{v}^\mathsf{a}\delta_\mathsf{v}^\mathsf{a}$.

During the setup phase, parties interactively generate the $[\cdot]$ sharing of value $\delta_\mathsf{v}^\mathsf{a}$. During the online phase, P_i for $i \in \{0,1\}$ locally computes $[\mathsf{v}^\mathsf{a}]_i = i\cdot\Delta_\mathsf{v}^\mathsf{a} + (1-2\Delta_\mathsf{v}^\mathsf{a})\cdot[\delta_\mathsf{v}^\mathsf{a}]_i$ and executes $\mathsf{SHARE}^\mathbf{A}(P_i,[\mathsf{v}^\mathsf{a}]_i)$ to generate $\langle[\mathsf{v}^\mathsf{a}]_i\rangle^\mathbf{A}$. This is followed by parties locally computing $\langle\mathsf{v}^\mathsf{a}\rangle^\mathbf{A} = \langle[\mathsf{v}^\mathsf{a}]_0\rangle^\mathbf{A} + \langle[\mathsf{v}^\mathsf{a}]_1\rangle^\mathbf{A}$.

Now we describe how to generate $[\delta_\mathsf{v}^\mathsf{a}]$ in the setup phase, given the $[\cdot]$-sharing of bit δ_v. Since $\delta_\mathsf{v} = [\delta_\mathsf{v}]_0 \oplus [\delta_\mathsf{v}]_1$, we can write $\delta_\mathsf{v}^\mathsf{a} = [\delta_\mathsf{v}^\mathsf{a}]_0 + [\delta_\mathsf{v}^\mathsf{a}]_1 - 2([\delta_\mathsf{v}^\mathsf{a}]_0\,[\delta_\mathsf{v}^\mathsf{a}]_1)$. The parties first execute cOT_ℓ^1 with P_0 as sender and P_1 as receiver. P_0 inputs the correlation $f_j(x) = x + [\delta_\mathsf{v}]_0^\mathsf{a}$ and obtains $(s_0 = r, s_1 = r + [\delta_\mathsf{v}]_0^\mathsf{a})$. P_1 inputs the choice bit as $[\delta_\mathsf{v}]_1$ and obtains $s_{[\delta_\mathsf{v}]_1} = r + [\delta_\mathsf{v}]_1\cdot[\delta_\mathsf{v}]_0^\mathsf{a}$ as the output. P_0 locally sets $[([\delta_\mathsf{v}]_0^\mathsf{a}\,[\delta_\mathsf{v}]_1^\mathsf{a})]_0 =$

$-r$ while P_1 sets $[([\delta_v]_0^a\,[\delta_v]_1^a)]_0 = s_{[\delta_v]_1}$. Party P_i for $i \in \{0,1\}$ locally sets the $[\cdot]$-share of $[\delta_v^a]$ as $[\delta_v^a]_i = (1-i)\cdot[\delta_v]_0^a + i\cdot[\delta_v]_1^a - 2\,[([\delta_v]_0^a\,[\delta_v]_1^a)]_i$.

B2A. To convert a value $v \in \mathbb{Z}_{2^\ell}$ from its $\langle\cdot\rangle^{\mathbf{B}}$-sharing to its equivalent arithmetic sharing $\langle v\rangle^{\mathbf{A}}$, one simple solution is to follow steps similar to the Y2A conversion. Here, parties evaluate a Boolean subtraction circuit with $\langle v\rangle^{\mathbf{B}}$ and $\langle r\rangle^{\mathbf{B}}$ as the inputs, where r denotes a random value chosen by P_0. In addition, P_0 executes $\mathsf{SHARE}^{\mathbf{A}}(P_0, r)$ to generate $\langle r\rangle^{\mathbf{A}}$ as well. After the evaluation, the value $(v-r)$ is reconstructed to P_1, who further generates $\langle v-r\rangle^{\mathbf{A}}$. Parties then locally compute $\langle v\rangle^{\mathbf{A}} = \langle v+r\rangle^{\mathbf{A}} - \langle r\rangle^{\mathbf{A}}$.

As the above solution results in a non-constant round protocol in the online phase, we propose a *novel* round efficient variant which makes use of the Bit2A protocol. Our protocol was inspired from [31] that proposed a similar solution for the four party honest majority case. Here we make use of the fact that $v = \sum_{j=0}^{\ell-1} 2^j \cdot v[j]$ where $v[j]$ denotes the j^{th} bit of v. Since the parties possess $\langle v[j]\rangle^{\mathbf{B}}$ for each $j \in [0,\ell)$, they execute Bit2A conversion on $\langle v[j]\rangle^{\mathbf{B}}$ to generate its arithmetic equivalent $\langle v[j]\rangle^{\mathbf{A}}$. This results in a communication corresponding to ℓ instances of Bit2A conversions.

We observe that the online cost can be brought down to just 2 ring elements using the following approach. For each bit $v[j]$, parties locally compute the $[\cdot]$-sharing corresponding to $(v[j])^a$ as mentioned in Bit2A. Now, instead of generating the $\langle\cdot\rangle^{\mathbf{A}}$-share corresponding to each bit, P_i for $i \in \{0,1\}$ locally computes $[v]_i = \sum_{j=0}^{\ell-1} 2^j \cdot [(v[j])^a]_i$ and executes $\mathsf{SHARE}^{\mathbf{A}}(P_i, [v]_i)$ to generate $\langle [v]_i\rangle^{\mathbf{A}}$. Both parties then locally compute $\langle v\rangle^{\mathbf{A}} = \langle [v_0]\rangle^{\mathbf{A}} + \langle [v_1]\rangle^{\mathbf{A}}$. It is easy to verify that $v = [v]_0 + [v]_1$.

4.2 Special Conversions

For the three special conversions described below, the inputs are either Boolean shares or a mix of Boolean and arithmetic shares. The goal is to compute the equivalent arithmetic sharing of the product of the inputs. These conversions use the techniques of the Bit2A protocol (§4.1).

a) Protocol $\mathsf{PQ}(\langle p\rangle^{\mathbf{B}}, \langle q\rangle^{\mathbf{B}}) : \langle p\rangle^{\mathbf{B}}\langle q\rangle^{\mathbf{B}} \to \langle pq\rangle^{\mathbf{A}}$

Prep: $[\delta_p^a], [\delta_q^a], [\delta_p^a\delta_q^a]$

$(pq)^a = (\Delta_p^a + (1-2\Delta_p^a)\delta_p^a)(\Delta_q^a + (1-2\Delta_q^a)\delta_q^a)$

b) Protocol $\mathsf{PV}(\langle p\rangle^{\mathbf{B}}, \langle v\rangle^{\mathbf{A}}) : \langle p\rangle^{\mathbf{B}}\langle v\rangle^{\mathbf{A}} \to \langle pv\rangle^{\mathbf{A}}$

Prep: $[\delta_p^a], [\delta_p^a\delta_v]$

$(pv)^a = (\Delta_p^a + (1-2\Delta_p^a)\delta_p^a)(\Delta_v - \delta_v)$

c) Protocol $\mathsf{PQV}(\langle p\rangle^{\mathbf{B}}, \langle q\rangle^{\mathbf{B}}, \langle v\rangle^{\mathbf{A}}) : \langle p\rangle^{\mathbf{B}}\langle q\rangle^{\mathbf{B}}\langle v\rangle^{\mathbf{A}} \to \langle pqv\rangle^{\mathbf{A}}$

Prep: $[\delta_p^a], [\delta_q^a], [\delta_p^a\delta_q^a], [\delta_p^a\delta_v], [\delta_q^a\delta_v], [\delta_p^a\delta_q^a\delta_v]$

$(pqv)^a = (\Delta_p^a + (1-2\Delta_p^a)\delta_p^a)(\Delta_q^a + (1-2\Delta_q^a)\delta_q^a)(\Delta_v - \delta_v)$

During the online phase, parties locally generate a $[\cdot]$-sharing of the value to be computed followed by executing the $\mathsf{SHARE}^{\mathbf{A}}$ protocol on it to generate its equivalent arithmetic sharing. Then, parties locally add the resulting arithmetic shares to obtain the final result. The difference lies in the setup required for each of the conversions. The expression provided above shows the desired result in terms of corresponding Δ and δ values and the data (labelled as Prep) to be prepared in the setup phase.

As observed in the Bit2A protocol, the online phase of all these conversions consists of both parties executing arithmetic sharing of a single element resulting in one round with a communication of just 2 ring elements. We provide a detailed description of the conversions in the full version [83].

5 Building Blocks for Applications

In this section, we provide details for our building blocks that form the core of the applications that we explore in §6. We provide the formal details and communication cost analysis in the full version [83].

5.1 Scalar Product

Given the arithmetic sharing of n-element vectors $\vec{\mathbf{a}}, \vec{\mathbf{b}}$, the goal is to generate $\langle y\rangle^{\mathbf{A}}$ where $y = \vec{\mathbf{a}} \odot \vec{\mathbf{b}} = \sum_{j=1}^{n} a_i b_i$. One trivial way is to invoke the multiplication protocol from §3.1.3 corresponding to each of the n underlying multiplications. This would result in online communication linear in the vector size n. We now show how to make the online communication *independent* of the vector size.

The parties first execute the preprocessing corresponding to each of the n multiplications in parallel. Here we observe that there is no need to sample the shares of $[\delta_{y_j}]$ corresponding to each of the underlying multiplications. Instead, the parties locally sample the shares of $[\delta_y]$. During the online phase, parties first locally compute the $[\cdot]$-sharing of value Δ_{y_j} where y_j denotes $a_j b_j$. P_i for $i \in \{0,1\}$ now locally computes $[\Delta_y]_i = \sum_{j=1}^{n} [\Delta_{y_j}]_i$. This is followed by the parties mutually exchanging $[\Delta_y]$-shares to reconstruct Δ_y.

Compared with the state-of-the-art 2PC solutions in ABY [39] which require communication of $4n$ elements in the online phase, our protocol requires an online communication of just 2 ring elements.

5.2 Matrix Multiplication

Here we provide the details for extending our 2PC multiplication (§3.1.3) to the matrix setting. We abuse the notation slightly and use '$+$' for addition of matrices and '$-$' for subtraction. Also, we follow the $\langle\cdot\rangle$-sharing semantics for matrices as well. For $\mathbf{X}^{m\times n}$, we have $\Delta_{\mathbf{X}} = \mathbf{X} + [\delta_{\mathbf{X}}]_0 + [\delta_{\mathbf{X}}]_1$. Here $\Delta_{\mathbf{X}}, [\delta_{\mathbf{X}}]_0$ and $[\delta_{\mathbf{X}}]_1$ are matrices with dimension $m \times n$ and $x_{i,j}$ denote the $[i:j]$-th entry of $\mathbf{X}$.

Given $\mathbf{A}^{p\times q}, \mathbf{B}^{q\times r}$, protocol MATMULT proceeds as follows: During the setup phase, for $i \in [p], j \in [q], k \in$

$[r]$, parties execute $\mathsf{setupMULT}(\left[\delta_{\mathsf{a}_{i,j}}\right], \left[\delta_{\mathsf{b}_{j,k}}\right])$ to generate $\left[\delta_{\mathsf{a}_{i,j}\mathsf{b}_{j,k}}\right]$. This results in a $[\cdot]$-sharing of $\gamma_{\mathbf{AB}} = \delta_{\mathbf{A}} \circ \delta_{\mathbf{B}}$ among P_0, P_1. During the online phase, parties locally compute a $[\cdot]$-sharing of $\Delta_{\mathbf{C}}$ using the following relation:

$$\begin{aligned}\Delta_{\mathbf{C}} &= \mathbf{C} + \delta_{\mathbf{C}} = \mathbf{A} \circ \mathbf{B} + \delta_{\mathbf{C}} = (\Delta_{\mathbf{A}} - \delta_{\mathbf{A}}) \circ (\Delta_{\mathbf{B}} - \delta_{\mathbf{B}}) + \delta_{\mathbf{C}} \\ &= \Delta_{\mathbf{A}} \circ \Delta_{\mathbf{B}} - \Delta_{\mathbf{A}} \circ \delta_{\mathbf{B}} - \delta_{\mathbf{A}} \circ \Delta_{\mathbf{B}} + \gamma_{\mathbf{AB}} + \delta_{\mathbf{C}}.\end{aligned}$$

Finally, parties mutually exchange $[\Delta_{\mathbf{C}}]$ and obtain $\Delta_{\mathbf{C}}$ completing the protocol. Our protocol improved the online communication from $\mathrm{O}(pqr)$ to $\mathrm{O}(pr)$ ring elements, eliminating the dependency on dimension q.

5.3 Depth-Optimized Circuits

Parallel-prefix Adders (PPA) offer a depth-optimized solution to the binary addition between two ℓ-bit binary numbers. The best-known PPAs have $\log_2(\ell)$ depth [47]. Using ideas from [7,47], we design a PPA using two, three, and four input AND gates combined and obtain depth-optimized PPAs. For a 64-bit ring, we achieve a $2\times$ improvement in depth over existing designs along with a reduction in online communication.

Circuit	ℓ	#AND2		#AND3	#AND4	Depth	
Adder	8	15	(24)	6	1	2	(3)
BitExt	8	7	(14)	4	1	2	(3)
Adder	64	216	(384)	184	179	3	(6)
BitExt	64	41	(126)	27	47	3	(6)

Table 5: Depth-optimized Circuits for ℓ-bit rings. Previous circuits from ABY3 [73] are given in brackets.

As shown in [73], the PPA circuit can be optimized to obtain just the most significant bit (MSB), which we denote as Bit Extraction (BitExt) circuits. The efficiency gain in our PPA construction extends to BitExt circuits as well. Tab. 5 provides a summary of the results.

5.4 Comparison

As pointed out in [30,73], checking $\mathsf{x} < \mathsf{y}$ in the Fixed-Point Arithmetic (FPA) representation is equivalent to checking the sign of $\mathsf{v} = \mathsf{x} - \mathsf{y}$, which is stored in the MSB position of v.

The corresponding protocol LT begins with parties locally computing $\langle \mathsf{v} \rangle = \langle \mathsf{x} \rangle - \langle \mathsf{y} \rangle$. Let $\mathsf{v} = \mathsf{a} + \mathsf{b}$ where $\mathsf{a} = -[\delta_{\mathsf{v}}]_0$ and $\mathsf{b} = \Delta_{\mathsf{v}} - [\delta_{\mathsf{v}}]_1$. P_0, P_1 execute $\mathsf{SHARE}^{\mathbf{B}}$ on a, b respectively to generate its equivalent boolean sharing. The parties then use the Bit Extraction (BitExt, §5.3) circuit to compute $\mathsf{MSB}(\mathsf{v})$ in the boolean sharing format.

5.5 Truncation

In Fixed-Point Arithmetic (FPA), repeated multiplications result in an overflow with the fractional part doubling up in size after each multiplication. The naive solution of choosing a large enough ring to avoid the overflow is impractical for ML algorithms where the number of sequential multiplications is large. To tackle this, truncation [31,73,75] is used where the result of the multiplication is brought back to the FPA representation by chopping off the last x bits.

Below we explain how to perform truncation without affecting the communication cost for the multiplication. Our protocol is inspired by SecureML [75] and works as follows: During the online phase of multiplication, the parties first locally compute $[\mathsf{y}]$ directly instead of $[\Delta_{\mathsf{y}}]$. This is possible since $[\mathsf{y}] = [\Delta_{\mathsf{y}}] - [\delta_{\mathsf{y}}]$. Now each party locally truncates $[\mathsf{y}]$ to obtain the truncated value denoted by $[\mathsf{y}^t]$. This is followed by parties executing the $\mathsf{SHARE}^{\mathbf{A}}$ protocol on $[\mathsf{y}^t]$ to generate its arithmetic sharing. Finally, the parties locally compute $\langle \mathsf{y}^t \rangle^{\mathbf{A}} = \langle [\mathsf{y}^t]_0 \rangle^{\mathbf{A}} + \langle [\mathsf{y}^t]_1 \rangle^{\mathbf{A}}$. The correctness of the method follows trivially from SecureML.

5.6 MAX2 / MIN2

The MAX2 protocol is used to compute the maximum among two values a, b in a secure manner given $\langle \mathsf{a} \rangle^{\mathbf{A}}$ and $\langle \mathsf{b} \rangle^{\mathbf{A}}$. For this, the parties execute the LT protocol from §5.4 on $\langle \mathsf{a} \rangle^{\mathbf{A}}, \langle \mathsf{b} \rangle^{\mathbf{A}}$ to obtain $\langle \mathsf{u} \rangle^{\mathbf{B}} = \langle a < b \rangle^{\mathbf{B}}$. Note that $\mathsf{MAX2}(\mathsf{a}, \mathsf{b}) = \mathsf{u} \cdot (\mathsf{b} - \mathsf{a}) + \mathsf{a}$. Hence, parties can use the PV protocol from §4.2 to compute the desired result. The MIN2 protocol proceeds similarly except that $\mathsf{MIN2}(\mathsf{a}, \mathsf{b}) = \mathsf{u} \cdot (\mathsf{a} - \mathsf{b}) + \mathsf{b}$.

5.7 MAX3 / MIN3

Given the arithmetic sharing $\langle \mathsf{a} \rangle^{\mathbf{A}}, \langle \mathsf{b} \rangle^{\mathbf{A}}, \langle \mathsf{c} \rangle^{\mathbf{A}}$, the goal of the MAX3 protocol is to find the maximum value among the three. For this, we optimize the solution proposed by [78] which results in an improvement of $24.5\times$ in terms of the communication and $1.3\times$ in rounds in the online phase. The parties first securely compare the pairs $(\mathsf{a}, \mathsf{b}), (\mathsf{a}, \mathsf{c})$ and (b, c) using the LT protocol from §5.4 and obtain $\langle \mathsf{u}_1 \rangle^{\mathbf{B}}, \langle \mathsf{u}_2 \rangle^{\mathbf{B}}$ and $\langle \mathsf{u}_3 \rangle^{\mathbf{B}}$ respectively. Here $\mathsf{u}_1 = 1$ if $\mathsf{a} < \mathsf{b}$ and 0 otherwise. u_2 and u_3 are defined likewise . Now the maximum among the three, denoted by y, can be written as $\mathsf{y} = \overline{\mathsf{u}_1} \cdot \overline{\mathsf{u}_2} \cdot \mathsf{a} + \mathsf{u}_1 \cdot \overline{\mathsf{u}_3} \cdot \mathsf{b} + \mathsf{u}_2 \cdot \mathsf{u}_3 \cdot \mathsf{c}$.

Given $\langle \mathsf{u}_1 \rangle^{\mathbf{B}}, \langle \mathsf{u}_2 \rangle^{\mathbf{B}}, \langle \mathsf{u}_3 \rangle^{\mathbf{B}}$ and $\langle \mathsf{a} \rangle^{\mathbf{A}}, \langle \mathsf{b} \rangle^{\mathbf{A}}, \langle \mathsf{c} \rangle^{\mathbf{A}}$, the parties can use the PQV protocol from §4.2 to obtain each term in the expression for y and can locally add them to obtain the desired result. As an optimization, we can combine the online phase corresponding to all three executions of the PQV protocol into one. This reduces the online communication from six to two ring elements.

The protocol for MIN3, which computes the minimum among the three values can be obtained by slightly modifying the protocol for MAX3. The difference lies in the expression for computing the minimum which will now be $\mathsf{y} = \mathsf{u}_1 \cdot \mathsf{u}_2 \cdot \mathsf{a} + \overline{\mathsf{u}_1} \cdot \mathsf{u}_3 \cdot \mathsf{b} + \overline{\mathsf{u}_2} \cdot \overline{\mathsf{u}_3} \cdot \mathsf{c}$.

We observe that the protocol described above can be modified slightly to compute the index of the maximum (or minimum) among a set of three values. We use

ArgMax/ArgMin to denote such a protocol and the details are given in the full version [83].

5.8 Non-linear Activation Functions

We show how to compute two of the most widely used non-linear activation functions for PPML: ReLU and Sigmoid. ReLU function, defined as $\mathsf{ReLU}(v) = \max(0, v)$, can be written as $\mathsf{ReLU}(v) = \overline{u}v$, where $u = 1$ if $v < 0$ and 0 otherwise. To compute this, parties first execute the LT protocol from §5.4 on v to obtain $\langle u \rangle^{\mathbf{B}}$ followed by executing the PV protocol from §4.2 on $\langle \overline{u} \rangle^{\mathbf{B}}$ and $\langle v \rangle^{\mathbf{A}}$ to obtain the desired result. For Sigmoid, we use the MPC-friendly version [30, 73, 75] defined as $\mathsf{Sig}(v) = \overline{u_1}u_2(v + 1/2) + \overline{u_2}$, where $u_1 = 1$ if $v + 1/2 < 0$ and $u_2 = 1$ if $v - 1/2 < 0$.

5.9 Maxpool and Minpool

Given the arithmetic sharing of an n-element vector $\vec{x} = (x_1, \ldots, x_n)$ of values with $x_j \in \mathbb{Z}_{2^\ell}$ for $j \in \{1, \ldots, n\}$, the goal of the Maxpool protocol is to compute the arithmetic sharing of the maximum value among the n values.

For this, parties arrange the n values into an N-ary tree (tournament) composed of MAXN blocks with depth $\log_N(n)$ and evaluate in a top-down fashion [64]. In the recent work of [78], a maxpool using MAX3 was proposed where three values are compared at a time. In this work, we use our optimized MAX3 protocol from §5.7 as the building block for computing Maxpool. The improvement in rounds as well as communication of our MAX3 protocol over [78] directly translates to this case as well. We provide an empirical comparison for the Maxpool protocol in §6.1. Using MIN3 instead of MAX3 will directly provide a solution for Minpool, where the goal is to find the minimum among the values.

5.10 Equality Testing

Given $\langle a \rangle^{\mathbf{A}}, \langle b \rangle^{\mathbf{A}}$, the goal of the Equality Testing (EQ) protocol is to check whether $a \stackrel{?}{=} b$ or not. An equivalent formulation of the problem [18, 78] is to check if all the bits of $a - b$ are 0 or not. This simple primitive is crucial in building efficient protocol for applications like Circuit-based Private Set Intersection [85, 87, 88] (cf. §6.3), the Table Lookup Protocol from [40], and Data Mining [18].

We begin with the observation that if $x = y$, then using our sharing semantics we can write $\Delta_x - [\delta_x]_0 - [\delta_x]_1 = \Delta_y - [\delta_y]_0 - [\delta_y]_1$. Assuming $v_0 = (\Delta_x - [\delta_x]_0) - (\Delta_y - [\delta_y]_0)$ and $v_1 = [\delta_x]_1 - [\delta_y]_1$, the problem now reduces to checking whether $v_0 \stackrel{?}{=} v_1$ or not. Note that the value v_i can be locally computed by party P_i for $i \in \{0, 1\}$.

Protocol EQ proceeds as follows: P_i for $i \in \{0, 1\}$ locally computes v_i and executes $\mathsf{SHARE}^{\mathbf{B}}$ to generate $\langle v_i \rangle^{\mathbf{B}}$. The parties then compute $\langle v \rangle^{\mathbf{B}} = \mathsf{NOT}(\langle v_0 \rangle^{\mathbf{B}} \oplus \langle v_1 \rangle^{\mathbf{B}})$. Note that checking $v_0 = v_1$ is the same as checking whether all the bits of v are 1 or not. For this, the parties use AND4 gates and a tree structure, where 4 bits are taken at a time and the AND of them is computed in one go. This approach improves the round complexity by a factor of 2 over the traditional approach using AND2 gates. In concrete terms for a 64 bit ring, our solution improves over the protocol of [18] by $2\times$ in online rounds and by $2.4\times$ in online communication.

6 Applications and Benchmarks

All secure two-party applications using Boolean sharing (**B**) or Arithmetic sharing (**A**) directly benefit from our improvement in the online phase of our protocols. In this section, we give four applications with further improvements: i) AES which benefits from AND3 gates (§6.2), ii) Circuit-based Private Set Intersection (PSI) which benefits from our improved Equality Tests (§6.3), ii) Biometric Matching which benefits from our new *dimension-independent* Scalar Product and Minpool protocols (§6.4), and iv) Privacy-Preserving Machine Learning (PPML), specifically training and inference of Logistic Regression and Neural Networks which benefit from many of our improved protocol building blocks (§6.5). Since Maxpool/Minpool is an essential building block for several applications like K-means clustering [25], face-recognition [93], and fingerprint-matching [15, 43], we provide a separate analysis for Maxpool in §6.1.

To showcase the practicality of our constructions, we have implemented our protocols and compare them with their closest competitors. We implemented our protocols using the ENCRYPTO library [41] in C++17 over a 64-bit ring. Each experiment is run 15 times and the average values are reported. The benchmarking is performed over a LAN of 25Gbps bandwidth and a WAN of 75Mbps bandwidth. Over the LAN, we use two machines, each equipped with a 3.5 GHz Intel (R) Xeon (R) Gold 6144 CPU and 64 GB of RAM. The WAN was instantiated using n1-standard-8 instances of Google Cloud[1] with machines located in East Australia (P_0) and South East Asia (P_1). Over the WAN, machines are equipped with 2.3 GHz Intel Xeon E5 v3 (Haswell) processors supporting hyper-threading, with 8 vCPUs, and 30 GB of RAM. The average round-trip time (rtt), which was taken as the time for communicating 128 KB of data, turned out to be 0.056 ms for LAN and 60.19 ms for WAN.

6.1 Maxpool

Here we provide an empirical analysis of our Maxpool protocol from §5.9 and compare it with its competitors. We consider vectors with dimensions $n \in \{1024, 65536\}$. We have evaluated both round-optimized and communication-optimized variants of the Maxpool protocol. In the round-optimized variant proposed by SecureML [75], a garbled circuit is used to evaluate the maximum among n elements. This method requires converting Arithmetic shares to Yao shares and back, which can be tackled using A2Y and Y2A

[1] https://cloud.google.com

conversions. In the communication-optimized variant, we use the tree-based approach where either two or three elements are compared at a time as described in §5.9.

Ref.	Type	n = 1,024		n = 65,536	
		Comm [KB]	Rounds	Comm [KB]	Rounds
[75]	GC	2,056	4	131,584	4
ABY2.0	GC	1,024	**2**	65,536	**2**
[78]	MAX2	258	50	16,512	80
ABY2.0	MAX2	**53**	40	**3,408**	64
[78]	MAX3	492	35	31,679	55
ABY2.0	MAX3	63	28	4,080	44

Table 6: Online communication and rounds of Maxpool protocols. Best results in bold. n is the number of input elements.

Based on the building block used to instantiate Maxpool, the analysis can be divided into three cases – i) *Case I:* where the garbled circuit is used, ii) *Case II:* only MAX2 is used, and iii) *Case III:* a mix of MAX3 and MAX2 are used. For Case I, we compare with SecureML [75], while ours is compared with [78] for the rest. Table 6 summarizes the cost for the online phase of the Maxpool protocol. It is evident from the table that our protocols outperform [75, 78] in both communication and rounds for the online phase in all three cases.

For Case I, our round-optimized variant has a 2× improvement over SecureML [75] in both online communication and rounds. This is due to our efficient A2Y and Y2A conversions. For Case II, we improve upon [78] by a factor of 6.2× in online communication and 1.3× in rounds. Similarly, for Case III, the respective improvements over [78] are 9.6× and 1.3×. For cases II&III, while the improvement in online rounds is due to our efficient comparison protocol, improvement in communication is primarily contributed by our PQV protocol from §4.2. We also note that [78] improved the online rounds by 1.4× by switching from MAX2 to MAX3 as the building block for Maxpool at the expense of 1.9× higher online communication. In contrast, our solution improves the online rounds by 1.4× with a minimal overhead of 1.2× in online communication.

For the round-optimized variant, our protocol incurs an additional communication of just 2KB over SecureML in the setup phase. For the communication-optimized variant, we improve upon [78] for both MAX2 and MAX3 in terms of communication in the setup phase. This improvement results from our improved comparison protocol.

6.2 Improved S-box for AES

In a privacy-preserving AES [51, 86], the goal is to enable P_0 to encrypt her message x using a key k held P_1. The privacy guarantee is that P_0 gets the corresponding ciphertext while leaking nothing else. This has several applications in PSI [48, 58] and encrypted databases [2, 22]. Since the MixColumns and AddRoundKey operations can be evaluated using only free XOR gates [51], the focus was shifted to building efficient protocols for evaluating S-boxes as its core block. While [21] gives a depth-optimized S-box of 34 AND gates with an AND-Depth of 4, [19] gives a size-optimized solution with 32 AND gates and AND-Depth 6.

We give a new construction for the AES S-box that results in an effective AND-Depth of only 3. On a high level, we start with the three-layer construction of [19, 21] and optimize the middle layer (inversion layer) by replacing some of the AND2 gates with AND3 gates. This optimization is crucial since AES-128, AES-192 and AES-256 have 10, 12, and 14 sequential calls to layers of S-boxes resulting in a respective saving of 10, 12, and 14 rounds of interaction over [21]. We provide the empirical analysis in Table 7 and defer a detailed description to the full version [83].

Cipher	Ref.	#AND	Setup	Online	
			Comm [KB]	Comm [KB]	Rounds
AES 128	[21]	5,440	88.98	2.66	40
	[19]	5,120	**83.75**	2.50	60
	ABY2.0	5,440	98.13	**1.33**	**30**

Table 7: Communication and rounds for Secure evaluation of AES. Best results in bold.

In the setup phase, we used 4-OT_1^1 for AND2 gates and 8-OT_4^1 for AND3 gates. With the optimization of [40] applied, one instance of 4-OT_1^1 requires communication of 134 bits while 8-OT_4^1 takes 253 bits. Our protocol outperforms [21] and [19] in terms of both online communication and rounds.

6.3 Circuit-Based PSI

Circuit-based PSI [50] allows us to efficiently compute *variants* of the Private Set Intersection (PSI) functionality by securely evaluating a Boolean circuit. Today's most efficient protocols in this area [85, 87–89] do this by using hashing techniques and then evaluating a Boolean circuit that checks for equality among several bit strings using secure 2PC.

In fact, for today's most efficient circuit-based PSI protocol of [87], the majority of the computation, as well as communication, is spent on this two-party Equality Checking protocol. To be precise, 96% of the overall communication (cf. [87, Tab. 3]) and 34% – 63% of the overall runtime (cf. [87, Tab. 5]) is spent on Equality Checking. Plugging in our efficient Equality Checking protocol from §5.10 into the PSI protocol of [87] results in a direct improvement of ≈ 1.3× in runtime and ≈ 2.4× in communication in the online phase.

6.4 Biometric (Minimum Euclidean Distance)

Given a database owner with m biometric samples $(\vec{s_1}, \ldots, \vec{s_m})$ and a party with its biometric sample $\vec{c}$, the goal of privacy-preserving biometric matching is to find out the "minimum distance" of $\vec{c}$ from the database. This method is used for various traits of biometrics such as face-recognition [42, 49] and fingerprint-matching [15, 51]. Some of these works use the Squared Euclidean Distance (SED) as the metric to

compute the distance between two vectors. For two n-element vectors $\vec{a}, \vec{b}$, SED is defined as $\mathsf{SED}(\vec{a},\vec{b}) = \sum_{j=1}^{n}(a_j - b_j)^2$. Note that for $\vec{y} = \vec{a} - \vec{b}$, $\mathsf{SED}(\vec{a},\vec{b}) = \vec{y} \odot \vec{y}$.

In our framework, P_0 is the database owner while P_1 is the party with the sample to be checked. For finding the nearest sample securely, the parties first generate an arithmetic sharing of both the database samples and the query according to our sharing semantics. Given $\langle \vec{s_j} \rangle^{\mathbf{A}}$ for $j \in \{1,\ldots,m\}$ and $\langle \vec{c} \rangle^{\mathbf{A}}$, the parties locally compute $\langle \vec{x_j} \rangle^{\mathbf{A}} = \langle \vec{s_j} \rangle^{\mathbf{A}} - \langle \vec{c} \rangle^{\mathbf{A}}$. This is followed by running the dot product protocol from §5.1 on each $\langle \vec{x_j} \rangle^{\mathbf{A}}$ with itself to generate $\langle y_j \rangle^{\mathbf{A}} = \langle \vec{x_j} \odot \vec{x_j} \rangle^{\mathbf{A}}$. Note that the vector $\langle \vec{y} \rangle^{\mathbf{A}} = \{\langle y_1 \rangle^{\mathbf{A}}, \ldots, \langle y_m \rangle^{\mathbf{A}}\}$ represents the SED of the query with each of the database samples. To find the minimum among the elements of $\vec{y}$ given the arithmetic sharing of its elements, the parties can use either of the two methods described below.

In the first method, P_0 generates a garbled circuit that can compute the minimum among m inputs and sends this circuit to P_1. The parties then execute the A2Y conversion on each $\langle y_j \rangle^{\mathbf{A}}$ for $j \in \{1,\ldots,m\}$ to generate $\langle y_j \rangle^{\mathbf{Y}}$. P_1 evaluates the circuit to obtain the desired result in $\langle \cdot \rangle^{\mathbf{Y}}$-sharing. This method will result in a constant round solution, but the communication will be large. Another option is to use our Minpool protocol from §5.9 which results in a communication-efficient solution, but will require a non-constant number of rounds.

Ref.	Type	$m = 1{,}024$		$m = 4{,}096$		$m = 16{,}384$	
		Rounds	Comm [KB]	Rounds	Comm [KB]	Rounds	Comm [KB]
[39]	A+Y	5	2,312	5	9,248	5	36,992
ABY2.0	A+Y	**3**	1,040	**3**	4,160	**3**	16,640
[78]	A+B	36	748	41	3,003	46	12,014
ABY2.0	A+B	29	**51**	33	**205**	37	**818**

Table 8: Online rounds and communication of Minimum Euclidean Distance. Best results in bold. m is the number of biometric samples.

An empirical analysis for the online phase of the two aforementioned variants is given in Tab. 8. We consider databases with $m \in \{1{,}024, 4{,}096, 16{,}384\}$ samples. Each biometric sample has a dimension of $n = 8$.

For the round-optimized variant, we improve upon ABY [39] by 2.2× in communication and and by 1.6× in rounds in the online phase. Similarly, for the communication-optimized variant, our improvements over [78] are 14.7× in communication and 1.3× in rounds. The overhead in the setup cost for our protocol over ABY [39] and [78] is similar to that of Maxpool (§6.1) since Minpool forms the majority of the computation for Biometric Matching.

6.5 Privacy-Preserving Machine Learning (PPML)

In the domain of PPML [30, 31, 73, 75], we show that Logistic Regression and Neural Networks can be substantially improved with our building blocks. While we chose the above applications, our building blocks are sufficient to perform training and inference of Linear Regression and Convolutional Neural Networks [31] as well as inference of Support Vector Machines [30] and Binarized Neural Networks [27].

The training phase for the aforementioned algorithms consists of two stages: (i) a *forward propagation* phase, where the model computes the output given the input; and (ii) a *backward propagation* phase, where the model parameters are adjusted according to the difference in the computed output and the actual one. The inference phase can be viewed as one pass of the forward propagation alone. In our work, we use the technique of Batching [73, 75], where the entire set of samples is divided into batches of size B and a combined update function is applied to the weight vectors.

For the training phase, we follow [31, 73] and benchmark the *number of iterations per minute* (#it/min) over both LAN and WAN. The values are reported over batch sizes of $\{128, 256, 512\}$ and with feature sizes $n \in \{100, 900\}$. For the inference, we report the online runtime as well as the *throughput* (TP) for the aforementioned feature sizes. Runtime shows the impact of rounds on the overall performance, while TP denotes the numbers of queries the framework can process in a minute and allows to analyse the impact of communication.

Logistic Regression. In Logistic Regression, one iteration comprises of updation of the weight vector $\vec{\mathbf{w}}$ using the gradient descent algorithm (GD) as follows:

$$\vec{\mathbf{w}} = \vec{\mathbf{w}} - \frac{\alpha}{B}\mathbf{X}_i^T \circ (\mathsf{Sig}(\mathbf{X}_i \circ \vec{\mathbf{w}}) - \mathbf{Y}_i)\,.$$

Here α denotes the learning rate and $\mathbf{X}_i$ denotes a subset of batch size B, randomly selected from the entire dataset in the i-th iteration.

Batch Size	Ref.	LAN (#it/min)		WAN (#it/min)	
		$n = 100$	$n = 900$	$n = 100$	$n = 900$
128	[75]	29,112	27,273	108	104
	ABY2.0	**176,471**	**149,626**	**162**	**162**
256	[75]	25,829	24,058	107	97
	ABY2.0	**163,043**	**117,188**	**162**	**162**
512	[75]	23,292	22,247	104	83
	ABY2.0	**110,906**	**98,847**	**162**	**162**

Table 9: Comparison of the online throughput of ABY2.0 and SecureML [75] for Logistic Regression Training. Best results are in bold and larger is better. n is the number of features.

For the case of training, the data owner possesses the matrices $\mathbf{X}, \mathbf{Y}$ and the initial weights ($\vec{\mathbf{w}}$) are all set to 0. During the forward propagation, $\mathbf{X}_i \circ \vec{\mathbf{w}}$ is first computed followed by applying the sigmoid (Sig) function on it. During the backward propagation, the weight vector is updated according to the equation above. The update function requires computation of a series of matrix multiplications, which can be achieved using our dot product protocol from §5.1. The

operations of subtraction as well as multiplication by a public constant can be performed locally.

Tab. 9 gives our benchmarks for Logistic Regression training. Over SecureML [75], we have improvements in the range 4.4×-6.1× for LAN and in the range 1.5×-2.0× for WAN. The improvement stems from our round efficient comparison protocol from §5.4 that forms the building block for the activation function ReLU as well as our scalar product protocol from §5.1 that has a communication independent of the size of the vector. Note that over WAN, the throughput of our protocol remains unchanged across feature sizes as well as batch sizes. This discrepancy is due to the effect of communication on the rtt. In detail, the rtt is in the order of microseconds for LAN and scales with the communication size, whereas rtt in the WAN is in the order of milliseconds and does not scale with communication up to a threshold, within which all our protocols operate.

Parameter	Ref.	LAN		WAN	
		n = 100	n = 900	n = 100	n = 900
Runtime (ms)	[75]	1.60	1.69	496.08	504.96
	ABY2.0	**0.29**	**0.29**	**308.16**	**308.16**
Throughput (Queries/min)	[75]	5,342.61	1,193.01	16.08	3.58
	ABY2.0	**42,372.41**	**42,371.11**	**39.88**	**39.88**

Table 10: Comparison of the online runtime and throughput of ABY2.0 and SecureML [75] for Logistic Regression Inference. Best results in bold. n is the number of features.

Tab. 10 gives our benchmarks for Logistic Regression inference. We improve the online runtime over SecureML [75] by 5.5× for LAN and 1.6× for WAN, and the online throughput by 7.9×-35.5× in LAN and 2.5×-11.1× in WAN.

Neural Networks (NN). Neural Networks are stronger than regression algorithms since they can learn more complex relationships between high dimensional input and output data.

Batch Size	Ref.	LAN (#it/min)		WAN (#it/min)	
		n = 100	n = 900	n = 100	n = 900
128	[75]	3,593	3,559	17	17
	ABY2.0	**12,448**	**12,343**	**42**	**42**
256	[75]	3,578	3,521	17	17
	ABY2.0	**9,259**	**9,156**	**42**	**42**
512	[75]	3,330	3,323	15	15
	ABY2.0	**9,177**	**9,146**	**42**	**42**

Table 11: Comparison of the online throughput of ABY2.0 and SecureML [75] for NN Training. Best results in bold and larger is better. n is the number of features.

In our work, we follow previous works [30, 73, 75] and consider a Neural Network with two hidden layers, each having 128 nodes followed by an output layer of 10 nodes. We use ReLU as the activation function over the nodes. Moreover, for training we use the MPC-friendly variant of the softmax function [75] which is defined as $f(v_i) = \mathsf{ReLU}(v_i)/\sum_{j=1}^{m}\mathsf{ReLU}(v_j)$. The division is performed using a garbled circuit.

Tab.11 gives our benchmarks for NN Training. Over SecureML [75], we have improvements in the range 2.7×-3.46× for LAN and 2.4×-2.8× for WAN. Here the improvement is further boosted with our implementation of the softmax function that requires 2 online rounds as opposed to 4 rounds in SecureML.

Parameter	Ref.	LAN		WAN	
		n = 100	n = 900	n = 100	n = 900
Runtime (ms)	[75]	8.68	8.77	1,759.92	1,759.95
	ABY2.0	**2.66**	**2.66**	**744.12**	**744.12**
TP (queries/min)	[75]	62.02	40.89	0.19	0.12
	ABY2.0	**30,796.99**	**30,795.17**	**92.39**	**91.57**

Table 12: Comparison of the online runtime and throughput of ABY2.0 and SecureML [75] for NN Inference. Best results in bold. n is the number of features.

Tab. 12 gives our benchmarks for NN Inference. Here we improve the online runtime of SecureML [75] by a factor of 3.3× in LAN and 2.4× in WAN. Regarding the online throughput, we observe huge improvements in the range 496×–754× for both LAN and WAN. This improvement is primarily due to our efficient dot product protocol from §5.1 which has a dimension-independent online communication.

Setup Costs for PPML. We incur a minimal overhead of just 1.6% over SecureML [75] in terms of communication in the setup phase for Logistic Regression, while the overhead is 0.7% for the case of Neural Networks. The overhead results from the expensive communication required by our activation functions (Sigmoid and ReLU) over the garbled circuit-based solutions of SecureML [75].

Acknowledgements

Arpita Patra would like to acknowledge financial support from SERB MATRICS (Theoretical Sciences) Grant 2020 and Google India AI/ML Research Award 2020. Ajith Suresh would like to acknowledge financial support from Google PhD Fellowship 2019.

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 DFG — SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by BMBF and HMWK within ATHENE.

References

[1] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys*, 2018.

[2] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, 2015.

[3] A. Aly, E. Orsini, D. Rotaru, N. P. Smart, and T. Wood. Zaphod: Efficiently combining LSSS and garbled circuits in SCALE. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019.

[4] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE S&P*, 2017.

[5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, 2013.

[6] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, 2007.

[7] A. Beaumont-Smith and C. Lim. Parallel prefix adder design. In *IEEE Symposium on Computer Arithmetic*, 2001.

[8] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[9] D. Beaver. Precomputing oblivious transfer. In *CRYPTO*, 1995.

[10] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC*, 1990.

[11] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE S&P*, 2013.

[12] A. Ben-Efraim, M. Nielsen, and E. Omri. Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing. In *ACNS*, 2019.

[13] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.

[14] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. In *PETS*, 2016.

[15] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, 2011.

[16] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, C. Wierzynski, and H. Yalame. MP2ML: A mixed-protocol machine learning framework for private inference. In *ARES*, 2020.

[17] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, 2008.

[18] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 2012.

[19] J. Boyar, P. Matthews, and R. Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 2013.

[20] J. Boyar and R. Peralta. Concrete multiplicative complexity of symmetric functions. In *International Symposium on Mathematical Foundations of Computer Science*, 2006.

[21] J. Boyar and R. Peralta. A small depth-16 circuit for the AES S-box. In *IFIP International Information Security Conference*, 2012.

[22] L. T. Brandão, N. Christin, and G. Danezis. Toward mending two nation-scale brokered identification systems. In *PETS*, 2015.

[23] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *CCS*, 2007.

[24] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner. GSHADE: Faster privacy-preserving distance computation and biometric identification. In *IH&MMSEC*, 2014.

[25] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *CCS*, 2007.

[26] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *CCS*, 2018.

[27] M. Byali, H. Chaudhari, A. Patra, and A. Suresh. Flash: Fast and robust framework for privacy-preserving machine learning. In *PETS*, 2020.

[28] M. Byali, A. Joseph, A. Patra, and D. Ravi. Fast secure computation for small population over the Internet. In *CCS*, 2018.

[29] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile devices. *Journal of Computer Security*, 2016.

[30] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh. ASTRA: High throughput 3PC over rings with application to secure prediction. In *CCSW*, 2019.

[31] H. Chaudhari, R. Rachuri, and A. Suresh. Trident: Efficient 4PC framework for privacy preserving machine learning. In *NDSS*, 2020.

[32] J. I. Choi, D. J. Tian, G. Hernandez, C. Patton, B. Mood, T. Shrimpton, K. R. Butler, and P. Traynor. A hybrid approach to secure function evaluation using SGX. In *ASIACCS*, 2019.

[33] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. Spd$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In *CRYPTO*, 2018.

[34] R. Cramer, S. Fehr, Y. Ishai, and E. Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, 2003.

[35] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, and N. Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *IEEE S&P*, 2019.

[36] I. Damgård, M. Geisler, and M. Krøigaard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 2008.

[37] I. Damgård, C. Orlandi, and M. Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. In *CRYPTO*, 2018.

[38] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[39] D. Demmler, T. Schneider, and M. Zohner. ABY – A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[40] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner. Pushing the communication barrier in secure computation using lookup tables. In *NDSS*, 2017.

[41] ENCRYPTO Utils. https://github.com/encryptogroup/ENCRYPTO_utils.

[42] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS*, 2009.

[43] D. Evans, Y. Huang, J. Katz, and L. Malka. Efficient privacy-preserving biometric identification. In *NDSS*, 2011.

[44] O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[45] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

[46] S. D. Gordon, S. Ranellucci, and X. Wang. Secure computation with low communication from cross-checking. In *ASIACRYPT*, 2018.

[47] D. M. Harris. A taxonomy of parallel prefix networks. In *Asilomar Conference on Signals, Systems and Computers*, 2003.

[48] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.

[49] W. Henecka, S. Kögl, A. R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for automating secure two-party computations. In *CCS*, 2010.

[50] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[51] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*, 2011.

[52] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint*, 2018. http://arxiv.org/abs/1803.05961.

[53] N. Husted, S. Myers, A. Shelat, and P. Grubbs. GPU and CPU parallelization of honest-but-curious secure two-party computation. In *CCS*, 2013.

[54] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, 1989.

[55] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[56] M. H. S. Javadi, M. H. Yalame, and H. R. Mahdiani. Small constant mean-error imprecise adder/multiplier for efficient VLSI implementation of mac-based applications. *IEEE Transactions on Computers*, 2020.

[57] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *USENIX Security*, 2018.

[58] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert. Mobile private contact discovery at scale. In *USENIX Security*, 2019.

[59] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In *CCS*, 2016.

[60] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, 2018.

[61] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *CCS*, 2013.

[62] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, 1988.

[63] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, 2013.

[64] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, 2009.

[65] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *Journal of Computer Security*, 2013.

[66] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008.

[67] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: super-fast and robust privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2020. https://eprint.iacr.org/2020/592.

[68] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, 2012.

[69] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 1980.

[70] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. http://yann.lecun.com/exdb/mnist/.

[71] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[72] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 2009.

[73] P. Mohassel and P. Rindal. ABY^3: A mixed protocol framework for machine learning. In *CCS*, 2018.

[74] P. Mohassel, M. Rosulek, and Y. Zhang. Fast and secure three-party computation: Garbled circuit approach. In *CCS*, 2015.

[75] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

[76] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 2005.

[77] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, 1999.

[78] S. Ohata and K. Nuida. Communication-efficient (client-aided) secure two-party protocols and its application. In *FC*, 2020.

[79] E. Orsini, N. P. Smart, and F. Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In *CT-RSA*, 2020.

[80] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[81] A. Patra and D. Ravi. On the exact round complexity of secure three-party computation. In *CRYPTO*, 2018.

[82] A. Patra, P. Sarkar, and A. Suresh. Fast actively secure OT extension for short secrets. In *NDSS*, 2017.

[83] A. Patra, T. Schneider, A. Suresh, and H. Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. *IACR Cryptology ePrint Archive*, 2020. https://eprint.iacr.org/2020/1225.

[84] A. Patra and A. Suresh. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*, 2020.

[85] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*, 2015.

[86] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, 2009.

[87] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, 2019.

[88] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT*, 2018.

[89] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security*, 2018.

[90] D. Rathee, T. Schneider, and K. Shukla. Improved multiplication triple generation over rings via RLWE-based AHE. In *CANS*, 2019.

[91] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS*, 2018.

[92] D. Rotaru and T. Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *INDOCRYPT*, 2019.

[93] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, 2009.

[94] S. Sharma, C. Xing, and Y. Liu. Privacy-preserving deep learning with SPDZ. In *The AAAI Workshop on Privacy-Preserving Artificial Intelligence*, 2019.

[95] Stanford. CS231n: Convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/.

[96] O. Tkachenko, C. Weinert, T. Schneider, and K. Hamacher. Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In *ASIACCS*, 2018.

[97] M. H. Yalame, M. H. Farzam, and S. B. Sarmadi. Secure two-party computation using an efficient garbled circuit by reducing data transfer. In *Applications and Techniques in Information Security (ATIS)*, 2017.

[98] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS*, 1986.

[99] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015.

[100] Y. Zhang, A. Steele, and M. Blanton. PICCO: A general-purpose compiler for private distributed computation. In *CCS*, 2013.

[101] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *IEEE S&P*, 2019.

A Preliminaries

A.1 Oblivious Transfer (OT)

In a 1-out-of-n Oblivious Transfer [54, 76] (OT) over ℓ-bit messages, the sender S inputs n messages $(x_1,\ldots,x_n)$ each of length ℓ bits, while the receiver R inputs the choice $c \in \{1,\ldots,n\}$. R receives x_c as output while S receives $\perp$ as output. The privacy guarantee is that S learns nothing about c, while R learns nothing about the inputs of S other than x_c. We use n-OT_ℓ^m to denote m instances of 1-out-of-n OT on ℓ bit inputs.

OT is a fundamental building block for MPC [62] and requires expensive public-key cryptography [54]. The technique of OT Extension [5, 55, 63, 82] allows us to generate many OTs from a small number (equal to the security parameter) of base OTs at the expense of symmetric-key operations alone. This reduces the cost of OT mainly to highly efficient symmetric-key primitives. Concretely, the OT Extension implementation of [5] generates around 1 million 2-OT_ℓ^1 per second with passive security. An orthogonal line of work considered pre-computation of OT [9], where all the cryptographic operations can be shifted to a setup phase, independent of the function to be evaluated. This technique enables a very efficient online phase for protocols that use OT. In the semi-honest setting, the state-of-the-art solution for OT extension [5] has communication $\kappa+2\ell$ bits per OT for 2-OT_ℓ^1 where κ denotes the computational security parameter.

A correlated OT (cOT) [5] is a variant of the traditional OT where the sender's input messages are correlated. In a cOT, the sender inputs a correlation function $f()$ and obtains the message pair $(x_0 \in_R \{0,1\}^\ell, x_1 = f(x_0))$ as the output. The receiver, on the other hand, inputs her choice c and obtains x_c as output. We use cOT_ℓ^m to denote m instances of 1-out-of-2 correlated OT on ℓ bit inputs. In the semi-honest setting, cOT_ℓ^1 has communication $\kappa+\ell$ bits [5].

A.2 Secure 2PC

Homomorphic Encryption (HE). The homomorphic property allows us to compute a ciphertext from a set of ciphertexts such that the plaintext underlying the former is a function of the underlying plaintexts of the latter. Towards this, one party called client generates a key-pair (pk,sk) for the HE scheme and sends pk to the other party called server. To perform a secure computation operation, the client encrypts its data using pk and sends this to the server. Now the server can locally compute the ciphertext corresponding to the operation and return the encrypted result to the client. The client can now decrypt the received ciphertext using her private key sk. An *additively HE* allows us to generate the ciphertext corresponding to the sum of the underlying plaintexts by doing operations on the ciphertexts. Prominent examples of additively HE schemes are Paillier [80], DGK [36] and RLWE-AHE [90]. On the other hand, fully homomorphic encryption schemes allow arbitrary computations under encryption but are less efficient. See [1] for a more detailed description.

Garbled Circuits (GC). In the two-party setting, Yao's garbled circuit protocol [72, 98] provides a constant-round solution. This method is particularly useful in high-latency networks like the Internet. Here, one party called *garbler*

generates the garbled circuit (GC) corresponding to the function to be evaluated. On a high level, garbling the circuit consists of associating two keys per wire corresponding to the bit values of $\{0,1\}$ and preparing garbled tables corresponding to each gate in the circuit. The garbler then sends the GC to the other party called *evaluator*. The evaluator, upon obliviously obtaining the keys corresponding to the inputs via OT, evaluates the GC and obtains the output.

Today's most efficient solution for garbled circuits is the combination of point-and-permute [10], free-XOR [66], fixed-key AES [11], and half-gates [99]. With these optimizations, each AND gate requires communication 2κ bits in the setup phase, and XOR gates have no communication. GC-based protocols perform in the online phase symmetric-key operations for each AND gate and need substantial memory to store the garbled tables. To avoid storing the garbled tables, their generation and transfer can be pipelined [49,51], but this shifts all the setup communication to the online phase.

Secret Sharing (SS). In the SS-based protocols, two parties compute a function in a secret-shared manner. Here, for every wire with value v, party P_i for $i \in \{0,1\}$ holds an additive sharing of the value denoted by $[\mathsf{v}]_i$ such that $\mathsf{v} = [\mathsf{v}]_0 + [\mathsf{v}]_1 \pmod{2^\ell}$. All the linear gates can be evaluated non-interactively. To securely evaluate a multiplication gate, parties use Beaver's [8] circuit randomization technique where the additive sharing of a random arithmetic triple is generated in the setup phase (cf. §3.1.1). The shares of the triple are then used in the online phase to compute the shares of the product. This requires communication of 4 ring elements per multiplication gate in the online phase. Later, [12] reduced online communication to 2 ring elements using a function-dependent preprocessing.

In this line of work, the GMW protocol [45] takes a function represented as Boolean circuit (i.e., $\ell = 1$) and the values are secret-shared using XOR-based secret sharing. To pre-compute a multiplication triple $(\mathsf{c}_1 \oplus \mathsf{c}_2) = (\mathsf{a}_1 \oplus \mathsf{a}_2) \wedge (\mathsf{b}_1 \oplus \mathsf{b}_2)$, the solution of [5] which uses 1-out-of-2 OT, requires 2κ bits of communication. As shown in [40], this cost can be improved by factor $1.2\times$ by using the 1-out-of-N OT extension of [63].

A.3 Comparison with Turbospeedz [12] and [78]

Comparison with Turbospeedz [12]. For the 2-input multiplication, Turbospeedz [12] presented a protocol that reduces the online communication of SPDZ-style protocols from 4 to 2 ring elements using a function-dependent preprocessing. Turbospeedz first executes a SPDZ-like preprocessing where random multiplication triples are generated. These triples are then associated to the multiplication gates using additional values that they call "external values" (cf. [12], §3.2). On the contrary, we obtain the preprocessing data directly and hence save communication of 4 ring elements as well as storage of 5 ring elements when compared with Turbospeedz. Tab. 13 provides the communication and storage required for the 2-input multiplication protocol of ABY [39], Turbospeedz [12] and ABY2.0.

Phase	Parameter	ABY [39]	Turbospeedz [12]	**ABY2.0**
Setup	Storage	3ℓ	9ℓ	4ℓ
	Communication	$\lvert\mathsf{Triple}\rvert$	$\lvert\mathsf{Triple}\rvert + 4\ell$	$\lvert\mathsf{Triple}\rvert$
Online	Storage	5ℓ	5ℓ	$\mathbf{3\ell}$
	Communication	4ℓ	$\mathbf{2\ell}$	$\mathbf{2\ell}$
Total	Storage	8ℓ	14ℓ	7ℓ
	Communication	$\lvert\mathsf{Triple}\rvert + 4\ell$	$\lvert\mathsf{Triple}\rvert + 6\ell$	$\lvert\mathsf{Triple}\rvert + 2\ell$

Table 13: Comparison of ABY2.0 with ABY [39] and Turbospeedz [12] in terms of storage and communication for a single multiplication. All values are given in bits. $\lvert\mathsf{Triple}\rvert$ denotes the communication required to generate a multiplication triple. Best values for the online phase are marked in bold.

For the multi-input multiplication (fan-in of N), the tree-based method (multiplying N elements by taking two at a time) requires $\log_2(N)$ rounds for both ABY [39] and Turbospeedz [12], while it requires communication of $4(N-1)$ ring elements for ABY [39] and $2(N-1)$ elements for Turbospeedz [12] in the online phase.

Comparison with [78]. Recently, [78] proposed round-efficient solutions for multi-input multiplication using a preprocessing for which the communication cost grows exponentially with the fan-in of the multiplication gate. However, for an N-input multiplication, [78] requires an online communication of $2N-2$ ring elements. On the contrary, ABY2.0 requires only an online communication of 2 ring elements and the preprocessing cost remains same as that of [78]. Note that since the preprocessing cost grows exponentially with the number of inputs to the multiplication gate, [78] considered only up to 5-input multiplication gates in their work. In our work, we use three and four input multiplication gates.

MULT *when input parties are the computing parties:* For the case of a two-input multiplication gate, [78] considered a special case where the input parties are the computing parties (cf. [78], §3.4). For this case, [78] proposed a protocol for which the online communication is 2 ring elements. For the same setting, we observe that our solution results in a protocol with zero online communication. To see this, recall the online phase of our multiplication protocol $\mathsf{MULT}(\langle\mathsf{a}\rangle, \langle\mathsf{b}\rangle)$ (Fig. 2). The modified protocol is as follows: During the online phase, party P_i for $i \in \{0,1\}$ locally computes $[\mathsf{ab}]_i = i \cdot \Delta_{\mathsf{ab}} - \Delta_{\mathsf{a}}[\delta_{\mathsf{b}}]_i - \Delta_{\mathsf{b}}[\delta_{\mathsf{a}}]_i + [\delta_{\mathsf{ab}}]_i$. Now to generate $\langle\cdot\rangle$-shares corresponding to $\mathsf{y} = \mathsf{ab}$, the parties locally set $[\delta_{\mathsf{y}}]_i = -[\mathsf{ab}]_i$ and $\Delta_{\mathsf{y}} = 0$. It is easy to see that $\mathsf{y} = \Delta_{\mathsf{y}} - [\delta_{\mathsf{y}}]_0 - [\delta_{\mathsf{c}}]_1 = 0 - ([\mathsf{ab}]_0 + [\mathsf{ab}]_1) = \mathsf{ab}$.

F FLUTE: Fast and Secure Lookup Table Evaluations (IEEE S&P'23)

[BHS+23] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, H. Yalame. **"FLUTE: Fast and Secure Lookup Table Evaluations"**. In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/499. Code: https://encrypto.de/code/FLUTE. IEEE, 2023, pp. 515–533. CORE Rank **A***. Appendix F.

https://doi.org/10.1109/SP46215.2023.10179345

G Breaking the Size Barrier: Universal Circuits meet Lookup Tables (ASIACRYPT'23)

[DGS+23] Y. Disser, D. Günther, T. Schneider, M. Stillger, A. Wigandt, H. Yalame. **"Breaking the Size Barrier: Universal Circuits meet Lookup Tables"**. In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Vol. 14438. LNCS. Online: https://ia.cr/2021/809. Code: https://encrypto.de/code/LUC. Springer, 2023, pp. 3–37. CORE Rank **A**. Appendix G.

https://link.springer.com/chapter/10.1007/978-981-99-8721-4_1

Breaking the Size Barrier: Universal Circuits Meet Lookup Tables

Yann Disser, Daniel Günther( ), Thomas Schneider, Maximilian Stillger, Arthur Wigandt, and Hossein Yalame

Technical University of Darmstadt, Darmstadt, Germany
disser@mathematik.tu-darmstadt.de,
{guenther,schneider,yalame}@encrypto.cs.tu-darmstadt.de,
maximilian.stillger@arcor.de, arthur.wigandt@protonmail.com

Abstract. A Universal Circuit (UC) is a Boolean circuit of size $\Theta(n \log n)$ that can simulate any Boolean function up to a certain size n. Valiant (STOC'76) provided the first two UC constructions of asymptotic sizes $\sim 5n \log n$ and $\sim 4.75n \log n$, and today's most efficient construction of Liu et al. (CRYPTO'21) has size $\sim 3n \log n$. Evaluating a public UC with a secure Multi-Party Computation (MPC) protocol allows efficient Private Function Evaluation (PFE), where a private function is evaluated on private data.

Previously, most UC constructions have only been developed for circuits consisting of 2-input gates. In this work, we generalize UCs to simulate circuits consisting of $(\rho \to \omega)$-Lookup Tables (LUTs) that map ρ input bits to ω output bits. Our LUT-based UC (LUC) construction has an asymptotic size of $1.5\rho\omega n \log \omega n$ and improves the size of the UC over the best previous UC construction of Liu et al. (CRYPTO'21) by factors 1.12×–2.18× for common functions. Our results show that the greatest size improvement is achieved for $\rho = 3$ inputs, and it decreases for $\rho > 3$.

Furthermore, we introduce Varying Universal Circuits (VUCs), which reduce circuit size at the expense of leaking the number of inputs ρ and outputs ω of each LUT. Our benchmarks demonstrate that VUCs can improve over the size of the LUC construction by a factor of up to 1.45×.

Keywords: universal circuit · private function evaluation · multi-party computation

1 Introduction

A *Universal Circuit* (UC) $\mathcal{U}$ is a Boolean circuit that can simulate any Boolean circuit C consisting of n_i inputs, n_g gates, and n_o outputs. The UC $\mathcal{U}$ takes, in addition to the function's input x, a set of programming bits p^C defining the circuit C that $\mathcal{U}$ simulates, i.e., the UC computes $\mathcal{U}(x, p^C) = C(x)$.

Valiant [51] proposed the first two UC constructions known as *2-way* and *4-way* split UCs with asymptotically optimal size $\Theta(n \log n)$ and depth $\mathcal{O}(n)$, where $n = n_i + n_g + n_o$ is the size of the simulated circuit C. Kolesnikov and

J. Guo and R. Steinfeld (Eds.): ASIACRYPT 2023, LNCS 14438, pp. 3–37, 2023.
https://doi.org/10.1007/978-981-99-8721-4_1

Schneider [34] gave the first practical implementation of a UC of non-optimal asymptotic size $\mathcal{O}(n \log^2 n)$. A line of work [7,22,31,36,57] followed with the common goal to minimize the size of Valiant's UC construction. Recently, Liu et al. [37] provided today's most efficient UC construction of size $\sim 3n \log n$.

All of these works designed UCs to simulate Boolean gates with 2 inputs and 1 output. However, Valiant's UC construction can be generalized to simulate circuits with $(\rho \to 1)$-LUT, namely *Lookup-Tables* with ρ inputs $x_1, \ldots, x_\rho$ and one output y and can compute arbitrary functionalities f as $y = f(x_1, \ldots, x_\rho)$ [48].

In this work, we propose LUT-based UCs (LUC) that evaluate circuits composed of $(\rho \to \omega)$-LUTs having ω output bits $y_1, \ldots, y_\omega$ and are programmed to compute $y_i = f^i(x_1, \ldots, x_\rho)$ for $1 \leq i \leq \omega$ and an arbitrary functionality f^i. In addition, we introduce *Varying UCs (VUCs)* that can simulate circuits consisting of $(\rho \to \omega)$-LUTs with varying numbers of inputs ρ and outputs ω, thereby leaking the number of in- and outputs of each LUT. VUCs have various applications (summarized in Sect. 1.1) like logic locking [55], which enables the designer to provide the foundry of a chip with a "locked" version of the original circuit. Once the locked circuit on the chip is fabricated, authorized users can regain access to the original functionality by using a secret key.

On top of our new UC constructions, we provide implementations of our constructions and analyze the size optimization of simulating LUT-based circuits with LUCs and VUCs compared to using traditional Boolean circuit-based UCs.

1.1 Applications of (Varying) Universal Circuits

The most prominent application for UCs is **Private Function Evaluation (PFE)** [6], which can be seen as a generalization of *Secure Multi-Party Computation* (MPC) [20,54]. In MPC, a set of k parties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ jointly compute a publicly known circuit C on their respective private inputs $x_1, \ldots, x_k$ and obtain nothing but the result $C(x_1, \ldots, x_k)$. In PFE, the circuit C that shall be computed is private information as well, i.e., party $\mathcal{P}_1$ with circuit input C and parties $\mathcal{P}_{2 \leq i \leq k}$ with data inputs $x_2, \ldots, x_k$ run a protocol that yields nothing but $C(x_2, \ldots, x_k)$ and parties $\mathcal{P}_{2 \leq i \leq k}$ do not learn any information about the circuit C.

PFE can be implemented via MPC by means of UCs as follows: The parties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ run an MPC protocol that evaluates the universal circuit $\mathcal{U}$ as public circuit on the secret inputs p^C of party $\mathcal{P}_1$ and $x_2, \ldots, x_k$ of parties $\mathcal{P}_2, \ldots, \mathcal{P}_k$, resulting in $\mathcal{U}(p^C, x_2, ..., x_k) = C(x_2, ..., x_k)$. In summary, PFE based on UCs is a very generic approach. It can simply be plugged into arbitrary MPC frameworks without any modification to the underlying MPC protocol, resulting in the same security level (semi-honest, covert, or malicious) as the underlying MPC framework. In addition, PFE is completely compatible with the features included in MPC like *secure outsourcing* [27] and *non-interactive computation* [36]. PFE is applicable for situations where customers aim to use a service from companies who want to hide *how* they perform the computation and do not learn the customer's data.[1]

[1] UC-based PFE, unlike PFE based on Fully Homomorphic Encryption (FHE) [19,32], relies primarily on symmetric encryption and involves far less computation.

As a trade-off between privacy and efficiency, a variant of PFE called **Semi-Private Function Evaluation (SPFE)** was proposed [21,43]. Unlike PFE, SPFE does not hide the entire function, but leaks the topology of certain sub-functions. SPFE can be applied in PFE scenarios where specific function components are known publicly. This approach is particularly useful in cases where certain function details have already been disclosed, often for promotional purposes. An example of this is car insurance companies offering discounts to experienced drivers.

Beyond (S)PFE, UCs have many applications like hiding policy circuits in attribute-based encryption [8,18], multi-hop homomorphic encryption [19], verifiable computation [16], program obfuscation [58], and hardware logic locking [40].

In this work, we introduce **Varying Private Function Evaluation (VPFE)** whose privacy-guarantee lies in between PFE and SPFE. Similar to PFE, our Varying UCs (VUCs) for VPFE hide the topology and functionality of the LUTs in the circuit, but leak their number of in- and outputs. This has applications in logic obfuscation techniques called logic locking, as demonstrated in prior studies such as LUT-Lock [26] and eFPGA [11]. These techniques proposed using LUTs to achieve secure logic locking on Application-Specific Integrated Circuit (ASIC) designs by removing critical elements and mapping them to custom LUTs. As shown in [11, Fig. 3], the adversary can only determine the number of inputs and outputs of a LUT, while the LUT's configuration bits are hidden, which is exactly our setting for VPFE using VUCs. Without this knowledge, there is no adversarial information leakage [11, Tab. 5]. Therefore, our VUCs can be used for secure logic locking while additionally hiding the topology of the circuit.

1.2 Outline and Our Contributions

So far, UC-based PFE research considered synthesis of the input circuit (to generate a small number of 2-input gates) and construction of the UC (to minimize its size) as independent tasks. In our work, we show that using multi-input/output LUTs these two tasks can be combined to yield a better size. After giving the preliminaries in Sect. 2 and summarizing the two UC constructions of Valiant [51] (Sect. 3.2) and Liu et al. [37] (Sect. 3.3), we contribute the following:

LUT-based UC (LUC) Construction (Sect. 4). Valiant's UC construction can be generalized to support the evaluation of $(\rho \to 1)$-LUT-based circuits by merging for the ρ inputs ρ instances of its basic building block called edge-universal graph [48, App. A]. This leads to a total size of $\sim 1.5\rho n \log n$ using Liu et al.'s [37] UC construction. In our work, we extend this into a novel UC construction to simulate for the first time functions composed of $(\rho \to \omega)$-LUTs with ρ inputs and ω outputs. Our construction is general, can be applied to all UC constructions based on Valiant's framework [51] and improvements by Liu et al. [37], and fits into the definition of UCs (cf. Definition 1 on page 6).

Size Improvements of LUCs for Basic Primitives (Sect. 4.3). Table 1 shows the history of improvements in UC sizes. Taking (V)PFE as our greatest motivation for improving UC sizes, we study three basic building blocks that

Table 1. Asymptotic sizes of various UC constructions and improvements over previous works. Previous UCs were for 2-input gates, whereas our LUC construction is generalized to ρ-input LUTs.

Universal Circuit	Asymptotic Size	Improvement over previous work	Fanin
Valiant's 2-way [51]	$5n \log n$	-	2
Valiant's 4-way [51]	$4.75n \log n$	1.05×	2
Zhao et al.'s 4-way [57]	$4.5n \log n$	1.06×	2
Liu et al.'s 2-way [37]	$3n \log n$	1.5×	2
Our LUC	$\mathbf{1.5}\rho n \log n$	$\mathbf{1.12 - 2.18}\times$	ρ

can be used to construct more complex functionalities for common PFE applications: We compare the asymptotic circuit sizes when evaluating our new LUC construction with UCs for equivalent binary gates (cf. Table 2) and achieve size improvements of factor 1.67× for full adders, 2.67× for comparisons, and 2× for multiplexers.

Varying UC (VUC) Construction (Sect. 5). In several applications (cf. Sect. 1.1, Sect. 5.2) only the programming of the LUTs needs to be hidden, but not their dimensions, i.e., we can leak their number of in- and outputs. For this, we introduce *Varying Universal Circuits* (VUC) which are circuits that can simulate other LUT-based circuits while hiding their topology and the LUT programmings, but leak the LUTs' number of in- and outputs. We give the first VUC construction that eliminates the leading ρ factor of our LUC construction (cf. Table 1), while still maintaining its general design, i.e., we can transform all UC constructions to our new VUC construction.

Implementation (Sect. 6.1). We provide the first implementation of today's most efficient UC construction of Liu et al. [37] which is of independent interest and our LUC and VUC constructions.[2] Moreover, we integrate these three UC implementations into the MPC framework ABY [13] for PFE. To create LUT-based circuits, we used the hardware circuit synthesis tool Yosys-ABC [10,53] and Synopsis Design Compiler [5] for LUT-Mapping. We optimize LUT-based PFE by combining LUTs with overlapping inputs and multiple outputs. However, hardware synthesis tools do not by default support mapping to multiple output LUTs. To address this, we post-process the single-output LUT circuits produced by the synthesis tool to convert them to multi-output LUT circuits.

Evaluation (Sect. 6.3). We experimentally evaluate our LUC and VUC constructions for various LUT sizes, and compare them with the previous best construction of Liu et al. [37]. The asymptotic UC sizes and improvements over previous works are given in Table 4 for LUC and in Table 6 for VUC. Our new LUC constructions outperform the state-of-the-art UC [37] in terms of circuit sizes by up to 2.18×.

[2] Our code is published under the MIT license at: https://encrypto.de/code/LUC.

1.3 Related Work

Universal Circuits (UCs). Valiant [51] defined universal circuits, showed that they have a lower bound of size $\Omega(n \log n)$, and proposed two asymptotically size-optimal constructions using a 2-way or a 4-way recursive structure of sizes $\sim 5n \log n$ and $\sim 4.75n \log n$, respectively. Hence, relevant research challenges left are reducing the prefactor and the concrete UC sizes. Valiant's constructions can be generalized to simulate circuits composed of $(\rho \rightarrow 1)$-LUTs as shown by Sadeghi and Schneider [48, App. A] which is summarized in Sect. 4.1.

A modular UC construction of non-optimal size $\sim 1.5n \log^2 n + 2.5n \log n$ was proposed and implemented by Kolesnikov and Schneider [34]. Their construction beats Valiant's construction for small circuits thanks to small prefactors. Motivated to provide more efficient PFE, Kiss and Schneider [31] implemented Valiant's 2-way split construction and proposed a more efficient hybrid construction combining the 2-way split construction with the modular construction of [34]. Lipmaa et al. [36] generalized Valiant's construction to a k-way split construction and proved that the optimal value for k is 3.147, i.e., $k \in \{3, 4\}$ when k is an integer. Günther et al. [22] modularized Valiant's construction, implemented the more efficient 4-way split construction, gave a generic edge-embedding algorithm for k-way split constructions, and showed that the 3-way split construction with Valiant's framework is less efficient than the 2-way split construction. Zhao et al. [57] improved Valiant's 4-way split construction to size $\sim 4.5n \log n$, which is today's most efficient asymptotic size for UCs in Valiant's framework. Alhassan et al. [7] proposed and implemented a scalable hybrid UC construction combining Valiant's 2-way and 4-way split constructions with Zhao et al.'s improvements [57]. Most recently, Liu et al. [37] reduced redundancies in Valiant's framework and provided today's most efficient UC construction of size $\sim 3n \log n$ based on Valiant's 2-way split construction, showed that $k = 2$-way split is the most efficient in their new UC framework, and already almost reached their computed lower bound of $\sim 2.95n \log n$. We provide the first implementation of their construction and use it as a basis for our UC constructions for LUT-based circuits.

Private Function Evaluation (PFE). Katz and Malka [30] designed a constant-round two-party PFE protocol with linear communication complexity based on homomorphic public-key encryption. Holz et al. [24] optimized and implemented the protocol of [30], demonstrating its superiority over the hybrid UC implementation of Alhassan et al. [7] already for circuits with a few thousand gates. Liu et al. [38] provide a constant-round actively secure two-party PFE protocol with linear complexity. However, all these protocols are not generic and hence not directly compatible with arbitrary MPC frameworks, which makes them less flexible. For instance, these protocols cannot easily be extended to multiple parties. Ji et al. [25] demonstrated the evaluation of private RAM programs using four servers, building the first PFE of non-Boolean and non-arithmetic functions. In fact, recent PFE applications relied on so-called Semi-Private Function Evaluation (SPFE) where not necessarily the whole

function needs to be hidden from the other parties, but selected parts of the function can be leaked. The first SPFE construction and implementation was proposed by Paus et al. [43] who provided several building blocks that can be programmed with one function out of a class of functions (e.g., ADD/SUB whose circuits have the same topology). Recently, Günther et al. [21] built an SPFE framework that allows to split the function into public and private components, embed the private components into UCs, and merge them into one Boolean circuit that is evaluated via MPC. They demonstrated their framework on computing car insurance tariffs and observed that some information of the function is public, e.g., that experienced drivers usually get discounts.

MPC on LUTs. In the area of secure multi-party computation (MPC), prior work noticed that 2-input/1-output gates can be extended into multi-input/multi-output gates to reduce the circuit evaluation overhead [14,23,39,41,45]. In Yao's Garbled Circuit (GC) setting, Fairplay [39] implemented MPC protocols to evaluate gates with up to 3-input gates. The TASTY framework [23] implemented ρ-input garbled gates using the garbled row reduction optimization [44]. Recently, [45] proposed an MPC protocol that works on circuits with multi-input/multi-output gates instead of working on circuits with 2-input gates. Another line of work in the secret-sharing setting aims to optimize the rounds and communication of the online phase without using Yao's GC protocol: [14] extended 2-input AND gates to the general N-input case using LUTs. Recently, ABY2.0 [41] extended AND gates from the 2-input to the multi-input setting with a constant online communication complexity at the cost of exponential offline communication in the number of inputs. In addition, Syncirc [42] handles the circuit generation with multi-input gates by using industry-grade hardware synthesis tools [10,53].

2 Preliminaries

We refer to the size of a circuit n as the sum of its number of inputs n_i, gates n_g, and outputs n_o: $n = n_i + n_g + n_o$.

Definition 1 (Universal Circuit [7,51]). *A Universal Circuit $\mathcal{U}$ for n_i inputs, n_g gates, and n_o outputs is a Boolean circuit that can be programmed to compute any Boolean circuit C with n_i inputs, n_g gates, and n_o outputs by defining programming bits p^C such that $\mathcal{U}(x, p^C) = C(x)$ for any input $x \in \{0,1\}^{n_i}$.*

2.1 Graph Theory

Let $G = (V, E)$ be a directed graph and $v \in V$. The indegree (resp. outdegree) of v which is the number of incoming (resp. outgoing) edges is denoted by $\deg^+(v)$ (resp. $\deg^-(v)$). G has fanin (resp. fanout) ρ if $\deg^+(v) \leq \rho$ (resp. $\deg^-(v) \leq \rho$) for all $v \in V$. We denote by $\Gamma_\rho(n)$ all directed acyclic graphs with at most n nodes and fanin/fanout ρ for $\rho, n \in \mathbb{N}$. For $U \subset V$, $G[U] \coloneqq \{U, \{e = (u, v) \in$

$E : u, v \in U\}\}$ denotes the subgraph induced by U. We omit the index G in the above definitions if G is clear from the context.

Let $G = (V, E) \in \Gamma_\rho(n)$. A topological order for G is a map $\eta_G \colon V \to \{1, ..., |V|\}$ such that $\forall (u, v) \in E : \eta_G(u) < \eta_G(v)$.

We represent Boolean circuits as directed acyclic graph $G \in \Gamma_\rho(n)$ for some $\rho > 1$. However, almost all previous works [22,31,36,37,51,57] restricted the circuits, that are simulated via UCs, to fanin/fanout $\rho = 2$. The reason for this restriction can be found in the structure of universal circuits according to Valiant's [51] and Liu et al.'s [37] constructions. On a high level, a universal circuit (UC) for simulating circuits $C \in \Gamma_\rho(n)$ is composed of ρ so-called *Edge-Universal Graphs* (EUGs) each of size $\mathcal{O}(n \log n)$, i.e., the total size of the UC grows linearly with the maximum fanin/fanout ρ of the gates in the simulated circuit C.

Definition 2 (Edge-Embedding [7,36,37,51]). *Let $G = (V, E)$ and $G' = (P, E')$ be directed graphs with $P \subset V$ and G' acyclic. An edge-embedding from G' into G is a map $\psi \colon E' \to \mathcal{P}_G$, where $\mathcal{P}_G$ denotes the set of all paths in G, with the following properties:*

- *$\psi(e')$ is a u-v-path (in G) for all $e' = (u, v) \in E'$,*
- *$\psi(e')$ and $\psi(\tilde{e}')$ are edge-disjoint paths for all $e', \tilde{e}' \in E'$ with $e' \neq \tilde{e}'$.*

Definition 3 (Edge-Universal Graph [7,36,37,51]). *A directed graph $G = (V, E)$, denoted as $\mathcal{U}_\rho(n)$ with ordered pole set $P \coloneqq \{p_1, ..., p_n\} \subset V$ is called an Edge-Universal Graph for $\Gamma_\rho(n)$ if:*

- *G is acyclic,*
- *Every acyclic $G' = (P, E') \in \Gamma_\rho(n)$ that is order-preserving, i.e., $\forall e = (p_i, p_j) \in E' \Rightarrow i < j$, can be edge-embedded into G.*

On a high level, the graph $G' = (P, E')$ in Definitions 2 and 3 represents a Boolean function that is embedded into the graph $G = (V, E)$, which represents the UC, where $P \subset V$ is the pole set of size $|P| = n$, which represents the inputs, gates, and outputs of the function represented in G'. As an EUG requires that every $G' \in \Gamma_\rho(n)$ can be edge-embedded into G, the UC built by the EUG can compute any function represented by a graph in the set $\Gamma_\rho(n)$.

EUGs for $\Gamma_2(n)$ graphs were constructed by merging two EUGs for $\Gamma_1(n)$ graphs (cf. Definition 4 and Fig. 1) [7,22,31,36,37,51,57]. Thus, research focused on minimizing the size of general EUGs for $\Gamma_1(n)$ graphs as these can be merged to EUGs for arbitrary $\Gamma_\rho(n)$ graphs by merging ρ instances of $\Gamma_1(n)$ EUGs (cf. Corollary 1).

Definition 4 (Merging of EUG). *Let $G = (V, E)$ and $\bar{G} = (\bar{V}, \bar{E})$ be two EUG for $\Gamma_\rho(n)$ and $\Gamma_{\bar{\rho}}(n)$ with the same pole order and $V \cap \bar{V} = P$. Then $\hat{G} = (V \cup \bar{V}, E \cup \bar{E})$ is called the merging of G and $\bar{G}$ with pole set P.*

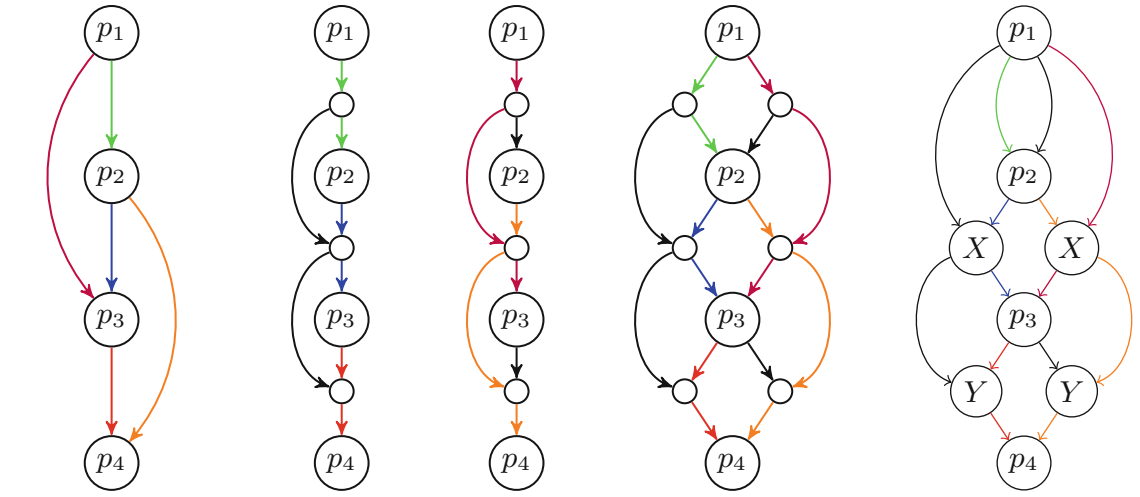
Proposition 1. *The merging of a $\Gamma_\rho(n)$ and a $\Gamma_{\bar{\rho}}(n)$ EUG is a $\Gamma_{\rho+\bar{\rho}}(n)$ EUG.*

We prove Proposition 1 in Appendix A of the full version [15].

Corollary 1 ([51, Corollary 2.2]). *An EUG for $\Gamma_\rho(n)$ can be constructed by merging ρ EUGs for $\Gamma_1(n)$.*

Proof. Let $G = (V, E)$ be a $\Gamma_1(n)$ EUG with pole set P. Create $\rho - 1$ copies of G with the same pole set and merge these graphs successively. Correctness follows directly by applying Proposition 1 ρ times. □

We call the UCs that are constructed according to Corollary 1 *LUT-based UCs* (LUCs) and this construction was first mentioned in [48, App. A]. In Sect. 5, we introduce our so-called *Varying UC (VUC) construction* that is constructed by two instances of $\Gamma_1(n)$ EUGs but still allows to edge-embed graphs with arbitrary fanin ρ.



(a) $\Gamma_2(4)$ graph (b) $\mathcal{U}_1(4)$ (c) $\mathcal{U}_1(4)$ (d) merged $\mathcal{U}_2(4)$ (e) UC

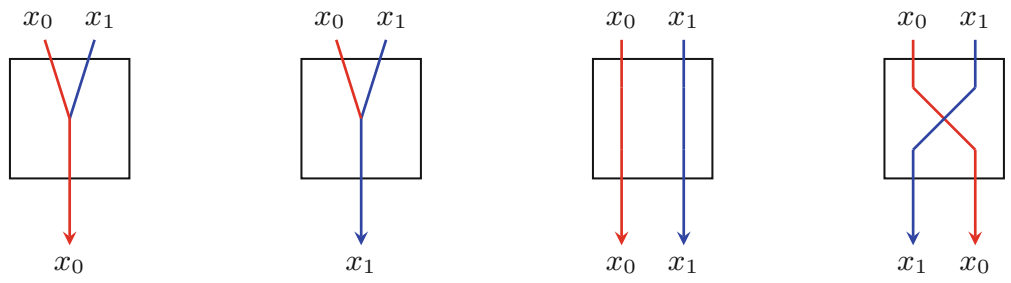
Fig. 1. (a) shows the $\Gamma_2(4)$ graph with already partitioned edge sets E_1 and E_2, (b) and (c) show the EUGs in which the edge sets E_1 resp. E_2 are embedded, (d) shows the merged EUG with all edges embedded, (e) shows the resulting UC, where p_1 is an input, and p_2, p_3, p_4 are translated to universal gates.

2.2 Building Universal Circuits from Edge-Universal Graphs

Boolean Circuits. A Boolean circuit is a directed acyclic graph whose nodes are Boolean inputs, (binary) gates, and outputs, with directed edges representing the wires. A Boolean gate is a function $z\colon \{0,1\}^k \to \{0,1\}$ for $k \in \mathbb{N}$. However, we can always divide a k-input gate into $\mathcal{O}(2^k)$ binary gates using Shannon's expansion theorem [49]. Unfortunately, we cannot avoid an exponential blow-up of the number of gates by this transformation [52, Theorem 2.1]. The two most prominent minimization methods for Boolean circuits are due to Karnaugh [29] and Quine-McCluskey [46]. As already mentioned, the UC constructions by Valiant [51] and Liu et al. [37] are designed to embed $\Gamma_\rho(n)$ graphs,

thus we possibly need to reduce the outdegree of the gates to ρ by using so-called copy gates which just copy their inputs [51, Corollary 3.1].[3]

From Edge-Universal Graphs to Universal Circuits. The translation from an EUG $G = (V, E)$ into a UC is depicted in Fig. 1 and works as follows. First, the nodes of circuit C to be embedded in G are considered as the poles $P \subset V$ of the EUG. A pole $p \in P$ is translated into an input or output wire, if p corresponds to an input or output in C, or into a so-called *Universal Gate*, if p corresponds to a gate in C. Universal gates take k inputs ($k = 2$ in the previous works [7,37,51]), 2^k programming bits, compute one output, and can be programmed to simulate any k input Boolean gate by specifying the truth table with the programming bits. We can implement universal gates with a binary tree of $2^k - 1$ multiplexers (Y-switches) spanned over the 2^k programming bits, where the correct programming bit specified by the k inputs is forwarded to the output (more details in [7,37]).[4]



(a) Y-switch (b) X-switch

Fig. 2. Switching blocks with programming bit p (from [34]).

The remaining nodes in the set $V \setminus P$ are for connecting the routes between the poles. A node $v \in V \setminus P$ is translated as follows:

- if v has two incoming edges and one outgoing edge, it is translated into a multiplexer/Y-switch (cf. Fig. 2a). A multiplexer has two inputs x_0 and x_1 and a programming bit p and outputs one bit, namely x_p. It is implemented with 1 AND gate and 2 XOR gates [34].
- if v has two incoming edges and two outgoing edges, it is translated into an X-switch (cf. Fig. 2b). An X-switch has two inputs x_0 and x_1, one programming bit p and outputs two bits, namely (x_p, x_{1-p}). It is implemented with 1 AND gate and 3 XOR gates [34].

[3] Note that a Universal Circuit can also compute circuits with less than the specified number of inputs, gates, and outputs by using dummy values with no functionality.

[4] In Yao's garbled circuit protocol [54], the UC's universal gates can be implemented as garbled tables when the function holder takes over the garbling part.

– if v has one incoming wire, it is replaced by a single wire that connects all of the outgoing edges.

The programming bits of the nodes are derived from the edge-embedding.

3 UC Constructions

In this section, we summarize the general guidelines for constructing edge-universal graphs (Sect. 3.1), present the original idea of Valiant [51] (Sect. 3.2), and describe the state-of-the-art construction of Liu et al. [37] (Sect. 3.3).

3.1 General EUG Constructions

The strategy for building UCs via EUGs is to construct $\Gamma_1(n)$ EUGs of smallest size, merging ρ instances of these (cf. Corollary 1) to construct a $\Gamma_\rho(n)$ EUG ($\rho = 2$ for binary gates), and translating this EUG into a UC. Valiant [51] proposed the first two designs for $\Gamma_1(n)$ EUGs, today known as 2-way and 4-way constructions, having asymptotic sizes of $\sim 2.5n \log n$ and $\sim 2.375n \log n$. Recently, Liu et al. [37] extended Valiant's framework, simplified the construction, and achieved an EUG based on the 2-way approach of asymptotically optimal size of $\sim 1.5n \log n$, which almost reaches their computed lower bound of $\sim 1.475n \log n$. The concrete construction principle of both frameworks is the same.

Let us assume we aim to construct a $\Gamma_1(n)$ EUG $G = (V, E)$ for a circuit of size n with a k-way construction and pole set $P \subset V$. First, we put k distinguished poles from the set P into a block called *superpole* that has k inputs and k outputs. Within this superpole, we can route edge-disjointly between its inputs and poles, and between its poles and outputs. In total, we have $\lceil n/k \rceil$ superpoles built by the poles set P. The k inputs and outputs of each superpole then can be used as poles for k instances of a $\Gamma_1(\lceil n/k \rceil - 1)$ nested EUG, which on a high level allows to find edge-disjoint paths between the superpoles of G.[5]

More formally, a superpole shall be able to edge-embed any so-called *augmented k-way block* (similar to an augmented DAG in [37]). An augmented k-way block is a map that defines the routes between the inputs and poles of the superpole, and between poles and other poles and outputs.

Definition 5 (Augmented k-way Block). *An augmented k-way block $G = (V, E)$ for pole set P, superpole inputs I, and superpole outputs O is a directed graph such that*

– $V = P \cup I \cup O$, $P \cap I = P \cap O = \emptyset$ *and* $|I| = |O| = k$,
– $G[P] := (P, E^P)$ *has fanin/fanout 1,*
– $E = E^P \cup E^{io}$ *with* E^{io} *satisfying*

[5] We distinguish between EUGs and nested EUGs as the recursively constructed nested EUGs differ from its first EUG in Liu et al.'s construction [37].

- *(Soundness) Every* $e \in E^{io}$ *satisfies either* $e = (in, p)$ *or* $e = (p, out)$ *for* $p \in P, in \in I, out \in O$,
- *(Completeness) For every source (resp. sink)* $p \in P$*, there exists at most one* $in \in I$ *(resp.* $out \in O$*) such that* $(in, p) \in E^{io}$ *(resp.* $(p, out) \in E^{io}$*).*

The set of all augmented k*-way blocks for* P, I, O *is denoted by* $\mathcal{B}_k(P, I, O)$*.*

Definition 6 (k-way Superpole). *A* k*-way superpole* $SP(k)$ *is a tuple* $SP(k) = (G = (V, E), P, \mathcal{P}, \mathcal{I}, \mathcal{O})$ *with pole set* $P \subset V$*, with following conditions:*

- $P = \mathcal{P} \cup \mathcal{I} \cup \mathcal{O}$ *with* $|\mathcal{I}| = |\mathcal{O}| = k$ *and* $\mathcal{P} \cap \mathcal{I} = \mathcal{P} \cap \mathcal{O} = \emptyset$,
- G *can edge-embed every* $G' \in \mathcal{B}_k(\mathcal{P}, \mathcal{I}, \mathcal{O})$.

We denote the input recursion points $\mathcal{I}$ of a k-way superpole as $\{in_1, in_2, ..., in_k\}$ and the output recursion points $\mathcal{O}$ as $\{out_1, out_2, ..., out_k\}$. These nodes serve as the inputs and outputs to the superpole and will be the poles of the next recursion, i.e., of the next nested EUG. We neither require the sets $\mathcal{I}$ and $\mathcal{O}$ to be disjoint nor that the recursion points of different superpoles must be disjoint. In fact, Valiant [51] merges the output recursion points of the i-th superpole with the input recursion points of the $(i+1)$-th superpole. On a high level, a

Algorithm 1: `Valiant`(P, k)

Input : Poles $P := \{p_1, ..., p_n\}$, split parameter k
Output : $\Gamma_1(n)$ EUG $G = (V, E)$, pole set P, sub-graphs $G^*, G^1, ..., G^k$

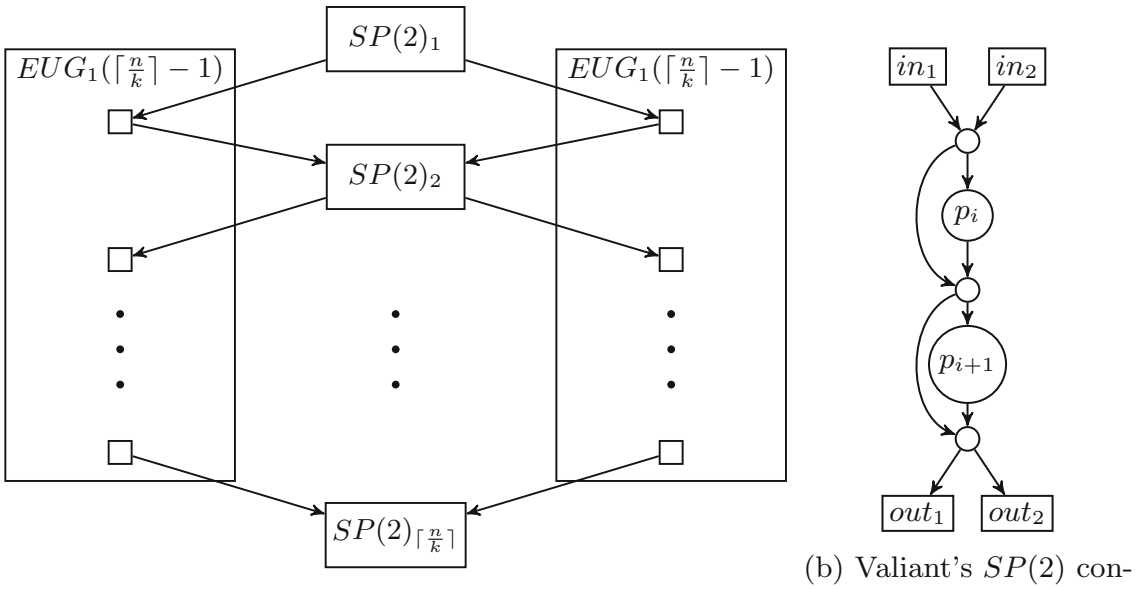
$V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
$\mathcal{O}_0 \leftarrow$ *create* k *dummy nodes*
for $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$ **do**
  $\mathcal{P}_i \leftarrow \{p_{k(i-1)+1}, ..., p_{ki}\}$
  `// Use` $\mathcal{O}_{i-1}$ `as input recursion points to this superpole (cf. Fig. 3b)`
  $SP(k)_i = (G_i = (V_i, E_i), P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i) \leftarrow$ `Createsuperpole`$(\mathcal{P}_i, \mathcal{O}_{i-1}, k)$; $G^* \leftarrow G^* \cup \{G_i\}$
  $V \leftarrow V \cup V_i, E \leftarrow E \cup E_i$
for $i \leftarrow 1$ **to** k **do**
  if $n \leq k$ **then**
    $G^i \leftarrow (\emptyset, ..., \emptyset)$ `// Recursion base`
  else
    `// Take the` i`-th output recursion point of each superpole (but the last) as the poles for the next sub EUG`
    $P^i \leftarrow \{\mathcal{O}_1[i], \mathcal{O}_2[i], ..., \mathcal{O}_{\lceil \frac{n}{k} \rceil - 1}[i]\}$
    $(G^i = (V^i, E^i), ...) \leftarrow$ `Valiant`(P^i, k)
    $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
return $G = (V, E), P, G^*, G^1, ..., G^k$

superpole in a nested EUG U^1, i.e., an EUG that is derived as a recursion from a larger EUG U, has k entry points to an input of k distinguished superpoles in U as well as k exit points from an output of k distinguished superpoles.

3.2 Valiant's EUG Construction [51]

Definition 7 (Valiant EUG). *A Valiant EUG $G = (V, E)$ with pole set $P \subset V$ and sub-graphs $G^*, G^1, ..., G^k$ is created by Algorithm 1 (*`Valiant`*). We also use the notation* $\texttt{Valiant}_k(n)$ *for a Valiant EUG with n poles and split parameter k.*

Valiant's k-way EUG construction is built recursively as depicted in Fig. 3a. A $\Gamma_1(n)$ EUG is a chain of $\lceil n/k \rceil$ superpoles $SP(k)_1 = (G_1 = (V_1, E_1), P_1, \mathcal{P}_1, \mathcal{I}_1, \mathcal{O}_1), \ldots, SP(k)_{\lceil n/k \rceil} = (G_{\lceil n/k \rceil} = (V_{\lceil n/k \rceil}, E_{\lceil n/k \rceil}), P_{\lceil n/k \rceil}, \mathcal{P}_{\lceil n/k \rceil}, \mathcal{I}_{\lceil n/k \rceil}, \mathcal{O}_{\lceil n/k \rceil})$ (lines 3–6 in Algorithm 1). $\texttt{Createsuperpole}(P, \mathcal{O}, k)$ creates a superpole with poles P, input recursion points $\mathcal{O}$, and split parameter k, e.g., Valiant's $k = 2$-way superpole $SP(2)$ (Fig. 3b). The sets $\mathcal{O}_1, \ldots, \mathcal{O}_{\lceil n/k-1 \rceil}$, each of size k, then recursively build the poles of the nested EUGs in the next recursion step (lines 7–13 in Algorithm 1), i.e., we build k nested EUGs $G^1 = (V^1, E^1), \ldots, G^k = (V^k, E^k)$ with pole sets $P^1, \ldots, P^k$, where $G^i \in \Gamma_1(\lceil n/k \rceil - 1)$ and $P^i = (\mathcal{O}_1[i], \ldots, \mathcal{O}_{\lceil n/k-1 \rceil}[i])$. Note that $\mathcal{I}_i := \mathcal{O}_{i-1}$ for all $1 < i \leq \lceil n/k \rceil$ as the k outputs of $G_i \in SP(k)_i$ are pairwise merged with the respective k inputs of $G_{i+1} \in SP(k)_{i+1}$. The creation of the first output recursion points $\mathcal{O}_0$ is a technical trick, and not needed because these nodes will never be used, but it simplifies the definition of the algorithm by avoiding a case distinction. An



(a) Valiant's $EUG_1(n)$ construction.

(b) Valiant's $SP(2)$ construction.

Fig. 3. (a) shows Valiant's 2-way split construction of $EUG_1(n)$ using two instances of $EUG_1(\lceil \frac{n}{k} \rceil - 1)$. (b) shows the corresponding superpole $SP(2)$ construction for the EUG.

advantage of this recursive method is that we can also recursively reduce the edge-embedding problem to finding paths between poles of the nested EUGs. Assuming we can easily edge-embed paths from inputs to poles and from poles to outputs within the superpoles, we can reduce finding a path from a pole located in $SP(k)_i$ to a pole in $SP(k)_j$ to the problem of finding a path from $\mathcal{O}_i[x]$ to $\mathcal{O}_{j-1}[x]$ for $i, j \in [\lceil n/k \rceil]$, $i < j$, where x is the index of the target output of the superpoles' internal edge-embedding for the concrete poles. Existing UC implementations [7,22] split the edge-embedding into two sub-tasks: (a) the superpole edge-embedding that takes care that the paths within a superpole are defined in a correct manner, and (b) the recursion-point edge-embedding which chooses the correct paths at the recursion points. We define the following theorem and refer to [7,37] for its proof:

Theorem 1. *Let $G = (V, E)$ be a Valiant EUG with pole set $P \subset V$ of size $|P| = n$ and sub-graphs $G^*, G^1, ..., G^k$. Then G is an EUG for $\Gamma_1(n)$.*

3.3 Liu et al.'s EUG Construction [37]

Definition 8 (Liu$^+$ EUG). *A Liu$^+$ EUG $G = (V, E)$ with pole set $P \subset V$ and sub-graphs $G^*, G^1, \ldots, G^k$ is created by Algorithm 2 (`Liu`$^+$). We also use the notation $\texttt{Liu}_k^+(n)$ for a Liu$^+$ EUG with n poles and split parameter k.*

We refer to Appendix B of the full version [15] for a complete description of the construction of Liu et al. [37] including Algorithm 2. In the subsequent sections of this work, we leverage the following theorem and refer to [37] for its proof:

Theorem 2 cf. [37, Theorem 4]). *Let $G = (V, E)$ be a Liu$^+$ EUG with pole set $P \subset V$ of size $|P| = n$ and sub-graphs $G^*, G^1, ..., G^k$. Then G is an EUG for $\Gamma_1(n)$ with size bounded by*

$$\frac{|SP(k)| - k}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n).$$

4 Evaluating LUTs with UCs

In this section, we extend the UC constructions from Sect. 3 to be able to simulate $(\rho \to \omega)$-LUT-based circuits. In Sect. 4.1, we first review the construction of [34,51] to evaluate $(\rho \to 1)$-LUT-based circuits, i.e., circuits that consist of LUTs with ρ inputs and one output. Then, in Sect. 4.2, we extend this to our LUT-based UCs (LUCs) that allows the UC to simulate $(\rho \to \omega)$-LUT-based circuits. Finally, in Sect. 4.3, we analyze the most important building blocks for PFE applications, describe how to implement them with LUTs, and show their theoretical improvement over evaluating the same building blocks with Boolean circuits.

4.1 UCs for LUTs with Multiple Inputs [48,51]

Valiant [51] proposed a method to integrate LUTs with more than two inputs into UCs and its size has been computed in [48].

We can get a UC with n copies of $(\rho \to 1)$-LUT from a $\Gamma_\rho(n)$ EUG that is merged by ρ instances of $\Gamma_1(n)$ EUGs according to Corollary 1. Each pole of U that is not an input or an output can then be implemented as a LUT with ρ inputs.

Corollary 2. *An EUG for $\Gamma_\rho(n)$ for $\rho \in \mathbb{N}_{\geq 2}$ can be constructed with size at most $1.5\rho n \log_2(n) + \mathcal{O}(n)$.*

Proof. Construct ρ instances of $\texttt{Liu}_2^+(n)$ and merge them. By Corollary 1, this yields an EUG for $\Gamma_\rho(n)$ with size bounded by $1.5\rho n \log_2(n) + \mathcal{O}(n)$. □

4.2 UCs for LUTs with Multiple In- and Outputs

In order to support $(\rho \to \omega)$-LUTs with $\omega > 1$ outputs in UCs, we propose a general solution that is compatible with the original UC constructions of Valiant [51] and Liu et al. [37]. The high level idea is as follows: For every $(\rho \to \omega)$-LUT that is represented by pole v_i, we add $\omega - 1$ so-called *auxiliary* poles to the EUG and the real pole v_i forwards its inputs directly to these auxiliary poles. The real pole and its auxiliary poles each compute and output one of the LUT's output. Concretely, the first pole takes the ρ inputs of the LUT using any of the above UC constructions and computes the first output of the LUT. The remaining poles copy the ρ inputs of the first poles by direct connections and compute the remaining outputs of the LUT, resulting in a chain of ω poles.

We define the class of $\Gamma_{\rho,\omega}(n)$ graphs that is used to map n $(\rho \to \omega)$-LUTs to a graph $G \in \Gamma_{\rho,\omega}(n)$. As the poles of the EUG are the nodes of G, we need to add for each additional output of the i-th LUT (denoted as pole $v_{i,1}$ in G) in total $\omega - 1$ additional poles (denoted as $v_{i,2}, \ldots, v_{i,\omega}$). These added poles $v_{i,j>1}$ use the inputs from pole $v_{i,1}$ and thus, they all have in-degree 0 (cf. condition 3 in Definition 9). We define $\Gamma_{\rho,\omega}(n)$ as follows:

Definition 9 ($\Gamma_{\rho,\omega}(n)$). *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V \coloneqq \{v_{1,1}, \ldots, v_{1,\omega}, v_{2,1}, \ldots, v_{2,\omega}, \ldots, v_{n,1}, \ldots, v_{n,\omega}\}$ and $\rho, \omega \in \mathbb{N}$. Then $G \in \Gamma_{\rho,\omega}(n)$ if:*

- $|V| \leq n\omega$,
- $|\{v_{i,j} \in V\}| \leq \omega \ \forall i \in [n]$,
- $deg^+(v_{i,1}) \leq \rho \wedge deg^+(v_{i,2}) = \cdots = deg^+(v_{i,\omega}) = 0$,
- $deg^-(v_{i,j}) \leq \rho \ \forall i \in [n] \ \forall j \in [\omega]$.

To easily build an EUG with only marginal modifications, we show that $\Gamma_{\rho,\omega}(n)$ is also a $\Gamma_\rho(n\omega)$ graph:

Proposition 2. *Let $G \in \Gamma_{\rho,\omega}(n)$. Then $G \in \Gamma_\rho(n\omega)$.*

Algorithm 2: $\texttt{Liu}^+(P,k)$

Input : Poles $P := \{p_1, ..., p_n\}$, split parameter k
Output : $\Gamma_1(n)$ EUG $G = (V, E)$, pole set P, sub-graphs $G^*, G^1, ..., G^k$

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $\lceil \frac{n}{k} \rceil$ **do**
  $\mathcal{P}^i \leftarrow \{p_{k(i-1)+1}, ..., p_{ki}\}$
  $SP(k)_i = (G_i = (V_i, E_i), P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i) \leftarrow \texttt{Createsuperpole}(\mathcal{P}_i, k)$
  $G^* \leftarrow G^* \cup \{G_i\}$
  $V \leftarrow V \cup V_i, E \leftarrow E \cup E_i$
for $i \leftarrow 1$ **to** k **do**
  if $n \leq k$ **then**
    $G^i \leftarrow (\emptyset, ..., \emptyset)$ `// Recursion base`
  else
    `// Take the` i`-th output recursion point of each superpole as the poles for the next sub EUG`
    $P^i \leftarrow \{\mathcal{O}_1[i], \mathcal{O}_2[i], ..., \mathcal{O}_{\lceil \frac{n}{k} \rceil - 1}[i], \mathcal{O}_{\lceil \frac{n}{k} \rceil}[i]\}$
    $(G^i = (V^i, E^i), ...) \leftarrow \texttt{Liu}^+(P^i, k)$
    $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
foreach $(u, v) \in E$ **do**
  if *$u \in s$ and v is recursion point for some superpole $s \in G^*$* **then**
    $G^x \leftarrow$ *the EUG in which v is a pole*
    $E \leftarrow E \setminus \{(u, v)\}$
    $w \leftarrow \Gamma^-_{G^x}(v)$
    $E \leftarrow E \setminus \{(v, w)\}$
    $E \leftarrow E \cup \{(u, w)\}$
  else if *u is recursion point for some superpole $s \in G^*$ and $v \in s$* **then**
    $G^x \leftarrow$ *the EUG in which u is a pole*
    $E \leftarrow E \setminus \{(u, v)\}$
    $w \leftarrow \Gamma^+_{G^x}(u)$
    $E \leftarrow E \setminus \{(w, u)\}$
    $E \leftarrow E \cup \{(w, v)\}$
remove all recursion points from V
return $G = (V, E), P, G^*, G^1, \ldots, G^k$

Proof. Let $G = (V, E) \in \Gamma_{\rho,\omega}(n)$. Obviously, it holds that $|V| \leq n\omega$ (condition 1 in Definition 9). Further, for all $v \in V$ it holds that $\deg^+(v) \leq \rho$ and $\deg^-(v_{i,j}) \leq \rho$ from conditions 3 and 4 in Definition 9. Thus, $G \in \Gamma_\rho(n\omega)$. □

Now, we can build EUGs for multi-input and multi-output LUTs.

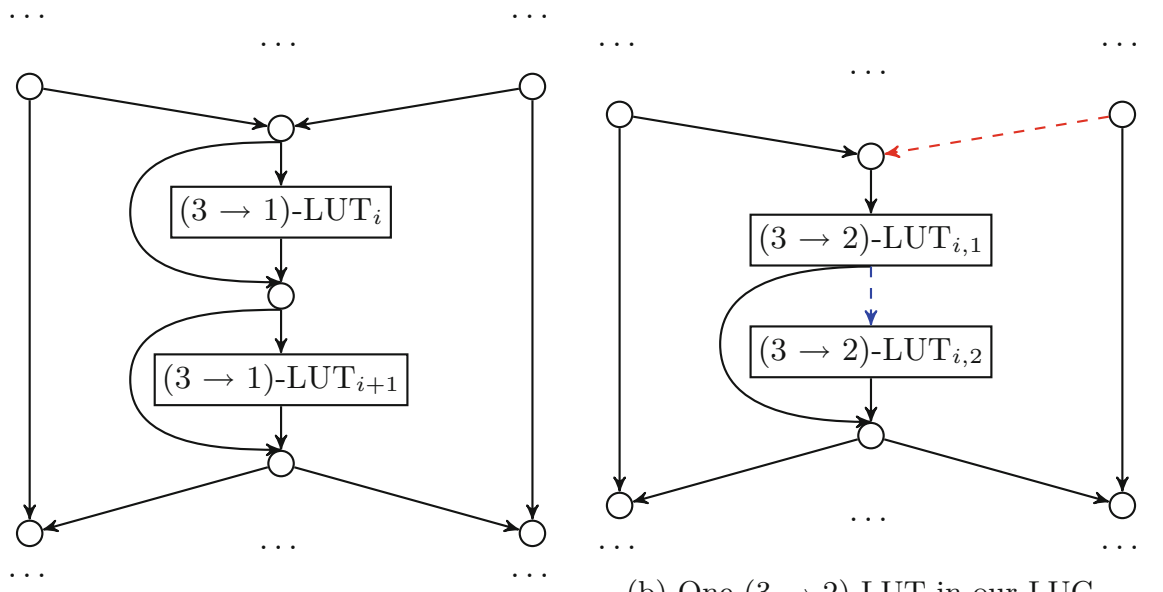
Corollary 3. *Let $\rho, \omega \in \mathbb{N}$. Then there exists a EUG for $\Gamma_{\rho,\omega}(n)$ with size bounded by*

$$1.5\rho n\omega \log_2(n\omega) + \mathcal{O}(n\omega).$$

Proof. **Step 1:** Create a $\Gamma_\rho(n\omega)$ EUG $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}})$ with a topologically ordered pole set $P \subset V^{\mathcal{U}}$ that has the form $(..., v_{i-1,\omega}, v_{i,1}, ..., v_{i,\omega}, v_{i+1,1}, ...)$ for all $i \in [n]$, i.e., the original pole $v_{i,1}$ directly preceding the auxiliary poles $v_{i,j}$ for $1 < j \leq \omega$: We do this by creating a Liu$^+$ EUG $\mathcal{U}$ with pole set P and split parameter $k = 2$. Then we merge ρ instances of it. By Theorem 2 with $|SP(2)| = 5$ [37] and Corollary 1, this yields a $\Gamma_\rho(n\omega)$ EUG of size at most $1.5\rho n\omega \log_2(n\omega) + \mathcal{O}(n\omega)$.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$ with pole set $P \subset V^{\bar{\mathcal{U}}}$: Let $v_{i,j}$ be an auxiliary pole of $v_{i,1}$ for $i \in [n], 1 < j \leq \omega$. Remove all of its incoming edges and replace each of them with an edge connecting the original pole $v_{i,1}$ with the auxiliary pole $v_{i,j}$, i.e., remove $(w, v_{i,j}) \in E^{\mathcal{U}}$ for $w \in V^{\mathcal{U}}$ and replace it by $(v_{i,1}, v_{i,j})$. This yields ρ edges $(v_{i,1}, v_{i,j})$ per auxiliary pole $v_{i,j}$ (one for each EUG instance). Thus, $E^{\mathcal{U}}$ becomes a multi set. The graph that results from modifying $\mathcal{U}$ in the just described way is denoted by $\bar{\mathcal{U}}$ and its pole set is denoted by P.

Step 3: Embed any graph $G = (P, E) \in \Gamma_{\rho,\omega}(n)$ into $\bar{\mathcal{U}}$: To show that $\bar{\mathcal{U}}$ is a EUG for $\Gamma_{\rho,\omega}(n)$, we need to define an edge-embedding ψ from G into $\bar{\mathcal{U}}$. Thanks to Proposition 2, it holds that $G \in \Gamma_\rho(n\omega)$. Note that the "relative topological order" is maintained, i.e., $\eta_G(v_i) < \eta_G(v_{i+1})$ for $i \in [n]$. However, although $\bar{\mathcal{U}}$ has $n\omega$ poles, it is not an EUG for all $\Gamma_\rho(n\omega)$ graphs as all poles $v_{i,j>1}$ are directly connected to pole $v_{i,1}$ via the edge $(v_{i,1}, v_{i,j>1})$ for $i \in [n], j \in [\omega]$. Thus,



(a) Two (3 → 1)-LUTs in our LUC.

(b) One (3 → 2)-LUT in our LUC.

Fig. 4. Embedding of (3 → 1)-LUTs (a) and (3 → 2)-LUTs (b) in a single superpole of our LUC construction. The blue line in (b) indicates that the inputs of the first LUT part are forwarded to the second LUT part. Each of the LUT parts in (b) generate one output with the same inputs, thus building together a (3 → 2)-LUT. The red edge is optional and can be removed as only one input in the superpole is needed.

we cannot find edge-disjoint paths from any pole $v_{k<i,l}$ to $v_{i,j>1}$ for $k \in [n]$, $l \in [\omega]$, as these would all use an ingoing edge of pole $v_{i,1}$. So, we need to show that all nodes $v_{i,j>1} \in G$ have indegree 0 to ensure that no edge-disjoint path needs to end up at pole $v_{i,j>1} \in \bar{\mathcal{U}}$. This, however, is fulfilled due to condition 3 of Definition 9, i.e., there exists no edge $e = (v_{k<i,l}, v_{i,j>1}) \in G$ for which an embedding $\psi(e)$ needs to be defined (the same argument holds for edges $e = (v_{i,l<j}, v_{i,j}) \in G$).

So far, we showed that G only contains edges $e = (v_{k<i,l}, v_{i,1}) \in G$, which are the only ones to edge-embed into $\bar{\mathcal{U}}$. However, as we just added additional edges to poles $v_{i,1}$ and no outgoing edges from any poles in $\bar{\mathcal{U}}$ have been removed, we can get the edge-embedding ψ directly from Corollary 2. □

In Fig. 4, we present our LUC construction for the embedding of both $(3 \rightarrow 1)$-LUTs and $(3 \rightarrow 2)$-LUTs within a single superpole. Specifically, in Fig. 4a, a superpole consists of two $(3 \rightarrow 1)$-LUTs, each having three individual inputs and one output. In contrast, in Fig. 4b, a $(3 \rightarrow 2)$-LUT requires two poles, limiting the embedding capacity to a single $(3 \rightarrow 2)$-LUT within one superpole. We achieve this by implementing each pole as a $(3 \rightarrow 1)$-LUT in our LUC construction, effectively combining them to form a $(3 \rightarrow 2)$-LUT. The second part of the LUT shares the same inputs as the first part (indicated by the blue edge in Fig. 4b), eliminating the need for an additional node between the two poles. The two outputs of the $(3 \rightarrow 2)$-LUT are forwarded to the lower node and can then propagate to the nested EUGs through this node. As an optimization, we can remove one incoming edge from the superpole (indicated by the red edge in Fig. 4b) since only one outer input is utilized.

4.3 Improvement

In this section, we show improvements of our LUC for several basic building blocks like full adder (FA), comparator (CMP), and multiplexer (MUX). As summarized in Table 2, our basic building blocks are smaller than the previous constructions [7,22,31,37] in UC size by factor $\approx 1.67\times$–$2.67\times$. Note that we compute improvement factors based only on the prefactor. The actual enhancements will be greater as also the logarithmic term is improved. This UC size reduction is achieved by merging 2-input gates into larger multi-input LUTs.

Full Adder (FA): The optimized implementation of a FA uses four 2-input XOR gates and one 2-input AND gate (cf. [33, Fig. 2]). We can implement a FA using only one $(3 \rightarrow 2)$-LUT, resulting in an improvement by $\approx 1.67\times$ in LUC size (cf. Table 2). The embedding of a $(3 \rightarrow 2)$-LUT in our LUC is depicted in Fig. 4b.

Comparator (CMP): The 1-bit comparator consists of three 2-input XOR gates and one 2-input AND gate (cf. [33, Fig. 6]). Our improved LUT-based instantiating for CMP uses only one $(3 \rightarrow 1)$-LUT, resulting in an improvement of $\approx 2.67\times$ in LUC size (cf. Table 2). The embedding of a $(3 \rightarrow 1)$-LUT in our LUC is depicted in Fig. 4a.

Multiplexer (MUX): The MUX block can be instantiated with two 2-input XOR gates and one 2-input AND gate (cf. [35, Fig. 2]). In our approach, MUX can be instantiated with only one $(3 \to 1)$-LUT, resulting in an improvement of $\approx 2\times$ in the LUC size (cf. Table 2). The embedding of a $(3 \to 1)$-LUT in our LUC is depicted in Fig. 4a.

Complex Building Blocks. We now present several motivating examples that benefit from improvements of our basic building blocks.

Addition and Subtraction. An l-bit addition is composed of a chain of l Full Adders (FA) (cf. [33, Fig. 1]. An l-bit subtraction is defined as $x - y = x + y + 1$ and can be constructed similarly to an addition circuit using l FAs (cf. [33, Fig. 3]. Using our FA construction, the LUC size of the addition and subtraction is improved by $\approx 1.67\times$.

Table 2. LUC sizes for basic building blocks which can be used to construct more complex functionalities. b denotes the frequency of occurrence of the specific building blocks within the circuit.

Building Block (BB)	Boolean Circuit		LUT-based Circuit		Improvement
	# Gates	Asympt. UC Size	LUT type	Asympt. LUC Size	
FA	4 XOR 1 AND	$15b \log_2 5b + \mathcal{O}(b)$	$(3 \to 2)$-LUT	$9b \log_2 b + \mathcal{O}(b)$	$1.67\times$
CMP	3 XOR 1 AND	$12b \log_2 4b + \mathcal{O}(b)$	$(3 \to 1)$-LUT	$4.5b \log_2 b + \mathcal{O}(b)$	$2.67\times$
MUX	2 XOR 1 AND	$9b \log_2 3b + \mathcal{O}(b)$	$(3 \to 1)$-LUT	$4.5b \log_2 b + \mathcal{O}(b)$	$2\times$

Multiplication. Multiplication of two l-bit numbers can be composed of l^2 of 2-input AND gates ($(2 \to 1)$-LUT) and $(l-1)$ l-bit adders [33]. Using the efficient implementation for LUT-based adders, the LUC size of the multiplication circuit is improved by $\approx 1.67\times$.

Multiplexer. An l-bit multiplexer circuit can be composed of l parallel MUX blocks (cf. [35, Fig. 9]) to select one of the l-bit inputs. So, using our LUT-based MUX has $\approx 2\times$ improvement for an l-bit multiplexer.

Comparison. An l-bit comparison circuit can be composed of a chain of l CMP blocks (cf. [33, Fig. 5]). Thus, our CMP construction improves the LUC size of the comparison circuit by $\approx 2.67\times$. A minimum circuit which selects the minimum value of a list of m l-bit values is composed of l-bit comparison and multiplexer circuits (cf. [33, Fig. 8]) and hence is improved by $\approx 2.3\times$.

5 Our Varying UC (VUC) Construction

In many applications, sub-functionalities are naturally implemented by LUTs with higher dimension, e.g., Sboxes in AES. In this case, we aim to put single LUTs with a higher dimension (e.g., $\rho = 8$) into the UC. Using our LUC construction for this concrete example, we would need to compose the UC of 8 instances of $\Gamma_1(n)$ EUGs, even if we only need the full $(8 \rightarrow 1)$-LUT few times in the whole circuit.[6] Thus, our aim is to find a way to use single LUTs with input dimension of $\rho > 3$ without a massive influence on the total circuit size.

In this section, we present our Varying UC (VUC) construction, which deviates from the conventional universal circuits (UCs) that have been widely studied [7,22,31,34,36,37,51]. Traditionally, UCs have been designed to conceal both the topology and the gate functionality of the simulated function, and have relied on the use of fixed computational units, namely universal 2-input gates or, like in our work, $(\rho \rightarrow \omega)$-LUTs with a globally fixed number of inputs ρ and outputs ω. A VUC, however, allows for the use of different programmable computational units, thereby leaking information about the types of units used. In particular, we focus on VUCs built using $(\rho \rightarrow \omega)$-LUTs with varying numbers of inputs and outputs, thereby revealing the dimensions of the individual LUTs.

Definition 10 (Varying Universal Circuit (VUC)). *A Varying Universal Circuit $\mathcal{V}$ for n_i inputs, the ordered list of n_g gates $\mathcal{G} = (\mathcal{G}_1, \ldots, \mathcal{G}_{n_g})$ of varying input and output dimensions, and n_o outputs is a Boolean circuit that can be programmed to compute any Boolean circuit C with n_i inputs, n_o outputs, and n_g gates that can be topologically ordered into $\mathcal{G}$ by defining a set of programming bits p^C such that $\mathcal{V}(x, p^C) = C(x)$ for all possible input values $x \in \{0,1\}^{n_i}$.*

In Sect. 5.2, we discuss several applications of VUCs as well as their leakage.

5.1 The VUC Construction

First, we show how to build our VUC for evaluating different $(\rho \rightarrow 1)$-LUTs with varying input dimensions ρ. Later in this section, we show how to extend this construction to evaluate any $(\rho \rightarrow \omega)$-LUTs with varying input and output dimensions ρ and ω. In our VUC construction, we keep building our UC from only two instances of a $\Gamma_1(n)$ EUG, independent of the LUT sizes. This reduces the overhead of our LUT-based UC construction that merges ρ instances of the

[6] An alternative would be to decompose the larger LUTs into multiple smaller ones using Shannon expansion [49].

large $\Gamma_1(n)$ EUG for $(\rho \to 1)$-LUT. We do this by adding auxiliary poles u to the EUG whose task is to collect up to two inputs and forward these inputs via direct edges to a real pole v to push the indegree of v to ρ. Definition 11 defines $\Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n)$ graphs, which classify the graphs that can be edge-embedded into our VUC construction, namely, the vectors P^+ and P^- specify the maximum indegree and outdegree of each LUT in our circuit that we aim to evaluate with the UC. Our VUC design additionally allows the evaluation of functions that only use a single type of ρ input LUTs by setting $\mathrm{P}^+ = \mathbb{1}\rho$,[7] i.e., each LUT in the circuit can have at most ρ inputs and the resulting VUC implements each universal gate as a $(\rho \to 1)$-LUT. In this case, the VUC is a real LUT-based UC (LUC) and can be used for PFE. In the case for VPFE, the universal gates of the UC have different implementations and therefore leak the specific input sizes of all LUTs.

Definition 11 ($\Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n)$)**.** *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V \coloneqq \{v_1, ..., v_n\}$ and $\mathrm{P}^+, \mathrm{P}^- \in \mathbb{N}^n$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n)$ if:*

- $|V| \leq n$,
- $deg^+(v_i) \leq \mathrm{P}^+_i \wedge deg^-(v_i) \leq \mathrm{P}^-_i \ \forall i \in [n]$.

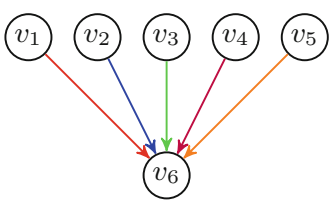
If $\mathrm{P}^{+/-} = \mathbb{1}\rho$ for some $\rho \in \mathbb{N}$, we write ρ instead of $\mathbb{1}\rho$.

In this sense, Corollary 2 yields a $\Gamma_{\rho,\rho}(n)$ EUG. In the following, we describe our VUC construction. An example of the whole EUG creation and the embedding process is depicted in Fig. 5. The explicit creation of the used auxiliary graph is given by Algorithm 3.

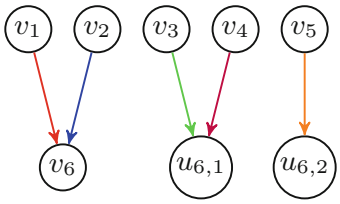
The key observation for our VUC construction is that, when merging two instances of $\Gamma_1(n)$ EUGs, each of the n poles (excluding inputs and outputs) can take two inputs, and *can*, but not necessarily need to, compute one output. We can use this observation to merge poles in order to collect $\rho > 2$ inputs for our LUT. For example, looking at Fig. 5, a $(5 \to 1)$-LUT consists of the three poles p_6, p_7, and p_8, where pole p_6 (resp. p_7) just collects two (resp. one) inputs, but does not compute any output. Instead, the ingoing edges are forwarded to pole p_8 (dashed lines) and the outgoing edges (dotted gray lines) are removed. Pole p_8 now has, in addition to its two regular ingoing edges, three additional ingoing edges that come directly from poles p_6 and p_7. On a high level, we can merge $\lceil \rho/2 \rceil$ poles into one $(\rho \to 1)$-LUT, while the first $\lceil \rho/2 \rceil - 1$ so-called *auxiliary poles* each collect up to two inputs for the LUT which are then directly forwarded to the last pole, which takes the last two inputs of the LUT and computes the output.

[7] $\mathbb{1}$ denotes the vector where each entry is 1.

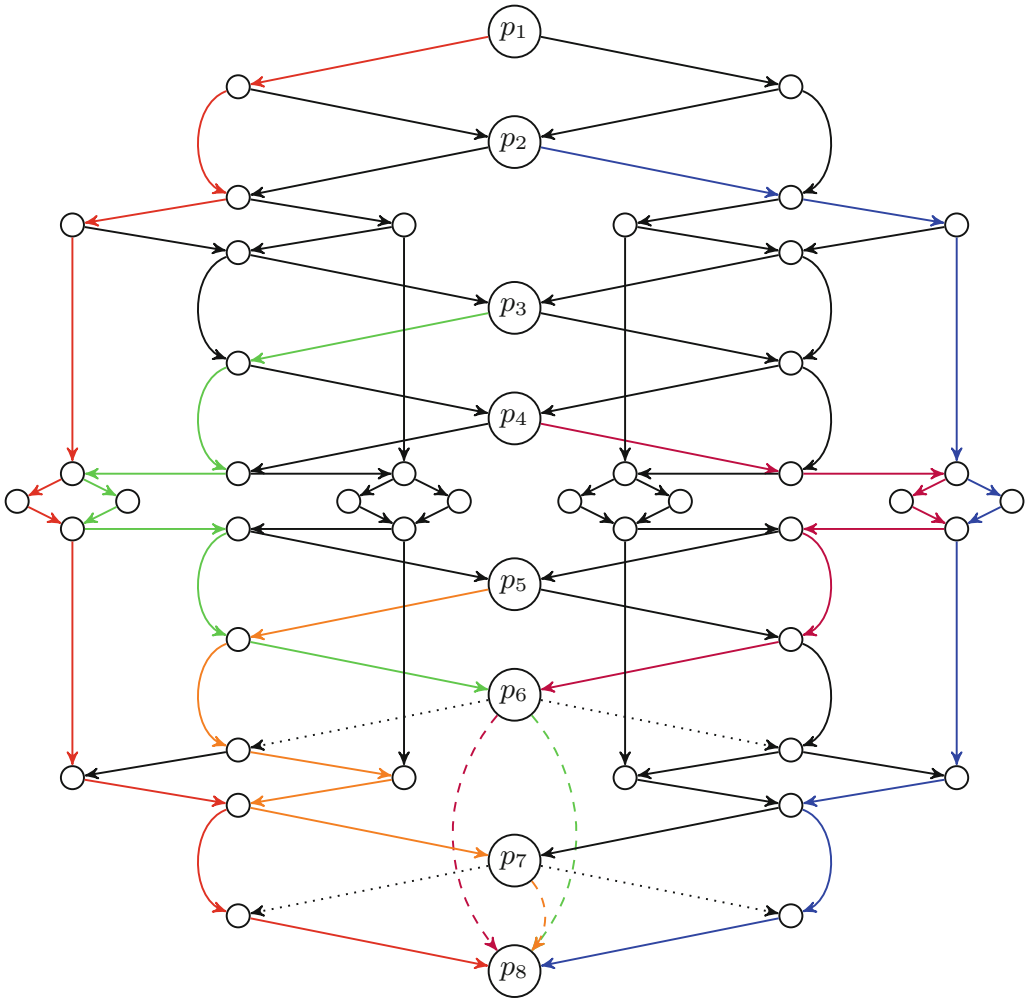
More formally, we begin by constructing an auxiliary graph $\bar{G}$. For each pole p that has $\rho > 2$ incoming edges, we create an auxiliary pole for each two additional inputs, i.e., $\lceil \rho/2 - 1 \rceil$ auxiliary poles. Then, we replace all except two edges from pole p by edges to the auxiliary poles. The purpose of the auxiliary poles is to forward their inputs to the original multi-input pole. The resulting



(a) Original graph



(b) Corresponding auxiliary graph



(c) Edge-embedding of the original graph. First, the edges from the auxiliary graph are embedded. Then, dotted gray edges are removed from the EUG, while dashed edges are added to the EUG, resp. to the edge-embedding. The result is an edge-embedding for the original graph. Now we can replace the ingoing edges to p_6 by directed edges to the multi-input pole p_8. The auxiliary pole p_7 becomes a Y-Switch that only forwards the orange wire.

Fig. 5. Our varying UC construction for $\rho = 5$ inputs.

Algorithm 3: `AuxiliaryGraph`(G)

Input : $G = (V, E) \in \Gamma_{\mathrm{P}^+,2}(n)$

Output : $\bar{G} = (\bar{V}, \bar{E}) \in \Gamma_2(n + \Delta)$ with $\Delta = \sum_{i=0}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$

```
1  Ḡ = (V̄, Ē) ← (V, ∅)
2  foreach v_i ∈ V do
3      j ← 0
4      foreach e = (w, v_i) ∈ E do
5          if j ≥ 2 then
6              if j ≡ 0 (mod 2) then
7                  V̄ ← V̄ ∪ {u_{i,j/2}}
8              Ē ← Ē ∪ {(w, u_{i,⌈j/2⌉})}
9          else
10             Ē ← Ē ∪ {e}
11         j ← j + 1
```

EUG $\mathcal{U}$ then guarantees that there can be a path from any pole with lower order to the corresponding auxiliary poles.

If there is a multi-input gate with an odd number of inputs ρ, then there will be one auxiliary pole in $\bar{G}$ with only one input. In this case, we can share this auxiliary pole for two poles if both have an odd number of inputs (which is always the case in the special case of PFE). This concrete auxiliary pole is then later translated into an X-switching block so that the inputs can be forwarded to the correct LUT.

Theorem 3. *Let* $\mathrm{P}^+ \in \mathbb{N}^n$. *Then there exists an EUG for* $\Gamma_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3(n + \Delta)\log_2(n + \Delta) + \mathcal{O}(n + \Delta),$$

where $\Delta \coloneqq \sum_{i=1}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$.

Proof. **Step 1:** Create a $\Gamma_2(n + \Delta)$ EUG $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}})$ with a topologically ordered pole set P that has the form $(..., v_{i-1}, u_{i,1}, ..., u_{i,\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil}, v_i, ...)$ for all $i \in [n]$, i.e., the auxiliary poles $u_{i,j}$ for $j \in [\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil]$ are directly preceding the original pole v_i: We do this by creating a Liu$^+$ EUG $\mathcal{U}$ with pole set P and split parameter 2. Then we merge two instances of it. By Theorem 2 and Corollary 1, this yields a $\Gamma_2(n + \Delta)$ EUG of size at most $3(n + \Delta)\log_2(n + \Delta) + \mathcal{O}(n + \Delta)$.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$ with pole set $P \subset V^{\bar{\mathcal{U}}}$: Let $u_{i,j}$ be an auxiliary pole of v_i for $i \in [n], j \in [\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil]$. Remove all of its

outgoing edges and replace each of them with an edge connecting the auxiliary pole to the original multi-input pole, i.e., remove each $(u_{i,j}, w) \in E^{\mathcal{U}}$ for $w \in V^{\mathcal{U}}$ and replace it by $(u_{i,j}, v_i)$. This yields two edges $(u_{i,j}, v_i)$ per auxiliary pole $u_{i,j}$. Thus, $E^{\mathcal{U}}$ becomes a multi set. If P_i^+ is odd and $j = 1$, add only one of these edges instead of two (otherwise, v_i would have too many ingoing edges). The graph that results from modifying $\mathcal{U}$ in the just described way is denoted by $\bar{\mathcal{U}}$.

Step 3: Embed any graph $G = (P, E) \in \Gamma_{\mathrm{P}^+,2}(n)$ into $\bar{\mathcal{U}}$: For this, we construct a $\Gamma_2(n+\Delta)$ graph using auxiliary poles for nodes with indegree higher than 2 by setting $\bar{G} = (\bar{V}, \bar{E}) = \texttt{auxiliaryGraph(G)} \in \Gamma_2(n+\Delta)$ (Algorithm 3). Note that the "relative topological order" is maintained, i.e., $\eta_{\bar{G}}(v_i) < \eta_{\bar{G}}(v_{i+1})\ \forall i \in [n]$. Edge-embedding $\bar{G}$ into $\bar{\mathcal{U}}$ yields $\psi\colon \bar{E} \to \mathcal{P}_{\bar{\mathcal{U}}}$. To show that $\bar{\mathcal{U}}$ is a $\Gamma_{\mathrm{P}^+,2}(n)$ EUG, we need to define an edge-embedding $\bar{\psi}$ from G into $\bar{\mathcal{U}}$: Note that for edges $e = (v_i, v_l) \in G \setminus \bar{G}$, i.e., edges whose endpoints are not auxiliary poles, ψ already yields edge-disjoint v_i-v_l-paths and we can set $\bar{\psi}(e) = \psi(e)$ for those edges.

Now consider edges $e = (v_i, v_l) \in G \cap \bar{G}$, i.e., the endpoints of those edges are transformed into an auxiliary pole in $\bar{G}$. For each e, there is exactly one $\bar{e} = (v_i, u_{l,j}) \in \bar{G}$ for $j \in [\lceil \frac{\deg^+(v_l)-2}{2} \rceil]$ (line 8 in Algorithm 3). Now set $\bar{\psi}(e) = \psi(\bar{e}) + (u_{l,j}, v_l)$ for one of the possibly two edges $(u_{l,j}, v_l)$ that were added to $\bar{\mathcal{U}}$ before. Obviously, this yields a v_i-v_l-path. Since there are at most two edges connecting to an auxiliary pole, we can choose a unique last edge for each path. Because the paths in the image of ψ were already edge-disjoint, also the paths in the image of $\bar{\psi}$ are edge-disjoint. Thus, $\bar{\psi}$ is an edge-embedding of G into $\bar{\mathcal{U}}$. □

Theorem 3 gives us an EUG that can be used to build VUCs for $(\rho \to 1)$-LUTs with varying parameter ρ and can thus be used for VPFE. Next, we consider VUCs for a fixed constant ρ which yields classical PFE.

Corollary 4. *Let* $\mathrm{P}^+ = \mathbb{1}\rho \in \mathbb{N}^n$ *for* $\rho > 2$. *Then there exists a EUG for* $\Gamma_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3\lceil \frac{\rho}{2} n \rceil \log_2(\lceil \frac{\rho}{2} n \rceil) + \mathcal{O}(\lceil \frac{\rho}{2} n \rceil).$$

Proof. We follow the proof of Theorem 3 and highlight the differences.

Step 1: Create a $\Gamma_2(\lceil \frac{\rho}{2} n \rceil)$ EUG $\mathcal{U}$ with topologically ordered pole set P that has the form $(..., v_{i-1}, u_{i,1}, ..., u_{i,\lceil \frac{\rho-2}{2} \rceil}, v_i, u_{i+1,1}, ..., u_{i+1,\lfloor \frac{\rho-2}{2} \rfloor}, v_{i+1}, ...)$ as described in step 1 in the proof of Theorem 3.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$ with pole set $P \subset V^{\bar{\mathcal{U}}}$ as described in step 2 in the proof of Theorem 3 with one difference: If ρ is odd, we share one auxiliary pole $u_{i,1}$ for two consecutive original poles v_i and v_{i+1}, i.e., we add the two edges $(u_{i,1}, v_i)$ and $(u_{i,1}, v_{i+1})$.

Step 3: Edge-embed G into $\bar{\mathcal{U}}$ as described in step 3 in the proof of Theorem 3 with one difference: If ρ is odd, the auxiliary graph $\bar{G} = (\bar{V}, \bar{E})$ shares one

auxiliary pole $u_{i,1}$ for two consecutive original poles v_i and v_{i+1}, i.e., $u_{i+1,1}$ is removed from $\bar{V}$ and the edge $(w, u_{i+1,1})$ is replaced by the edge $(w, u_{i,1})$. As $u_{i,1}$ and $u_{i+1,1}$ both have indegree 1, $u_{i,1}$ now has indegree 2. □

Multi-output Support for VUCs. An auxiliary graph that represents multi-output LUTs is a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$ graph as defined in Definition 12, i.e., $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$ classifies the graphs that can be edge-embedded into our UC construction. Here, P^+ is a vector of size n that specifies the indegree of each node in the auxiliary graph and thus represents the maximum number of inputs of each LUT in the UC. P^- is a constant that specifies the maximum outdegree of each node in the auxiliary graph/of each LUT in our circuit that we aim to evaluate with the UC. Similarly, Ω^- describes the number of distinguished outputs of the LUTs, i.e., P^- specifies the number of copies we have for each output of a LUT in our circuit, while Ω^- sets the number of outputs for each LUT.

As later, when embedding G into the EUG, each output of a LUT represents a separate value, i.e., we need to put each output into an individual pole. As the poles of the EUG are the nodes of the auxiliary graph, we need to add for each additional output of the i-th LUT in total $\Omega_i^- - 1$ additional poles. In Definition 12, we denote the outputs of the i-th LUT with $v_{i,1}, \ldots, v_{i,\Omega_i^-}$.

Definition 12 ($\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$)**.** *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V := \{v_{1,1}, \ldots, v_{1,\Omega_1^-}, v_{2,1}, \ldots, v_{2,\Omega_2^-}, \ldots, v_{n,1}, \ldots, v_{n,\Omega_n^-}\}$ and $\mathrm{P}^+, \mathrm{P}^-, \Omega^- \in \mathbb{N}^n$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$ if:*

- $|V| \leq \sum_{i=1}^{n} \Omega_i^-$,
- $|\{v_{i,j} \in V\}| \leq \Omega_i^- \ \forall i \in [n]$,
- $deg^+(v_{i,1}) \leq \mathrm{P}_i^+ \wedge deg^+(v_{i,2}) = \cdots = deg^+(v_{i,\Omega_i^-}) = 0$,
- $deg^-(v_{i,j}) \leq \mathrm{P}_i^- \ \forall i \in [n] \ \forall j \in [\Omega_i^-]$.

To easily build an EUG with only marginal modifications, we show that a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}$ is also a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-}$ graph:

Proposition 3. *Let $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n + \Delta)$, where $\Delta := \sum_{i=1}^{n} \Omega_i^- - 1$.*

Proof. Let $G = (V, E) \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$. It holds that $|V| \leq \sum_{i=1}^{n} \Omega_i^- = n + \Delta$ where $\Delta = \sum_{i=1}^{n} \Omega_i^- - 1$ (condition 1 in Definition 12). Further, for all $v \in V$ it holds that $\deg^+(v) \leq \mathrm{P}_i^+$ and $\deg^-(v_{i,j}) \leq \mathrm{P}_i^-$ from conditions 3 and 4 in Definition 12. □

We can build VUCs using Corollary 5 and UCs with constant ρ and ω using Corollary 6, whose prove directly follows from Corollarys 4 and 5.

Corollary 5. *Let* $\mathrm{P}^+, \Omega^- \in \mathbb{N}^n$*. Then there exists a EUG for* $\Gamma_{\mathrm{P}^+,2,\Omega^-}(n)$ *with size bounded by*

$$3(n+\Delta)\log_2(n+\Delta)+\mathcal{O}(n+\Delta),$$

where $\Delta \coloneqq \sum_{i=1}^{n}(\max\{\lceil\frac{\mathrm{P}_i^+-2}{2}\rceil,0\}+\Omega_i^- -1)$.

Proof. Let $G=(P,E)\in\Gamma_{\mathrm{P}^+,2,\Omega^-}(n)$ be the graph to be embedded in an EUG with $P=\{v_{1,1},\ \ldots,\ v_{1,\Omega_1^-},\ v_{2,1},\ \ldots,\ v_{2,\Omega_2^-},\ \ldots,\ v_{n,1},\ \ldots,\ v_{n,\Omega_n^-}\}$. We can transform G into a $\Gamma_{\mathrm{P}^+,2}(n+\Delta')$ graph where $\Delta' \coloneqq \sum_{i=1}^{n}\Omega_i^- - 1$. Using Theorem 3, we get an EUG for $\Gamma_{\mathrm{P}^+,2}(n+\Delta')$ that is bounded by

$$3(n+\Delta'+\Delta'')\log_2(n+\Delta'+\Delta'')+\mathcal{O}(n+\Delta'+\Delta''),$$

where $\Delta'' \coloneqq \sum_{i=1}^{n}\max\{\lceil\frac{\mathrm{P}_i^+-2}{2}\rceil,0\}$ and setting $\Delta \coloneqq \Delta'+\Delta''$ yields an EUG of the given size.

We need to add some more edges to the resulting EUG $\bar{\mathcal{U}}=(V^{\bar{\mathcal{U}}},E^{\bar{\mathcal{U}}})$ with pole set $P\subset V^{\bar{\mathcal{U}}}$, namely the inputs of the first pole associated with the LUT need to be forwarded to all remaining output poles of the same LUT as follows:

$\forall i\in[n]:\forall v_{i,1}\in P:\forall(u,v_{i,1})\in E^{\bar{\mathcal{U}}}:\forall v_{i,j}\in P, j>1: E^{\bar{\mathcal{U}}}=E^{\bar{\mathcal{U}}}\cup(u,v_{i,j})$. □

Corollary 6. *Let* $\mathrm{P}^+=\rho\in\mathbb{N}^n$ *for* $\rho>2$ *and* $\Omega^-=\mathbb{1}\omega\in\mathbb{N}^n$ *for* $\omega>1$*. Then there exists an EUG for* $\Gamma_{\mathrm{P}^+,2,\Omega^-}(n)$ *with size bounded by*

$$3(\lceil(\frac{\rho}{2}+\omega-1)n\rceil)\log_2(\lceil(\frac{\rho}{2}+\omega-1)n\rceil)+\mathcal{O}(\lceil(\frac{\rho}{2}+\omega-1)n\rceil).$$

5.2 Applications of Varying UCs (VUCs)

If we use a VUC instead of a UC in MPC-based PFE, we get Varying Private Function Evaluation (VPFE). VPFE allows a set of k parties $\mathcal{P}_1,\ldots,\mathcal{P}_k$, to jointly compute a circuit C held by $\mathcal{P}_1$ on private data $x_2,\ldots,x_k$ held by $\mathcal{P}_{i\geq 2}$ to obtain nothing but $C(x_2,\ldots,x_k)$, and $\mathcal{P}_{i\geq 2}$ learn nothing about C but the dimensions of all its LUTs. Thus, VPFE does not leak the whole topology of sub-circuits like SPFE (cf. Sect. 1.1), but leaks more information than PFE.

We can reduce the leakage by randomly changing the sequence of LUTs according to the topological order of the simulated circuit. In this way, building blocks (e.g., full adders) do not occur as a whole block of consecutive LUTs of the same dimension in the VUC. The function would be mapped to different sequences of dimensions and thus we would remove fingerprints of certain functions. So, even multiple building blocks of different circuit layers can be mixed in a sequence. This technique, however, still allows to exclude certain functions when they cannot be mapped to the given sequence of dimensions.

Some applications, such as logic locking (cf. [11, Fig. 3] and Sect. 1.1) do not require full privacy of the evaluated function and allow for the leakage of the sequence of dimensions of the used LUTs. However, in general PFE applications, even knowledge of the LUT sizes may reveal too much information about the protected function. Our analysis (cf. Sect. 4.3) and our benchmarks (cf. Sect. 6.3) demonstrate that many functionalities can be reduced to 3-input LUTs. Consequently, we benefit from using LUCs with 3-input LUTs in most cases. This observation is not surprising, as most arithmetic operations can be reduced to full adders (3-input LUTs), and only a small number of sub-functionalities benefit from using LUTs with more than 3 inputs. However, when adding only one of these larger LUTs, the overall size of the LUC would be significantly increased, as a complete EUG graph would need to be added to the circuit for each additional input of all LUTs, even if the higher dimension is used only once. Therefore, VUCs are well-suited for embedding circuits with a limited number of various LUT combinations, such as $(3 \rightarrow 1)$-LUT and $(8 \rightarrow 8)$-LUT, resulting in significant size improvements. By implementing simple functionalities with $(3 \rightarrow 1)$-LUTs and allowing complex functionalities with $(8 \rightarrow 8)$-LUTs, a wide range of possible functions can be achieved without compromising critical information (which can always be implemented using a single LUT type). The $(8 \rightarrow 8)$-LUTs offer a vast set of 256 combinations, enabling the implementation of a large and diverse collection of functionalities. Despite the inclusion of these additional combinations, the resulting leakage remains limited.

There are many PFE applications that benefit from such a setting, including credit checking [17], user-specific tariff calculations [21], and medical diagnosis [9]. All these applications rely on sub-functionalities such as classifiers. A classifier utilizes a mapping table to look up a class based on input data, and then outputs the determined class. To illustrate, a car insurance tariff calculator may use a classifier to establish a basic price based on the type of car a potential customer drives. Multi-input LUTs, such as $(8 \rightarrow 8)$-LUTs, can efficiently implement these classifiers as they provide exactly such a table lookup. By incorporating individually tailored multi-input LUTs in a VUC, we can benefit from overall size improvements over the LUC construction, while still maintaining the internal implementation of the classifier, including the computation performed to obtain the address of the lookup whose topology is hidden.

6 Implementation and Evaluation

We implement our proposed UC constructions using the MPC framework ABY [13] to provide a fair comparison to previous PFE works on UCs [7,22,31]. MPC frameworks supporting multi-input garbled circuits [39] reduce the communication of evaluating a single ρ input LUT to $2^\rho - 1$ ciphertexts. In ABY, we implement ρ input LUTs as a multiplexer tree consisting of $2^\rho - 1$ 2-input AND gates, requiring $2(2^\rho - 1)$ ciphertexts using half-gates [56]. This could be further reduced to $1.5(2^\rho - 1)$ ciphertexts using three-halves garbling [47].

We benchmark our LUC construction (cf. Sect. 4) and compare it with the most recent UC of Liu et al. [37] that simulates circuits with binary gates. Moreover, we evaluate our VUC construction (cf. Sect. 5) to show the improvement over Liu et al.'s UC [37] and our LUC construction. All results in this section use the EUG construction by Liu et al. [37] to construct the underlying Γ_1 EUGs. We discuss the LUT generation in Sect. 6.1, details about our UC compilation in Sect. 6.2, and experimental results in Sect. 6.3.

6.1 LUT Generation

Hardware synthesis is a crucial process in electronic design automation that involves converting an abstract function description into a functionally equivalent logic implementation. This transformation is achieved through the utilization of various optimization and technology mapping algorithms. These algorithms have been extensively researched and developed over the course of many years. The resulting circuit implementation is typically dependent on the target hardware platform and the manufacturing technology employed. The two most common target hardware platforms are Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs).

This work specifically focuses on exploiting multi-input LUTs, which are fundamental components of FPGAs (which consist of logic cells containing programmable LUTs) and their corresponding synthesis tools. Although ASIC synthesis tools can also map to multi-input gates, this process is laborious, impractical, and necessitates the creation of large libraries to accommodate all possible LUTs for each input size. Thus, we chose FPGA synthesis tools. The market offers commercial FPGA synthesis tools like Intel Quartus Prime [1], VTR [2], XST [4], and Vivado Synthesis tools by Xilinx [3]. However, these tools synthesize LUT-based circuits tailored to the specific features of their respective devices. For instance, most current FPGA devices support a maximum of 6-input LUTs. In our work, we aim to generate circuits with up to 8-input LUTs, which, to the best of our knowledge, is not supported by mainstream commercial tools.

In this work, similar to [12,14], we leverage the mapping capabilities of the open-source tools Yosys [53] and ABC [10]. Yosys allows us to transform the circuit descriptions into a network of low-level logic operations represented in an intermediate format. Subsequently, ABC [10] organizes this network into a Directed Acyclic Graph (DAG) and maps it to a depth-optimized circuit composed of LUTs. It is worth noting that ABC [10] does not inherently support mapping to multi-output LUTs. To overcome this limitation, we perform post-processing on the single-output LUT circuits generated by ABC [10] and convert them into multi-output LUT circuits. Additionally, we use integrated Intellectual Property (IP) libraries within the commercial ASIC synthesis tool Synopsys Design Compiler (DC) [5], to generate circuit netlists for more complex functionalities such as floating-point operations. These circuits are initially created as Boolean netlists by Synopsys DC [5], and we subsequently remap them to LUT-based representations using the Yosys-ABC toolchain [10,53].

6.2 UC Compilation

Let C denote the circuit to be embedded and ρ the maximum fan-in of the circuit.

1. Parsing the circuit: The circuit is input in the Secure Hardware Definition Language (SHDL) [39] and parsed into the internal graph representation. We reduce the fan-out of the graph to the allowed fan-in ρ for LUCs (cf. Sect. 4) and 2 for VUCs (cf. Sect. 5) by using copy gates. For VUCs, the auxiliary graph (cf. Theorem 3) is generated. Here, we denote the auxiliary graph by G and the former graph with possibly reduced fan-out by $\bar{G}$.

2. Splitting G into Γ_1 graphs and creating Γ_1 EUGs: Using the LUC construction yields ρ Γ_1 graphs. For each Γ_1 graph, we create a Γ_1 EUG. Possible EUGs are Valiant's EUG [51] and the 2-way split EUG of Liu et al. [37]. If we use the VUC construction, we get two Γ_1 graphs.

3. Edge-embedding the Γ_1 graphs and merging them: Each Γ_1 graph is edge-embedded into the corresponding Γ_1 EUG. This edge-embedding is coded directly into the control bits of the X- and Y-Switches of the EUG. The concrete algorithm uses a slightly modified version of the edge-embedding algorithm in [22]. Then, the Γ_1 EUGs are merged into a Γ_ρ EUG (LUC) or into a Γ_2 EUG (VUC).

4. Basic optimizations and correctness checking: We remove edges connecting to an input pole as they will never be used and replace copy gates with wires. Then we remove isolated nodes or change X- to Y-Switching nodes if one edge was removed before. We check the correctness of the edge-embedding by checking for each edge (u, v) in G, if there is a path leading from u to v.

5. Setting the gates of the EUG: In the VUC construction, we replace the auxiliary poles with wires connecting directly to the actual pole or a Y-Switch if only one input is forwarded. Analogously to step 4, we check the correctness of the edge-embedding to $\bar{G}$. For each node in G, we set the programming bits of the corresponding EUG pole. We determine the order of inputs and then set the programming bits accordingly. This also involves padding the programming bits if the gate has more inputs. Note that these additional inputs are likely to occur since each Universal Gate outputs ρ (in LUC construction) or 2 (in VUC construction) wires, independent of whether they are used in G or not. We pad the programming bits such that additional and undesired inputs are ignored.

6. Transforming the EUG into an ABY compatible UC: As a final step, we topologically order the EUG and output it in the UC format compatible with ABY [13]. Then, each node, along with its incoming and outgoing wires, is written into a circuit file. At the same time, the programming bits are written into a separate programming bits file.

6.3 Experimental Results

Setup. Like previous works [22,31,37], we benchmark a set of real-world circuits from [50]. In addition, we consider other useful functions like Karatsuba multiplication [28], Manhattan and Euclidean distance [14], and floating-point operations [14]. For each functionality, we give the sizes of the resulting circuit, as well as communication and runtime complexity when the UC is evaluated with an MPC protocol. In order to show the improvement of our work, we use two identical machines with a LAN connection of 10 Gbit/s bandwidth and a round-trip time of 1 ms. Each machine is equipped with an Intel Core i9-7960X@2.8 GHz with 128GB DDR4 RAM. All measurements are averaged over 10 executions.

Table 3. Number of AND and XOR gates per building block in our UCs.

Building block	AND gates	XOR gates
X-switching block [35]	1	3
Y-switching block [35]	1	2
Universal Gate with $k \geq 2$ inputs	$2^k - 1$	$2^{k+1} - 2$

LUC Improvement. As we have Universal Gates of different sizes, we cannot just count the number of nodes in the EUG to compare the implementations. As underlying MPC protocol for UC-based PFE we use Yao's protocol [56] using free XORs [35], so XOR gates can be evaluated without communication. Therefore, we count the number of non-free AND gates to instantiate the building blocks of the UC (cf. Table 3). We experimentally compared our implementations with the best existing UC-based PFE construction of Liu et al. [37]. We provide our results for our LUT-based UC constructions in Table 4. In our circuit generation, we vary possible choices for $(\rho \rightarrow \omega)$-LUTs and select the ones with highest improvement. We can see from Table 4 that our LUT-based UC construction is always smaller than that of [37] by $1.12 - 2.18\times$.

For a comparison of the improvements in PFE, we securely evaluate our generated UCs with the GMW-based SP-LUT protocol [14] and Yao's GC protocol [56]. In Table 5, we show the runtime and communication of our LUT-based UC construction (LUC) compared to the most recent UC construction of Liu et al. [37] as baseline using Yao [56] and GMW [20]. Our new UC construction is the fastest implementation: Compared to the baseline using Yao [56], the total

Table 4. Comparison of the sizes of our LUT-based UC construction (LUC, cf. Sect. 4) and the best previous UC construction of Liu et al. [37] as baseline (in number of AND gates) measured with our implementations. The smallest size is marked in **bold** and always achieved by our UCs. The sizes for our UC is the best combinations for ($\rho \to \omega$)-LUT for $\rho \in \{2, ..., 8\}$ inputs and $\omega \in \{1, \ldots, 8\}$ outputs for the benchmarked circuit.

Circuit	Circuit size (# AND gates)		Improvement (×)	LUT sizes ($\rho \to \omega$)
	UC of [37]	Our LUC		
AES	**1,779,105**	**1,779,105**	1.00	$2 \to 1$
DES	1,269,537	**1,130,037**	1.12	$3 \to 1$
MD5	3,293,262	**1,724,221**	1.91	$3 \to 1$
SHA-1	4,872,501	**2,559,602**	1.90	$3 \to 1$
SHA-256	10,652,234	**5,351,972**	1.99	$3 \to 1$
Add_32	6,926	**3,907**	1.77	$3 \to 2$
Add_64	17,006	**8,963**	1.90	$3 \to 2$
Comp_32	2,519	**1,278**	1.97	$3 \to 1$
Mult_32x32	347,274	**177,081**	1.96	$3 \to 3$
Karatsuba_32x32	286,933	**156,888**	1.83	$3 \to 3$
MD256	327,203	**150,046**	2.18	$3 \to 2$
ED64	1,852,419	**947,679**	1.95	$3 \to 3$
FP-Add_32	113,620	**90,964**	1.25	$3 \to 1$
FP-Mul_32	293,125	**247,859**	1.18	$3 \to 1$
FP-Exp2_32	2,008,269	**1,548,079**	1.30	$3 \to 1$
FP-Div_32	372,101	**236,300**	1.57	$3 \to 1$
FP-Sqrt_32	176,176	**118,873**	1.48	$3 \to 1$
FP-Comp_32	6,387	**5,628**	1.13	$4 \to 4$
FP-Log_32	1,936,813	**1,499,538**	1.29	$3 \to 1$

runtime for our sample circuits is faster by a factor of $1.14-2\times$. The communication improvements over the baseline using Yao [56] are $1.12-2.25\times$. The runtime of Yao's protocol is $3.83 - 11.5\times$ faster than that of the LUT-based protocols which can be explained by the constant round complexity of Yao's protocol. The SP-LUT protocol [14] always has the lowest communication, achieving factor $1.19 - 2.44\times$ less communication than Yao's protocol. In Table 5, it is evident that the baseline employing Yao [56] exhibits superior runtime performance and lower total communication overhead than the baseline employing the GMW protocol [20].

Table 5. Runtime and communication for our LUT-based UC construction (cf. Sect. 4) compared to the state-of-the-art UC of [37] when evaluated with ABY [13]. We include the LAN evaluation time (in seconds) and the total communication (in Megabytes) between the parties in LUT-based [14], Yao sharing [56], as well as in GMW sharing [20]. The best values are marked in **bold**.

UC construction	UC of [37]				Our LUT-based UC (LUC)			
MPC protocol	Yao [56]		GMW [20]		Yao [56]		SP-LUT [14]	
Circuit	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)
AES	**1.811**	80.315	142.193	96.845	**1.811**	80.315	13.187	**28.427**
DES	1.282	57.271	101.645	68.071	**1.124**	50.570	9.233	**24.311**
MD5	3.471	148.348	238.847	168.991	**1.832**	76.638	26.642	**46.013**
SHA1	5.184	220.065	343.344	246.708	**2.756**	113.859	27.268	**58.641**
SHA-256	11.571	481.412	722.650	528.122	**5.878**	238.364	54.082	**123.045**
Add_32	0.018	0.314	1.139	0.594	**0.009**	0.177	0.224	**0.148**
Add_64	0.026	0.770	2.323	1.230	**0.017**	0.404	0.452	**0.319**
Comp_32	0.008	0.117	0.265	0.203	**0.004**	0.062	0.139	**0.055**
Mult_32x32	0.350	15.626	31.497	19.650	**0.212**	7.300	4.144	**4.531**
Karatsuba_32x32	0.292	12.901	27.214	16.597	**0.191**	6.469	3.685	**4.020,1**
MD256	0.337	14.801	29.037	18.326	**0.193**	6.592	4.234	**4.544**
ED64	1.924	83.552	142.147	97.558	**1.046**	39.704	17.524	**24.572**
FP-Add_32	0.164	5.105	11.780	6.859	**0.139**	4.003	2.903	**2.426**
FP-Mul_32	0.350	13.178	28.215	16.988	**0.308**	10.949	6.217	**4.579**
FP-Exp2_32	2.292	90.555	155.881	106.023	**1.612**	68.651	21.531	**38.330**
FP-Div_32	0.458	16.743	33.686	21.024	**0.296**	10.443	5.918	**6.528**
FP-Sqrt_32	0.223	7.915	18.910	10.433	**0.168**	5.237	3.417	**3.442**
FP-Comp_32	0.014	0.290	1.066	0.553	**0.012**	0.235	0.226	**0.118**
FP-Log_32	2.083	87.330	151.423	102.369	**1.600**	66.510	20.198	**36.287**

VUC Improvement. Table 6 shows that our VUC construction which – other than LUC – leaks the fanin of the individual LUTs is up to 2.90× smaller than Liu et al.'s UC [37] when evaluated with Yao's protocol [56], the total runtime for our sample circuits is faster by $1.1 - 2.85\times$ and the communication is improved by $1.06 - 2.96\times$. This shows that significant speedups can be achieved when giving up some function privacy.

Note that during the process of compiling our VUC construction, our tool conducts an initial verification to determine whether the LUC construction results in a better size than VUC, and, if so, proceeds to compile a LUC. Nonetheless, in the majority of cases, VUC yields a better size by a factor of up to 1.45×. The superiority of VUC over LUC is strongly influenced by the circuit design. Specifically, if the circuit can primarily be constructed using Look-Up Tables (LUTs) with identical input dimensions, the overall size is better than VUC. However, if the circuit can be effectively constructed using LUTs with differing input dimensions, VUC performs better.

Table 6. Sizes, runtime, and communication for our VUC construction (cf. Sect. 5). We include LAN evaluation times (in seconds) and total communications (in Megabytes) between the parties in LUT-based [14] as well as in Yao sharing [56]. We show the size improvement of VUC over the UC of [37] and LUC construction (cf. Sect. 4) in the last two columns. Note that VUCs reveal the LUTs' dimensions, showcasing the enhancements obtained by sacrificing some circuit privacy.

Circuit	Size	Yao [56]		SP-LUT [14]		Size Improv.(×) VUC/UC of [37]	Size Improv.(×) VUC/LUC
		Time	Comm.	Time	Comm.		
AES	1,584,047	1.724	71.753	142.221	2.607	1.13	1.13
DES	960,854	0.98	43.441	93.79	1.66	1.32	1.17
MD5	1,191,566	1.22	52.89	23.01	42.77	2.76	1.45
SHA-1	2,559,602	2.76	113.86	27.27	58.64	1.90	1.00
SHA-256	4,591,982	4.91	201.98	52.22	108.43	2.32	1.17
Add_32	3,907	0.01	0.18	0.23	0.15	1.77	1.00
Add_64	8,963	0.02	0.40	0.45	0.32	1.90	1.00
Comp_32	1,188	0.01	0.05	0.04	0.04	2.12	1.08
Mult_32x32	130,053	0.14	5.51	1.42	4.06	2.67	1.36
Karatsuba_32x32	112,829	0.12	5.01	1.41	3.41	2.54	1.40
MD256	112,829	0.13	5.01	1.37	4.09	2.90	1.33
ED64	947,679	1.05	39.70	17.52	24.58	1.95	1.00
FP-Add_32	90,964	0.14	4.00	2.90	2.43	1.25	1.00
FP-Mul_32	185,968	0.18	8.11	2.04	3.65	1.58	1.33
FP-Exp2_32	1,265,869	1.34	55.72	19.38	25.16	1.59	1.22
FP-Div_32	181,904	0.18	7.89	1.91	5.27	2.05	1.30
FP-Sqrt_32	89,311	0.10	3.84	1.07	2.70	1.97	1.33
FP-Comp_32	5,269	0.01	0.22	0.11	0.093	1.21	1.07
FP-Log_32	1,230,530	1.31	54.16	16.17	24.69	1.57	1.22

Acknowledgements. This project received funding from the ERC under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the DFG within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

References

1. Intel Quartus Prime Software. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html
2. Verilog to Routing. https://verilogtorouting.org/
3. Vivado 2023.1 - Logic Synthesis. https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0018-vivado-synthesis-hub.html
4. XST Synthesis. https://www.xilinx.com/products/design-tools/xst.html
5. Synopsys Inc., Design Compiler (2010). http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler

6. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. JoC (1990)
7. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and Scalable Universal Circuits. JoC (2020)
8. Attrapadung, N.: Fully Secure and Succinct Attribute Based Encryption for Circuits from Multi-linear Maps. Cryptology ePrint Archive, Report 2014/772 (2016)
9. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_26
10. Berkeley Logic Synthesis and Verification Group: ABC: A system for sequential synthesis and verification. http://www.eecs.berkeley.edu/alanmi/abc/
11. Bhandari, J., et al.: Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction. CoRR: abs/2111.04222 (2021)
12. Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., Yalame, H.: FLUTE: fast and secure lookup table evaluations. In: S&P (2023)
13. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
14. Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: NDSS (2017)
15. Disser, Y., Günther, D., Schneider, T., Stillger, M., Wigandt, A., Yalame, H.: Breaking the Size Barrier: Universal Circuits meet Lookup Tables. Cryptology ePrint Archive, Report 2022/1652 (2022)
16. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: CCS (2014)
17. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: ACM Conference on Electronic Commerce (2005)
18. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-based encryption for circuits from multilinear maps. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 479–499. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_27
19. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-hop homomorphic encryption and rerandomizable yao circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_9
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC (1987)
21. Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Poster: framework for semi-private function evaluation with application to secure insurance rate calculation. In: CCS (2019)
22. Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_16
23. Henecka, W., Kögl, S., Sadeghi, A., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-Party Computations. In: CCS (2010)
24. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 401–420. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_20
25. Ji, K., Zhang, B., Lu, T., Ren, K.: Multi-party private function evaluation for RAM. IEEE Trans. Inf. Forensics Secur. **18**, 1252–1267 (2023)

26. Kamali, H.M., Azar, K.Z., Gaj, K., Homayoun, H., Sasan, A.: LUT-lock: a novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection. In: ISVLSI (2018)
27. Kamara, S., Raykova, M.: Secure outsourced computation in a multi-tenant cloud. In: IBM Workshop on Cryptography and Security in Clouds (2011)
28. Karatsuba, A.A., Ofman, Y.P.: Multiplication of many-digital numbers by automatic computers. In: SSSR Academy of Sciences (1962)
29. Karnaugh, M.: The map method for synthesis of combinational logic circuits. Trans. Am. Inst. Electrical Eng. **72**, 593–599 (1953)
30. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_30
31. Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27
32. Kluczniak, K.: Circuit privacy for FHEW/TFHE-style fully homomorphic encryption in practice. Cryptology ePrint Archive, Report 2022/1459 (2022)
33. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved garbled circuit building blocks and applications to auctions and computing minima. In: CANS (2009)
34. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: FC (2008)
35. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
36. Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant's Universal Circuit: Improvements, Implementation, and Applications. Cryptology ePrint Archive, Report 2016/017 (2016)
37. Liu, H., Yu, Yu., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of valiant's universal circuits: simpler, tighter and more compact. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 365–394. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_13
38. Liu, Y., Wang, Q., Yiu, S.: Making private function evaluation safer, faster, and simpler. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022. LNCS, vol. 13177, pp. 349–378. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97121-2_13
39. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security (2004)
40. Masserova, E., Garg, D., Mai, K., Pileggi, L., Goyal, V., Parno, B.: Logic Locking-Connecting Theory and Practice. Cryptology ePrint Archive, Report 2022/545 (2022)
41. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: improved mixed-protocol secure two-party computation. In: USENIX Security (2021)
42. Patra, A., Schneider, T., Suresh, A., Yalame, H.: SynCirc: efficient synthesis of depth-optimized circuits for secure computation. In: HOST (2021)
43. Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_6

44. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15
45. Pohle, E., Abidin, A., Preneel, B.: Poster: fast evaluation of S-boxes in MPC. In: NDSS (2022)
46. Quine, W.V.: The problem of simplifying truth functions. The American Mathematical Monthly (1952)
47. Rosulek, M., Roy, L.: Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12825, pp. 94–124. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84242-0_5
48. Sadeghi, A.R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: ICISC (2008)
49. Shannon, C.E.: The synthesis of two-terminal switching circuits. Bell Syst. Tech. J. **28**, 59–98 (1949)
50. Smart, N., Tillich, S.: Bristol Fashion MPC circuits. https://homes.esat.kuleuven.be/nsmart/MPC/old-circuits.html
51. Valiant, L.G.: Universal Circuits (Preliminary Report). In: STOC (1976)
52. Wegener, I.: The Complexity of Boolean Functions. Wiley, New York (1987)
53. Wolf, C., Glaser, J., Kepler, J.: Yosys - a free Verilog synthesis suite. In: Austrian Workshop on Microelectronics (2013)
54. Yao, A.C.: How to generate and exchange secrets (Extended Abstract). In: FOCS (1986)
55. Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J., Sinanoglu, O.: Provably-secure logic locking: from theory to practice. In: CCS (2017)
56. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8
57. Zhao, S., Yu, Yu., Zhang, J., Liu, H.: Valiant's universal circuits revisited: an overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 401–425. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_15
58. Zimmerman, J.: How to obfuscate programs directly. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 439–467. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_15

H Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater (IEEE S&P'24)

[BSS+24] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, H. Yalame. **"Don't Eject the Impostor: Fast Three-Party Computation with A Known Cheater."** In: *IEEE Symposium on Security and Privacy (IEEE S&P)*. Online: https://ia.cr/2023/1744. Code: https://encrypto.de/code/MOTION-FD. IEEE, 2024. CORE Rank **A***. Appendix H.

https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00164