TECHNISCHE
UNIVERSITÄT
DARMSTADT

Towards Deployable MPC: Flexible and Efficient Tools for Real-World Applications

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

Oleksandr Tkachenko, M.Sc.
born in Kherson, Ukraine

Advisors: Prof. Dr.-Ing. Thomas Schneider
Prof. Dr. Benny Pinkas

Date of submission: 02.05.2022
Date of defense: 23.06.2022

D 17
Darmstadt, 2022

Tkachenko, Oleksandr
Towards Deployable MPC: Flexible and Efficient Tools for Real-World Applications
Darmstadt, Technical University of Darmstadt
Publication year of the dissertation on TUprints: 2022
URN: urn:nbn:de:tuda-tuprints-212537
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/21253
Date of defense: 23.06.2022

Erklärung

Hiermit versichere ich, Oleksandr Tkachenko, 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. 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, 02.05.2022

Oleksandr Tkachenko, M.Sc.

Abstract

Secure multi-party computation (MPC) allows two or more parties to compute an arbitrary function on their private inputs while revealing nothing beyond the output of the function. MPC is a generic technique that can be used to build privacy-preserving solutions for a large number of real-world, privacy-sensitive applications such as collaborative medical research or machine learning on sensitive data. This, however, poses big challenges: 1) the current tools for MPC are fine-tuned for particular tasks and are challenging to adopt or extend to other tasks which is though required in many use cases, and 2) MPC imposes a significant communication and computational overhead compared to the cleartext computation, rendering most "naïvely constructed" privacy-preserving protocols impractical for the real-world use.

We address both challenges in this thesis. Firstly, we design and implement an open-source flexible and efficient framework for MPC and secure outsourcing, where MPC is outsourced to a smaller number of non-colluding parties who gain no information about the inputs nor about the intermediate or final results of the computation. Secondly, we design, implement, and evaluate the first linear-communication circuit-based private set intersection (PSI) protocol, i.e., it can can be combined with arbitrary MPC functionalities, e.g., reveal the intersection only if the intersection size is larger than some threshold. Thirdly, we design, implement, and evaluate two MPC protocols for specific tasks: 1) WiFi fingerprint based indoor localization and 2) similar sequence queries on genomic data.

Generic MPC Tools and Protocols To run MPC protocol for real tasks, we need tools for MPC that can be used to implement arbitrary applications in a privacy-preserving way. There exist several open-source MPC frameworks but all of them are either efficient or they can be modified without a significant time investment, but not both at the same time. As a consequence, a significant amount of time is spent by researchers in an attempt to understand and modify MPC frameworks, which often crash after incautious modifications, thus slowing down the research.

To address this problem, we develop MOTION, an open-source C++ framework for implementing and combining MPC protocols and primitives. In contrast to prior work, MOTION combines both efficiency and flexibility. It allows to implement and combine MPC protocols and optimizations while learning and modifying only the relevant parts of the framework, which is due to the novel asynchronous design.

In MOTION, we combine three different generic MPC protocols for two or more parties. We also design and implement novel efficient conversions between some of those protocols and optimize low-level building blocks. Finally, we show that MOTION is often substantially more efficient than other MPC frameworks.

While MOTION can evaluate arbitrary functionalities, we also investigate a more specific but yet generic class of functionalities. PSI is a well-known building block for privacy-preserving applications, and it has generated a long line of research. PSI allows two parties to compute set intersection on their private input sets without revealing any information beyond the

intersection. The notion of PSI we improve in this part of the thesis is called circuit-based PSI (C-PSI), and it allows to perform arbitrary MPC while hiding the intersection result, e.g., the parties can compute complex statistics on the intersection result.

We design the first C-PSI protocol with linear communication based on sophisticated hashing techniques and oblivious programmable pseudo-random functions. Beyond better asymptotic communication complexity, we show that an implementation of our C-PSI protocol is significantly more efficient compared to prior work in terms of both computation and communication.

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

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *Transactions on Privacy and Security (TOPS)* 25 (2 2022). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION. CORE Rank A. Appendix A.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, pp. 122–153. CORE Rank A*. Appendix B.

MPC Applications & Optimizations MPC is not yet widely adopted in real applications. The main reason is the high communication and computational overhead of MPC compared to cleartext computation. Furthermore, the design of efficient MPC protocols requires expert knowledge in computer science, cryptography, and circuit design. Tools such as HyCC (Büscher et al., CCS'19) lift this requirement for small applications but are yet not suitable for large and complex problems.

In this thesis, we build MPC-based protocols for two privacy-sensitive real-world applications: 1) indoor localization and 2) similar sequence queries on genomic databases. To improve the efficiency of our protocols, we use optimized building blocks such as our currently smallest circuit for finding k-nearest neighbors.

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

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E. S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 448–463. Appendix C.

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Full version: https://ia.cr/2021/029. ACM, 2019, pp. 315–327. CORE Rank A. Appendix D.

This thesis contributes to making MPC more flexible but also more efficient for a generic use by optimizing low-level building blocks, and it introduces efficient privacy-preserving solutions for two concrete real-world applications.

Zusammenfassung

Sichere Mehrparteienberechnungen (im Englischen *secure multi-party computation*, MPC) erlauben zwei oder mehr Parteien, eine beliebige Funktion basierend auf geheimen Eingabewerten auszuwerten, ohne etwas außer der Ausgabewerten preiszugeben. MPC ist eine generische Technik, die für die Realisierung Privatsphäre-schützender Lösungen für eine Vielzahl praktischer Anwendungen mit Bezug auf private Daten eingesetzt werden kann. Beispielsweise ist MPC für medizinische Kooperationsforschung und maschinelles Lernen auf sensiblen Daten einsetzbar. Hierbei gibt es jedoch zwei Herausforderungen: 1) die aktuellen Tools für MPC sind für konkrete Aufgaben entworfen worden und sind nur schwer für andere Aufgaben einsetz- oder anpassbar. Dies ist allerdings in vielen Anwendungsszenarien unvermeidbar. 2) Im Vergleich zur Klartextberechnung bringt MPC mit sich einen erheblichen Mehraufwand in Bezug auf die Kommunikations- und Rechenkosten mit sich, was die "naive" Konstruktion der Privatsphäre-schützenden Protokolle in der Regel ineffizient für praktische Anwendungen macht.

In dieser Dissertation gehen wir beide Herausforderungen an. Erstens konstruieren und implementieren wir ein quelloffenes, flexibles und effizientes Framework für MPC und sicheres Outsourcing, wo MPC an eine kleinere Anzahl von nicht-kolludierenden Parteien ausgelagert wird. Diese Parteien erhalten keine Informationen über die Eingabe-, Zwischen- und Ausgabewerte. Zweitens konstruieren, implementieren und evaluieren wir das erste Protokoll für Schaltkreis-basierte private Schnittmengenberechnung (im Englischen *private set intersection*, PSI) mit linearen Kommunikationskosten. Schaltkreis-basiertes PSI ermöglicht beliebige weitere Berechnungen auf der Schnittmenge, z.B. dass die Schnittmenge preisgegeben wird, wenn eine Mindestgröße erreicht wird. Drittens konstruieren, implementieren und evaluieren wir zwei MPC-Protokolle für spezifische Aufgaben: 1) WiFi-Fingerprint-basierte Indoor-Lokalisierung und 2) Similärgensequenzanfragen.

Generische MPC-Tools und MPC-Protokolle Um MPC-Protokolle für praktische Aufgaben zu verwenden, werden Tools für MPC benötigt, die die Privatsphäre-schützende Implementierung beliebiger Anwendungen ermöglichen. Es existieren bereits mehrere quelloffene MPC-Frameworks, die allerdings entweder effizient oder leicht anpassbar sind, aber nicht beides gleichzeitig. Als Folge müssen Forscher einen erheblichen Anteil ihrer Zeit mit Bemühungen verbringen, die existierenden MPC-Frameworks zu verstehen und zu modifizieren. Diese stürzen außerdem durch unvorsichtige Modifikationen häufig ab. Dies führt zu beträchtlicher Verlangsamung der MPC-Forschung.

Um dieses Problem anzugehen, entwickeln wir MOTION, ein quelloffenes C++-Framework für die Implementierung und Kombinierung der MPC-Protokolle und Primitive. Im Gegensatz zu früheren Ansätzen verfügt MOTION über beide Eigenschaften: Effizienz und Flexibilität. Außerdem ermöglicht MOTION die Implementierung und Kombinierung der MPC-Protokolle und MPC-Optimierungen. Dabei müssen nur die relevanten Komponenten des Frameworks verstanden beziehungsweise geändert werden, was dem asynchronen Design zu verdanken ist.

In MOTION sind bereits drei verschiedene generische MPC-Protokolle kombiniert, die zwei oder mehr Parteien zulassen. Darüber hinaus entwickeln und implementieren wir neue Konvertierungen zwischen einiger dieser Protokolle und optimieren die MPC-Grundbausteine. Schließlich zeigen wir, dass MOTION häufig wesentlich schneller ist als andere MPC-Frameworks.

Über MOTION hinaus, das für beliebige Anwendungen verwendet werden kann, erforschen wir auch eine spezifischere aber zugleich generische Funktionalitätsklasse. PSI ist ein weitverbreiteter Baustein für Privatsphäre-schützende Anwendungen, das eine lange Reihe von Forschungsergebnissen generiert hat. PSI ermöglicht zwei Parteien die Schnittmenge ihrer geheimen Eingabewerte zu berechnen, ohne etwas außer der Schnittmenge preiszugeben. In diesem Teil der Dissertation verbessern wir das Konzept von PSI, das auf Schaltkreisen basiert (im Englischen *circuit-based PSI*, C-PSI). Dieses ermöglicht weitere beliebige sichere Berechnungen auf dem Ergebnis von C-PSI, ohne die Schnittmenge preiszugeben. Beispielsweise können die Parteien komplexe Statistiken auf der Schnittmenge berechnen.

Wir konstruieren das erste Protokoll für C-PSI mit linearen Kommunikationskosten. Unser Protokoll basiert auf ausgefeilten Hashing-Techniken und "oblivious" programmierbaren pseudozufälligen Funktionen. Außerdem zeigen wir, dass unser Protokoll für C-PSI nicht nur niedrigere asymptotische Kommunikationskosten im Vergleich zu früheren Arbeiten aufweist, sondern auch konkret wesentlich niedrigere Rechen- und Kommunikationskosten hat.

Dieser Abschnitt der Dissertation basiert auf folgenden zwei Publikationen:

[BDST22] L. Braun, D. Demmler, T. Schneider, O. Tkachenko. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *Transactions on Privacy and Security (TOPS)* 25 (2 2022). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION. CORE Rank A. Appendix A.

[PSTY19] B. Pinkas, T. Schneider, O. Tkachenko, A. Yanai. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Bd. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, S. 122–153. CORE Rank A*. Appendix B.

MPC-Anwendungen & MPC-Optimierungen MPC wird derzeit noch nicht flächendeckend in praktischen Anwendungen verwendet. Der Hauptgrund dafür sind die hohen Kommunikations- sowie Rechenkosten von MPC im Vergleich zu Klartextberechungen. Darüber hinaus ist für die Konstruktion effizienter MPC-Protokolle Expertenwissen in den Bereichen Informatik, Kryptographie und Schaltkreis-Design notwendig. Tools wie HyCC (Büscher et al., CCS'19) heben diese Anforderung für kleine Anwendungen auf, sind aber für große und komplexe Probleme nicht einsetzbar.

In dieser Dissertation konstruieren wir MPC-basierte Protokolle für zwei praktische Anwendungen mit sensiblen Daten: 1) Indoor-Lokalisierung und 2) Similärgensequenzeanfragen. Um die Performance unserer Protokolle zu verbessern, verwenden wir optimierte Bausteine wie unseren derzeit kleinsten Schaltkreis für die Suche nach den k nächsten Nachbarn eines Eingabeelements.

Dieser Abschnitt der Dissertation basiert auf folgenden zwei Publikationen:

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E. S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, S. 448–463. Appendix C.

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Full version: https://ia.cr/2021/029. ACM, 2019, S. 315–327. CORE Rank A. Appendix D.

Diese Dissertation trägt dazu bei, die generische Nutzung von MPC flexibler sowie effizienter zu machen, was durch Optimierung von Grundbausteinen erreicht wird. Außerdem stellt diese Dissertation effiziente Privatsphäre-schützende Lösungen für zwei konkrete praktische Anwendungen vor.

My Contributions

This thesis comprises four scientific publications co-authored by me, my PhD advisor Thomas Schneider and several great researchers: Lennart Braun (Aarhus University, Denmark), Daniel Demmler (Universität Hamburg, Germany), Kimmo Järvinen (Xiphera Ltd., Finland), Helena Leppäkoski (Tampere University, Finland), Elena-Simona Lohan (Tampere University, Finland), Benny Pinkas (Bar-Ilan University, Israel), Philipp Richter (Tampere University, Finland), Avishay Yanai (VMware, Israel) and Zheng Yang (Southwest University, China). I thank all my co-authors for these very successful collaborations and detail my own contributions below.

Chapter 2 is based on a joint work with Lennart Braun, Daniel Demmler, and Thomas Schneider [**BDST22**] and on a joint work with Benny Pinkas, Thomas Schneider, and Avishay Yanai [**PSTY19**].
In [**BDST22**], in a collaborative effort we designed an open-source secure multi-party computation (MPC) framework, MOTION, which features 1) a unique flexible yet efficient design, 2) a unique combination of MPC protocols and 3) low-level optimizations of MPC protocols and conversions. With the indispensable help of my former student assistant Lennart Braun, I implemented, optimized, and evaluated MOTION, resulting into more than 30 thousand lines of code and tests. Furthermore, I improved the efficiency of a crucial generic MPC building block, correlated oblivious transfer, and implemented it in MOTION. Our framework is publicly available under the MIT license: `https://encrypto.de/code/MOTION`.
In [**PSTY19**], I contributed to the design of our circuit-based private set intersection (C-PSI) protocol by designing and implementing a practically efficient instantiation of our C-PSI protocol based on three-way Cuckoo hashing. Furthermore, I implemented, evaluated and compared it with prior best C-PSI protocol in terms of run time efficiency. The implementation of our protocol is open-source and licensed under the MIT license: `https://encrypto.de/code/OPPRF-PSI`.

Chapter 3 is based on a joint work with Kimmo Järvinen, Helena Leppäkoski, Elena-Simona Lohan, Philipp Richter, Thomas Schneider, and Zheng Yang [**JLL+19**], and on a joint work with Thomas Schneider [**ST19**].
In [**JLL+19**], I designed several highly efficient size-optimized and depth-optimized building blocks for computing indoor localization privately in an outsourcing scenario, i.e., for securely computing distance metrics, selecting elements with smallest distance, and obliviously accessing elements in a secret-shared array. Our size-optimized k-nearest neighbor circuit is currently the state of the art. I implemented and evaluated the designed building blocks and, using the quantization accuracy statistics from Philipp Richter, showed that we constructed the first practically efficient protocol for privacy-preserving indoor localization.
In [**ST19**], collaboratively with Thomas Schneider, I designed an efficient protocol for privately finding similar genomic sequences in an outsourced database that outperforms the previous best, homomorphic encryption-based, protocol by multiple orders of magnitude. I implemented and evaluated a prototype of our protocol and showed its applicability to large or whole-genome databases.

Contents

1 Introduction

Today's technological progress is growing at an exponential rate [But16]. Powerful technologies have been invented and the progress does not seem to stop in the near future. But with those technologies, unfortunately, also comes the danger of their misuse or mistakes due to negligence as well as human errors. Therefore, any new technology needs to be developed and used with caution, even if its use brings immediate benefits.

The motivation of this thesis is the ongoing digitization and data collection that explosively gained pace since the invention of the internet, which counted 4.66 billion active users worldwide as of January 2021 [Joh21]. The digitization is ubiquitous - most of our actions on the internet are logged. Some data like the URL of a visited website and the IP address is often logged by many parties simultaneously, e.g., by the web server, internet exchange point (IXP), etc. The digitization is large-scale - the annual amount of the internet traffic is predicted to triple from 2017 to 2022 and achieve a rate of 150 TB/s [Cis19].

The digitization affects most areas of our lives and transforms the way we perceive the world and communicate with it, producing enormous amounts of data that can be logged and stored by third parties. The data collection paves the way for data analysis that is useful in many ways, e.g., it can enable more powerful machine learning based solutions.

On the other hand, privacy is seen as a fundamental right by many. Furthermore, a user can have absolutely valid privacy concerns, e.g., with regard to how their data will be used by third parties or with regard to potential data leaks. However, there is often less a user can do to avoid the data collection except not using the respective service. Thus, it is crucial to find ways to protect privacy in the digital world while still being able to use the benefits of data.

Privacy regulations such as the general data protection regulation (GDPR) in the European Union (EU) have been imposed to protect by law the rights of people with respect to their privacy. However, despite that the users often strive for better privacy and that the privacy regulations such as GDPR are in place since a long time, a high level of privacy is yet notoriously hard to achieve [ABL20].

This problem is manyfold. There are very few privacy-preserving alternatives for many useful services that are privacy-preserving by design and scale to real-world data input sizes, e.g., private analyses on genomic data, where each DNA sample consists of millions of characters. Websites may track the activities of their visitors even if they opt out of third-party cookies [MBS20]. Data leaks may ruin the efforts for protecting privacy of honest and lawful parties [Gre21; con22; Red22].

Therefore, additional protecting measures are necessary to ensure the privacy of the users. The focus of this thesis is on secure multi-party computation (MPC), a cryptographic building block that can be used to provide services and functionalities that allow multiple parties to compute a public function on their private inputs and reveal nothing but the result of the computation. It is possible to construct services and applications that are privacy-preserving by design using MPC, but it incurs a large overhead in terms of communication and computation. The existing building blocks for MPC lack in efficiency and it is not trivial to come up with MPC protocols that solve complex real-world problems with acceptable overhead. Furthermore, the current tools for MPC are built for efficiency and are hard to use and extend, let alone adopt them in real-world applications. We address all aforementioned problems in this thesis.

Use Cases of MPC MPC finds application in many real-world use cases, e.g., in secure medical research [CWB18; **ST18**; **TWSH18**; **ST19**], location privacy [SCYW14; **RYT+18**; YJ18; **JLL+19**], secure distributed key management [LN18], private machine learning [ASKG19; WGC19] or inference [MZ17; JVC18; MR18; **RWT+18**; CCPS19; RSC+19; BCPS20; CRS20; KRC+20; RRK+20; KPPS21; PSSY21; KPRS22; PS22]. MPC is not only an instrument to enhance privacy in already existing yet privacy-invasive use cases such as machine learning, but it also acts as enabler of particular use cases such as distributed genomic analysis, where sharing the genomic data across medical institutions in the clear is prohibited by law.

Application of MPC received attention not only from the research community but also from industry. Recently, the MPC Alliance[1] was formed from mainly industrial companies to increase the market awareness and adoption of MPC. The first large-scale deployment of MPC is known to be for the private double auction organized by the Danish government [BCD+09] for privately computing fair market prices. Cybernetica[2], a company based in Estonia, provides a proprietary MPC-based system for generic secret sharing called Sharemind [BLW08], and it has been used in multiple case studies such as private satellite collision detection [Sha]. Sepior[3], Partisia[4] and Unbound Security[5] provide MPC based solutions for secure distributed key management. The latter has recently been taken over by Coinbase[6]. Intel has integrated the ABY framework [DSZ15] for secure two-party computation (2PC) into their nGraph-HE system [BLCW19] for private neural network inference [BCD+20]. In a recent cooperation, Mozilla and Meta designed a new system for privacy-preserving advertising [Tho22] based on MPC.

[1] https://www.mpcalliance.org
[2] https://cyber.ee
[3] https://sepior.com
[4] https://partisia.com
[5] https://www.unboundsecurity.com
[6] https://www.coinbase.com

2 Generic MPC Tools and Protocols

The paradigm of secure multi-party computation (MPC) went a long way from being a theoretical concept for playing mental games in the 1980s [GMW87] to practical protocols and prototypes that are available nowadays. The most widely used MPC protocols are garbled circuits (GCs) [Yao86; RR21] and GMW [GMW87].

The techniques used in MPC are becoming increasingly sophisticated. For example, a recent line of work combined multiple MPC protocols to use the most efficient technique for different parts of the computation to improve the overall efficiency [HKS+10; DSZ15; MR18; Kel20; ACC+21; PSSY21].

However, most of the existing MPC tools provide only low-level MPC components and require manual synchronization, and thus deep understanding of different parts of the used tool. This poses an obstacle in effectively combining MPC protocols and hinders a reuse of existing MPC implementations. Therefore, the key enabler of the real-world use of MPC are flexible yet efficient frameworks for implementing MPC protocols, optimizations and applications.

Furthermore, generic MPC building blocks are important for and find their use in many applications. The big advantage of the generic MPC building blocks compared with the special-purpose ones is that they can be reused and improved constructions will immediately result in better efficiency in multiple applications. Efficient special-purpose building blocks, on the other hand, need to be designed for each application separately, which requires expert knowledge and additional security proofs, and is thus time-consuming.

In this thesis, we focus on a generic MPC building block called circuit-based private set intersection (C-PSI). It does not reveal the intersection and can be used in combination with arbitrary functionalities. For example, C-PSI can be used for private contact discovery or private calculation of ad conversion rates. The extensive recent research indicates the still existing room for improvement in C-PSI.

This chapter is based on the following publications:

[BDST22] L. Braun, D. Demmler, T. Schneider, O. Tkachenko. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *Transactions on Privacy and Security (TOPS)* 25 (2 2022). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION. CORE Rank A. Appendix A.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, pp. 122–153. CORE Rank A*. Appendix B.

In this chapter, we describe the design and implementation of MOTION, our flexible and efficient MPC framework for implementing, combining, and benchmarking MPC protocols and applications (cf. section 2.1.1). The relevant publication can be found in appendix A. Furthermore, we describe the first protocol for C-PSI with only linear communication in the input size (cf. section 2.1.2). The relevant publication can be found in appendix B.

2.1 Our Contributions

2.1.1 Flexible and Efficient Framework for Implementing MPC

In this section, we describe our new flexible and efficient MPC framework dubbed MOTION [**BDST22**] as well as protocol-level improvements we introduce in our work. Our framework combines several important features that serve as a fundament for non-invasive integration and combination of efficient cryptographic primitives and protocols and even facilitate new deployment scenarios of MPC, e.g., MPC in middleware.

MOTION combines the arithmetic and boolean version of the Goldreich-Micali-Wigderson (GMW) protocol [GMW87] and the Beaver-Micali-Rogaway (BMR) protocol [BMR90; BLO16] based on oblivious transfer (OT) extension [IKNP03; ALSZ13], secure against semi-honest adversaries. Furthermore, we introduce a few protocol-level improvements:

1. We show how precomputed correlated oblivious transfer (C-OT), in contrast to general oblivious transfer (G-OT), can be used without increasing the number of communication rounds.
2. We give a practically efficient construction of the BMR protocol with low communication that, in contrast to prior work, uses (pre-computed) OT instances in a black-box way which simplifies its implementation.
3. We design a conversion from BMR to arithmetic Goldreich-Micali-Wigderson (A-GMW) that does not require interaction between the parties in the online phase, i.e., the interactive operations can be computed prior to when the inputs to the protocol are defined.

In the following, we give more details on our contributions.

Flexible implementation and combination of MPC protocols In MOTION the evaluation of a gate, i.e., a conversion or a primitive operation in an MPC protocol, is asynchronous and is powered by fibers using Boost Fibers[1], which can be considered as more efficient (user-space) threads. Beyond MOTION, we are aware of only one MPC framework, MPyC [Sch18], that provides a similar level of asynchrony but inherently worse efficiency due to the less efficient programming language (Python) it is implemented in. More concretely, each gate is submitted as a task to a fiber pool, which spawns a fiber for each task. The main feature of fibers is that they can arbitrarily be paused and resumed, e.g., if a gate needs to wait for another gate or a message from another party. This has the following advantages:

1. Implementation and combination of gates does not depend on the gate types or their dependencies, since it cannot block the overall execution, e.g., if the evaluation of a gate takes multiple (blocking) rounds until it is finished.
2. Fibers allow to decouple the implementations of the gates from the way they are evaluated, and they make the gates and wires "waitable", making the implementation and combination of different gates very flexible and easy to use, without the need of invasive changes.

The communication in MOTION is based on only serialized messages, omitting the need for synchronizing the communication of different gates and primitives. This also enables new use cases such as MPC in browsers using WebSockets or server applications that are not in full control of a network socket, while still being highly efficient. The reliance of the MPC software on raw network sockets has been considered as a major obstacle for deploying MPC prototypes in practical real-world applications [Hal18]. We give directions for future work regarding the use of communication channels in MOTION in section 4.2.2.

MOTION provides multiple layers of abstraction for different purposes. At the low abstraction level, the developer can control how the framework works in the finest details, e.g., it is easy to change the strategy of how particular gates or gate types are evaluated or to use primitives such as OT directly, or even to combine their use with the existing (higher-level) functionalities. Most cryptographic primitives and some MPC protocols implement their API via "providers", which give the developer an easy to use interface to request and use primitives and gates as C++ objects. On a higher level, the developer can use C++ objects to construct a circuit that builds primitive gates available in different protocols, e.g., XOR in BMR and boolean GMW. The currently highest level of abstraction in MOTION implements optimized circuits for particular operations, e.g., integer arithmetic, which can be constructed using simple C++ operators.

Our framework gives the developer full control over its functionality but it does not automatically optimize the implemented circuits and thus requires expert knowledge in circuit design and cryptography. To make MOTION available to developers without expert knowledge, we integrate the HyCC toolchain [BDK+18] into MOTION. HyCC allows to write programs in ANSI C that automatically get translated to optimized circuits for mixed-protocol MPC.

[1] https://www.boost.org/doc/libs/1_78_0/libs/fiber/doc/html/index.html

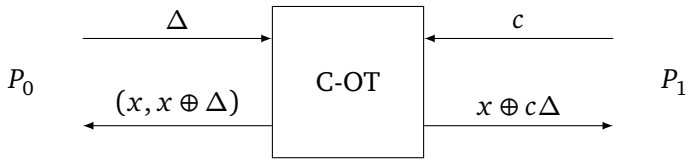

Figure 2.1: Ideal functionality of the correlated oblivious transfer (C-OT) [ALSZ13].

C-OT with one online communication round Correlated oblivious transfer (C-OT) is a building block widely used in MPC to realize AND, multiplication and multiplexer gates [ALSZ13]. The ideal functionality of C-OT is depicted in Figure 2.1. The communication complexity for using C-OT on ℓ-bit Δ is $\kappa + \ell$ bits, where κ denotes the symmetric security parameter. C-OT can be precomputed on random values [Bea95] and the computation of C-OT is split into two phases: the phase where the inputs are not yet known, which we denote as setup, and the phase where the inputs are known, which we denote as online. The advantage of the setup/online paradigm is that the setup phase can be precomputed at any time when the parties are idle, e.g., overnight. In the setup phase, the communication is κ bits and in the online phase only $\ell + 1$ bits, where 1 bit is sent for the correction if the randomly chosen selection bit was incorrect. The reduction of the communication in the online phase by κ bits makes a significant difference especially for small values of ℓ, since κ is the dominant factor in the communication requirements of most practical applications of C-OT.

OT precomputation was initially designed for general oblivious transfer (G-OT), where the sender holds two messages and the receiver holds a selection bit. The online phase of precomputed G-OT consists of two consecutive steps: 1) the receiver sends the correction bit and 2) the sender swaps particular parts if the correction bit was 1. The communication rounds in the aforementioned protocol are prohibitive for MPC protocols such as GMW, which requires $O(d \cdot \mathsf{AND})$ communication rounds, where d is the circuit multiplicative depth and AND is the number of communication rounds required to evaluate an AND gate. To circumvent this, Beaver's triples [Bea91] are normally used, which increase the protocol and implementation complexity.

In our work, we show how to compute C-OT in only one online round instead of two. In a nutshell, we observe that, in contrast to G-OT, the message sent by the sender in the online phase is always equivalent, with swapped terms how it is computed. The term that can change due to the correction bit value is the output of the sender, which gets swapped if the correction bit equals 1. Our protocol has approximately equal communication complexity as the Beaver triple-based protocol but requires one instead of two communication rounds and is much simpler to compute. Furthermore, pre-computed OTs are substantially easier to use in flexible MPC frameworks that employ the setup-online computation paradigm such as MOTION.

More efficient BMR garbling The BMR garbling can abstractly be seen as an extension of the two-party Yao's garbling [Yao86]. In the latter, a garbler "encrypts" the function to compute, represented as a Boolean circuit, and sends it to the receiver along with the output

keys. The evaluator can "evaluate" or process the garbled circuit and obtain the encrypted outputs, which can be decrypted using the output keys.

The BMR protocol extends the concept of garbling to the setting with more than two parties. In a nutshell, all parties act as garblers as well as the evaluators. In contrast to Yao garbling, BMR garbling is performed in a secure protocol, s.t. no single party knows the garbling data in the clear. In this way, BMR garbling is secure against all but one corrupted parties.

Ben-Efraim et al. [BLO16] proposed an efficient protocol for BMR garbling based on two invocations of C-OT based secure multiplication protocols. The first C-OT is used to securely multiply the secret-shared permutation bits, which determine if the cleartext value on the wire is inverted. The second C-OT is used to multiply the secret-shared permutation of the output wire with the secret-shared key belonging to the gate. In our work, we show how to reduce the communication required for the second multiplication.

Each invocation of C-OT requires $\kappa + \ell$ bits of communication [ALSZ13], where ℓ is the bit length of the transmitted message and κ is the symmetric security parameter. We show that the κ term can be removed for the second multiplication if we "inverse" the order of the multiplication, i.e., we multiply each bit in the key with a bit string containing a batch of output permutation bits. This approach results in an amortized communication improvement of BMR garbling by approximately a factor of 2 for large batch sizes compared to [BLO16].

BMR to arithmetic GMW w/o online interaction We provide a new gate construction for converting a value x shared in BMR to arithmetic GMW without interaction in the online phase.

Our new conversion is based on the use of a random value r that is shared in BMR as well as arithmetic GMW. Since r is random, it can be shared in the setup phase. First, all parties generate a random share of r in arithmetic GMW, which can be done locally. Then, they use a protocol for converting an arithmetic GMW share to a BMR share, which is another, existing conversion in MOTION. In the online phase, the parties 1) subtract r from x in BMR, 2) reveal it to one of the parties and 3) add the pre-shared r to x to obtain a valid arithmetic GMW share of x. All of the aforementioned operations in the online phase are local, since 1) the BMR circuit is evaluated locally, 2) the output keys in BMR are transmitted already in the setup phase and 3) addition is a local operation in arithmetic GMW. Additionally, we provide an optimization that reduces the communication by a linear factor in the number of parties at the cost of increasing the number of communication rounds in the setup phase.

Efficiency The most important building block in MOTION is OT extension [IKNP03; ALSZ13]. MOTION is only slightly less efficient for generating OTs compared to the libOTe library [Rin], which is highly optimized on the low level. The difference in the abstraction level in MOTION and libOTe is that the latter runs a particular type of OT, e.g., G-OT on 128-bit string, and blocks a network socket. On the one hand, the low level of abstraction in libOTe enables implementation optimizations such as more efficient buffer allocation and alignment for predefined bit lengths. On the other hand, using multiple OT types requires

either blocking a network socket and pipelining different OTs, or it can be run using parallel sockets. However, either of the mentioned solutions in libOTe is manual and quickly gets infeasible for complex MPC protocols. MOTION, in contrast, allows for arbitrary, flexible composition of any of the implemented OTs that use the same network socket, together with the used MPC protocols. For MOTION, we measured 10–60 % slower run times compared to libOTe, depending on the OT flavor and network setting. Considering the much higher level of abstraction and communication serialization in MOTION, we see this as acceptable overhead and leave further low-level optimizations, which we believe are possible, as future work.

Regarding the efficiency of MPC, MOTION provides comparable or even better run-time efficiency compared to other MPC frameworks in many cases, despite the high level of abstraction and communication serialization. For example, our experiments showed that MOTION is faster than MP-SPDZ [Kel20] by at least 8.9× when evaluating a batch of hybrid circuits for biometric matching. For evaluating a batch of AES or SHA circuits, MOTION shows efficiency comparable to MP-SPDZ and is from equally fast to 9× faster than the GMW implementation by Choi et al. [CHK+12]. The interested reader will find more details about our benchmarking parameters and results in section 8 of Appendix A.

To summarize, MOTION was either comparably fast or faster than other frameworks. The only exception is the well-established ABY framework [DSZ15] for secure two-party computation (2PC), which was always most efficient in our experiments among all the benchmarked frameworks. However, ABY is a 2PC framework and is very challenging to extend to generic MPC due to many low-level optimizations and general lack of flexibility.

We found two aspects to be crucial for the efficiency of MOTION:

1. The high abstraction level in MOTION leads to higher run times, especially due to the dynamic synchronizations of gates and primitives. The way we chose to reduce this overhead is to use single instruction multiple data (SIMD) instructions, which allow to evaluate gates in a vectorized fashion. This enables evaluation of multiple gates at the same time, e.g., for computing the dot product, and/or parallel evaluation of multiple circuits. SIMD instructions are also available in ABY [DSZ15] and MP-SPDZ [Kel20].

2. Due to its high flexibility, MOTION often allocates memory for small (batches of) objects. Contrary to that, less flexible frameworks often hard code such parameters allowing for more efficient memory allocation. The standard memory allocation in C++, i.e., using `malloc`[2], is often slow for small memory buffers of flexible sizes, especially in highly parallel programs [LBC+19]. Fortunately, many more suitable memory allocation libraries exist such as Google's `tcmalloc`[3], which is available in MOTION. Our experiments showed that `tcmalloc` has a significant positive effect on the performance of MOTION but not on the performance of other, less flexible MPC frameworks. Finding the most

[2] https://linux.die.net/man/3/malloc
[3] http://goog-perftools.sourceforge.net/doc/tcmalloc.html

efficient method for memory allocation in MOTION and other MPC frameworks is subject to further research.

2.1.2 First C-PSI protocol with linear communication

One of generic building blocks that can be used in MPC is circuit-based private set intersection (C-PSI), which allows two parties to privately compute the set intersection on their private inputs, yielding the secret-shared output that can be used in arbitrary further computations. The advantage of state-of-the-art C-PSI protocols over MPC based C-PSI protocols is that the former are more efficient both asymptotically and concretely. C-PSI is interesting for several real-world applications, e.g., in computing genetic similarity or in joining databases. Another line of work aimed at designing special-purpose protocols for particular functionalities such as private set intersection (PSI) cardinality [CGT12; DD15; EFG+15], where only the intersection size is revealed, or PSI sum [IKN+17; IKN+20; MPR+20], where the output of the PSI protocol is the sum of the payloads corresponding to the matched elements. Some special-purpose PSI protocols can be simpler or more efficient than their C-PSI analogues, but they are not trivial to extend to other functionalities and normally require additional proofs of security, in contrast to C-PSI.

Here, we describe our work on C-PSI with linear communication, which is asymptotically and concretely more efficient than the C-PSI protocol by Pinkas et al. [PSWW18] with $\omega(n)$ complexity, where n is the input size of each party. Our construction is strongly inspired by the OT phasing protocol introduced by Pinkas et al. [PSSZ15], which is based on the hashing to bins approach. Their construction relies on the interplay between two hash tables of linear size, where one party uses Cuckoo hashing [PR01] and the other party uses simple hashing. To map input elements to their hash tables, the parties use h different hash functions. Using Cuckoo hashing, each bin in the table contains only one element. The respective party tries to insert all their input elements to the table and invokes a remapping procedure in case of collision. If a collision cannot be resolved, the element is added to a special list called stash. Using simple hashing, each bin can contain multiple elements and all elements in the party's input set are added to the bins for each of the h hash functions. It is easy to see that if one party uses Cuckoo hashing and the other party uses simple hashing for mapping their respective elements, it is guaranteed that computing the set inclusion on each pair of elements would yield the set intersection. The authors of [PSSZ15] show that the maximum size of a bin in the simple hash table for their parameters is $O(\log n / \log\log n)$ and the total complexity is thus $O(n \log n / \log\log n)$.

The 2D Cuckoo hashing [PSWW18] shows an approach with only constant a number of comparisons per bin but it still requires a stash of size $\omega(1)$, where each element of stash must be compared to all elements in other party's input set. Thus, this yields a total complexity of $\omega(n)$ due to stash.

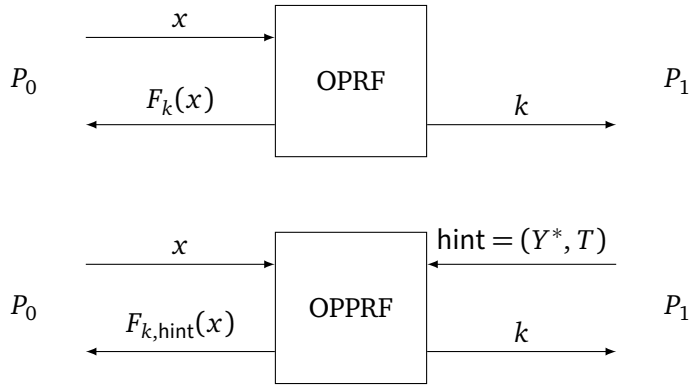

Figure 2.2: Ideal functionalities of oblivious pseudo-random function (OPRF) and oblivious programmable pseudo-random function (OPPRF). In OPRF party P_0 obtains $F_k(x)$ for its input x and party P_1 obtains k that can be used to locally compute $F_k(y)$ for all y in its input set Y. The OPPRF functionality is similar with the only difference that P_1 inputs a hint that contains input-target pairs of form (y_i^*, t_i), where $y_i^* \in Y^*, t_i \in T$, making $F_{k,\mathsf{hint}}(x)$ point to $t_i \in T$ if $x == y_i \in Y^*$ or to some pseudo-random value otherwise. Similarly to OPRF, P_1 can invoke $F_{k,\mathsf{hint}}(y)$ locally on any values of y.

Efficient C-PSI based on OPPRF The C-PSI protocol is based on a primitive called oblivious programmable pseudo-random function (OPPRF) that is based on a simpler functionality called oblivious pseudo-random function (OPRF). In a nutshell, OPPRF allows two parties, P_0 and P_1, to evaluate a pseudo-random function F, where P_1 can "program" OPPRF to point to particular target values for particular inputs, while producing pseudo-random outputs for other input values. The communication complexity of OPPRF is linear in the number of programmed values. We depict and briefly explain both functionalities in Figure 2.2.

The underlying idea of our C-PSI protocol is to invoke OPPRF for each respective pair of bins. Namely, for each pair of bins at position i, P_0 computes $F_{k,\mathsf{hint}}(\mathsf{CH}_i)$, where CH_i is the element in bin i of the Cuckoo hash table, and P_1 programs OPPRF to point to a pseudo-random target value t_i for all its values in SH_i, i.e., the elements in bin i of the simple hash table. This reduces the comparison of both bins to comparing $F_{k,\mathsf{hint}}(\mathsf{CH}_i)$ with t_i. Unfortunately, this still has $\log n / \log\log n$ complexity per pair of bins, since P_1 needs to encode the maximum number of values to hide the real number of elements in the bin. However, since the total number of elements in both hash tables is known to both parties, we show how OPPRF can be extended to batch OPPRF to reduce the communication complexity to only $\mathrm{O}(1)$ per bin. The OPPRF construction we use is based on polynomial interpolation and has either $\mathrm{O}(n^2)$ complexity and excellent practical efficiency using the textbook Lagrange interpolation algorithm or $\mathrm{O}(n \log n)$ complexity and mediocre practical efficiency using Fast Fourier Transformation (FFT). We show how the tables can be split into "mega-bins", thus reducing the number of bins per one interpolation and getting the best of both worlds.

At this point, the only remaining problem is the $\omega(n)$ communication complexity for processing the stash. We evaluate two methods for mitigating this problem: 1) a novel dual execution approach and 2) the use of 2 instead of 3 hash functions that, on the one hand, increases the size of the hash table, but, on the other hand, omits the stash due to significantly better bin utilization.

C-PSI with linear communication via dual execution We make the observation that the following elegant approach leads to linear complexity of superlinear C-PSI protocols, whose superlinearity is only due to the stash. Our optimization has three steps:

1. The parties run a C-PSI protocol with stash, where P_0 ignores their $\texttt{Stash}_0$, and they obtain $\texttt{Output}_1$. This subprotocol has linear complexity.
2. The parties reverse roles, where P_0 adds elements from $\texttt{Stash}_0$ as input to their simple hash table and P_1 adds all their elements to a Cuckoo hash table and the parties obtain $\texttt{Output}_2$. As in the first step, P_1 ignores their $\texttt{Stash}_1$ in the protocol. This subprotocol also has linear complexity.
3. The parties compute the "naïve" set intersection on $\texttt{Stash}_0$ and $\texttt{Stash}_1$ as inputs, where they compare each element against each other's element in MPC, and they obtain $\texttt{Output}_3$. Their stashes are of size $O(\log n)$ and this subprotocol has thus only sublinear complexity, i.e., $O(\log^2 n)$.

Observe that $\texttt{Output}_1||\texttt{Output}_2||\texttt{Output}_3$ is exactly the intersection between both parties' input sets, where $||$ denotes the concatenation.

More efficient C-PSI with linear communication via three-way Cuckoo hashing As another possibility to realize C-PSI, we consider a construction inspired by circuit phasing-based C-PSI [PSSZ15; PSZ18], where it was shown that the stash can be avoided with high probability using three or more hash functions due to the significantly better bin utilization using more than two hash functions. This approach results in even better communication and computation than dual execution.

Linear C-PSI with payloads In the previous best C-PSI protocol [PSWW18], both parties map their elements to bins of a constant size. Hence, it is trivial for them to add payloads corresponding to the elements, e.g., the amount of money spent in different stores by the same person. In our protocol, P_0 can trivially add their payloads as a separate table, since there is always only at most one element in the hash table. For P_1 it is less trivial since each bin may contain multiple elements due to simple hashing. Naïvely finding the correct payload for a matched element would result in the superlinear overall complexity.

To circumvent this, we design an additional protocol for mapping the correct payloads from the simple hash table based on OPPRF. Here, the parties first run our C-PSI protocol and obtain secret-shared zeros or ones for each bin in their tables. Next, they reuse their hash tables in the following way: for each bin, P_1 holds a pair (y, p) for each element y with p being the respective payload, and they generate a uniformly random target value t of

the maximum length of the payload. Then, P_1 programs OPPRF to map to $t \oplus p$ on input y and sets t as its secret share of the payload, thus creating a valid share of $(t \oplus p, t)$ if $x == y$. Similarly to our C-PSI protocol, OPPRF is batched over multiple bins to achieve linear communication complexity.

Efficiency Compared to the previous-best C-PSI protocol [PSWW18], our construction based on three-way Cuckoo hashing is 2.8–4.2× faster in local area network (LAN) and 3.9–5.8× faster in wide area network (WAN). We identify two main bottlenecks of our protocol in terms of run time: 1) circuit evaluation and 2) polynomial interpolation. The first is due to the "textbook" comparisons of bins and the hash table size blowup due to Cuckoo hashing. Improving the comparison protocol or reducing the table size would immediately improve the run time efficiency. The second is due to the asymptotically expensive polynomial interpolation which is one of the main building blocks of our OPPRF protocol. A more efficient OPPRF protocol would immediately improve the run time efficiency. In terms of communication, our protocol is 10.1–12.8× more efficient than [PSWW18] for arbitrary input bit lengths.

2.2 Related work

In this section, we describe the MPC frameworks related to MOTION [**BDST22**] such as frameworks for mixed-protocol MPC and flexible MPC frameworks that allow non-invasive modifications. The interested reader will find some additional information on several MPC frameworks in the systematization of knowledge by Hastings et al. [HHNZ19].

2.2.1 MPC Frameworks

Hybrid MPC frameworks Here, we give an overview on MPC frameworks that allow to mix MPC protocols. Unless stated otherwise, the described frameworks are secure against semi-honest adversaries.

Sharemind [BLW08] is a proprietary, closed-source framework that combines arithmetic and boolean secret sharing in the three-party honest majority setting and operates in both security settings: passive and active.

TASTY [HKS+10] is a Python framework that combines GCs and additively homomorphic encryption (AHE) for 2PC. The idea behind TASTY is to securely compute arithmetic parts of a function using AHE and use GCs to compute the non-linear parts of the function such as comparisons. This approach reduces the communication for secure computation of many functions.

The ABY framework [DSZ15] is a very efficient and popular in the MPC community tool for mixed-protocol semi-honest 2PC, implemented in C++. In terms of flexibility, unfortunately, ABY constitutes several blocks of highly coherent code that is hard to modify without breaking

the existing functionality. The main reason we suspect to be the cause of the lack of flexibility in ABY is that efficiency can often be improved by writing code that is fine-tuned for a specific task, consequently sacrificing the flexibility.

ABY^3 [MR18] is a framework for privacy-preserving machine learning (ML) with both passive or active security based on the combination of arithmetic and boolean secret sharing as well as GCs in the three-party honest majority setting.

MP-SPDZ [Kel20] is a large set of passively as well as actively secure MPC protocols implemented in C++. It supports a variety of security models and corruption thresholds. Unfortunately, the MPC protocols implemented in MP-SPDZ are yet not trivial to combine.

SCALE-MAMBA [ACC+21] is a toolset for, mainly, actively secure full-threshold MPC. It consists of a domain-specific language MAMBA and an "MPC engine" that is run by a backend called SCALE. It is a closed system which, naturally, limits its use in combination with other systems. Moreover, although its extensive documentation provides many details on how SCALE works and how user instructions can be used and extended, there is very little information about integrating and combining new MPC protocols.

Flexible MPC frameworks MPyC [Sch18] is a flexible MPC framework implemented in Python. Although MPyC shares some features with MOTION, e.g., asynchronous gate evaluation that allows non-invasive addition, modification and combination of gates and functionalities, it is not positioned as a highly efficient framework, mainly because its code is implemented in a scripting language. Furthermore, the communication in MPyC is not serialized which limits its flexibility. The main goal of MPyC is to provide better usability, e.g., for educational purposes.

Follow-up works Recently, Braun et al. [BCS21] extended MOTION with the ABY2.0 [PSSY21] 2PC protocols and conversions for secure ML inference. Furthermore, they implemented an interpreter for the commonly used open neural network exchange (ONNX)[4] format that is directly run as a secure protocol in MOTION [**BDST22**].

2.2.2 C-PSI Protocols

Huang et al. [HEK12] were the first to overcome the quadratic computation and communication of the "naïve" C-PSI — where each element in X is compared with each element in Y — using their sort-compare-shuffle approach and achieved $O(n \log n)$ overhead. In a nutshell, the parties 1) locally sort their own input sets, 2) securely merge the two sorted lists into one sorted list, 3) securely remove duplicates and 4) securely randomly shuffle the result. Another interesting aspect of this scheme is that it largely relies on MPC in a black-box way and thus can be extended to different settings, e.g., to C-PSI where parties do not hold the elements in the clear or to C-PSI with more than two parties. The latter, however, would increase the circuit size by up to $O(\log n)$ due to less efficient merging.

[4] https://onnx.ai/

Pinkas et al. [PSSZ15] designed PSI and C-PSI protocols, whose ideas are still used in the most efficient state-of-the-art protocols. They proposed to use Cuckoo hashing [PR01], which allows to bound the position of an element in the hash table of size $O(n)$ to only a constant number of possible addresses. Cuckoo hashing uses h different hash functions, tried out iteratively in case of collision, to map each element to an empty bin in the hash table. If an element cannot be mapped to an empty bin directly or by remapping other elements in the table that cause the collision, the element gets stored in a special list called stash. While one party uses Cuckoo hashing, the other party maps its elements using simple hashing, where each element is added to all bins to which the hash functions map to, and bound the maximum size of a bin to $O(\log n / \log\log n)$. In this way, Pinkas et al. reduce the problem of computing C-PSI to securely computing set inclusion between the respective bins in the Cuckoo and simple hash table, resulting in $O(n \log n / \log\log n)$ overall communication complexity. Although each element in the stash needs to be compared with each element from the input set, the stash contains maximally only $\omega(1)$ elements, thus having no negative effect on the total complexity.

A subsequent work [PSWW18] introduced a 2D Cuckoo hashing technique for C-PSI, which reduces the number of comparisons for comparing the elements in a pair of bins to $O(1)$ by more effectively mapping elements to bins. This not only reduces the asymptotic complexity but also significantly improves the concrete efficiency compared to [PSSZ15]. However, this scheme still requires a stash of size $\omega(1)$ resulting in a total communication of $\omega(n)$.

In [**PSTY19**], we improve over [PSSZ15] and [PSWW18] and introduce the first C-PSI scheme with linear communication complexity. Similarly to [PSSZ15] and [PSWW18], our scheme is also based on Cuckoo hashing.

Follow-up works Multiple works further improved on [**PSTY19**] in terms of concrete efficiency.

Rindal and Schoppmann [RS21] designed a C-PSI protocol with not only linear communication but also linear computation. They use vector oblivious linear evaluation (VOLE) and silent oblivious transfer (S-OT) [SGRR19] to reduce the concrete communication over [**PSTY19**] at the cost of higher computation and analyze trade-offs between higher computation or communication. Moreover, they use their ideas to construct efficient *PSI* protocols (i.e., with plaintext outputs) for both semi-honest and malicious setting, taking inspiration by [PRTY20] for the latter.

Garimella et al. [GMR+21] introduced new protocols for C-PSI and circuit-based private set union (C-PSU) using oblivious switching. Their C-PSI protocol has comparable running time and slightly better communication than [**PSTY19**] but leaks the intersection size.

Chandrad et al. [CGS22] improve the construction of [**PSTY19**] for better performance. Concretely, they construct a new primitive called relaxed batch OPPRF, which has only linear computation instead of $O(n \log n)$ or $O(n^2)$ in [**PSTY19**]. Furthermore, their concrete efficiency is also, in total, better by 2.8× in terms of computation and by 2.3× in terms of communication.

3 MPC Applications & Optimizations

There are many use cases in the real world where user privacy is violated and privacy-preserving alternatives are not available or where it is impossible to provide a service due to privacy regulations. Secure multi-party computation (MPC) can realize privacy protection for arbitrary applications and thus has the potential to solve the aforementioned issue. However, efficient MPC protocols are challenging to design, and it is often even unclear whether a practically efficient MPC solution is possible for a specific problem, especially in the current era of big data and low-latency services.

In this chapter, we describe our contributions to designing efficient secure two-party computation (2PC) protocols in the outsourcing scenario that solve real-world problems in the areas of localization and genomic privacy.

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E. S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 448–463. Appendix C.

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Full version: https://ia.cr/2021/029. ACM, 2019, pp. 315–327. CORE Rank A. Appendix D.

In the first part of this chapter (cf. section 3.1.1), we detail our efficient 2PC design and optimizations for the problem of privacy-preserving indoor localization (PPIL). This can be used for example if a user wants to compute their coordinate in a shopping mall but does not want to reveal their location to the server. To improve the efficiency, we quantize Wi-Fi signals to a smaller dimension and verify the resulting accuracy in a field experiment. Furthermore, we provide multiple optimizations on the algorithmic and protocol level. Our solution achieves a subsecond response latency and is the first that may be sufficiently efficient for the real-world use. The respective paper can be found in Appendix C.

In the second part of this chapter (cf. section 3.1.2), we describe our 2PC-based solution to the problem of securely finding similar genomic sequences in a large outsourced database, aggregated from several smaller databases. Our solution extends a prior work by Asharov et al. [AHLR18] with an efficient building block to enable 2PC outsourcing. Compared to the previous best protocol, which is based on homomorphic encryption (HE), our protocol is by multiple orders of magnitude more efficient in terms of both computation and communication. More details can be found in Appendix D.

3.1 Our Contributions

3.1.1 First Practically Efficient Protocol for Privacy-Preserving Indoor Localization Based on MPC

Indoor localization (IL) is a valuable service, e.g., for navigation to the sought gate in an airport or finding a shop or a cinema in a shopping mall. There exist different methods for IL, e.g., based on bluetooth low energy (BLE) or Wi-Fi fingerprints. The latter is especially useful, since it requires only the existing infrastructure, i.e., the Wi-Fi access points (APs), and thus can be used "out of the box". On the other hand, global navigation satellite systems (GNSS) such as the widely used global positioning system (GPS) often provide insufficient signal strength inside buildings and thus are not suitable for IL.

In non-private IL, the client sends their collected Wi-Fi fingerprint to the server that computes the client's coordinates and returns the result. However, the privacy of the user is violated in this scenario, since the server can track the user across the building.

Alternatively, the server can send its IL model to the client, and the client performs localization locally. However, the construction of a good IL model is a time-consuming and costly process because the time and device bias must be eliminated for good accuracy. Therefore, the server may not want to reveal the model to the client due to economic incentives.

PPIL is a solution that enables IL without the user sacrificing their privacy or the server losing economic incentives for providing the IL service. However, PPIL requires private computation based on cryptography and increases the cost of IL significantly. Namely, PPIL based on MPC requires hundreds of megabytes of computation and significant computation which is prohibitive for mobile clients.

In our work, we provide the first practically efficient PPIL protocols. We consider an outsourcing scenario where two non-colluding servers run MPC. We find and compare the most MPC-friendly distance metrics. To reduce the cost of PPIL, we explore signal quantization to reduce the bit length of the signals and show its effect on IL accuracy in a field experiment. We explore two directions for optimizing our PPIL protocols: optimizations for low depth and for low size of the circuit. Our protocols are based on custom-tailored building blocks and improve some of them, e.g., our size-optimized circuit for computing k nearest neighbors (k-NN) is currently the state of the art.

MPC-friendly distance metrics We find four MPC-friendly distance metrics: Manhattan, Euclidean, Sørensen and Kumar-Hassebrook. These distance metrics differ in terms of efficiency and accuracy, but they have in common that they can be computed and compared using only additions, multiplications and comparisons, which are MPC-friendly operations. Furthermore, we devise a new distance metric from the Sørensen distance, which is more friendly for arithmetic operations and show that it provides accuracy comparable to the original Sørensen distance.

Signal quantization Wi-Fi signal strengths are normally stored as 8-bit values. To improve the performance of our protocols, we investigate the effect of the quantization of Wi-Fi signal strengths to smaller values. We conduct a field experiment in a real building and show that quantization to 4 bits reduces the accuracy by 0.1–5 %, depending on the distance metric. Moreover, even the quantization to 2 and 1 bits still achieves meaningful accuracy. Using 1-bit quantization, we design a garbled circuit (GC) based PPIL protocol that may be efficient enough for PPIL without outsourcing.

Size-optimized vs. depth-optimized circuit for PPIL It is known that different MPC protocols have different performance tradeoffs and thus require different approaches to achieve the best performance [SZ13]. However, finding the best suited building blocks and their combinations is a non-trivial task and is similar to finding the best algorithm for solving a problem and/or compiling high-level code to the most efficient assembly representation.

Our PPIL protocols can be split into two high-level parts: distance computation and finding k minima along with their respective coordinates. We explore two directions for optimizing our PPIL protocols: for circuit size and for circuit depth. For the size-optimized protocols, we compute the k nearest neighbors (k-NN) algorithm and for the depth-optimized algorithms we compute the Bitonic sort and take the first k elements. In total, we devise 13 different PPIL protocols and analyze their performance.

Furthermore, we devise new efficient building blocks: 1) a size-optimized k-NN circuit that is by 1.6× smaller for k=3 neighbors than the previous best circuit [SHSK15] due to a more efficient swap operation, and 2) a single instruction multiple data (SIMD) friendly low-depth circuit for oblivious array access, which is important for practical efficiency.

Efficiency Our most efficient PPIL protocol is based on computing the Euclidean distance in arithmetic Goldreich-Micali-Wigderson (GMW) and the subsequent coordinate computation in a GC using our k-NN circuit. With at least 4-bit long received signal strengths (RSSs), it runs in 1 s of total time and only 0.15 s of online time, and it requires only 167 MB of total communication and 2.7 MB of online communication. Our only faster protocol is the protocol for computing the Manhattan distance completely in a GC on 1-bit RSSs, where the computation of the distance between two coordinates boils down to computing the Hamming distance, which is significantly more efficient. This protocol runs in only 0.5 s of total time and 0.2 s of online time, and it requires only 5.2 MB of total communication and only a few dozen bytes of online communication. This enhanced efficiency at the cost of lower accuracy makes our 1-bit protocol a good candidate for the use on mobile clients without outsourcing. On the other hand, our PPIL protocols using more accurate distance metrics require 2.6–3.0 s of total run time and slightly less than one second of online time, making them usable in certain scenarios, e.g., if a user is localized continuously but not in real time and/or the use case requires high accuracy.

3.1.2 Efficient MPC protocol for SSQ

Recent advances in and the subsequent drastic reduction in costs of genome sequencing has given rise to the now booming personal genomics industry. However, the current state of affairs is that the user entrusts their data to the service provider in the clear. And although the law is very strict on how to handle genomic or medical data in general, the service provider in this case is trusted, thus increasing the danger of a misuse.

In contrast to other types of personal data, genomic data contains an exceptional amount of information about the individual and their relatives [HAHT15; Ayd16]. The contained information ranges from the eye color to predispositions to particular diseases, and this is only what we already know today. The leakage of genomic data may cause concrete harm, e.g., if a person does not get a job due to a predisposition to a disease or has to pay a higher price for the health insurance.

On the other hand, personal genomics can bring a tremendous improvement to many people's lives. The use case that we consider in this work is the querying of a genomic database for finding genomic sequences that are similar to the input sequence. This may not only help to find (yet unknown) relatives but also to improve healthcare, e.g., a doctor may obtain additional information from diagnoses of similar patients if making a diagnose for their patient is difficult. Another use case for similar sequence queries (SSQs) is crime solving [Jon10], which helped to solve some widely known cold cases in criminal investigations [Jon18]. However, all these use cases may violate the privacy of the genomic data owners, since significant amounts of sensitive data are revealed in each cleartext query.

Methods The underlying technique for finding similar genomic sequences is called SSQ and it is realized using the edit distance (ED), which computes the minimum number of additions, deletions and substitutions required to transform one string into another. It has quadratic complexity in the string size that is in the order of millions of symbols for the parts of the genome that change from person to person. The quadratic blowup draws the straightforward translation of the SSQ functionality to MPC being infeasible to run on real data.

Methods for reducing the complexity of ED exist, although by reducing accuracy. Their MPC counterparts are custom-tailored to the two-party setting and do no allow secure data aggregation from many parties due to the assumption that both servers see their own data in the clear. We extend the protocol of Asharov et al. [AHLR18] to the outsourcing setting and replace their main building block by a multiplication of a secret-shared bit by a secret-shared integer. We show a simple way how to implement this multiplication at the same communication complexity as the simpler case of the multiplication of a secret-shared string by a secret-shared bit. The resulting SSQ protocol has only twice the communication of the original protocol.

Outsourced computation of approximate ED To reduce the complexity of computing the ED, Asharov et al. [AHLR18] introduce a look-up table (LUT)-based approach for computing the approximate ED. In a nutshell, they precompute EDs for the variants of the genome that

were actually observed in their local database. Then, the client selects a LUT value using oblivious transfer (OT). The approximation part of this approach comes into play when a desired ED has not been precomputed by the server. In this case, an error is likely to occur, which the authors empirically analyze and show that it is small.

Their protocol relies on a multiplication of cleartext values held by the client and server, respectively. We replace their multiplication of cleartext values by a multiplication of secret shares and show that two additive correlated oblivious transfers (C-OTs) [ALSZ13] are sufficient for a multiplication of a secret-shared integer by a secret-shared bit, similar to the simpler case of secret-shared bit-string multiplication based on C-OT [ALSZ13]. Furthermore, we discuss the complexity of aggregating LUTs of many clients into one, potentially more accurate LUT, which, however, incurs additional overhead.

Efficiency We explore the limits of our outsourced SSQ protocol by analyzing how well it performs for large databases and whole genomes. For a database containing a million 10-thousand-character-long sequences, our protocol requires less than an hour and a quarter of terabyte communication in a local area network (LAN). For a database containing a thousand whole genomes, each consisting of 75 million characters, our protocol requires half a day and 7 TB of communication. Compared to the previous state of the art [CHW18], our protocol is by more than 20,000× faster and requires at least 16× less communication. More details can be found in appendix D.

3.2 Related Work

3.2.1 Privacy-Preserving Indoor Localization

A few early works [LSZ+14; SCYW14; ZVHW14] mainly used Paillier-based HE [Pai99] to construct protocols for PPIL. However, they require dozens of seconds or even minutes to run and thus do not meet real-world efficiency demands. Konstantinidis et al. [KCZ+15] proposed to use k-anonymity [Swe02] to preserve privacy in PPIL, which is based on adding noise to data and, therefore, orthogonal to our work. [ZCZL16] propose an efficient HE-based PPIL protocol, which, however, we showed to be insecure [**RST19**]. Jang and Järvinen [YJ18] improved on other HE-based works and construct a more efficient HE-based PPIL protocol, but its computation overhead is still high and prohibitive, e.g., for mobile devices, which is the main use case of IL.

Follow-up works In our subsequent work [**RYT+18**], we extend our quantization analysis and verify our results on data collected in further field experiments in multiple buildings.

In our other subsequent work [**RST19**], we design attacks against [ZCZL16], an existing PPIL protocol. We show how to efficiently learn the server's model for IL from query responses only and analyze different strategies for constructing queries for efficient learning.

Nieminen and Järvinen [NJ20] introduced further optimizations to HE-based PPIL and constructed a protocol that combines HE and GCs. Their protocol requires only around a megabyte of communication and takes only a few seconds to complete for small databases, thus making their protocol applicable to non-real-time PPIL, e.g., for continuous localization of a walking person.

Wu et al. [WFL20] reduced the problem of PPIL to PPIL for a sub-area on a map. However, their solution leaks a fuzzy location of the user and requires a trusted initializer. Although the benchmarks in [WFL20] show runtimes of only hundreds of milliseconds, they are hard to compare with other works, since they run the complete protocol locally using non-mobile hardware despite that their scenario involves a mobile client.

Fathalizadeh et al. [FMA22] continued the work in [KCZ+15] and construct further PPIL schemes based on adding noise to data. Another line of work [ZLZ+20; ZGS21] investigates the level of privacy that can be achieved by querying many fake fingerprints in the clear along with the "real" fingerprint, also by further adding noise to achieve better privacy.

3.2.2 Secure SSQ

The first protocols for secure SSQs were proposed in [AKD03] and extended to the ousourcing scenario in [AL05]. Both works use the ED for SSQs, which incur $O(n^2)$ overhead, drawing this form of SSQs inefficient for long strings. Jha et al. [JKS08] and Zhu and Huang [ZH17] further improved upon them, but the asymptotic overhead of their protocols still remained quadratic.

Wang et al. [WHZ+15] proposed an extremely efficient approximation for the ED that is based on set size difference. Their approach can process a genome-wide query over one million patients in about 3 hours, but, unfortunately, it works only for data with very small divergence (less than 0.5 % variability between individuals), which is not always true for genome data. Aziz et al. [AAM17] introduce other two approximations for the ED, which, however, are inaccurate for long sequences and the communication requirements of their protocols are unclear. Mahdi et al. [MHM17] instantiate SSQ with Hamming distance, which is a very efficient but very inaccurate metric for genomic data. Cheng et al. [CHW18] design an SSQ protocol based on HE in the outsourcing scenario with two non-colluding parties. We show that our SSQ protocol is significantly more efficient than [CHW18].

Follow-up works Zhu et al. [ZAVV21] recently showed how to use a hierarchical index data structure efficiently to substantially reduce the cost of SSQs in the cloud. To achieve this efficiency, they introduce a method for merging such data structures obtained from multiple clients that is run once prior to querying. Compared to [**ST19**], their scheme is slightly less efficient for short genome sequences and orders of magnitude more efficient for long genome sequences. Furthermore, [ZAVV21] does not require semi-trusted cloud servers, in contrast to [**ST19**]. However, their scheme is fairly complex, which might hinder its adoption, and requires merging multiple hierarchical indices, which takes significantly more time than the

query itself. Extrapolating their benchmarks, it would take at least days or weeks to merge indices for real genomes.

4 Conclusion and Future Work

In this final chapter, we draw a conclusion about the work described in this thesis in section 4.1. Then, we describe a few directions and ideas for future work in section 4.2.

4.1 Summary

This thesis significantly contributed to the research on practically efficient protocols and tools for secure multi-party computation (MPC), outsourcing, applications of MPC and their implementations.

We designed an efficient and flexible framework for MPC and outsourcing in [**BDST22**] and improved the efficiency on the implementation and protocol level. Our framework provides a high level of abstraction for implementing MPC protocols and primitives, while maintaining high efficiency. In contrast to other efficient MPC frameworks, it allows to easily implement and combine cryptographic protocols and primitives.

As another important step towards practically efficient MPC, in [**PSTY19**] we improved circuit-based private set intersection (C-PSI), a generic building block in MPC, and provided the first C-PSI protocol with linear communication complexity. We introduced and analyzed several optimizations such as removing the need of stash in Cuckoo hashing, batching multiple oblivious programmable pseudo-random functions (OPPRFs) and reduction of the computation complexity by evaluating batches of OPPRFs over a subset of the inputs, without increasing the linear communication complexity. Although our protocol is not straightforward to extend to allow using payloads by both parties, we give a protocol how to efficiently realize this.

Finally, we showed in [**JLL+19**; **ST19**] how to use MPC in the outsourcing setting to solve important real-world problems.
In [**JLL+19**], we designed two MPC protocols for privacy-preserving indoor localization (PPIL): depth-optimized and size-optimized. We improved some of the building blocks, e.g., our size-optimized circuit for computing k nearest neighbors (k-NN) is currently the state of the art. We further improved the performance of our PPIL protocols by quantizing the input data, and we verified the resulting accuracy in a field experiment. We analyzed different distance metrics and identified the MPC-friendly ones. Our most efficient protocol requires only 150 ms in the online phase.
In [**ST19**], we extended the MPC protocol by Asharov et al. [AHLR18] for securely computing

similar sequence queries (SSQs) between a client, who holds the query data, and a server, who holds the database, to an outsourcing scenario, where two non-colluding third parties hold a privately aggregated database. Our SSQ protocol is based on a simple and efficient technique to compute a secret bit-integer multiplication. Compared to the previous best SSQ protocol in the outsourcing scenario [CHW18], ours is four orders of magnitude faster and one order of magnitude more communication-efficient.

4.2 Future Work

Here, we give some ideas for future work.

4.2.1 More Efficient MPC Protocols and Primitives

Many cryptographic building blocks such as oblivious random access machine (ORAM) [Gol87] and private information retrieval (PIR) [CGKS95] can be used in combination with MPC to improve the efficiency. Recently, multiple novel efficient cryptographic building blocks such as function secret sharing (FSS) [BGI15] have been introduced that can even be used to construct new MPC building blocks or protocols [BCG+21]. However, as for now only a few protocols have been constructed in this way and further work is needed. Moreover, it would be interesting to see how well such building blocks can solve (further) real-world problems.

4.2.2 Flexible & Efficient Tools for MPC

Better efficiency of flexible MPC tools MOTION provides means for flexible implementation and combination of cryptographic primitives and MPC protocols. However, a higher degree of flexibility usually comes from high-level abstractions and the need for synchronization of multiple routines such as gate evaluation that both incur additional overhead. Therefore, it is important to find ways to avoid this additional overhead, e.g., by automatically batching multiple gates together. For this, different strategies can be utilized depending on the hardware, software and network setting.

Flexible format for MPC An even more ambitious goal would be to design a format for MPC functionalities that would allow to flexibly store, modify and optimize complex circuits at low costs. Such a format does not exists right now in the MPC community but would be highly desirable. Furthermore, such a format would allow automatic optimizations and knowledge transfer between MPC frameworks to avoid re-implementations.

Performance analysis of communication channels for MPC The high level of abstraction in MOTION allows to replace many crucial building blocks, such as the internal communication interface, by a different implementation. For example, MOTION's serialized communication allows to use transport layer protocols that do not preserve the message order such as the reliable user datagram protocol (RUDP) [IET99] or other communication protocols such as WebSockets [IET11], ZeroMQ [Pie] or even Signal [Shl21; Sig]. This flexibility of MOTION can be made use of to analyze the efficiency tradeoffs of different communication channels in MPC, similar to [BOSS20]. The efficiency of MPC with the latest secure transport layer protocols such as QUIC [IET19] and TLS1.3 [Res18], including its zero-round trip time (RTT) handshake, would also be of great interest. A systematization of knowledge in this direction would bring more clarity to the real cost of MPC in practical settings and potentially allow for more precise cost models.

MPC on different hardware and in browsers It would also be interesting to see how well MPC performs on different platforms and architectures when implemented in a low-level programming language and with hardware acceleration, e.g., many modern ARM devices are shipped with so-called cryptography extensions (CE)[1] that allow to use hardware acceleration for advanced encryption standard (AES), probably the most crucial building block of the state-of-the-art efficient MPC protocols. Furthermore, after the ongoing work on implementing the translation of Boost fibers in `emscripten`[2] is finished, it will likely be possible to compile MOTION to WebAssembly, thus enabling its use in web browsers, with very low implementation effort.

4.2.3 More efficient C-PSI and new settings

Unbalanced C-PSI The current C-PSI protocols consider the setting where all parties have inputs of the same size. It would be interesting to extend C-PSI protocols to settings where parties have unbalanced inputs, e.g., C-PSI with a messenger service holding a database with millions of phone numbers and a client holding their contact book with only hundreds of phone numbers, and evaluate its efficiency. Finding constructions in the setup-online paradigm with the online complexity in the size of the smaller input set would greatly increase the practical value of unbalanced C-PSI.

Cuckoo hashing parametrization insights from PSI for K-PIR One of the most popular approaches for realizing PSI and C-PSI is called hashing to bins (cf. section 2.2.2). Different sets of parameters have been considered and analyzed for hashing to bins in the scope of private set intersection (PSI) and C-PSI [PSSZ15; PSWW18; PSZ18]. However, also other techniques such as keyword private information retrieval (K-PIR) use Cuckoo hashing to map the inputs to a constant number of locations [ALP+19]. In contrast to PSI, K-PIR involves a server or multiple non-colluding servers holding a public database and one client who wants to obtain an element from the database without revealing to the server(s) which element was

[1] https://developer.arm.com/documentation/ddi0500/e/CJHDEBAF
[2] https://emscripten.org/index.html

obtained. The term "keyword" in K-PIR describes the fact that even if the database contains a (small) subset of all possible indices, the overhead for querying such a database remains linear in the database size. One technique to realize K-PIR is to map the database elements to a Cuckoo hash table and the client tries to retrieve the desired element from all its possible positions. It is possible to use the existing parameters for Cuckoo hashing to avoid collisions in K-PIR but it would be even better to conduct an analysis for smaller parameters, since the setting of PSI and K-PIR is different. Namely, the database in K-PIR is normally public and it does not leak additional information if we select non-random hash functions for mapping the database. So the question here is how small can the Cuckoo hash table be s.t. we can find h hash functions in t trials that allow to map n elements without collision.

4.2.4 New Applications & Optimizations

The need of revisiting the GMW vs. Yao tradeoff Schneider and Zohner [SZ13] analyzed several boolean circuits regarding their efficiency for different MPC protocols. Furthermore, they compared the efficiency of Goldreich-Micali-Wigderson (GMW) and garbled circuits (GCs) using the best suited circuits, and they came to the conclusion that none of the protocols is best in all scenarios. Their analysis mainly showed that often GMW requires some blowup in communication to reduce the depth of the circuit by sacrificing its size optimization, which quickly becomes a bottleneck in slower networks and/or for large parallelizable circuits. However, since then many optimizations, such as silent oblivious transfer (S-OT) [BCG+19] and more efficient garbling [RR21], have been introduced toward more efficient generic MPC. The communication of both protocols was very similar when the analysis by Schneider and Zohner was performed, whereas now the tradeoffs are very different. For example, an open question is to what extent S-OT-based GMW removes the disadvantage of the size blow-up in low-depth optimizations and whether even more aggressive depth optimizations would produce more efficient circuits in certain settings.

Three halves GC for multi-input gates The three halves garbling scheme by Rosulek and Roy [RR21] reduces the communication cost of garbling AND gates by linearly combining (more) garbled values of smaller size compared to the previous garbling schemes, resulting in a smaller overall size of the garbled material. The best communication their scheme achieves for a 2-input AND gate is $\sim 1.5\kappa$. It is an interesting question to which extent their approach could improve the communication of multi-input and/or multi-output gates, e.g., AND gates with n inputs.

Bibliography

[AAM17] M. M. A. AZIZ, D. ALHADIDI, N. MOHAMMED. **"Secure approximation of edit distance on genomic data"**. In: *BMC Medical Genomics* 10.2 (2017), pp. 55–67.

[ABL20] A. ACQUISTI, L. BRANDIMARTE, G. LOEWENSTEIN. **"Secrets and Likes: The Drive for Privacy and the Difficulty of Achieving it in the Digital Age"**. In: *Journal of Consumer Psychology* 30.4 (2020), pp. 736–758.

[ACC+21] A. ALY, K. CONG, D. COZZO, M. KELLER, E. ORSINI, D. ROTARU, O. SCHERER, P. SCHOLL, N. SMART, T. TANGUY. **"SCALE–MAMBA v1.12: Documentation"**. 2021.

[AHLR18] G. ASHAROV, S. HALEVI, Y. LINDELL, T. RABIN. **"Privacy-preserving search of similar patients in genomic data"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* (2018), pp. 104–124.

[AKD03] M. J. ATALLAH, F. KERSCHBAUM, W. DU. **"Secure and private sequence comparisons"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2003, pp. 39–44.

[AL05] M. J. ATALLAH, J. LI. **"Secure outsourcing of sequence comparisons"**. In: *International Journal of Information Security* 4.4 (2005), pp. 277–287.

[ALP+19] A. ALI, T. LEPOINT, S. PATEL, M. RAYKOVA, P. SCHOPPMANN, K. SETH, K. YEO. **"Communication-Computation Trade-offs in PIR"**. In: *Cryptology ePrint Archive* 2019/1483 (2019). IACR.

[ALSZ13] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. **"More Efficient Oblivious Transfer and Extensions for Faster Secure Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2013, pp. 535–548.

[ASKG19] N. AGRAWAL, A. SHAHIN SHAMSABADI, M. J. KUSNER, A. GASCÓN. **"QUOTIENT: two-party secure neural network training and prediction"**. In: *Computer and Communications Security*. ACM. 2019, pp. 1231–1247.

[Ayd16] E. AYDAY. **"Cryptographic solutions for genomic privacy"**. In: *Financial Cryptography and Data Security (FC)*. Springer. 2016, pp. 328–341.

[BCD+09] P. BOGETOFT, D. L. CHRISTENSEN, I. DAMGÅRD, M. GEISLER, T. JAKOBSEN, M. KRØIGAARD, J. D. NIELSEN, J. B. NIELSEN, K. NIELSEN, J. PAGTER. **"Secure multiparty computation goes live"**. In: *Financial Cryptography and Data Security (FC)*. Springer. 2009, pp. 325–343.

[BCD+20] F. BOEMER, R. CAMMAROTA, D. DEMMLER, T. SCHNEIDER, H. YALAME. **"MP2ML: A mixed-protocol machine learning framework for private inference"**. In: *Availability, Reliability and Security (ARES)*. 2020, pp. 1–10.

[BCG+19] 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.

[BCG+21] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, M. Rathee. **"Function secret sharing for mixed-mode and fixed-point secure computation"**. In: *Advances in Cryptology – EUROCRYPT*. Springer. 2021, pp. 871–900.

[BCPS20] M. Byali, H. Chaudhari, A. Patra, A. Suresh. **"FLASH: fast and robust framework for privacy-preserving machine learning"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2020.2 (2020), pp. 459–480.

[BCS21] L. Braun, R. Cammarota, T. Schneider. **"A Generic Hybrid 2PC Framework with Application to Private Inference of Unmodified Neural Networks"**. In: *NeurIPS'21 Workshop Privacy in Machine Learning*. 2021.

[BDK+18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *Computer and Communications Security (CCS)*. ACM, 2018, pp. 847–861.

[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). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION.

[Bea91] D. Beaver. **"Efficient multiparty protocols using circuit randomization"**. In: *Advances in Cryptology – CRYPTO*. Springer. 1991, pp. 420–432.

[Bea95] D. Beaver. **"Precomputing Oblivious Transfer"**. In: *Advances in Cryptology – CRYPTO*. Springer. 1995, pp. 97–109.

[BGI15] E. Boyle, N. Gilboa, Y. Ishai. **"Function secret sharing"**. In: *Advances in Cryptology – EUROCRYPT*. Springer. 2015, pp. 337–367.

[BLCW19] F. Boemer, Y. Lao, R. Cammarota, C. Wierzynski. **"nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data"**. In: *Computing Frontiers*. 2019, pp. 3–13.

[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.

[BLW08] D. Bogdanov, S. Laur, J. Willemson. **"Sharemind: A framework for fast privacy-preserving computations"**. In: *European Symposium on Research in Computer Security (ESORICS)*. Springer. 2008, pp. 192–206.

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

[BOSS20] M. Brandt, C. Orlandi, K. Shrishak, H. Shulman. **"Optimal Transport Layer for Secure Computation"**. In: *Security and Cryptography (SECRYPT)*. SciTePress, 2020.

[But16] D. Butler. **"Tomorrow's world: technological change is accelerating today at an unprecedented speed and could create a world we can barely begin to imagine"**. In: *Nature* 530.7591 (2016), pp. 398–402.

[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*. 2019, pp. 81–92.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan. **"Private information retrieval"**. In: *Foundations of Computer Science (FOCS)*. IEEE. 1995, pp. 41–50.

[CGS22] N. CHANDRAN, D. GUPTA, A. SHAH. **"Circuit-PSI with Linear Complexity via Relaxed Batch OPPRF"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETS)* 2022.1 (2022), pp. 353–372.

[CGT12] E. D. CRISTOFARO, P. GASTI, G. TSUDIK. **"Fast and Private Computation of Cardinality of Set Intersection and Union"**. In: *Cryptology And Network Security (CANS)*. Vol. 7712. Springer, 2012, pp. 218–231.

[CHK+12] S. G. CHOI, K.-W. HWANG, J. KATZ, T. MALKIN, D. RUBENSTEIN. **"Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces"**. In: *Cryptographers' Track at the RSA Conference (CT-RSA)*. Springer. 2012, pp. 416–432.

[CHW18] K. CHENG, Y. HOU, L. WANG. **"Secure similar sequence query on outsourced genomic data"**. In: *Asia Conference on Computer and Communications Security (ASIACCS)*. ACM. 2018, pp. 237–251.

[Cis19] CISCO. **"Cisco Visual Networking Index: Forecast and Trends, 2017–2022"**. https://twiki.cern.ch/twiki/pub/HEPIX/TechwatchNetwork/HtwNetworkDocuments/white-paper-c11-741490.pdf. Retrieved on 03.03.2022. 2019.

[con22] W. CONTRIBUTORS. **"2020 United States federal government data breach"**. https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach. Retrieved on 15.03.2022. 2022.

[CRS20] H. CHAUDHARI, R. RACHURI, A. SURESH. **"Trident: Efficient 4PC framework for privacy preserving machine learning"**. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2020.

[CWB18] H. CHO, D. J. WU, B. BERGER. **"Secure genome-wide association analysis using multiparty computation"**. In: *Nature biotechnology* 36.6 (2018), pp. 547–551.

[DD15] S. K. DEBNATH, R. DUTTA. **"Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter"**. In: *Information Security Conference (ISC)*. Vol. 9290. LNCS. Springer, 2015, pp. 209–226.

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

[EFG+15] R. EGERT, M. FISCHLIN, D. GENS, S. JACOB, M. SENKER, J. TILLMANNS. **"Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters"**. In: *Australasian Conference on Information Security (ACISP)*. Vol. 9144. LNCS. Springer, 2015, pp. 413–430.

[FMA22] A. FATHALIZADEH, V. MOGHTADAIEE, M. ALISHAHI. **"On the Privacy Protection of Indoor Location Dataset using Anonymization"**. In: *Computers & Security* (2022), p. 102665.

[GMR+21] G. GARIMELLA, P. MOHASSEL, M. ROSULEK, S. SADEGHIAN, J. SINGH. **"Private Set Operations from Oblivious Switching"**. In: *Public-Key Cryptography (PKC)*. Springer. 2021, pp. 591–617.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **"How to Play any Mental Game or a Completeness Theorem for Protocols with Honest Majority"**. In: *Symposium on Theory of Computing (STOC)*. ACM, 1987, pp. 218–229.

[Gol87] O. GOLDREICH. **"Towards a theory of software protection and simulation by oblivious RAMs"**. In: *Symposium on Theory of Computing (STOC)*. 1987, pp. 182–194.

[Gre21] J. GREIG. **"Losses from BitMart breach reach $200 million"**. https://www.zdnet.com/article/bitmart-breach-losses-reach-200-million/. Retrieved on 15.03.2022. 2021.

[HAHT15] M. HUMBERT, E. AYDAY, J.-P. HUBAUX, A. TELENTI. **"On non-cooperative genomic privacy"**. In: *Financial Cryptography and Data Security (FC)*. Springer. 2015, pp. 407–426.

[Hal18] S. HALEVI. **"Advanced Cryptography: Promise and Challenges."** In: *CCS'18*. ACM, 2018.

[HEK12] Y. HUANG, D. EVANS, J. KATZ. **"Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?"** In: *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2012.

[HHNZ19] M. HASTINGS, B. HEMENWAY, D. NOBLE, S. ZDANCEWIC. **"SoK: General Purpose Compilers for Secure Multi-Party Computation"**. In: *Security and Privacy (S&P)*. IEEE, 2019, pp. 1220–1237.

[HKS+10] W. HENECKA, S. K ÖGL, A.-R. SADEGHI, T. SCHNEIDER, I. WEHRENBERG. **"TASTY: tool for automating secure two-party computations"**. In: *Computer and Communications Security (CCS)*. ACM. 2010, pp. 451–462.

[IET11] IETF. **"The WebSocket Protocol"**. https://datatracker.ietf.org/doc/html/rfc6455. 2011.

[IET19] IETF. **"QUIC: A UDP-Based Multiplexed and Secure Transport"**. https://tools.ietf.org/html/draft-ietf-quic-transport-23. 2019.

[IET99] IETF. **"Reliable UDP (RUDP) Protocol"**. https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00. 1999.

[IKN+17] M. ION, B. KREUTER, E. NERGIZ, S. PATEL, S. SAXENA, K. SETH, D. SHANAHAN, M. YUNG. **"Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions"**. In: *Cryptology ePrint Archive* 2017/738 (2017). IACR.

[IKN+20] M. ION, B. KREUTER, A. E. NERGIZ, S. PATEL, S. SAXENA, K. SETH, M. RAYKOVA, D. SHANAHAN, M. YUNG. **"On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality"**. In: *European Symposium on Security and Privacy (S&P)*. IEEE, 2020, pp. 370–389.

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *Advances in Cryptology – CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 145–161.

[JKS08] S. JHA, L. KRUGER, V. SHMATIKOV. **"Towards practical privacy for genomic computation"**. In: *Security and Privacy (S&P)*. IEEE. 2008, pp. 216–230.

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E. S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 448–463.

[Joh21] J. JOHNSON. **"Statista: Global digital population as of January 2021"**. https://www.statista.com/statistics/617136/digital-population-worldwide/. Accessed on 02.01.2022. 2021.

[Jon10] T. JONES. **"The rise of DNA analysis in crime solving"**. https://www.theguardian.com/politics/2010/apr/10/dna-analysis-crime-solving. Retrieved on 04.03.2022. 2010.

[Jon18] T. JONES. **"How taking a home genetics test could help catch a murderer"**. https://www.theguardian.com/science/2018/dec/01/how-home-dna-tests-are-solving-cold-cases-golden-state-killer. Retrieved on 04.03.2022. 2018.

[JVC18] C. JUVEKAR, V. VAIKUNTANATHAN, A. CHANDRAKASAN. **"GAZELLE: A low latency framework for secure neural network inference"**. In: *USENIX Security Symposium*. 2018, pp. 1651–1669.

[KCZ+15] A. KONSTANTINIDIS, G. CHATZIMILIOUDIS, D. ZEINALIPOUR-YAZTI, P. MPEIS, N. PELEKIS, Y. THEODORIDIS. **"Privacy-preserving indoor localization on smartphones"**. In: *Knowledge and Data Engineering* 27.11 (2015), pp. 3042–3055.

[Kel20] M. KELLER. **"MP-SPDZ: A versatile framework for multi-party computation"**. In: *Computer and Communications Security (CCS)*. 2020, pp. 1575–1590.

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

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

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

[LBC+19] P. LIÉTAR, T. BUTLER, S. CLEBSCH, S. DROSSOPOULOU, J. FRANCO, M. J. PARKINSON, A. SHAMIS, C. M. WINTERSTEIGER, D. CHISNALL. **"snmalloc: A message passing allocator"**. In: *Memory Management*. ACM. 2019, pp. 122–135.

[LN18] Y. LINDELL, A. NOF. **"Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody"**. In: *Computer and Communications Security (CCS)*. ACM. 2018, pp. 1837–1854.

[LSZ+14] H. LI, L. SUN, H. ZHU, X. LU, X. CHENG. **"Achieving privacy preservation in WiFi fingerprint-based localization"**. In: *IEEE INFOCOM-IEEE Conference on Computer Communications*. IEEE. 2014, pp. 2337–2345.

[MBS20] C. MATTE, N. BIELOVA, C. SANTOS. **"Do Cookie Banners Respect My Choice?: Measuring Legal Compliance of Banners from IAB Europe's Transparency and Consent Framework"**. In: *Security and Privacy (S&P)*. IEEE. 2020, pp. 791–809.

[MHM17] M. S. R. MAHDI, M. Z. HASAN, N. MOHAMMED. **"Secure sequence similarity search on encrypted genomic data"**. In: *Connected Health: Applications, Systems and Engineering Technologies (CHASE)*. IEEE/ACM. 2017, pp. 205–213.

[MPR+20] P. MIAO, S. PATEL, M. RAYKOVA, K. SETH, M. YUNG. **"Two-Sided Malicious Security for Private Intersection-Sum with Cardinality"**. In: *Advances in Cryptology – CRYPTO*. Vol. 12172. LNCS. Springer, 2020, pp. 3–33.

[MR18] P. MOHASSEL, P. RINDAL. **"ABY3: A mixed protocol framework for machine learning"**. In: *Computer and Communications Security (CCS)*. ACM. 2018, pp. 35–52.

[MZ17] P. MOHASSEL, Y. ZHANG. **"SecureML: A system for scalable privacy-preserving machine learning"**. In: *Security and Privacy (S&P)*. IEEE. 2017, pp. 19–38.

[NJ20] R. NIEMINEN, K. JÄRVINEN. **"Practical privacy-preserving indoor localization based on secure two-party computation"**. In: *Mobile Computing* 20.9 (2020), pp. 2877–2890.

[Pai99] P. PAILLIER. **"Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 223–238.

[Pie] PIETER HINTJENS AND OTHERS. **"ZeroMQ RFC"**. https://rfc.zeromq.org/. Retrieved on 07.02.2022.

[PR01] R. PAGH, F. F. RODLER. **"Cuckoo Hashing"**. In: *European Symposium on Algorithms (ESA)*. Vol. 2161. LNCS. Springer, 2001, pp. 121–133.

[PRTY20] B. PINKAS, M. ROSULEK, N. TRIEU, A. YANAI. **"PSI from PaXoS: Fast, Malicious Private Set Intersection"**. In: *Advances in Cryptology – EUROCRYPT* 12106 (2020), p. 739.

[PS22] A. PATRA, A. SURESH. **"BLAZE: blazing fast privacy-preserving machine learning"**. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved mixed-protocol secure two-party computation"**. In: *USENIX Security Symposium*. 2021.

[PSSZ15] B. PINKAS, T. SCHNEIDER, G. SEGEV, M. ZOHNER. **"Phasing: Private Set Intersection Using Permutation-based Hashing"**. In: *USENIX Security Symposium*. USENIX Association, 2015, pp. 515–530.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, pp. 122–153.

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *Advances in Cryptology (EUROCRYPT)*. Vol. 10822. LNCS. Springer, 2018, pp. 125–157.

[PSZ18] B. PINKAS, T. SCHNEIDER, M. ZOHNER. **"Scalable Private Set Intersection Based on OT Extension"**. In: *Transactions on Privacy and Security (TOPS)* 21.2 (2018). ACM, 7:1–7:35.

[Red22] I. C. o. t. RED CROSS. **"Sophisticated cyber-attack targets Red Cross Red Crescent data on 500,000 people"**. https://www.icrc.org/en/document/sophisticated-cyber-attack-targets-red-cross-red-crescent-data-500000-people. Retrieved on 15.03.2022. 2022.

[Res18] E. RESCORLA. **"The Transport Layer Security (TLS) Protocol Version 1.3"**. https://datatracker.ietf.org/doc/html/rfc8446. 2018.

[Rin] P. RINDAL. **"libOTe: an efficient, portable, and easy to use Oblivious Transfer Library"**. https://github.com/osu-crypto/libOTe. Retrieved on 20.02.2022.

[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.

[RRK+20] 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.

[RS21] P. RINDAL, P. SCHOPPMANN. **"VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE"**. In: *Advances in Cryptology – EUROCRYPT*. Springer. 2021, pp. 901–930.

[RSC+19] M. S. RIAZI, M. SAMRAGH, H. CHEN, K. LAINE, K. LAUTER, F. KOUSHANFAR. **"XONN: XNOR-based Oblivious Deep Neural Network Inference"**. In: *USENIX Security Symposium*. 2019, pp. 1501–1518.

[RST19] R. N. REITH, T. SCHNEIDER, O. TKACHENKO. **"Efficiently Stealing your Machine Learning Models"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2019, pp. 198–210.

[RWT+18] M. S. RIAZI, C. WEINERT, O. TKACHENKO, E. M. SONGHORI, T. SCHNEIDER, F. KOUSHANFAR. **"Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications"**. In: *Asia Conference on Computer and Communication Security (ASIACCS)*. ACM, 2018, pp. 707–721.

[RYT+18] P. RICHTER, Z. YANG, O. TKACHENKO, H. LEPPÄKOSKI, K. JÄRVINEN, T. SCHNEIDER, E. S. LOHAN. **"Received Signal Strength Quantization for Secure Indoor Positioning via Fingerprinting"**. In: *International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2018, pp. 1–6.

[Sch18] B. SCHOENMAKERS. **"MPyC—Python package for secure multiparty computation"**. In: *Workshop on the Theory and Practice of MPC (TPMPC)*. Code: https://github.com/lschoe/mpyc. 2018.

[SCYW14] T. SHU, Y. CHEN, J. YANG, A. WILLIAMS. **"Multi-lateral privacy-preserving localization in pervasive environments"**. In: *IEEE INFOCOM-IEEE Conference on Computer Communications*. IEEE. 2014, pp. 2319–2327.

[SGRR19] P. SCHOPPMANN, A. GASCÓN, L. REICHERT, M. RAYKOVA. **"Distributed Vector-OLE: Improved Constructions and Implementation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. 2019, pp. 1055–1072.

[Sha] SHAREMIND. **"Analyse and Process Encrypted Data Anonymously - Case Studies"**. https://sharemind.cyber.ee/analyse-process-encrypted-data. Retrieved on 02.03.2022.

[Shl21] O. SHLOMOVITS. **"MPC-Over-Signal"**. https://medium.com/zengo/mpc-over-signal-977db599de66. Retrieved on 07.02.2022. 2021.

[SHSK15] E. M. SONGHORI, S. U. HUSSAIN, A.-R. SADEGHI, F. KOUSHANFAR. **"Compacting privacy-preserving k-nearest neighbor search using logic synthesis"**. In: *Design Automation Conference (DAC)*. ACM/EDAC/IEEE. 2015, pp. 1–6.

[Sig] SIGNAL. **"Signal Specifications"**. https://signal.org/docs/. Retrieved on 06.02.2022.

[ST18] T. SCHNEIDER, O. TKACHENKO. **"Towards Efficient Privacy-Preserving Similar Sequence Queries on Outsourced Genomic Databases"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2018, pp. 71–75.

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Online: https://ia.cr/2021/029. ACM, 2019, pp. 315–327.

[Swe02] L. SWEENEY. **"k-anonymity: A model for protecting privacy"**. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[SZ13] T. SCHNEIDER, M. ZOHNER. **"GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits"**. In: *Financial Cryptography and Data Security (FC)*. Vol. 7859. LNCS. Springer, 2013, pp. 275–292.

[Tho22] M. THOMSON. **"Mozilla Blog: Privacy Preserving Attribution for Advertising"**. https://blog.mozilla.org/en/mozilla/privacy-preserving-attribution-for-advertising/. Retrieved on 16.02.2022. 2022.

[TWSH18] O. TKACHENKO, C. WEINERT, T. SCHNEIDER, K. HAMACHER. **"Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies"**. In: *Asia Conference on Computer and Communication Security (ASIACCS)*. ACM, 2018, pp. 221–235.

[WFL20] W. WU, S. FU, Y. LUO. **"Practical Privacy Protection Scheme In WiFi Fingerprint-based Localization"**. In: *Data Science and Advanced Analytics (DSAA)*. IEEE, 2020, pp. 699–708.

[WGC19] S. WAGH, D. GUPTA, N. CHANDRAN. **"SecureNN: 3-Party Secure Computation for Neural Network Training."** In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2019.3 (2019), pp. 26–49.

[WHZ+15] X. S. WANG, Y. HUANG, Y. ZHAO, H. TANG, X. WANG, D. BU. **"Efficient genome-wide, privacy-preserving similar patient query based on private edit distance"**. In: *Computer and Communications Security (CCS)*. ACM. 2015, pp. 492–503.

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

[YJ18] Z. YANG, K. JÄRVINEN. **"The death and rebirth of privacy-preserving WiFi fingerprint localization with Paillier encryption"**. In: *IEEE INFOCOM-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 1223–1231.

[ZAVV21] X. ZHU, E. AYDAY, R. VITENBERG, N. R. VEERARAGAVAN. **"Privacy-Preserving Search for a Similar Genomic Makeup in the Cloud"**. In: *Transactions on Dependable and Secure Computing (TDSC)* (2021).

[ZCZL16] T. ZHANG, S. S. CHOW, Z. ZHOU, M. LI. **"Privacy-preserving Wi-Fi fingerprinting indoor localization"**. In: *International Workshop on Security*. Springer. 2016, pp. 215–233.

[ZGS21] A. ZAYETS, C. GENTNER, E. STEINBACH. **"High-precision multipath-based indoor localization scheme with user privacy protection for dynamic NLoS environments"**. In: *Access* 9 (2021), pp. 116033–116049.

[ZH17] R. ZHU, Y. HUANG. **"Efficient privacy-preserving general edit distance and beyond"**. In: *Cryptology ePrint Archive* 2017/683 (2017). IACR.

[ZLZ+20] P. ZHAO, W. LIU, G. ZHANG, Z. LI, L. WANG. **"Preserving privacy in WiFi localization with plausible dummy locations"**. In: *Transactions on Vehicular Technology* 69.10 (2020), pp. 11909–11925.

[ZVHW14] J. H. ZIEGELDORF, N. VIOL, M. HENZE, K. WEHRLE. **"Poster: Privacy-preserving indoor localization"**. In: *Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM. 2014.

List of Figures

List of Abbreviations

2PC	secure two-party computation
A-GMW	arithmetic Goldreich-Micali-Wigderson
AES	advanced encryption standard
AHE	additively homomorphic encryption
AP	access point
BLE	bluetooth low energy
BMR	Beaver-Micali-Rogaway
CE	cryptography extensions
C-OT	correlated oblivious transfer
C-PSI	circuit-based private set intersection
C-PSU	circuit-based private set union
CE	cryptography extensions
ED	edit distance
EU	European Union
FFT	Fast Fourier Transformation
FSS	function secret sharing
G-OT	general oblivious transfer
GC	garbled circuit
GDPR	general data protection regulation
GMW	Goldreich-Micali-Wigderson
GNSS	global navigation satellite systems
GPS	global positioning system
HE	homomorphic encryption
IL	indoor localization
IXP	internet exchange point
***k*-NN**	k nearest neighbors
K-PIR	keyword private information retrieval
LAN	local area network
LUT	look-up table
ML	machine learning
MPC	secure multi-party computation
ONNX	open neural network exchange
ORAM	oblivious random access machine
OPRF	oblivious pseudo-random function
OPPRF	oblivious programmable pseudo-random function

OT	oblivious transfer
PIR	private information retrieval
PPIL	privacy-preserving indoor localization
PSI	private set intersection
RSS	received signal strength
RTT	round trip time
RUDP	reliable user datagram protocol
S-OT	silent oblivious transfer
SIMD	single instruction multiple data
SSQ	similar sequence query
VOLE	vector oblivious linear evaluation
WAN	wide area network

List of Own Publications

Peer-reviewed Publications

[BDST22] L. Braun, D. Demmler, T. Schneider, O. Tkachenko. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *Transactions on Privacy and Security (TOPS)* 25 (2 2022). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION. CORE Rank A. Appendix A.

[HKST22] K. Hamacher, T. Kussel, T. Schneider, O. Tkachenko. **"Practical Private Epistasis Analysis using MPC"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2022 (2022). CORE Rank A. Appendix B.

[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)*. Vol. 12727. LNCS. Online: https://ia.cr/2021/797. Code: https://encrypto.de/code/LLVM. Springer, 2021, pp. 99–121. CORE Rank B.

[JKS+18] K. Järvinen, Á. Kiss, T. Schneider, O. Tkachenko, Z. Yang. **"Faster Privacy-Preserving Location Proximity Schemes"**. In: *Cryptology And Network Security (CANS)*. Vol. 11124. LNCS. Full version: https://ia.cr/2018/694. Springer, 2018, pp. 3–22. CORE Rank B.

[JKS+20] K. Järvinen, Á. Kiss, T. Schneider, O. Tkachenko, Z. Yang. **"Faster Privacy-Preserving Location Proximity Schemes for Circles and Polygons"**. In: *IET Information Security* 14.3 (2020), pp. 254–265. CORE Rank C.

[JLL+19] K. Järvinen, H. Leppäkoski, E. S. Lohan, P. Richter, T. Schneider, O. Tkachenko, Z. Yang. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 448–463. Appendix C.

[PSTY19] B. Pinkas, T. Schneider, O. Tkachenko, A. Yanai. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, pp. 122–153. CORE Rank A*. Appendix B.

[RST19] R. N. Reith, T. Schneider, O. Tkachenko. **"Efficiently Stealing your Machine Learning Models"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2019, pp. 198–210.

[RWT+18] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, F. Koushanfar. **"Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Preliminary version: https://ia.cr/2017/1164. ACM, 2018, pp. 707–721. CORE Rank A.

[RYT+18] P. Richter, Z. Yang, O. Tkachenko, H. Leppäkoski, K. Järvinen, T. Schneider, E. S. Lohan. **"Received Signal Strength Quantization for Secure Indoor Positioning via Fingerprinting"**. In: *International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2018, pp. 1–6.

[ST18] T. Schneider, O. Tkachenko. **"Towards Efficient Privacy-Preserving Similar Sequence Queries on Outsourced Genomic Databases"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2018, pp. 71–75.

[ST19] T. Schneider, O. Tkachenko. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Full version: https://ia.cr/2021/029. ACM, 2019, pp. 315–327. CORE Rank A. Appendix D.

[TWSH18] O. Tkachenko, C. Weinert, T. Schneider, K. Hamacher. **"Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. ACM, 2018, pp. 221–235. CORE Rank A.

Appendices

A MOTION - A Framework for Mixed-Protocol Multi-Party Computation (TOPS'22)

[BDST22] L. BRAUN, D. DEMMLER, T. SCHNEIDER, O. TKACHENKO. **"MOTION - A Framework for Mixed-Protocol Multi-Party Computation"**. In: *Transactions on Privacy and Security (TOPS)* 25 (2 2022). Accepted for publication. Online: https://ia.cr/2020/1137. Code: https://encrypto.de/code/MOTION. CORE Rank A. Appendix A.

https://doi.org/10.1145/3490390

MOTION – A Framework for Mixed-Protocol Multi-Party Computation

LENNART BRAUN, Aarhus University
DANIEL DEMMLER, Universität Hamburg
THOMAS SCHNEIDER, TU Darmstadt
OLEKSANDR TKACHENKO, TU Darmstadt

We present MOTION, an efficient and generic open-source framework for mixed-protocol secure multi-party computation (MPC). MOTION is built in a user-friendly, modular, and extensible way, intended to be used as tool in MPC research and to increase adoption of MPC protocols in practice. Our framework incorporates several important engineering decisions such as full communication serialization, which enables MPC over arbitrary messaging interfaces and removes the need of owning network sockets. MOTION incorporates several novel performance optimizations that improve the communication complexity and latency, e.g., 2× better online round complexity of precomputed correlated Oblivious Transfer (OT).

We instantiate our framework with protocols for N parties and security against up to $N-1$ passive corruptions: the MPC protocols of Goldreich-Micali-Wigderson (GMW) in its arithmetic and Boolean version and OT-based BMR (Ben-Efraim et al., CCS'16), as well as novel and highly efficient conversions between them, including a non-interactive conversion from BMR to arithmetic GMW.

MOTION is highly efficient, which we demonstrate in our experiments. Compared to secure evaluation of AES-128 with N=3 parties in a high-latency network with OT-based BMR, we achieve a 16× better throughput of 16 AES evaluations per second using BMR. With this, we show that BMR is much more competitive than previously assumed. For N=3 parties and full-threshold protocols in a LAN, MOTION is 10×–18× faster than the previous best passively secure implementation from the MP-SPDZ framework, and 190×–586× faster than the actively secure SCALE-MAMBA framework. Finally, we show that our framework is highly efficient for privacy-preserving neural network inference.

Additional Key Words and Phrases: secure multi-party computation, hybrid protocols, efficiency, outsourcing

1 INTRODUCTION

Secure Multi-Party Computation (MPC) allows multiple parties to jointly compute a public function on their private inputs without revealing anything but the function's output. This concept was first introduced in the 1980s by Yao [74] and Goldreich-Micali-Wigderson [33] and was initially considered of merely theoretical interest. The seminal work of Fairplay [55] was the first to implement MPC protocols and showed that MPC can indeed be practical. A long line of research has followed since then and has shown MPC to be a viable solution for preserving privacy in applications such as auctions [15], stable matching [30], set intersection [41], biometric matching [13, 56], and machine learning [57, 77].

This was facilitated by implementations of generic MPC frameworks that can be used for multiple applications. However, many MPC frameworks have a somewhat limited scope: they allow computations only for a fixed number of parties, e.g., two [19, 29, 39] or three [14, 20, 58, 62, 64] of which at most one can be corrupted, they only implement a single MPC protocol [55, 70], or are custom-tailored towards a few use-cases [54]. Furthermore, most of the publicly available code of MPC frameworks as surveyed in [36] is in a rather prototypical state as its main purpose is to generate performance measurements. This is a big problem, as these implementations are unusable in practice and building on top of them in future research is often a tedious procedure that involves a significant amount of time and expertise in understanding poorly documented code, and fixing a multitude of existing problems and limitations.

In this work, we present MOTION, an MPC framework that overcomes these limitations and aims to be a piece of software of high quality and usability. MOTION is object-oriented, developer-friendly, and well-documented, 20% of

its code base are unit and component tests, and it is designed in a modular and extensible way, serving as a powerful tool for future MPC research and implementations. MOTION is a *generic solution* for implementing mixed-protocol MPC with two or more parties and guarantees security against all but one passively corrupted parties (full-threshold security). It supports multiple MPC protocols and can securely and efficiently convert between them in order to benefit from each protocol's strengths. Previous mixed-protocol MPC frameworks are either limited to two parties [29], require an honest majority [14, 20, 58, 62, 64], or are full-threshold but with stronger active security only and hence are less efficient [24]. The architecture of MOTION allows to implement further MPC protocols in different security models.

Our motivation is to enable MPC protocols in practical application scenarios with an *arbitrary* number of parties and full threshold security. By this, we increase the number of parties in order to strengthen the *security* guarantees of the protocols. The limitation to exactly two or three parties of previous works might be problematic for practical settings, where a larger number of participants wants to jointly compute on private data. Also, in outsourcing scenarios [47], where a very large number of clients securely outsource computation to a smaller number of non-colluding computing parties, it might be desirable to increase the number of computing parties to achieve better security guarantees. In contrast, the goal of many previous works, e.g., [14, 20, 58, 62, 64], was to improve *performance* by increasing the number of parties, while simultaneously effectively reducing the security guarantees because only a single party can be corrupted.

Our protocols include novel performance optimizations in order to enable privacy-preserving computations for large real-world applications. With AES and private inference using a convolutional neural network, we present examples of such applications. Biometric identification using the Euclidean distance, which we also demonstrate, is an example for a multi-party application where a client wants to privately check if a data point is included in a large data set that is provided by multiple data owners. One could imagine a security check where a fingerprint is tested against databases of known fingerprints supplied by different security agencies. This scenario also finds application in other domains, e.g., statistics or financial analysis.

We integrate MOTION with the HyCC compiler [17] that generates optimized circuits for hybrid MPC protocols. With this, MOTION can directly perform efficient MPC of functionalities that have been specified in the C programming language. This enables also developers with limited domain knowledge to use MPC for a large range of applications.

We provide a detailed performance evaluation of both the low-level building blocks and protocol parts, as well as our full-scale applications on large data sets. Our performance results provide new insights into the performance differences of MPC protocols, such as better performance of BMR [8] compared to GMW [33] (often even in low-latency networks) for deep *natural* circuits, e.g., for integer division, which could be of independent interest for protocol designers.

Paper Organization and Our Contributions

We summarize related work in §2 and provide preliminary information about notation, our setting, and security assumptions in §3. Our main contributions are the following:

The MOTION framework for mixed-protocol MPC. In §4, we describe the design rationales behind our MOTION framework. It is a well-engineered and modular framework for MPC, which is extensively tested and well-documented. We aim at high usability and will release our code[1] as open-source software under the permissive MIT license[2]. Our novel important features are the asynchronous evaluation of gates and oblivious transfers, which allows for a high level of abstraction in designing the protocols, and the full communication serialization that allows the use of MOTION in

[1] The code will be published shortly at https://encrypto.de/code/MOTION.
[2] https://choosealicense.com/licenses/mit/

server and web applications. Moreover, our implementation can optionally interleave ('pipeline') the evaluation of the input-independent setup and the input-dependent online phase and allows to implement arbitrary circuit evaluation strategies in an abstract way.

Full-threshold hybrid MPC with passive security. We implement the existing full-threshold passively secure MPC protocols Boolean and arithmetic GMW [33], and BMR [8, 11] for multiple parties that guarantee a high level of security because all but one party can be corrupted (full-threshold). In §5, we describe the used MPC building blocks in detail. In §6, we introduce optimizations of these protocols, including an improvement of precomputed correlated OTs (C-OTs) from 2 to 1 communication rounds. This is of independent interest and makes the direct use of precomputed C-OTs for AND gates in GMW even more efficient than the use of Multiplication Triples (MTs) [6], having slightly lower communication and takes 1 instead of 2 communication rounds in the input-independent setup phase. In §7, we provide efficient protocols for converting between all of the above mentioned MPC protocols. MOTION is the first framework that efficiently combines these protocols. We support direct processing of hybrid circuits generated using the HyCC compiler [17].

Performance and Applications. In §8, we evaluate the performance of MOTION and demonstrate its practical relevance by showing that securely computing real-world applications such as biometric identification, AES-128, SHA-256, and convolutional neural network inference with N parties and full-threshold security is highly efficient, especially with our protocol conversions.

We compare MOTION's performance to other full-threshold frameworks. For biometric matching with N=3 parties, MOTION outperforms the passively secure implementations in MP-SPDZ [48] by 10.4×–17.6×, and the actively secure implementation in SCALE-MAMBA [2] by more than two orders of magnitude.

2 RELATED WORK

Practical MPC has been a very active field of research, especially in the past decade. Here, we provide an overview of the results most relevant to MOTION. Besides several theoretical foundations, we also discuss related implementations.

2.1 Theoretical Foundations

Yao's garbled circuits [74] and the protocol by Goldreich, Micali and Wigderson (GMW) [33] were the seminal works that introduced MPC with two and multiple parties, respectively. The protocol of Beaver, Micali and Rogaway (BMR) [8] can be seen as a multi-party variant of Yao's protocol. We provide a more detailed overview of these protocols in §6. We denote Yao's protocol and BMR with Y, the GMW protocol using Boolean sharing with B and using arithmetic sharing with A.

2.2 MPC Implementations and Frameworks

A thorough overview and categorization of MPC implementations is given in [36]. In this section, we summarize *mixed-protocol* MPC implementations (that support multiple protocols) and compare them with MOTION in Tab. 1.

2-Party Solutions. Fairplay [55] was one of the first works that showed practical feasibility of MPC by providing an implementation of Yao's protocol. The TASTY framework [40] was the first mixed-protocol framework, and it combined Yao's GCs and Homomorphic Encryption (HE). The ABY framework [29] is a mixed-protocol framework for secure two-party computation based on Oblivious Transfer (OT) [4]. ABY showed that using OT yields better efficiency than using HE in the online phase. Researchers from Baidu have recently re-implemented two-party arithmetic and Yao

Table 1. Related mixed-protocol MPC frameworks with N parties, threshold t, and active (●) or passive (◑) security. We denote the license of unpublished source code as '—'.

Framework	N	t	Security	Protocols	License
ABY [29]	2	1	◑	$A/B/Y$	LGPL-3.0
PrivC [39]	2	1	◑	A/Y	—
TASTY [40]	2	1	◑	A/Y	no license
EzPC [19]	2	1	◑	A/Y	MIT
OPA Mixing [46]	2	1	◑	any 2 of $A/B/Y$	MIT
ABY^3 [58]	3	1	● or ◑	$A/B/Y$	MIT
Sharemind [14]	3	1	● or ◑	A/B	payware[3]
ASTRA [20]	3	1	● or ◑	A/B	—
BLAZE [62]	3	1	● or ◑	A/B	—
Trident [64]	4	1	●	$A/B/Y$	—
SCALE-MAMBA [2]	≥ 2	$N-1$	●	A/Y	MIT-like
MP-SPDZ [48]	≥ 2	$N-1$	● or ◑	A/B or Y	MIT-like
MOTION (this work)	≥ 2	$N-1$	◑	$A/B/Y$	MIT

sharing protocols from ABY in their product-level framework PrivC [39]. Patra et al. [61] improved the conversions from ABY for a more efficient online phase and design hybrid circuits for machine learning tasks. Recently, a hybrid solution partitioned protocols into a part that is executed using classical MPC primitives and a part that is evaluated in an Intel SGX enclave [21].

3- and 4-Party Solutions. ABY^3 [58] is a mixed-protocol implementation with a focus on privacy-preserving machine learning with exactly 3 parties. BLAZE [62] and ASTRA [20] further improve upon the performance of ABY^3 in the same setting. Trident [64] proposes hybrid 4-party protocols. Sharemind [14] is a framework for both integer arithmetic and Boolean operations. All of these frameworks only allow up to a single corruption, whereas MOTION provides full-threshold security, which is stronger given the same adversary model (cf. §3.4). A comparison between different number of corruptions against non-matching adversary models (e.g., passively secure full-threshold vs. actively-secure honest-majority protocols) is non-trivial and out of scope of this work.

N-Party Solutions. FairplayMP [10] is an extension of Fairplay that implements the BMR protocol [8] with the setup phase computed using the honest-majority BGW protocol [12]. Choi et al. [22] provide an N-party passively secure Boolean GMW implementation. SDPZ [24] is an actively secure MPC protocol based on arithmetic sharing in prime fields. SPD$\mathbb{Z}_{2^k}$ [23] introduced integer computations modulo 2^k for this approach. [25] gave an efficient implementation of SPD$\mathbb{Z}_{2^k}$ with applications to machine learning. Zaphod [3] allows to efficiently combine BMR [8, 38] with the SPDZ protocol, which is integrated in the SCALE-MAMBA implementation [2]. However, SCALE-MAMBA currently implements only actively secure MPC protocols, which have substantially higher overhead than passively secure ones. An alternative implementation of the SPDZ protocol is provided by MP-SPDZ [48] that also includes implementations of other protocols. MP-SPDZ has recently also included conversions between arithmetic and Boolean sharing. MPC protocols with a large number of parties were implemented in [73] as part of the EMP toolkit [71, 72], which contains also implementations of other MPC protocols. The BMR protocol was implemented in [11].

[3] Only a Sharemind-*emulator* is available for free.

2.3 Compilers for MPC Protocols

Another line of research focuses on directly compiling existing code into an MPC protocol. The PCF compiler [51] processes C code and creates a compact intermediate circuit format that is evaluated by an interpreter for the actively secure Yao-based two-party protocol of [52]. Wysteria [65] is a multi-party framework that implements the GMW protocol and offers type-based security and correctness checks. Frigate [59] is a verified compiler for creation of circuits for MPC protocols that can be securely evaluated with MPC implementations. PICCO [76] is a source-to-source compiler for C programs that builds on arithmetic secret sharing using bit decomposition for bit operations. Obliv-C [75] compiles a special-purpose C-based language to plain C for evaluating it in Yao's garbled circuits. EzPC [19] is a secure 2-party computation framework that allows to generate an efficient partitioning for mixed computations based on Yao's garbled circuits and arithmetic sharing. The authors of [46] show efficient algorithms for computing an optimal partitioning for mixing any two MPC protocols. The HyCC compiler [17] allows compilation of C code into efficient mixed-protocol MPC. MOTION directly supports the evaluation of circuits generated by HyCC. A combination of private memory access using Oblivious RAM (ORAM) and Yao's garbled circuits is implemented in ObliVM [53] that compiles programs from a Java-like language.

3 PRELIMINARIES

In this section, we provide background information about our setting, the adversary model, and define the notation.

3.1 Notation

We abbreviate $[i] = \{1, \ldots, i\}$. We denote the number of parties as N and the parties themselves as $P_1, \ldots, P_N$. A value x that is shared between N parties, is denoted as tuple $\langle x \rangle^S = \left(\langle x \rangle_1^S, \ldots, \langle x \rangle_N^S\right)$, where the superscript $S \in \{A, B, Y\}$ denotes the sharing type (cf. §6), and the subscript $i \in [N]$ denotes the i-th share of x that is held by party P_i. We write $\langle \boldsymbol{x} \rangle^B$ or $\langle \boldsymbol{x} \rangle^Y$ in bold font for a vector of ℓ shared bits, which we interpret as an ℓ-bit unsigned integer or element of $\mathbb{Z}_{2^\ell}$. $\mathsf{Share}_i^S(x)$ denotes party P_i sharing their private value x in sharing S with all parties, while $\mathsf{Share}^S(x)$ denotes that all parties create a sharing of a public value x in sharing S. $\mathsf{Rec}_i^S\left(\langle x \rangle^S\right)$ denotes the reconstruction of a shared value $\langle x \rangle^S$ such that only party P_i receives x and $\mathsf{Rec}^S\left(\langle x \rangle^S\right)$ denotes the reconstruction of a shared value $\langle x \rangle^S$ such that all parties receive x. We use the symmetric security parameter κ. With $x \in_R X$ we denote that x is drawn uniformly at random from the set X. We denote $\boldsymbol{x}[i]$ as the i-th element from $\boldsymbol{x}$.

3.2 Secure Multi-Party Computation (MPC)

MPC protocols are run by N parties and typically divided into a *setup phase*, that is independent of the parties' inputs and can be precomputed, and an *online phase*, that starts when the parties supply their private inputs.

3.3 Outsourcing Scenario

Alternatively to running our protocols directly between the N parties, they can also be used in an outsourcing scenario in a natural way. As described in more detail in [47], in an outsourcing scenario an arbitrary number of input parties secret-share their private data to N non-colluding computing parties, who have no insight into that data. These computing parties evaluate our N-party MPC protocols and send the secret-shared output to a set of output parties, who can then reconstruct the plaintext output. Input and output parties can be (partially) identical or distinct. This allows for a large number of input/output parties without significantly increasing communication complexity of the protocols,

since input sharing and output reconstruction are cheap 1-round operations and also provide security against actively corrupted input/output parties. The number of computing parties N can be chosen in accordance to performance and security requirements.

3.4 Adversary Model

The protocols we consider in this work are secure against passively corrupted (semi-honest) adversaries that follow the protocol specification but try to infer additional information about the other parties' private inputs by inspecting the protocol transcript. This passive attacker model is useful in scenarios where the involved parties trust each other but are legally constrained to keep information confidential, e.g., when computing statistics on sensitive health records. In the outsourcing scenario (cf. §3.3), a prime use case for our protocols, the computing parties are trusted to not collude with each other and run in a secured network. Discovering active attacks would lead to an immense loss of reputation and would hurt the business model of offering privacy-preserving services. From a research perspective, advances in the passive security model often lead to advances in stronger adversary models and serve as a performance baseline to show general feasibility of MPC-based applications. Moreover, techniques like attestation and several protocol extensions can be used for ensuring security against stronger adversaries, which we leave as future work.

4 ARCHITECTURE OF OUR MOTION FRAMEWORK

MOTION implements mixed-protocol MPC with an *arbitrary* number of $N \geq 2$ parties with full-threshold security against passive adversaries. Since communication complexity inherently scales with N, the focus of this work is to achieve practical performance for relatively small N, e.g., $N \leq 16$, as also commonly used in an outsourcing setting (cf. §3.3). Our framework allows to implement MPC protocols in different security models by design, such as honest majority and/or actively secure protocols.

MOTION is implemented in C++ and uses many of the modern features that were introduced in the C++17 standard. In the first place, it is a library and can easily be used in external projects. Our implementation has only few dependencies, making it OS-independent and fully compatible with the very liberal MIT license. MOTION requires only the following third-party libraries: `Boost`[4] (for network communication, logging, parsing command line arguments, fibers, and statistics), `flatbuffers`[5] (for communication serialization), `fmt`[6] (for string processing), optionally `Google Test`[7] (for unit and component tests), and `OpenSSL`[8] (for cryptographic primitives). MOTION does not depend on any third-party OT or MPC libraries. We took the software development process of our framework very seriously and designed the whole system with great care. Extensive component tests are included for all of the framework parts in order to ensure the correctness and security of our implementation. We routinely test MOTION for memory leaks using the valgrind[9] framework and for other bugs using various sanitizers[10], such as the undefined behavior and address sanitizer, enabling us to fix possible problems early on. We will make our code public on GitHub[11] and will actively support and extend it in the future. Currently, our codebase consists of 179 source files, totaling in 36 000 lines of code, 20% of which are tests.

[4] https://www.boost.org
[5] https://github.com/google/flatbuffers
[6] https://github.com/fmtlib/fmt
[7] https://github.com/google/googletest
[8] https://www.openssl.org
[9] https://valgrind.org
[10] https://github.com/google/sanitizers/wiki
[11] https://encrypto.de/code/MOTION

4.1 Novel Design Aspects

Besides designing MOTION with great care, we enhance its architecture by including several novel design aspects that improve its usability and facilitate new use cases. Although a few of our design aspects have at least partially been addressed in previous frameworks, *their combination* is novel. The extensibility of our framework paves the way to integrate also other optimizations such as different circuit evaluation strategies and new MPC protocols in the future.

4.1.1 Communication Serialization. As mentioned by Shai Halevi in his keynote talk at ACM CCS'18 [35], the requirement of MPC frameworks to own a TCP socket has in the past hindered their adoption, e.g., in server applications, which often run under constrained permissions or have a proprietary messaging interface. This restriction is solved in MOTION by using communication serialization. In MOTION, *all* the transferred messages are serialized and contain metadata sufficient to make messages recognizable without having to rely on an order preserving channel, e.g., a TCP socket. To the best of our knowledge, MOTION is the first MPC framework, whose communication is completely serialized. The most important benefit of the communication serialization is that our framework neither needs to own a separate TCP connection, nor does it rely on TCP (or similar protocols) as transport protocol, as it was the case for all previous MPC frameworks. Also, communication serialization allows for MPC to be based on low-latency network protocols (e.g., QUIC [42] or RUDP [43]), which can yield substantial performance improvements in MPC as shown in [16, 69], and also facilitates the real-world MPC use in previously infeasible scenarios, such as MPC in many proprietary networks, Web Services, Remote Procedure Calls (RPCs), or even via peer-to-peer messengers without establishing separate connections for the MPC framework. In order to demonstrate and evaluate the functionality of our framework we use `Boost` for TCP connections between the parties, but we stress that exchanging the communication parts with a different networking protocol is intended by design and would only involve minimal code changes.

Furthermore, we use `flatbuffers`' schema files to define how the serialized communication is structured in MOTION. The schema files are independent of the programming language and can be compiled to a variety of different programming languages such as Java, Go, JavaScript, Rust, and even Swift[12]. These schema files significantly reduce the overhead of implementing MOTION or parts of it in a different programming language, e.g., to enable MPC on iPhones, while still supporting the original messaging interface. This approach enables multi-party protocols communication between our original C++ framework and many heterogeneous devices, operating systems, and programming languages.

4.1.2 Provider-based use of MPC Primitives. In our framework, the developer does not need to know how the MPC primitives interact with the network stack. Instead of the synchronized use of different MPC primitives on a network interface directly, we provide convenient provider interfaces for Oblivious Transfer (OT), Multiplication Triples (MTs) [6], Shared Bits (SBs), and Square Pairs (SPs), where the requirements for a program run can be registered, and are then automatically handled without any further action from the developer. The providers return pointers to the registered objects, which provide a separate interface to execute the desired protocol, e.g., set inputs to the OT functionality or wait for a batch of MTs to be computed.

Besides the convenient user interface, our providers enable the developer to easily replace the computation procedure of a primitive partially or completely, e.g., use a semi-trusted third party that generates correlated randomness (e.g., MTs) and distributes it among the computing parties instead of computing it using expensive crypto (cf. [66]).

4.1.3 Multiple Layers of Abstraction. MOTION is both developer-friendly and function-rich. On the one hand, it provides a convenient way of developing MPC solutions without significant MPC knowledge using secure type classes,

[12] https://github.com/mzaks/FlatBuffersSwift

Listing 1. Code excerpt for efficiently computing minimum squared Euclidean distance in MOTION.

```
using namespace MOTION;
using suint = SecureUnsignedInteger;
using vec = std::vector<suint>;

// variable a is an arithmetic GMW share
// variable v is a vector of arithmetic GMW shares
// computes squared Euclidean distance between
// a and each element in v
// returns BMR share of min. sqr. Euclidean distance
suint MinSqrEuclideanDistance(suint& a, vec& v){
  vec res(v.size()); // result vector

  // automatic use of more efficient squaring
  auto a_sqr = a*a, two_a = 2*a;

  // compute squared Euclidean distance
  // (a-v[i])^2 = a^2 + v[i]^2 - 2a*v[i]
  for(unsigned int i = 0; i < res.size(); ++i)
    res[i] = a_sqr + v[i] * (v[i] - two_a);

  // convert each distance to BMR sharing
  for(auto& e : res)
    e = e->Convert<MPCProtocol::BMR>();

  // select initial minimum as 0-th element
  auto min = res[0];

  // find the minimum distance
  for(unsigned int i = 1; i < res.size(); ++i){
    auto smaller = res[i] < min;
    //    smaller ? res[i] : min
    min = smaller.MUX(res[i], min);
  }
  return min;
}
```

which provide overloaded C++ operators and can be used just as the classic C++ types. Moreover, all of our abstract APIs operate directly in C++. This simplifies error handling and omits the need for the developer to learn a new domain-specific language that is accepted by the framework. On the other hand, the developer can also use our API to get access to the low-level MPC primitives and analyze or modify the underlying routines, e.g., add new optimizations or even use the MPC primitives standalone. The latter allows for using our framework, e.g., to only compute base OTs or OT extension. When using our MPC protocols or the underlying primitives, developers do not require any knowledge about how the protocols or primitives work together or interact with the message passing interface. To the best of our knowledge, MOTION is the first MPC framework that provides such a high level of abstraction while allowing to use all primitives directly. The other MPC frameworks either translate a special-purpose language to circuits (e.g., [17]), or low-level code (e.g., [75]), or they process commands/circuits by a compiled interpreter (e.g., [2]).

We give a small code snippet in List. 1 to illustrate the simplicity of the code in MOTION. Note that the example depicted in List. 1 is *manually* optimized for efficiency, and also a straightforward implementation would be fully functional but less efficient. For the convenient use of MOTION by non-experts in MPC, we recommend to utilize our HyCC adapter for efficiency reasons (cf. §4.1.4). Also, we show an advanced example of functionality extension of MOTION in List. 2 in App. A, where we implement a secure two-party protocol for multiplying an integer known by one party by a secret-shared bit.

4.1.4 HyCC Integration. If the use of an abstract language instead of the direct use of C++ classes is desired, e.g., if the developer has no expert knowledge in MPC and thus is not able to manually implement and optimize MPC protocols, the developer can import efficient hybrid circuits generated by the HyCC compiler [17] from a subset of the C programming language using our HyCC adapter. MOTION fully supports the features of HyCC and follows its partitioning guidelines, which results in protocols that are tailored according to a user-specified optimization goal. *Previous works:* HyCC is integrated in the ABY framework [29] for N=2-party MPC.

4.1.5 Asynchronous Gate Evaluation. MOTION allows the secure evaluation of arbitrary circuits without additional information about their structure. Each gate depends on its parents, and some gates require network communication for their evaluation. Thus, we are faced with a complex dependency graph consisting of possibly millions of interdependent tasks, which need to be scheduled on the available CPU cores. Evaluation of each gate in a separate thread is clearly infeasible if not impossible due to constraints of the operating system. Our solution is to use fibers, i.e., threads implemented in userspace, which are run on a fixed number of worker threads. For this, we use the `Boost.Fiber` library[13]. When a fiber is blocked, e.g., because a message has not yet been received, the worker thread switches to a different fiber and continues to evaluate a different gate. Compared to the overhead of a context switch between threads on the operating system level, switching between fibers is lightweight and possible in less than 100 CPU cycles, which is typically about an order of magnitude less than for a context switch between threads. Another advantage is that fibers can be used in the same way as usual threads. Thus, the implementation of a protocol is straightforward since the developer has access to all common synchronization mechanisms. We adapted a work-stealing scheduling algorithm from `Boost.Fiber` to our own thread pool, which creates one worker thread per CPU core by default. Also, we designed and implemented a number of efficient synchronization primitives and asynchronous access mechanisms for fibers to handle interactions between gates and MPC primitives. Note that we do not put any constraints on the number of physical processor cores used by a party. Moreover, MOTION allows parties to run with a different number of threads, which is a common problem of synchronized MPC where the work is scheduled for a static number of threads and/or communication channels before the protocol evaluation, and an inconsistent number of threads either yields wrong results or causes a program crash, e.g., [29, 67].

We evaluate all gates separately as soon as their parent gates become ready. In the following, we highlight two benefits of this approach. Firstly, the asynchronous gate evaluation decreases the online time for evaluating unbalanced circuits. Consider a scenario with high network latency and two major subcircuits, as shown in Fig. 1: one consisting of only few data-dependent layers with many costly non-XOR gates (blue), and the other subcircuit containing much fewer non-XOR gates but consisting of a large number of interactive layers (green). If evaluated layer-wise (as it is done in many current MPC frameworks), the large subcircuit has a blocking effect on the deep circuit due to the longer evaluation time. In our framework, the default scheduler evaluates gates in first come first serve order. On the other hand, the possibility to replace the default scheduler by a custom one with a different evaluation strategy is intended by design. The goal of a custom scheduler can be to prioritize gates according to the maximum subcircuit depth (i.e., gates that lead to the deepest subcircuit are evaluated first) to minimize communication latency, or to synchronize the evaluation layer-wise to evaluate in a batch all gates in a layer to possibly save bandwidth.

This asynchrony is especially beneficial in networks with high latency, e.g., for trans-continental Internet connections. Also, we batch operations for all input wires and the contained SIMD values of each gate in order to reduce communication. Secondly, integration of new protocols into MOTION becomes easier, since gate evaluation is independent of

[13] https://www.boost.org/doc/libs/1_73_0/libs/fiber/

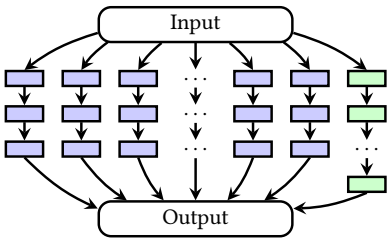

Fig. 1. Circuit with a large number of parallel gates (blue) and many data-dependent, sequential gates (green), that benefits from asynchronous gate evaluation.

other protocols by design (in contrast to most existing MPC frameworks). Thus, the developer does not need to know how the other parts of the framework work to integrate the new protocols.

Previous Works. Asynchronous gate evaluation was implemented in the no longer supported VIFF framework [26] using callbacks in Python. The callbacks, however, made the code unnecessarily complicated. The more recent VIFF derivate MPyC[14], uses native coroutines in Python. However, Python is a suboptimal choice for highly efficient MPC, because it is a scripting language and thus substantially less efficient than lower-level programming languages such as C and C++.

4.1.6 Code Vectorization. We design our code with vectorization of CPU instructions in mind to improve its efficiency. This goal is different from the MPC-level SIMD instructions (cf. §4.1.7) and affects the compiled code directly. Yet, explicit vectorization using architecture-specific instructions would limit the number of architectures MOTION supports. To achieve both better efficiency through (better) vectorization of CPU instructions and rich support for various architectures, most of our code is optimized such that the compiler can vectorize it automatically using the native instructions of the underlying architecture. This is achieved by using multiple techniques such as eliminating loop dependencies and branching, enforcing buffer alignment that matches the cache line size, and giving the compiler various hints to produce better code, e.g., using the `restrict` type qualifier. The few SSE instructions we used are supported by many architectures, and for those that do not support them, we automatically provide a slightly slower pure C++ code drop-in replacement.

Previous Works. To the best of our knowledge, so far the only MPC framework implemented in a low-level programming language with the possibility of cross-platform compilation is ABY [29].

4.1.7 Single Instruction Multiple Data (SIMD). We intentionally design the API of MOTION in a way that encourages the use of MPC-level Single Instruction Multiple Data (SIMD) instructions that process vectors of data instead of single data, e.g., vectors of bits instead of a single bit. This not only drastically reduces the memory footprint but also the required communication, since sending 1-bit values has significant overhead. This optimization results in a much better amortized efficiency and throughput, which we detail in §8. SIMD instructions are especially relevant for the outsourcing setting, where the outsourcing servers often simultaneously process data of many users. In our experiments, extensive use of SIMD instructions improved the throughput of MOTION by about an order of magnitude in both the LAN and WAN setting (cf. Tab. 4).

Previous Works. SIMD instructions have been used for $N \in \{2, 3\}$ in [14, 29, 68].

[14] https://github.com/lschoe/mpyc

4.1.8 Interleaved Setup and Online Phase. Circuit evaluation in MPC happens in one of two modes: sequential or interleaved ('pipelined'). The sequential mode runs the input-dependent online phase only after the input-independent setup has completed. This allows precise measurements of the setup and online phase communication and computation requirements, or full precomputation ahead of the online phase. Frameworks like ABY [29] support only this evaluation mode. The interleaved mode, on the other hand, runs both phases in parallel and facilitates possibly more efficient evaluation of the circuit in terms of load balancing, since the gates that otherwise would have been waiting for the setup phase to finish can be evaluated faster, thus improving the protocol latency.

Previous Works. A similar approach is used in SCALE-MAMBA [2]. However, it often overproduces correlated randomness in the setup phase, which is disadvantageous for small applications. In contrast, MOTION produces exactly the required amount of correlated randomness. To the best of our knowledge, we are the first framework to offer both sequential and interleaved circuit evaluation, giving the user the freedom of choice according to the use case.

4.2 Implemented Building Blocks

To make MOTION easy to use and extend, we design it in a completely different way compared to prior work. We did not use any existing implementations of the cryptographic protocols, since they would need to be completely redesigned. Below, we list the main components implemented in our framework that can also be of interest for other applications.

- OT* by Hauck and Loss [37] for base OTs (cf. §5.1.1).
- Providers for OT extension including Beaver's OT precomputation [7] and different OT flavors [5] (cf. §5.1.2): general, random (cf. §5.1.3), additively correlated, and XOR-correlated OTs (C-OTs) including our optimization of precomputed C-OT w.r.t. round complexity (cf. §5.1.3).
- Providers for Multiplication Triples (cf. §5.2), Squaring Pairs (cf. §5.3), and Shared Bits (cf. §5.4) using C-OTs.
- N-party full-threshold passively secure MPC protocols: Arithmetic GMW (cf. §6.1), Boolean GMW [33] (cf. §6.2), and BMR [8, 11] (cf. §6.3), and secure conversions between them (cf. §7).
- Plenty of utility classes, e.g., adapted `Boost.Fiber` for fibers, (aligned) bit vector, bit span, logger, run-time and communication statistics over multiple runs, function-encapsulating conditions, and reusable promises and futures.
- Unit and component tests for all of the implemented components and most of the utility classes.
- Application examples (cf. §8.2) that use MOTION as a library, e.g., MPC protocols for AES-128, SHA-256, minimum Euclidean distance, and Convolutional Neural Networks (CNNs).

5 MPC BUILDING BLOCKS

In this section, we describe the primitives that our protocols rely on, as well as our improvements to them.

5.1 Oblivious Transfer

Oblivious Transfer (OT) [63] is the basic building block for various generic and custom MPC protocols. It involves two parties, a sender $\mathcal{S}$ that inputs two messages (m_0, m_1), and a receiver $\mathcal{R}$ that inputs the choice bit $c \in \{0, 1\}$. The functionality outputs $\perp$ to $\mathcal{S}$ and only m_c to $\mathcal{R}$. It is guaranteed, that $\mathcal{S}$ does not learn c and that $\mathcal{R}$ does not learn m_{1-c}. This kind of OT is called 1-out-of-2 OT and it can be generalized to 1-out-of-n OT where $\mathcal{S}$ inputs $(m_0, \ldots, m_{n-1})$, $\mathcal{R}$ inputs $c \in \{0, \ldots, n-1\}$, and the functionality outputs $(\perp, m_c)$. Inherently, OT requires public-key cryptography [44], which is computationally expensive. However, the important *OT extension* technique proposed by Ishai et al. [45]

allows to use a small number of public-key-based "base OTs" and use them as seeds to compute a much larger number of OTs using significantly faster symmetric cryptography. OT can also be precomputed [7], moving a significant part of computation and communication from the online phase to the input-independent setup phase (cf. §3.2). To precompute an OT, $\mathcal{R}$ starts the OT extension protocol using a random $r \in_R \{0,1\}$. In the online phase, $\mathcal{R}$ sends $p := r \oplus c$ to $\mathcal{S}$, who swaps the messages if $p = 1$ and does nothing otherwise. Then, the parties proceed as in the original OT extension protocol. OT precomputation adds one sequential message to the OT extension protocol, thus increasing the number of communication rounds.

5.1.1 Base OTs. Abstractly speaking, the base OTs are computed as follows: for each $j \in [\kappa]$ (where κ is the symmetric security parameter, e.g., 128 bit), $\mathcal{S}$ inputs $(s_{j,0}, s_{j,1}) \in_R \{0,1\}^{2\kappa}$ and $\mathcal{R}$ inputs $c_j \in_R \{0,1\}$. $\mathcal{R}$ obtains s_{j,c_j} for each $j \in [\kappa]$. In this work, we use the base OT protocol by Hauck and Loss [37] (denoted as OT*). We use it in the random OT setting, i.e., (1) the choice bits of $\mathcal{R}$ are random, and (2) we omit the last step of the protocol for sending the messages to $\mathcal{R}$.

5.1.2 OT Extension. We use OT extension [45] with optimizations from [4, 5] and denote it as General OT (G-OT). The protocol is defined as follows: First, the parties run a base OT protocol with inverted roles. In the setup phase, $\mathcal{R}$ uses the sent messages from the base OTs to generate $\boldsymbol{T} \in \{0,1\}^{m \times \kappa}$ with $\boldsymbol{T}[j] = \mathrm{PRG}(s_{j,0})$ for each $j \in [\kappa]$ and sets $u_j = \mathrm{PRG}(s_{j,1}) \oplus \boldsymbol{T}[j] \oplus \boldsymbol{r}$, where m is the number of required OTs, $\boldsymbol{r}$ are $\mathcal{R}$'s real choices, and PRG is a pseudo-random generator. Then, $\mathcal{S}$ creates $\boldsymbol{V} \in \{0,1\}^{m \times \kappa}$ with $\boldsymbol{V}[j] = c_j u_j \oplus \mathrm{PRG}(s_{j,c_j})$ for each $j \in [\kappa]$, where c_j is the choice bit in the j-th base OT. Finally, both parties transpose their matrices: $\mathcal{S}$ sets $\boldsymbol{V'} = \boldsymbol{V}^T$ and $\mathcal{R}$ sets $\boldsymbol{T'} = \boldsymbol{T}^T$.

In the online phase, $\mathcal{S}$ sends to $\mathcal{R}$ $y_{i,0} := x_{i,0} \oplus H(i, \boldsymbol{V'}[i])$ and $y_{i,1} := x_{i,0} \oplus H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c})$ for each $i \in [m]$, where $H(\cdot)$ is a one-way pseudo-random function. $\mathcal{R}$ sets the output of the OT $i \in [m]$ as $x_{i,r_i} := y_{i,r_i} \oplus H(i, \boldsymbol{T'}[i])$. We instantiate both PRG and H using AES (cf. §5.5).

5.1.3 OT Flavors. In many cases, OT-based MPC protocols need to compute very specific functionalities using OT. Asharov et al. [4, 5] have first shown that OT extension can be done significantly more efficiently for specific tasks.

Random OT (R-OT). Random OT can essentially be seen as a truncated OT extension protocol with no inputs. The parties run the same protocol steps as in OT extension, but omit the last step where $\mathcal{S}$ masks his messages and sends them to $\mathcal{R}$. Instead, the parties only compute their masks and set them as the output of the protocol. Slightly more formally, $\mathcal{S}$ sets $(H(i, \boldsymbol{V'}[i]), H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c}))$ and $\mathcal{R}$ sets $H(i, \boldsymbol{T'}[i])$ as his output. R-OT can be used to compute other OT functionalities such as G-OT and correlated OT (C-OT) or to compute MTs in 2PC [5].

Correlated OT (C-OT). Correlated OT is a special OT flavor that is very well-suited for MPC. Its main use case is multiplication of a (secret-shared) bit or string by a (secret-shared) bit yielding a secret-shared multiplication result. The functionality of C-OT is as follows: $\mathcal{S}$ inputs bit-string x and $\mathcal{R}$ inputs bit r. The functionality outputs $(x_0, x_0 \odot x)$ to $\mathcal{S}$ and $x_0 \odot rx$ to $\mathcal{R}$, where x_0 is random and $\odot$ is usually bit-wise XOR, which we denote as XOR-correlated C-OT ($\mathrm{C}^{\oplus}$-OT), or addition $\bmod 2^\ell$, which we denote as additively correlated C-OT (C^{+}-OT). The $\mathrm{C}^{\oplus}$-OT results in an XOR-sharing of the multiplication, and C^{+}-OT is an additively shared multiplication. The difference to G-OT is that instead of sending two masked messages, $\mathcal{S}$ sends only one message $y_i = x_{i,1} \odot H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c}) = x_{i,0} \odot x_i \odot H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c}) = H(i, \boldsymbol{V'}[i]) \odot x_i \odot H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c})$ and $\mathcal{R}$ sets $x_{i,r_i} = r_i y_i \odot H(i, \boldsymbol{T'}[i])$ for each $i \in [m]$.

Our optimization for precomputed C-OT. We introduce a generic factor two improvement of the online round complexity of precomputed C-OT. To simplify the description of the C-OT precomputation, we reuse the R-OT protocol.

First, the parties compute an R-OT with the choice bit $r \in_R \{0,1\}$. In order to obtain the correct message mask for his real choice c, $\mathcal{R}$ sends $p = 0$ if $r = c$ and $p = 1$ otherwise. After obtaining p, $\mathcal{S}$ swaps the messages iff $p = 1$, and proceeds with the original protocol. Our improvement is based on the observation that in C-OT (in contrast to G-OT) the message of $\mathcal{S}$ is equal in both cases ($p = 0$ and $p = 1$), i.e., $y_i := H(i, \boldsymbol{V'}[i]) \odot x_i \odot H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c}) = H(i, \boldsymbol{V'}[i] \oplus \boldsymbol{c}) \odot x_i \odot H(i, \boldsymbol{V'}[i])$. However, p may change the *output of* $\mathcal{S}$. This is why $\mathcal{S}$ sends y_i independent of the choice bit and waits for receiving p to determine his own correct output, whereas $\mathcal{R}$ sends p, waits for y_i, and unmasks it. Our improved precomputed C-OT protocol results in two messages that are sent *independently*, which improves the online phase latency by a factor of 2. Its latency is equal to the original C-OT protocol without precomputation.

5.2 Multiplication Triples

Multiplication triples (MTs), proposed by Beaver [6] allow to reduce the online complexity of MPC protocols by precomputing random triples of the form $\left(\langle a\rangle^S, \langle b\rangle^S, \langle c\rangle^S\right)$ such that $c = a \cdot b$. Here, $S \in \{A, B\}$ denotes additive secret sharing over $\mathbb{Z}_{2^\ell}$ or $\{0,1\}$, respectively. In the online phase, the triples can be used to privately compute multiplications with only linear operations and reconstructions while avoiding costly cryptographic operations (cf. §6.1, §6.2).

5.2.1 Arithmetic MTs (A-MTs). For ℓ-bit A-MTs (over the ring $\mathbb{Z}_{2^\ell}$), we generalize the C$^+$-OT-based A-MT generation protocol by Demmler et al. [29] from two to N parties. Namely, each party P_i locally generates two random shares $\langle a\rangle_i^A \in_R \mathbb{Z}_{2^\ell}$ and $\langle b\rangle_i^A \in_R \mathbb{Z}_{2^\ell}$, and then the parties interactively compute $\langle c\rangle^A \leftarrow \langle a\rangle^A \cdot \langle b\rangle^A$. Note that the product can be written as $a \cdot b = \left(\sum_i \langle a\rangle_i^A\right) \cdot \left(\sum_i \langle b\rangle_i^A\right) = \sum_i \left(\langle a\rangle_i^A \cdot \langle b\rangle_i^A\right) + \sum_{i,j\neq i} \left(\langle a\rangle_i^A \cdot \langle b\rangle_j^A\right) \pmod{2^\ell}$. Each party P_i can compute $\langle a\rangle_i^A \cdot \langle b\rangle_i^A$ locally, and to compute $\langle a\rangle_i^A \cdot \langle b\rangle_j^A$ with $i \neq j$ we run the following secure multiplication protocol between P_i and P_j such that each of the two parties obtains an additive share of the product.

To perform a secure multiplication the two parties P_i and P_j, owning the values $x, y \in \mathbb{Z}_{2^\ell}$, respectively, run ℓ parallel C$^+$-OTs. Here, P_i acts as the sender and inputs x as correlation to each C$^+$-OT and obtains $r_k \in \mathbb{Z}_{2^\ell}$ from the k-th C$^+$-OT with $k \in [\ell]$. P_j acts as receiver and for each $k \in [\ell]$ inputs the k-th bit of y (denoted by $\boldsymbol{y}[k]$) to the k-th C$^+$-OT, and obtains $r_k + \boldsymbol{y}[k] \cdot x$ as output. Finally, P_i sets $z_i \leftarrow -\sum_{k=1}^{\ell} r_k 2^{k-1} \bmod 2^\ell$ and P_j sets $z_j \leftarrow \sum_{k=1}^{\ell} 2^{k-1}(r_k + \boldsymbol{y}[k] \cdot x) \bmod 2^\ell$ such that $z_i + z_j = x \cdot y \pmod{2^\ell}$. As observed in [29], the online communication for the C$^+$-OTs can be halved by omitting the most significant bits of the values that will be cut off by multiplication with 2^{k-1} modulo 2^ℓ in the subsequent computation. The total communication of ℓ-bit A-MT generation with N parties and symmetric security parameter κ is $\approx N(N-1)\ell(\kappa + \ell/2)$ bits, and requires two sequential messages.

5.2.2 Boolean MTs (B-MTs). In this work, we use C$^\oplus$-OT to compute B-MTs. The protocol is analogous to the one for A-MT computation (cf. §5.2.1) with the difference that here we use C$^\oplus$-OT instead of C$^+$-OT, and the 2× communication reduction does not apply. The total communication of the B-MT generation protocol is $N(N-1)(\kappa + 1)$ bits, and also requires two sequential messages.

5.3 Square Pairs

In addition to A-MTs, we also compute *square pairs* (SPs), introduced in [27], which are pairs of random secret-shared values $\left(\langle a\rangle^A, \langle c\rangle^A\right)$ such that $c = a^2$. They can be generated analogously to A-MTs (cf. §5.2.1) but require only a single secure multiplication between each pair of parties and, thus, only half the number of C$^+$-OTs (and hence communication) compared to MTs in the same sharing. SPs are used to compute squaring operations more efficiently than using a normal multiplication (cf. §6.1).

5.4 Shared Bits

Another form of precomputation are *shared bits* (SBs) [27], which are arithmetic sharings $\langle b \rangle^A$ over $\mathbb{Z}_{2^\ell}$ of random bits $b \in \{0,1\}$. Note that, from a shared bit $\langle b \rangle^A$ we can compute a Boolean sharing $\langle \boldsymbol{b} \rangle^B$ of the same bit with $\langle \boldsymbol{b} \rangle_i^B \leftarrow \langle b \rangle_i^A \bmod 2$. We generate SBs with an adapted version of Π_{RandBit} from [25], and use square pairs (cf. §5.3) to compute the required squaring in $\mathbb{Z}_{2^{\ell+2}}$ (cf. §6.1). Hence, $N(N-1)(\ell+2)/2$ C^+-OTs with additive correlation in the ring $\mathbb{Z}_{2^{\ell+2}}$ are needed per shared bit over $\mathbb{Z}_{2^\ell}$. We use SBs to convert from Boolean to arithmetic secret sharing (cf. §7.1).

5.5 Fixed-Key AES

The implemented OT extension protocol (cf. §5.1.2) and the garbling scheme in the BMR protocol (cf. §6.3) make extensive use of hash and pseudorandom functions. Thus, if instantiated inefficiently, these can become the main bottleneck of MPC. Modern CPUs have dedicated instruction sets for performing cryptographic operations such as AES in hardware (e.g., AES-NI on x86). As these instructions are substantially faster than a software implementation, it is natural to utilize these primitives to speed up higher level protocols. Since the AES key schedule is still quite inefficient, constructions using a fixed key have been used for garbling schemes [9]. In our framework, we instantiate hash and pseudorandom functions for OT extension and BMR garbling (cf. §6.3) with fixed-key AES following the approach of Guo et al. [34] and Ben-Efraim et al. [11].

5.6 Bandwidth-Saving Broadcast

Broadcast is a common building block in many of the protocols implemented in MOTION (cf. §6,§7). Typically, each party sends some data of size ℓ bit to every other party, whereupon the shares are accumulated, e.g., via XOR. Using point-to-point channels, this results in $N(N-1)\ell$ bit of total communication. Since we consider the passive security setting, we can reduce this to $2(N-1)\ell$ bit by letting everyone send its part to a designated party who performs the accumulation and broadcasts the result. Now the communication is no longer quadratic but instead linear in the number of parties. This comes at the cost of one additional round of communication.

6 MPC PROTOCOLS

In this section, we describe the established passively secure full-threshold MPC base protocols Arithmetic sharing (§6.1), Boolean sharing with GMW (§6.2), and Yao sharing with BMR (§6.3). We indicate their use in protocols as A, B, and Y, respectively. We provide a detailed analysis of the communication and computation cost, depending on the number of parties N and bit length ℓ for primitive operations and conversions in MOTION in Tab. 2.

6.1 Arithmetic Sharing (A)

As arithmetic sharing, we use a variant of the GMW protocol [33] over the ring $\mathbb{Z}_{2^\ell}$ with support for evaluating arithmetic circuits consisting of addition and multiplication gates. The protocol uses additive secret sharing, i.e., a value $x \in \mathbb{Z}_{2^\ell}$ is shared among the N parties as $\langle x \rangle^A = \left(\langle x \rangle_1^A, \ldots, \langle x \rangle_N^A\right) \in \mathbb{Z}_{2^\ell}^N$ such that $x = \sum_{j=1}^N \langle x \rangle_j^A \pmod{2^\ell}$ and party P_i holds share $\langle x \rangle_i^A$. In the following, we assume a fixed bit length ℓ.

For input sharing $\mathsf{Share}_i^A(x)$, party P_i samples $\langle x \rangle_1^A, \ldots, \langle x \rangle_N^A \in_R \mathbb{Z}_{2^\ell}$ such that $x = \sum_{j=1}^N \langle x \rangle_j^A \pmod{2^\ell}$, and sends $\langle x \rangle_j^A$ to party P_j. The communication can be avoided by sampling $\langle x \rangle_j^A$ from the output of a PRG whose seed is known only to parties P_i and P_j. We instantiate the PRG for sharing functionalities in GMW with AES-128 in counter mode. For the output reconstruction $\mathsf{Rec}_i^A\left(\langle x \rangle^A\right)$, each party P_j sends $\langle x \rangle_j^A$ to party P_i who computes $x \leftarrow \sum_{j=1}^N \langle x \rangle_j^A$.

Table 2. Total costs of primitive operations and conversions: the number of symmetric cryptographic operations, the number of bits sent by all parties, and the number of sequential messages required for the conversions.

	Computation [# symm. crypt. ops]		Communication [# bits]		# Messages	
	Setup	Online	Setup	Online	Setup	Online
ADD^A, XOR^B, XOR^Y	0	0	0	0	0	0
MUL^A	$2\ell N(N-1)$	0	$\ell N(N-1)(\kappa+\ell/2)$	$\ell N(N-1)$	2	1
AND^B	$2N(N-1)$	0	$N(N-1)\kappa$	$2N(N-1)$	1	1
AND^Y	$8N(N-1)$	N^2	$N(N-1)((N+1)4\kappa+1)$	0	6	0
Share^A, Share^B	$2(N-1)$	0	0	0	0	0
Share^Y	0	0	0	$(N-1)(N\kappa+1)$	0	2
Rec^A	0	0	0	$\ell N(N-1)$	0	1
Rec^B	0	0	0	$N(N-1)$	0	1
Rec^Y	0	0	$N(N-1)$	0	1	0
Y2B	0	0	0	0	0	0
B2Y	0	0	0	$N(N-1)(\kappa+1)$	0	2
A2B, *A2Y*	$(\ell-1)\cdot 8(2N^3-3N^2+N)$	$(\ell-1)(N^3-N^2)$	$(\ell-1)\cdot N(N-1)^2((N+1)4\kappa+1)$	$\ell N(N-1)(N\kappa+1)$	6	2
B2A, *Y2A* (via *B*)	$\ell N(N-1)(\ell+2)$	0	$\ell N(N-1)(\ell+2)(\kappa+\ell+2)/2$	$N(N-1)\ell$	2	1
Y2A (w/o online comm.)	$(\ell-1)N^2(17N-9)$	$(\ell-1)N^2$	$N(N-1)\big(\ell(N\kappa+2)+(\ell-1)N((N+1)4\kappa+1)\big)$	0	6	0

For $\text{Rec}^A\big(\langle x\rangle^A\big)$, each party P_j broadcasts $\langle x\rangle_j^A$ and computes $x \leftarrow \sum_{j=1}^N \langle x\rangle_j^A$. Alternatively, each P_j could send $\langle x\rangle_j^A$ to P_1, who reconstructs $x \leftarrow \sum_{j=1}^N \langle x\rangle_j^A$ and sends it back to all parties. This requires an additional round, but in total only $O(N\ell)$ instead of $O(N^2\ell)$ bits of communication, which can be used as a trade-off for low-latency networks with limited bandwidth.

Linear operations can be computed locally, i.e., without communication: E.g., for $\langle z\rangle^A \leftarrow a\cdot\langle x\rangle^A + \langle y\rangle^A + b$ with public values $a, b \in \mathbb{Z}_{2^\ell}$, party P_1 computes $\langle z\rangle_1^A \leftarrow a\cdot\langle x\rangle_1^A + \langle y\rangle_1^A + b$, and all other parties $P_2,\ldots,P_N$ compute $\langle z\rangle_i^A \leftarrow a\cdot\langle x\rangle_i^A + \langle y\rangle_i^A$.

Multiplications can be computed using multiplication triples (MTs) (cf. §5.2.1): Let $\big(\langle a\rangle^A, \langle b\rangle^A, \langle c\rangle^A\big)$ be an MT for $\mathbb{Z}_{2^\ell}$. For $\langle z\rangle^A \leftarrow \langle x\rangle^A \cdot \langle y\rangle^A$, the parties first reconstruct the input values masked with a, b from the MT as $d \leftarrow \text{Rec}^A\big(\langle x\rangle^A - \langle a\rangle^A\big)$, and $e \leftarrow \text{Rec}^A\big(\langle y\rangle^A - \langle b\rangle^A\big)$. Then, they jointly compute the result as the linear computation $\langle z\rangle^A \leftarrow \langle c\rangle^A + e\cdot\langle x\rangle^A + d\cdot\langle y\rangle^A - d\cdot e$. Multiplications can also be computed with less communication at cost of an additional communication round by applying the communication-saving reconstruction method described above to the computation of d and e. However, the total number of communication rounds and, therefore, also the online run-time of GMW depends linearly on the multiplicative depth of the circuit. Thus, we see the communication-saving reconstruction as a more expensive option for a small number of parties in arithmetic GMW.

Squaring is computed more efficiently with square pairs (SPs) (cf. §5.3) [27] using only half of the communication compared to a normal multiplication: Let $(\langle a\rangle^A, \langle c\rangle^A)$ be an SP for $\mathbb{Z}_{2^\ell}$. For $\langle z\rangle^A \leftarrow \langle x\rangle^A \cdot \langle x\rangle^A$, the parties compute $d \leftarrow \text{Rec}^A\big(\langle x\rangle^A - \langle a\rangle^A\big)$ and $\langle z\rangle^A \leftarrow \langle c\rangle^A + 2\cdot d\cdot\langle x\rangle^A - d^2$.

6.2 Boolean Sharing with GMW (B)

Boolean GMW [33] uses XOR-based secret sharing, which is equivalent to additive secret sharing in the ring $\mathbb{Z}_2$, where addition and multiplication correspond to XOR ($\oplus$) and AND ($\wedge$), respectively. Hence, this is a special case of the arithmetic sharing (cf. §6.1) with bit length $\ell = 1$, and allows the evaluation of Boolean circuits. A value $x \in \{0,1\}$ is shared among the N parties as $\langle x\rangle^B = \big(\langle x\rangle_1^B,\ldots,\langle x\rangle_N^B\big)$ such that $x = \bigoplus_{i=1}^N \langle x\rangle_i^B$ where party P_i holds $\langle x\rangle_i^B$. All basic operations are computed analogously to those in arithmetic sharing.

We write $\langle \boldsymbol{x}\rangle^B$ for a vector of ℓ shared bits interpreted as an ℓ-bit integer or element of $\mathbb{Z}_{2^\ell}$. In this context, $\langle \boldsymbol{x}\rangle^B + \langle \boldsymbol{y}\rangle^B$ denotes addition and $\langle \boldsymbol{x}\rangle^B \cdot \langle \boldsymbol{y}\rangle^B$ denotes multiplication in $\mathbb{Z}_{2^\ell}$. Basic operations are done using depth-optimized Boolean circuits [28, 68].

Analogously to §6.1, we use direct broadcast for the reconstruction of MTs. Significant effort was put into constructing low-depth circuits to achieve efficiency in GMW that is competitive with depth-independent Yao's garbled circuits [28, 68]. Therefore, doubling the latency due to the communication-saving reconstruction will likely be disadvantageous.

More efficient AND Gates without MTs. The use of Boolean Multiplication Triples (B-MTs) instead of direct secure bit multiplication is motivated by their very cheap online phase with exactly *one* communication round, low communication, and only cleartext operations. Computation of a B-MT requires a secure bit multiplication, which makes $2\binom{N}{2}$ calls to precomputed C-OT (cf. §5.1.3) in the setup phase, and $\mathsf{Rec}^B(\langle e\rangle^B, \langle d\rangle^B)$ in the online phase (cf. §5.2.2). The secure multiplication protocol using our optimized C-OT also requires only one round in the online phase (instead of two with non-optimized C-OT), but only *one* round in the setup phase (instead of two for MT). Moreover, it also transfers exactly 4 bits in the online phase between each (P_i, P_j) with $i \neq j$, but compared to MTs 4 bits *less* in the setup phase. Also, this saves $2N + 1$ cleartext additions and multiplications in total, and (identical to using MTs) it needs only cleartext operations in the online phase.

6.3 Yao Sharing with BMR (Y)

The BMR protocol [8] is an extension of Yao's garbled circuits protocol [74] to the multi-party case. Instead of the garbled circuit being constructed by one party and evaluated by the other, it is garbled by all parties collaboratively and then evaluated by each party locally. During the setup phase, the parties engage in a garbling protocol such that no set of up to $N - 1$ parties gains enough information to recover any intermediate values in the resulting garbled circuit. In the online phase, after the input values have been shared, the garbled circuit can be evaluated by each party without further communication. Therefore, the round complexity of the BMR protocol is independent of the circuit, whereas in GMW it is linear in the multiplicative depth of the circuit. In the following, we use the notation by [11]. Moreover, we implement the free-XOR technique for BMR introduced by [11], which allows to evaluate XOR gates for free, i.e., without any communication or cryptographic operations during the setup or online phase.

In the setup phase, each party P_i generates a global key offset $R^i \in_R \{0,1\}^\kappa$, and shares λ_w^i of random permutation bits $\lambda_w := \bigoplus_{j=1}^N \lambda_w^j$ and pairs of keys $k_{w,0}^i, k_{w,1}^i$ for each wire w in the circuit: If w is an input wire of the circuit such that party P_j provides that input, then $\lambda_w^i \in_R \{0,1\}$ if $i = j$ and $\lambda_w^i = 0$ otherwise. If w is *not* the output of an XOR gate, the share of the permutation bit $\lambda_w^i \in_R \{0,1\}$ and the key $k_{w,0}^i \in_R \{0,1\}^\kappa$ are chosen randomly. If w is the output of an XOR gate with input wires a, b, then $\lambda_w^i := \lambda_a^i \oplus \lambda_b^i$ and $k_{w,0}^i := k_{a,0}^i \oplus k_{b,0}^i$. The second key $k_{w,1}^i$ is in both cases implicitly defined as $k_{w,0}^i \oplus R^i$. If w is an output wire of the circuit, then all λ_w^i are sent to the party (or the parties) collecting that output.

Furthermore, the parties invoke the following garbling functionality $\mathcal{F}_{GC}$: It takes R^i and $\lambda_w^i, k_{w,0}^i, k_{w,1}^i$ for all wires w from each party P_i as inputs. Let F^2 be a double-key PRF and let $\circ$ denote concatenation in the following. We instantiate F^2 with a fixed-key AES construction [11, 34] (cf. §5.5). The garbling functionality $\mathcal{F}_{GC}$ computes for each AND gate g, and for all $j \in [N]$ and $\alpha, \beta \in \{0,1\}$:

$$\tilde{g}_{\alpha,\beta}^j \leftarrow \left(\bigoplus_{i=1}^N F^2_{k_{a,\alpha}^i, k_{b,\beta}^i}(g \circ j)\right) \oplus k_{w,0}^j \oplus \left(R^j \cdot ((\lambda_a \oplus \alpha)(\lambda_b \oplus \beta) \oplus \lambda_w)\right).$$

It outputs $\tilde{g}_{\alpha,\beta}^1 \circ \cdots \circ \tilde{g}_{\alpha,\beta}^N$ for all g and $\alpha, \beta \in \{0,1\}$.

In our framework, we instantiate $\mathcal{F}_{GC}$ with the OT-based protocol by [11] achieving a BMR instantiation with full corruption threshold. Their garbling protocol uses one bit $\mathrm{C}^{\oplus}$-OT and three correlated $\mathrm{C}^{\oplus}$-OTs of strings of length κ per pair of parties to generate the garbled tables of an AND gate with inputs a, b and output w:

First the parties securely compute $\lambda_{ab} := \lambda_a \cdot \lambda_b$ such that each party P_j receives a random share λ^j_{ab} using two 1-bit $\mathrm{C}^{\oplus}$-OTs per pair of parties. Then they locally compute $\lambda_{a\bar{b}w} := \lambda_a \cdot \overline{\lambda_b} \oplus \lambda_w$, by setting $\lambda^j_{a\bar{b}w} := \lambda^j_{ab} \oplus \lambda^j_a \oplus \lambda^j_w$, and analogously $\lambda_{\bar{a}bw} := \overline{\lambda_a} \cdot \lambda_b \oplus \lambda_w$ and $\lambda_{\bar{a}\bar{b}w} := \overline{\lambda_a} \cdot \overline{\lambda_b} \oplus \lambda_w$. As a third step, the parties securely compute $R^j \cdot ((\lambda_a \oplus \alpha) \cdot (\lambda_b \oplus \beta) \oplus \lambda_w)$ for all $j = 1, \ldots, N$ and $\alpha, \beta \in \{0, 1\}$. The multiplications can be done by eight κ-bit $\mathrm{C}^{\oplus}$-OTs per pair of parties: For all parties $P_j \neq P_i$, P_j inputs R^j as correlation and $P_i \neq P_j$ inputs λ_{abw}, $\lambda_{\bar{a}bw}$, $\lambda_{a\bar{b}w}$, and $\lambda_{\bar{a}\bar{b}w}$ as choice bits. Let $\rho^i_{j,\alpha,\beta}$ denote the resulting share of P_i of the product, i.e., the output of the corresponding $\mathrm{C}^{\oplus}$-OT. Finally the garbled tables $\{\tilde{g}^j_{0,0}, \tilde{g}^j_{0,1}, \tilde{g}^j_{1,0}, \tilde{g}^j_{1,1}\}_{j=1}^N$ are computed as follows: For $j = 1, \ldots, N$, and $\alpha, \beta \in \{0, 1\}$, party P_j broadcasts

$$F^2_{k^j_{a,\alpha}, k^j_{b,\beta}}(g \circ j) \oplus k^j_{w,0} \oplus \rho^j_{j,\alpha,\beta}$$

and all other parties P_i broadcast

$$F^2_{k^i_{a,\alpha}, k^i_{b,\beta}}(g \circ j) \oplus \rho^i_{j,\alpha,\beta}.$$

The XOR of these messages yields the table entry $\tilde{g}^j_{\alpha,\beta}$. Ben-Efraim et al. [11] noticed that one of the κ-bit $\mathrm{C}^{\oplus}$-OTs can be saved in the third step: Instead of computing and using $\rho^i_{j,1,1}$ as described above, the table entry $\tilde{g}^j_{1,1}$ is computed as XOR of the values

$$F^2_{k^j_{a,\alpha}, k^j_{b,\beta}}(g \circ j) \oplus k^j_{w,0} \oplus R^j \oplus \rho^j_{j,0,0} \oplus \rho^j_{j,0,1} \oplus \rho^j_{j,1,0}$$

from P_j and

$$F^2_{k^i_{a,\alpha}, k^i_{b,\beta}}(g \circ j) \oplus \rho^i_{j,0,0} \oplus \rho^i_{j,0,1} \oplus \rho^i_{j,1,0}$$

from each other party P_i.

Hazay et al. [38] improve the aforementioned garbling protocol making use of the fact that R^j is fixed. They set the choice bits in the base OTs to R^j which allows to compute C-OTs directly instead of computing them from R-OTs [60] and reduces the communication by κ bits for each C-OT. This optimization results in $20/16/13/11\%$ less communication for $N = 2/3/4/5$ parties, respectively. Using the bandwidth-saving broadcast from §5.6, the improvement is 20% and independent of the number of parties.

As an alternative to [38], we present a *novel* garbling optimization that uses the OT extension as a black box, allowing arbitrary OT extension instantiations, and achieves the same amortized communication cost. For a circuit consisting of m AND gates, the garbling protocol in [11], as described above, uses $3m \cdot N(N-1)$ $\mathrm{C}^{\oplus}$-OTs of κ-bit strings in total. Here, we show how to reduce this to $\kappa \cdot N(N-1)$ $\mathrm{C}^{\oplus}$-OTs of $3m$-bit strings. First, note that we can swap the inputs of the OTs and use κ $\mathrm{C}^{\oplus}$-OTs of 3-bit strings to compute the shares $\rho^i_{j,\alpha,\beta}$. Let $\hat{\lambda}^i_g \in \{0, 1\}^3$ be the triple $(\lambda^i_{abw}, \lambda^i_{a\bar{b}w}, \lambda^i_{\bar{a}bw})$ for the gate g. Then, we can use the bits of R^j as choice bits and $\hat{\lambda}^i_g$ as correlation in the $\mathrm{C}^{\oplus}$-OT. By using the concatenation of all $\hat{\lambda}^i_1, \ldots, \hat{\lambda}^i_m$ as correlation, we can use the same $\kappa N(N-1)$ $\mathrm{C}^{\oplus}$-OT for all gates.

During the online phase, each party P_i holds for a wire w a public value $\alpha_w = \lambda_w \oplus x$ (with permutation bit λ_w and real value x), keys $k^j_{w,\alpha}$ for $j = 1, \ldots, N$ (and $k^i_{w,1-\alpha} = k^i_{w,\alpha} \oplus R^i$), and an additive share λ^i_w of the permutation bit. We can write this in the form of a sharing as $\langle x \rangle^Y = \left(\lambda^1_x, \ldots, \lambda^N_x; \left(\alpha, k^1_{x,\alpha}, \ldots, k^N_{x,\alpha}\right)\right)$ where the part after the semicolon denotes public information.

Given the setup as described above, we now describe the basic operations of this sharing during the online phase: For $\mathsf{Share}_i^Y(x)$, party P_i (holding λ) broadcasts $\alpha = x \oplus \lambda$, and each party P_j broadcasts k_α^j. For $\mathsf{Rec}_i^Y(\langle x \rangle^Y)$, party P_i (holding λ) computes $x \leftarrow \alpha \oplus \lambda$. Let $\langle x \rangle^Y = (\lambda_x^1, \ldots, \lambda_x^N; (\alpha, k_{x,\alpha}^1, \ldots, k_{x,\alpha}^N))$, and $\langle y \rangle^Y = (\lambda_y^1, \ldots, \lambda_y^N; (\beta, k_{y,\beta}^1, \ldots, k_{y,\beta}^N))$ be two shared values. The XOR of these $\langle z \rangle^Y \leftarrow \langle x \rangle^Y \oplus \langle z \rangle^Y$ can be computed using a free-XOR technique: With $\gamma \leftarrow \alpha \oplus \beta$, and $k_{z,\gamma}^j \leftarrow k_{x,\alpha}^j \oplus k_{y,\beta}^j$ for all $j \in [N]$, we get $\langle z \rangle^Y = (\lambda_z^1, \ldots, \lambda_z^N; (\gamma, k_{z,\gamma}^1, \ldots, k_{z,\gamma}^N))$. For an AND gate g, the corresponding garbled tables must be decrypted: For $j \in [N]$, the next key is computed as $k_{z,\gamma}^j \leftarrow \tilde{g}_{\alpha,\beta}^j \oplus \bigoplus_{i=1}^N F_{k_{x,\alpha}^i, k_{y,\beta}^i}(g \circ j)$. Then, each party P_i can deduce γ by checking whether $k_{z,\gamma}^i = k_{z,0}^i$ or $k_{z,\gamma}^i = k_{z,1}^i$ holds. Basic operations can be done using size-optimized Boolean circuits [50, 70].

7 MPC PROTOCOL CONVERSIONS

We present secure and efficient conversions between the three protocols (cf. §6) to enable passively secure hybrid MPC. This allows to use different protocols for different parts of an application to exploit the respective advantages of each protocol: Additions and multiplications, for example, are more efficient in arithmetic GMW (A), whereas a Boolean circuit evaluated with Boolean GMW (B) or BMR (Y) is often the better choice for comparisons. We summarize the costs of all six conversions in Tab. 2 on page 15.

7.1 Boolean to Arithmetic Sharing – *B2A*

For converting a Boolean sharing $\langle \boldsymbol{x} \rangle^B$ of ℓ bits into an arithmetic sharing $\langle x \rangle^A$ over $\mathbb{Z}_{2^\ell}$ such that x equals $\boldsymbol{x}$ when interpreted as an element of $\mathbb{Z}_{2^\ell}$, we present two variants

Straightforward: Additive Masking. As described in prior work [3, 29, 40] in different settings, the conversion can be computed as follows: A random mask is added to the input value in the Boolean sharing. The result is reconstructed and shared again in the arithmetic sharing where the mask is subtracted.

The mask can be generated in the online phase by letting each party share a random value. However, since the mask is input-independent, it could also be generated by running a subprotocol during the setup phase. Here, we assume that we have a pair $(\langle r \rangle^A, \langle \boldsymbol{r} \rangle^B)$ of sharings of the same value $r \in_R \mathbb{Z}_{2^\ell}$. To convert the sharings, the parties compute $\langle \boldsymbol{t} \rangle^B \leftarrow \langle \boldsymbol{x} \rangle^B - \langle \boldsymbol{r} \rangle^B$, $t \leftarrow \mathsf{Rec}^B(\langle \boldsymbol{t} \rangle^B)$, $\langle t \rangle^A \leftarrow \mathsf{Share}^A(t)$, and $\langle x \rangle^A \leftarrow \langle t \rangle^A + \langle r \rangle^A$.

This requires at least $\Omega(\log \ell) + 1$ rounds of communication in the online phase for computing the subtraction circuit in GMW [18, 68] and the subsequent reconstruction. Moreover, one pair of sharings $(\langle r \rangle^A, \langle \boldsymbol{r} \rangle^B)$ for $r \in_R \mathbb{Z}_{2^\ell}$, generated in the setup phase, is required.

Optimized: Using Shared Bits. In our implementation, we adapt the approach from [25] for SPD$\mathbb{Z}_{2^k}$ to our setting and use so called *shared bits*. A shared bit is a pair of sharings $(\langle r \rangle^A, \langle r \rangle^B)$ of a random bit $r \in_R \{0, 1\}$.

Let $\langle \boldsymbol{x} \rangle^B = (\langle x_0 \rangle^B, \ldots, \langle x_{\ell-1} \rangle^B)$ with the least significant bit $\langle x_0 \rangle^B$. Given the shared bits $(\langle r_i \rangle^A, \langle r_i \rangle^B)$ for $i = 0, \ldots, \ell - 1$, we can convert $\langle \boldsymbol{x} \rangle^B$ into an arithmetic sharing as follows: For each bit $i = 0, \ldots, \ell - 1$, the parties compute in parallel $\langle t_i \rangle^B \leftarrow \langle x_i \rangle^B \oplus \langle r_i \rangle^B$, $t_i \leftarrow \mathsf{Rec}^B(\langle t_i \rangle^B)$, and $\langle x_i \rangle^A \leftarrow t_i + \langle r_i \rangle^A - 2t_i \langle r_i \rangle^A$. Thereafter, the output sharing is computed as $\langle x \rangle^A \leftarrow \sum_{i=0}^{\ell-1} 2^i \cdot \langle x_i \rangle^A$.

This costs only one round of communication for the reconstruction of the t_i during which $N(N-1)\ell$ bits are transmitted, and ℓ shared bits, which are generated during the setup phase (cf. §5.4).

7.2 Boolean to Yao Sharing – $B2Y$

The straight-forward way to do the $B2Y$ conversion of a shared value $\langle x\rangle^B$ would be that each party P_i reshares its Boolean share $\langle x\rangle_i^B$ in Yao sharing as $\langle x_i\rangle^Y \leftarrow \mathsf{Share}_i^Y\left(\langle x\rangle_i^B\right)$ and the parties compute $\langle x\rangle^Y \leftarrow \bigoplus_{j=1}^N \langle x_i\rangle^Y$. The sharing requires two rounds of communication and has a total communication cost of $N(N-1)(N\kappa+1)$ bits, which is in $O(N^3\kappa)$.

The properties of the BMR sharing allow the following natural optimization for the $B2Y$ conversion (also implemented by [3]): Let w be the BMR wire that is supposed to obtain the value x. Note that party P_i holds (in addition to its Boolean share $\langle x\rangle_i^B$) also a share λ_w^i of the random permutation bit $\lambda_w = \bigoplus_{j=1}^N \lambda_w^j$, and keys $k_{w,0}^i, k_{w,1}^i = k_{w,0}^i \oplus R^i$, which are generated during the BMR setup phase (cf. §6.3). For the conversion, each party P_i first broadcasts $\alpha_i \leftarrow \langle x\rangle_i^B \oplus \lambda_w^i$. Then, every party P_i computes $\alpha \leftarrow \bigoplus_{j=1}^N \alpha_j$ and broadcasts $k_{w,\alpha}^i$. Then $\langle x\rangle^Y := \left(\lambda_w^1,\ldots,\lambda_w^N; \left(\alpha, k_{w,\alpha}^1,\ldots,k_{w,\alpha}^N\right)\right)$ is a valid Yao sharing of x since $\alpha = \bigoplus_{j=1}^N \alpha_j = \bigoplus_{j=1}^N \langle x\rangle_i^B \oplus \lambda_w^j = x \oplus \lambda_w$. This optimized conversion requires also two rounds but only $N(N-1)(\kappa+1)$ bits of communication, which is in $O(N^2\kappa)$. This is an improvement by a factor of $\frac{(N\kappa+1)}{(\kappa+1)} \approx N$ over the straight-forward solution.

7.3 Yao to Boolean Sharing – $Y2B$

Let $\langle x\rangle^Y = \left(\lambda_x^1,\ldots,\lambda_x^N; \left(\alpha, k_{x,\alpha}^1,\ldots,k_{x,\alpha}^N\right)\right)$ be the Yao sharing of a value $x \in \{0,1\}$. As described in §6.3, the public value α is the real value x masked with the random permutation bit $\lambda_x = \bigoplus_{j=1}^N \lambda_x^j$, i.e., $\alpha = x \oplus \lambda_x$. Hence, the shared permutation bit is already a Boolean sharing $\langle \alpha \oplus x\rangle^B = (\lambda_x^1,\ldots,\lambda_x^N)$, and the parties compute $\langle x\rangle^B \leftarrow \langle \alpha \oplus x\rangle^B \oplus \alpha$ (cf. §6.1 and §6.2), i.e., party P_1 computes $\langle x\rangle_1^B \leftarrow \lambda_1 \oplus \alpha$ and all other parties $P_2,\ldots,P_N$ set $\langle x\rangle_j^B := \lambda_j$. Then, we have obtained a Boolean sharing $\langle x\rangle^B$ of x since $\bigoplus_{j=1}^N \langle x\rangle_j^B = \alpha \oplus \lambda = x$. $Y2B$ can be computed locally and hence is for free.

7.4 Arithmetic to Yao Sharing – $A2Y$

Given an arithmetic sharing $\langle x\rangle^A = (\langle x\rangle_1^A,\ldots,\langle x\rangle_N^A)$ over $\mathbb{Z}_{2^\ell}$ we want to obtain a Yao sharing $\langle \boldsymbol{x}\rangle^Y$ of ℓ bits such that $\boldsymbol{x}$ equals x when interpreted as element of $\mathbb{Z}_{2^\ell}$. To achieve this, every party P_i first shares its additive share of $\langle x\rangle^A$ in the Yao sharing: $\langle \boldsymbol{x}_i\rangle^Y \leftarrow \mathsf{Share}_i^Y\left(\langle x\rangle_i^A\right)$. Then, they compute $\langle \boldsymbol{x}\rangle^Y \leftarrow \sum_{j=1}^N \langle \boldsymbol{x}_j\rangle^Y$ using a Boolean circuit for addition. The conversion requires two rounds of communication for the sharing (cf. §6.3), and the evaluation of $N-1$ addition circuits in BMR with $O(\ell N)$ AND gates in total.

7.5 Arithmetic to Boolean Sharing – $A2B$

There are two options for converting an arithmetic sharing $\langle x\rangle^A$ over $\mathbb{Z}_{2^\ell}$ into a Boolean sharing $\langle \boldsymbol{x}\rangle^B$ of ℓ bits, such that $\boldsymbol{x}$ equals x when interpreted as element of $\mathbb{Z}_{2^\ell}$.

We can do the analogue of $A2Y$ (cf. §7.4) in Boolean sharing. However, this requires $O(\log N \cdot \log \ell)$ rounds of communication to compute the $N-1$ additions using depth-optimized addition circuits [18, 68].

In order to avoid the additional communication rounds, we first convert $\langle x\rangle^A$ to $\langle \boldsymbol{x}\rangle^Y$ (cf. §7.4), and then $\langle \boldsymbol{x}\rangle^Y$ to $\langle \boldsymbol{x}\rangle^B$ for free (cf. §7.3). Hence, $A2B$ has the same costs as $A2Y$.

7.6 Yao to Arithmetic Sharing – $Y2A$

Straightforward: Via $Y2B$ and $B2A$. We implemented the conversion of a Yao sharing $\langle \boldsymbol{x}\rangle^Y$ of ℓ bits into an arithmetic sharing $\langle x\rangle^A$ over $\mathbb{Z}_{2^\ell}$ by first converting $\langle \boldsymbol{x}\rangle^Y$ into a Boolean sharing $\langle \boldsymbol{x}\rangle^B$ for free with $Y2B$ (cf. §7.3), and then applying

$B2A$ (cf. §7.1) to obtain $\langle x \rangle^A$. Hence, the costs of $Y2A$ are the same as for $B2A$: one round of communication during the online phase.

Optimized: Without Online Communication. Furthermore, we present a *novel* conversion protocol that computes $Y2A$ *without any online communication*: This conversion requires a precomputed pair $\left(\langle r \rangle^A, \langle \boldsymbol{r} \rangle^Y\right)$ consisting of an arithmetic sharing $\langle r \rangle^A$ and a Yao sharing $\langle \boldsymbol{r} \rangle^Y$ of the same randomly chosen value $r \in_R \mathbb{Z}_{2^\ell}$. Since r is sampled independently of the overall protocol's inputs, it can be generated beforehand in the setup phase (see below). We first describe the *online* phase of the conversion. Given a sharing $\langle \boldsymbol{x} \rangle^Y$ of a value x and a pair as described above, we compute an arithmetic sharing $\langle x \rangle^A$ of x as follows: First, the input value x is masked with r in Yao sharing $\langle \boldsymbol{t} \rangle^Y \leftarrow \langle \boldsymbol{x} \rangle^Y - \langle \boldsymbol{r} \rangle^Y$. Then, the masked value is reconstructed $t \leftarrow \mathsf{Rec}^Y\left(\langle \boldsymbol{t} \rangle^Y\right)$, and shared arithmetically $\langle t \rangle^A \leftarrow \mathsf{Share}^A(t)$. Finally, the mask is removed in arithmetic sharing $\langle x \rangle^A \leftarrow \langle t \rangle^A + \langle r \rangle^A$. Note that each of these steps can be computed without any communication in the online phase: The subtraction circuit in Yao sharing and $\mathsf{Rec}^Y(\cdot)$ can be computed locally due to the properties of the BMR protocol (cf. §6.3). Also, $\mathsf{Share}^A(\cdot)$ and addition in arithmetic sharing do not require any online communication (cf. §6.1). The input-independent pair $\left(\langle r \rangle^A, \langle \boldsymbol{r} \rangle^Y\right)$ can be generated in the *setup* phase as follows: Every party P_i samples $\langle r \rangle_i^A \in_R \mathbb{Z}_{2^\ell}$ resulting in a sharing $\langle r \rangle^A = \sum_{j=0}^{N} \langle r \rangle_j^A$. Then $\langle \boldsymbol{r} \rangle^Y$ is obtained by applying a $A2Y$ conversion to $\langle r \rangle^A$ (cf. §7.4). To reduce communication costs, we can also compute the summation in Boolean sharing (cf. §7.5) and convert to BMR afterwards (cf. §7.2). This improves the setup phase by a factor of $O(N)$ in communication at the expense of more communication rounds in the setup phase.

8 PERFORMANCE EVALUATION

Along with the code base of MOTION, we provide the code and benchmarks for multiple applications that use MOTION as a C++ library. In this section, we evaluate the performance of MOTION and of these applications. We compare the applications' performance with other MPC implementations that also offer full-threshold security, i.e., protocols that *increase* their level of security by adding more parties. We do not compare with frameworks such as [14, 20, 58, 62, 64] that involve multiple parties for performance improvements, but offer only security against a single corruption, as this would not be meaningful comparison. We run benchmarks in two different environments: our own servers connected via a local network and several AWS servers.

Our servers: We evaluate the performance on five servers, each equipped with an Intel Core i9-7960X processor and 128 GB RAM, connected via a 10 Gbps network. For this benchmarking environment, we define two network settings to analyze how our framework behaves in different scenarios.

- LAN: The network is used as is with 10 Gbps bandwidth and 0.25 ms RTT. This setting represents parties in a fast LAN or an outsourcing scenario (cf. §3.3) with servers located in a network with low latency and high bandwidth, e.g., computing parties connected at an Internet Exchange Point (IXP).
- WAN: `tc`[15] is used to limit the network bandwidth to 1 Gbps and simulate an average RTT of 100 ms, simulating parties connected over the Internet. The scenarios covered by this setting are, for example, ad-hoc MPC over the Internet run by normal users and outsourcing computation to servers located in distinct locations, e.g., each server is owned by a different company in a different country.

AWS servers: To perform experiments with a larger number of parties, we use 10, 15, or 20 `r5.8xlarge` instances on AWS EC2, located in the same availability zone[16]. Each instance has 32 vCPUs using Intel Xeon Platinum 8175 or worse

[15] http://man7.org/linux/man-pages/man8/tc.8.html
[16] Exact location omitted for anonymous submission.

processors with 256 GiB memory and a 10 Gbps network connection[17]. We measured a bandwidth between 4.8 Gbps and 9.6 Gbps and an RTT between 0.043 ms and 0.079 ms among the instances. This setting represents two use cases: (1) a direct use of MPC between many parties, e.g., for privacy-preserving auctions, and (2) outsourcing to many servers of which all but one can be passively corrupted to achieve a very high level of privacy, e.g., for privacy-preserving computation on genomic data.

We average most of our benchmarks over 100 iterations. On AWS, we run 10 to 25 iterations to reduce the required time and costs. MOTION includes the functionality of automatically collecting extensive run-time and communication statistics. These numbers can be viewed for individual executions, and separate parts of protocols and primitives (e.g., OTs, MTs, etc.), as well as aggregated numbers for an entire batch of executions, including average numbers and their standard deviation. While MOTION can easily support other communication protocols (cf. §4.1.1), we used TCP in our benchmarks. We have run several benchmarks with the TCP traffic tunneled through a TLS channel using `stunnel`[18] and did not observe any noticeable performance overhead.

8.1 Microbenchmarks

We provide extensive microbenchmarks and communication requirements for primitive MPC operations and conversions, as well as microbenchmarks for integer operations in MOTION that can serve as guidelines for protocol design and cost estimation. Due to space limitations, we provide these results in App. A.

Table 3. Run-times in nanoseconds for one OT, amortized over 10 million parallel evaluations, averaged over 100 runs.

	G-OT		R-OT	$C^{\oplus}$-OT		C^{+}-OT			
Bit size	1	128	128	1	128	8	16	32	64
libOTe [67] LAN	—	120	—	—	130	—	—	—	—
MOTION LAN (this work)	151	196	77	131	147	119	121	126	134
libOTe [67] WAN	—	820	—	—	874	—	—	—	—
MOTION WAN (this work)	1 069	1 221	957	932	973	955	960	972	980

8.1.1 Performance of OT Extension (Tab. 3). In Tab. 3, we compare our OT extension implementation with the libOTe library by Peter Rindal [67]. The major part of libOTe is written in assembly and is, hence, very efficient. libOTe provides interfaces for single OT batches, which are explicitly associated with a communication channel, and operates directly on network sockets without message serialization and thus requires to manually synchronize all uses of different OTs. Taking the above into account, libOTe is easy to use and efficient in small MPC applications but, unfortunately, is often inconvenient for constructing complex MPC protocols. Our `OTProvider` class provides an abstract non-blocking API to request and use OTs without any knowledge about the underlying communication channel or other OTs.

We compare the efficiency of our OT extension implementation on 128-bit $C^{\oplus}$-OT, which is one of the core components of the BMR protocol (cf. §6.3) and 128-bit G-OT, which can be used for implementing other MPC protocols. For a total of 10 million parallel 128-bit $C^{\oplus}$-OT evaluations averaged over 100 runs, libOTe is only 10% faster than our `OTProvider`. In the same setting, libOTe's 128-bit G-OT implementation is $1.6\times$ faster than our 128-bit G-OT implementation. In the WAN setting, the performance difference is slightly smaller: libOTe outperforms our `OTProvider` for 128-bit C-OT by

[17] https://aws.amazon.com/ec2/instance-types/
[18] https://www.stunnel.org/

Table 4. Total (online+setup) run-times in seconds for biometric matching, comparing several implementations and protocols over various domains for N parties, bitlength ℓ and multiple database sizes. We benchmarked MOTION and ABY [29] with circuits generated with the HyCC compiler [17]. In MOTION, the run-times with SIMD are amortized over 192 / 32 parallel circuits for DB sizes of 1 024 / 4 096. Best run-times are in **bold**.

							LAN		WAN	
							DB Size		DB Size	
Implementation	Protocol	Domain	Security	N	Thresh.	ℓ	1 024	4 096	1 024	4 096
ABY [29]	A+B	$\mathbb{Z}_{2^\ell}$	passive	2	1	32	0.26	0.89	2.6	4.1
ABY [29]	A+Y	$\mathbb{Z}_{2^\ell}$	passive	2	1	32	**0.24**	**0.76**	**2.5**	**3.6**
MP-SPDZ [48]	MASCOT [49]	$\mathbb{F}_p$	active	3	2	32	45.78	174.90	1 150.4	4 596.0
MP-SPDZ [48]	MASCOT [49]	$\mathbb{F}_p$	passive	3	2	32	9.56	36.78	935.0	3 746.0
MP-SPDZ [48]	SPD$\mathbb{Z}_{2^k}$ [23]	$\mathbb{Z}_{2^\ell}$	active	3	2	64	57.25	231.53	1 643.8	6 580.2
MP-SPDZ [48]	SPD$\mathbb{Z}_{2^k}$ [23]	$\mathbb{Z}_{2^\ell}$	passive	3	2	64	3.89	13.80	1 126.1	4 500.5
MP-SPDZ [48]	FKOS15 [31]	binary	active	3	2	32	104.76	413.25	3 456.6	13 772.6
MP-SPDZ [48]	OT-based	binary	passive	3	2	32	4.17	14.80	1 346.4	5 289.3
SCALE-MAMBA [2]	Full-Threshold	$\mathbb{F}_p$	active	3	2	32	128.95	253.19	858.9	2 033.0
MOTION (**this work**)	A+B	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	5.74	19.99	15.4	41.3
MOTION (**this work**) w/ SIMD	A+B	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	**0.22**	**1.33**	**1.2**	**5.2**
MOTION (**this work**)	A+Y	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	5.71	21.78	10.2	29.2
MOTION (**this work**) w/ SIMD	A+Y	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	0.26	1.55	1.8	7.5

factor 1.1 and G-OT by factor 1.5. Taking into account the additional overhead for the communication serialization and the much higher level of abstraction in MOTION, the performance difference between the implementations is very small. To further improve the efficiency of our `OTProvider`, it is possible to replace parts of our code with assembly code as was done in libOTe. However, we aim to avoid this by design to make our code portable to different platforms like ARM.

8.1.2 Boolean Circuits: BMR vs. GMW. Since the state-of-the-art BMR protocol [11] undoubtedly incurs higher overhead than GMW, the authors of [11] created artificial circuits to show that circuits with very high depth can be evaluated faster in BMR than in GMW in high-latency networks. Here, we give a *real-world* example where BMR is more efficient than GMW even in the LAN setting with *low* network latency.

In the experiments on our servers, the evaluation of integer division circuits generated using HyCC [17] was always faster in BMR than in GMW. In the LAN setting, the difference was $1.1\times$–$1.4\times$, whereas in the WAN setting the factors were between $3\times$ and $5.3\times$. For the 3-party 64-bit integer division, the run-time difference between BMR and GMW in the WAN setting was 296 ms, which is equivalent to the run-time of 127 secure 64-bit additions or 30 secure 64-bit multiplications in 3-party GMW, and thus is significant. This substantial difference is due to the very high depth of the division circuit, which ranges from depth 65 for 8-bit division to depth 2 218 for 64-bit division. However, on the AWS servers with high bandwidth, low latency, and 10 to 20 parties, BMR performs worse and scales worse than GMW due to its substantially higher run-time and communication overhead. More detailed results are provided in Tab. 8 in App. A.

8.1.3 Comparison with N-Party GMW [22]. Compared to the passively secure N-party Boolean GMW implementation by Choi et al. [22], which requires amortized 4.61 µs to evaluate one AND gate by three parties, MOTION requires only 0.55 µs (cf. Tab. 7 in App. A), which is $8.4\times$ faster. Our better run-times can be explained by our more efficient OT extension implementation, and the use of MPC-level SIMD instructions (cf. §4.1.7). Both implementations were benchmarked on the same hardware.

8.2 Applications

In this section, we benchmark the run-times for the secure evaluation of real-world applications in MOTION and compare them with other full-threshold MPC frameworks. We run all these implementations on identical hardware using the same network conditions.

8.2.1 Biometric Matching (Tab. 4). Here, we analyze the overhead of moving from passively secure full-threshold MPC to actively-secure full-threshold MPC by comparing our framework with the SCALE-MAMBA framework [2] and with multiple protocols implemented in MP-SPDZ [48]. We also compare with the passively secure 2-party ABY framework [29]. As application, we use biometric matching that computes the Euclidean distance between a 4-dimensional sample and a database of biometric samples and then determines the minimum distance. A code example for the 2-dimensional case is provided in Listing 1 on page 8. We give the run-times in Tab. 4 for 2^{10} and 2^{12} database entries. Apart from benchmarks for the HyCC biomatch circuit [17] that is evaluated in a non-SIMD fashion, we provide a native MOTION implementation for the biometric matching with equivalent functionality but utilizing SIMD instructions evaluating 200 parallel circuits for 1 024 elements and 40 parallel circuits for 4 096 elements. The latter results in $16\times$–$34\times$ amortized speedup in the LAN setting and in a $42\times$–$221\times$ amortized speedup in the WAN setting (cf. Tab. 4).

Comparison with SCALE-MAMBA [2] & MP-SPDZ [48]. For SCALE-MAMBA, we set up a 3-party scenario with full-threshold security. For the passively secure versions of the MASCOT [49] and SPD$\mathbb{Z}_{2^k}$ [23] protocol, we compiled mixed circuits, which are more efficient, whereas the actively secure versions of these protocols turned out to be more efficient when running plain, non-hybrid circuits in the respective sharing. As a default we used values with a bitlength ℓ=32 bits, but had to run some measurements with ℓ=64 bit values, due to limitations of the respective implementation.

Comparing MOTION's run-times from Tab. 4 with those of the passively secure protocols of MP-SPDZ, we can see that the HyCC biometric matching circuit in MOTION is from $1.6\times$ slower to $1.7\times$ faster in the LAN setting, and $61\times$–$255\times$ faster in the WAN setting. With enabled SIMD support, MOTION outperforms MP-SPDZ and SCALE-MAMBA in all settings: it is at least $8.9\times$ / $271\times$ faster in the LAN / WAN setting than the fastest protocol implemented in MP-SPDZ (passive) or SCALE-MAMBA (active).

Comparison with ABY [29]. The passively secure two-party ABY framework outperforms most other implementations. As shown in Tab. 4, biometric matching in ABY is from slightly slower to $1.8\times$ faster than in MOTION in the LAN setting. This is mainly due to the higher cost of the *A2B* conversion in MOTION, which requires multiple addition circuits instead of one, and because BMR incurs higher communication and computation costs than two-party garbled circuits. In the WAN setting where the communication plays a greater role than in the LAN setting, we measured from $2.0\times$ faster to $1.4\times$ slower run-times in MOTION than in ABY, which is due to the more efficient communication using SIMD instructions in MOTION.

8.2.2 AES-128 and SHA-256 (Tab. 5). Here, we provide a comparison of the overhead needed to move from passively secure Secure Two-Party Computation (2PC) to passively secure full-threshold Secure N-Party Computation by comparing the ABY framework [29] with MOTION. Amortized run-times for securely evaluating 1 000 parallel invocations of AES-128 and SHA-256 in MOTION are given in Tab. 5. An important observation from this table is that for both AES and SHA the run-time of GMW (*B*) in the WAN setting is dominated by the online time, which cannot be precomputed,

Table 5. Total and online run-times in milliseconds for the evaluation of the Bristol Fashion circuits [1] for AES-128 with key scheduling and SHA-256 in GMW (B) and BMR (Y) executed with MOTION and Choi et al.'s GMW [22]. The run-times are amortized over 512 / 256 executions of AES-128 / SHA-256.

			LAN			WAN		
	Implementation		N=2	N=3	N=5	N=2	N=3	N=5
AES	Choi et al. [22]	B	27.3	42.2	84.8	28.9	44.4	90.0
	ABY [29]	B	0.2	—	—	8.5	—	—
	online	B	<0.1	—	—	6.6	—	—
	ABY [29]	Y	0.2	—	—	1.9	—	—
	online	Y	0.1	—	—	0.1	—	—
	MOTION (**this work**)	B	1.9	2.5	3.8	13.8	14.9	18.9
	online	B	0.4	0.5	0.8	7.3	7.7	8.1
	MOTION (**this work**)	Y	4.7	8.0	17.1	61.1	87.8	141.7
	online	Y	0.2	0.2	0.4	0.5	0.7	0.9
SHA	Choi et al. [22]	B	80.7	128.5	254.5	104.1	150.7	287.6
	ABY [29]	B	1.5	—	—	339.8	—	—
	online	B	0.7	—	—	334.1	—	—
	ABY [29]	Y	1.5	—	—	8.1	—	—
	online	Y	0.5	—	—	0.6	—	—
	MOTION (**this work**)	B	8.3	10.8	16.0	500.2	572.4	614.1
	online	B	2.7	3.2	4.5	479.6	547.9	538.4
	MOTION (**this work**)	Y	19.0	29.6	61.8	201.9	279.3	492.6
	online	Y	1.3	1.6	1.8	1.4	2.1	2.7

whereas the online time of BMR (Y) is only a small fraction of the total run-time. BMR has substantially higher run-times in the LAN setting, but has a much faster online phase in the WAN setting.

Comparison with N-Party GMW [22]. A comparison of our GMW (B) implementation with the passively secure GMW implementation of Choi et al. [22] is given in Tab. 5. In the LAN setting, MOTION is 14×–22× faster for AES and 10×–16× for SHA. In the WAN setting, MOTION is 2×–5× faster for AES and 1.7×–2× slower for SHA. Surprisingly, the Choi et al.'s implementation is almost as fast in the WAN as in the LAN setting, and is faster in the WAN setting even for N=5 parties than ABY [29] (that implements highly efficient N=2-party protocols), which we cannot explain. Note that we did not try to verify the correctness of their implementation. Also, MOTION is able to evaluate 10× as many SHA circuits in the *same* time, while Choi et al.'s implementation only supports circuits of very limited size. Moreover, their circuits have to be provided in a custom file format, whereas MOTION has builtin support for multiple circuit formats such as the commonly used Bristol (Fashion) format [1]. Also, Choi et al.'s implementation does not distinguish between setup and online phase.

Comparison with ABY [29]. In ABY using two-party Boolean GMW (B), we securely computed 100 000 AES evaluations in the LAN setting in 20.0 s. In contrast, MOTION requires 183.4 s using N=3-party GMW, which is 9.2× slower, and 303.8 s using N=5-party GMW, which is 15.2× slower. This difference results from the more efficient N=2-party protocols (that, however, provide weaker security guarantees than full-threshold protocols for more than two parties) implemented in ABY and the substantially higher level of abstraction in MOTION (cf. §4). Although the workload of each party increases with the total number of parties, the difference between three and five-party GMW in MOTION is only minor (1.65×) due to the substantially better load balancing with more parties. As expected, the high-depth

Table 6. Total run-times in seconds for CryptoNets [32] with HyCC hybrid circuits [17] for N parties and full threshold.

		LAN				WAN			
Implementation		N=2	N=3	N=4	N=5	N=2	N=3	N=4	N=5
ABY [29]	A+B	0.5	—	—	—	3.2	—	—	—
	A+Y	0.5	—	—	—	3.4	—	—	—
MOTION (**this work**)	A+B	3.5	4.2	4.9	5.7	6.7	8.0	9.8	13.2
	A+Y	3.6	4.2	4.9	5.8	6.7	8.6	12.6	13.1

SHA-256 circuit can be evaluated faster in Y than in B. However, for N=2 parties Yao's GCs are 25× faster than BMR in the WAN setting, which indicates the significant gap between the efficiency of both protocols. The MOTION run-times here are extrapolated from the run-times in Tab. 5.

Comparison with BMR [11]. The original passively secure OT-based BMR implementation by Ben-Efraim et al. [11] requires approximately 1 s (698±930 ms setup and 138±88 ms online time) for a single AES-128 evaluation by N=3 parties with 75 ms average network latency and 10 Gbps network bandwidth. MOTION takes 1.3 s in the same network setting (as their code is not publicly available, we use slightly different machines), which is similar to the run-times in [11]. By evaluating 1 000 AES circuits in parallel, we achieve an amortized run-time of 61 ms, which is at least 16× faster than [11].

8.2.3 Privacy-Preserving Machine Learning (Tab. 6). MOTION can be used for privacy-preserving machine learning. We give benchmarks for privately evaluating a convolutional neural network for handwriting recognition in Tab. 6. For our benchmarks, we use the hybrid circuits generated by HyCC [17] for CryptoNets [32] with ReLU as activation function. In the case of N=2 parties, MOTION is slower than ABY [29]: 7× in the LAN and 2× in the WAN setting, because our protocols are generic for N parties, whereas ABY has optimized protocols for exactly N=2 parties only. When increasing the number of parties and hence obtaining better security due to the full threshold protocols in MOTION, the performance of MOTION decreases only slightly, e.g., 1.6× for N=5 vs. N=2 parties in LAN and 2.0× in WAN.

Acknowledgments. This project has 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/251-805230, 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] V. A. Abril, P. Maene, N. Mertens, D. Sijacic, N. Smart. "'Bristol Fashion' MPC Circuits". *https://homes.esat.kuleuven.be/~nsmart/MPC/*. 2019.

[2] A. Aly, M. Keller, D. Rotaru, P. Scholl, N. P. Smart, T. Wood. "SCALE-MAMBA". *https://homes.esat.kuleuven.be/~nsmart/SCALE/*. 2018.

[3] A. Aly, E. Orsini, D. Rotaru, N. P. Smart, T. Wood. "Zaphod: Efficiently Combing LSSS and Garbled Circuits in SCALE". In: *Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC'19)*. ACM, 2019, pp. 33–44.

[4] G. Asharov, Y. Lindell, T. Schneider, M. Zohner. "More Efficient Oblivious Transfer and Extensions for Faster Secure Computation". In: *CCS'13*. ACM. 2013, pp. 535–548.

[5] G. Asharov, Y. Lindell, T. Schneider, M. Zohner. "More Efficient Oblivious Transfer Extensions". In: *Journal of Cryptology (JoC)* 3 (2017), pp. 805–858.

[6] D. Beaver. "Efficient Multiparty Protocols using Circuit Randomization". In: *CRYPTO'91*. Springer. 1991, pp. 420–432.

[7] D. Beaver. "Precomputing Oblivious Transfer". In: *CRYPTO'95*. Springer. 1995, pp. 97–109.

[8] D. Beaver, S. Micali, P. Rogaway. "The Round Complexity of Secure Protocols". In: *STOC'90*. ACM, 1990, pp. 503–513.

[9] M. Bellare, V. T. Hoang, S. Keelveedhi, P. Rogaway. "Efficient Garbling from a Fixed-Key Blockcipher". In: *S&P'13*. IEEE, 2013, pp. 478–492.

[10] A. Ben-David, N. Nisan, B. Pinkas. "FairplayMP: A System for Secure Multi-Party Computation". In: *CCS'08*. ACM. 2008, pp. 257–266.

[11] A. Ben-Efraim, Y. Lindell, E. Omri. "Optimizing Semi-Honest Secure Multiparty Computation for the Internet". In: *CCS'16*. ACM, 2016, pp. 578–590.

[12] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *STOC'88*. ACM. 1988, pp. 1–10.

[13] M. Blanton, P. Gasti. "Secure and Efficient Protocols for Iris and Fingerprint Identification". In: *ESORICS'11*. Springer, 2011, pp. 190–209.

[14] D. Bogdanov, S. Laur, J. Willemson. "Sharemind: A Framework for Fast Privacy-Preserving Computations". In: *ESORICS'08*. Springer, 2008, pp. 192–206.

[15] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, T. Toft. "Secure Multiparty Computation Goes Live". In: *FC'09*. Springer, 2009, pp. 325–343.

[16] M. Brandt, C. Orlandi, K. Shishak, H. Shulman. "Optimizing Transport Layer for Secure Computation". In: *IACR Cryptology ePrint Archive, Report 2019/836* (2019). https://ia.cr/2019/836.

[17] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. "HyCC: Compilation of Hybrid Protocols for Practical Secure Computation". In: *CCS'18*. ACM, 2018, pp. 847–861.

[18] N. Büscher, A. Holzer, A. Weber, S. Katzenbeisser. "Compiling Low Depth Circuits for Practical Secure Computation". In: *ESORICS'16*. Springer, 2016, pp. 80–98.

[19] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, S. Tripathi. "EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning". In: *EuroS&P'19*. IEEE, 2019, pp. 496–511.

[20] H. Chaudhari, A. Choudhury, A. Patra, A. Suresh. "ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction". In: *CCSW'19*. ACM, 2019, pp. 81–92.

[21] J. I. Choi, D. Tian, G. Hernandez, C. Patton, B. Mood, T. Shrimpton, K. R. B. Butler, P. Traynor. "A Hybrid Approach to Secure Function Evaluation Using SGX". In: *ASIACCS'19*. ACM, 2019, pp. 100–113.

[22] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, D. Rubenstein. "Secure Multi-Party Computation of Boolean Circuits with Applications to Privacy in On-Line Marketplaces". In: *CT-RSA'12*. Springer. 2012, pp. 416–432.

[23] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *CRYPTO'18*. Springer, 2018, pp. 769–798.

[24] I. Damgård, V. Pastro, N. P. Smart, S. Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO'12*. Springer, 2012, pp. 643–662.

[25] I. Damgård, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, N. Volgushev. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *S&P'19*. IEEE, 2019, pp. 1102–1120.

[26] I. Damgård, M. Geisler, M. Krøigaard, J. B. Nielsen. "Asynchronous Multiparty Computation: Theory and Implementation". In: *CRYPTO'09*. Code: http://viff.dk. Springer. 2009, pp. 160–179.

[27] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, N. P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: *ESORICS'13*. Springer, 2013, pp. 1–18.

[28] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni. "Automated Synthesis of Optimized Circuits for Secure Computation". In: *CCS'15*. ACM, 2015, pp. 1504–1517.

[29] D. Demmler, T. Schneider, M. Zohner. "ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation". In: *NDSS'15*. Internet Society, 2015.

[30] J. Doerner, D. Evans, A. Shelat. "Secure Stable Matching at Scale". In: *CCS'16*. ACM, 2016, pp. 1602–1613.

[31] T. K. Frederiksen, M. Keller, E. Orsini, P. Scholl. "A Unified Approach to MPC with Preprocessing Using OT". In: *ASIACRYPT'15*. Springer, 2015, pp. 711–735.

[32] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, J. Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy". In: *International Conference on Machine Learning (ICML'16)*. 2016, pp. 201–210.

[33] O. Goldreich, S. Micali, A. Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *STOC'87*. ACM, 1987, pp. 218–229.

[34] C. Guo, J. Katz, X. Wang, Y. Yu. "Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers". In: *S&P'20*. To appear. Online: https://ia.cr/2019/074. IEEE, 2020.

[35] S. Halevi. "Advanced Cryptography: Promise and Challenges." In: *CCS'18*. ACM, 2018, p. 647.

[36] M. Hastings, B. Hemenway, D. Noble, S. Zdancewic. "SoK: General Purpose Compilers for Secure Multi-Party Computation". In: *S&P'19*. IEEE, 2019, pp. 1220–1237.

[37] E. Hauck, J. Loss. "Efficient and Universally Composable Protocols for Oblivious Transfer from the CDH Assumption." In: *IACR Cryptology ePrint Archive, Report 2017/1011* (2017). https://ia.cr/2017/1011.

[38] C. Hazay, P. Scholl, E. Soria-Vazquez. "Low Cost Constant Round MPC Combining BMR and Oblivious Transfer". In: *ASIACRYPT'17*. Springer. 2017, pp. 598–628.

[39] K. He, L. Yang, J. Hong, J. Jiang, J. Wu, X. Dong, Z. Liang. "PrivC—A Framework for Efficient Secure Two-Party Computation". In: *Security and Privacy in Communication Networks*. Springer, 2019, pp. 394–407.

[40] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, I. Wehrenberg. "TASTY: Tool for Automating Secure Two-partY computations". In: *CCS'10*. ACM, 2010, pp. 451–462.

[41] Y. Huang, D. Evans, J. Katz. "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" In: *NDSS'12*. Internet Society, 2012.

[42] IETF. "QUIC: A UDP-Based Multiplexed and Secure Transport". *https://tools.ietf.org/html/draft-ietf-quic-transport-23*.

[43] IETF. "Reliable UDP (RUDP) Protocol". *https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00*.

[44] R. Impagliazzo, S. Rudich. "Limits on the Provable Consequences of One-Way Permutations". In: *STOC'89*. ACM. 1989, pp. 44–61.

[45] Y. Ishai, J. Kilian, K. Nissim, E. Petrank. "Extending Oblivious Transfers Efficiently". In: *CRYPTO'03*. Springer. 2003, pp. 145–161.

[46] M. Ishaq, A. L. Milanova, V. Zikas. "Efficient MPC via Program Analysis: A Framework for Efficient Optimal Mixing". In: *CCS'19*. ACM, 2019, pp. 1539–1556.

[47] S. Kamara, P. Mohassel, M. Raykova. "Outsourcing Multi-Party Computation". In: *IACR Cryptology ePrint Archive, Report 2011/272* (2011). https://ia.cr/2011/272.

[48] M. Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *CCS'20*. https://ia.cr/2020/521. ACM, 2020.

[49] M. Keller, E. Orsini, P. Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: *CCS'16*. ACM, 2016, pp. 830–842.

[50] V. Kolesnikov, A.-R. Sadeghi, T. Schneider. "Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima". In: *CANS'09*. Springer, 2009, pp. 1–20.

[51] B. Kreuter, B. Mood, A. Shelat, K. Butler. "PCF: A Portable Circuit Format for Scalable Two-party Secure Computation". In: *USENIX Security'12*. USENIX Association, 2013, pp. 321–336.

[52] B. Kreuter, A. Shelat, C.-H. Shen. "Billion-gate Secure Computation with Malicious Adversaries". In: *USENIX Security'12*. USENIX Association, 2012, pp. 285–300.

[53] C. Liu, X. S. Wang, K. Nayak, Y. Huang, E. Shi. "ObliVM: A Programming Framework for Secure Computation". In: *S&P'15*. IEEE, 2015, pp. 359–376.

[54] J. Liu, M. Juuti, Y. Lu, N. Asokan. "Oblivious Neural Network Predictions via MiniONN Transformations". In: *CCS'17*. ACM. 2017, pp. 619–631.

[55] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella. "Fairplay – A Secure Two-Party Computation System". In: *USENIX Security'04*. USENIX Association, 2004, pp. 287–302.

[56] A. Mittos, B. Malin, E. D. Cristofaro. "Systematizing Genome Privacy Research: A Privacy-Enhancing Technologies Perspective". In: *PETS* 1 (2019), pp. 87–107.

[57] P. Mohassel, Y. Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: *S&P'17*. IEEE, 2017, pp. 19–38.

[58] P. Mohassel, P. Rindal. "ABY^3: A Mixed Protocol Framework for Machine Learning". In: *CCS'18*. ACM, 2018, pp. 35–52.

[59] B. Mood, D. Gupta, H. Carter, K. Butler, P. Traynor. "Frigate: A Validated, Extensible, and Efficient Compiler and Interpreter for Secure Computation". In: *EuroS&P'16*. IEEE, 2016, pp. 112–127.

[60] J. B. Nielsen, T. Schneider, R. Trifiletti. "Constant Round Maliciously Secure 2PC with Function-Independent Preprocessing using LEGO". In: *NDSS'17*. 2017.

[61] A. Patra, T. Schneider, A. Suresh, H. Yalame. "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation". In: *USENIX Security'21*. To appear. Online: https://ia.cr/2020/1225. USENIX Association, 2021.

[62] A. Patra, A. Suresh. "BLAZE: Blazing Fast Privacy-Preserving Machine Learning". In: *NDSS'20*. Internet Society, 2020.

[63] M. O. Rabin. "How To Exchange Secrets with Oblivious Transfer". Tech. rep. Harvard Aiken Computation Laboratory, 1981.

[64] R. Rachuri, A. Suresh. "Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning". In: *NDSS'20*. Internet Society, 2020, pp. 23–26.

[65] A. Rastogi, M. A. Hammer, M. Hicks. "Wysteria: A Programming Language for Generic, Mixed-mode Multiparty Computations". In: *S&P'14*. IEEE, 2014, pp. 655–670.

[66] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, F. Koushanfar. "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications". In: *ASIACCS'17*. ACM, 2018, pp. 707–721.

[67] P. Rindal. "libOTe: an Efficient, Portable, and Easy to Use Oblivious Transfer Library". *https://github.com/osu-crypto/libOTe*.

[68] T. Schneider, M. Zohner. "GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits". In: *FC'13*. Springer, 2013, pp. 275–292.

[69] K. Shrishak, H. Shulman, M. Waidner. "Removing the Bottleneck for Practical 2PC (Poster)". In: *CCS'18*. ACM, 2018, pp. 2300–2302.

[70] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, F. Koushanfar. "TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits". In: *S&P'15*. IEEE, 2015, pp. 411–428.

[71] X. Wang. "A New Paradigm for Practical Maliciously Secure Multi-Party Computation". PhD thesis. PhD thesis. University of Maryland (College Park, Md.), 2018.

[72] X. Wang, A. J. Malozemoff, J. Katz. "EMP-toolkit: Efficient MultiParty computation toolkit". *https://github.com/emp-toolkit*. 2016.

[73] X. Wang, S. Ranellucci, J. Katz. "Global-Scale Secure Multiparty Computation". In: *CCS'17*. ACM, 2017, pp. 39–56.

[74] A. C.-C. Yao. "How to Generate and Exchange Secrets". In: *FOCS'86*. IEEE, 1986, pp. 162–167.

[75] S. Zahur, D. Evans. "Obliv-C: A Language for Extensible Data-Oblivious Computation". In: *IACR Cryptology ePrint Archive, Report 2015/1153* (2015). https://ia.cr/2015/1153.

[76] Y. Zhang, A. Steele, M. Blanton. "PICCO: A General-Purpose Compiler for Private Distributed Computation". In: *CCS'13*. ACM. 2013, pp. 813–826.

[77] W. Zheng, R. A. Popa, J. E. Gonzalez, I. Stoica. "Helen: Maliciously Secure Coopetitive Learning for Linear Models". In: *S&P'19*. IEEE, 2019, pp. 724–738.

A FRAMEWORK EXTENSIBILITY, AND RUN-TIME AND COMMUNICATION COSTS

Here, we show an advanced example of extensibility of the protocols implemented in MOTION in List. 2. Alternatively, this functionality can be implemented by inheriting from our `Gate` class to avoid the manual use of `Executor`.

As briefly summarized in §8.1, we ran extensive microbenchmarks in MOTION. We list the run-times for primitive operations, sharing, reconstruction and conversion in Tab. 7. The run-times for more complex building blocks, such as arithmetic operations, and comparisons are provided in Tab. 8.

Listing 2. Code excerpt in MOTION for *efficiently* multiplying an integer known by one party in the clear by a secret-shared bit computed somewhere in the circuit. The result of the multiplication is a secret-shared integer and can be used further in the circuit as input to the "standard" arithmetic GMW gates.

```
  using namespace MOTION;

/// \brief A secure two-party computation protocol for multiplying a
/// ring element known by one party in the clear by a bit secret-shared
/// between both parties, e.g., some intermediate result in Boolean GMW.
/// This protocol may be used, e.g., to aggregate statistics based on
/// threshold values and requires only one additively-correlated OT (ACOT).
///
/// @param s_bit secret-shared bit
/// @param val an unsigned integer number
/// \returns a secret-shared ring element in arith. GMW, i.e., s_bit * val

// knows share s_bit and integer val in the clear
template<typename T>
ArithmeticGMWWirePtr<T> MultiplyServer(BooleanGMWWirePtr s_bit, T val){
  auto& backend = s_bit->GetBackend();
  auto other_id = (backend.GetConfiguration().GetMyId() + 1) % 2;

  // register a sender OT object
  auto ot = backend.GetOTProvider(other_id).RegisterSendACOT<T>();

  // create an ArithmeticGMW wire for storing the result
  auto result = std::make_shared<ArithmeticGMWWire<T>>(backend);

  // submit the protocol as a task in form of an anonymous function
  // that will be executed in the online phase as a separate fiber
  backend.GetExecutor().EnqueueOnline(
  [s_bit, val, result, std::move(ot)]{
    ot->SetInput(val);
    ot->SendMessages();
    ot->ComputeOutput();
    // if s_bit is 1, we need to swap the output and s_bit might be a wire
    // deep in the circuit, so we need to wait until its value is computed
    s_bit->Wait();
    if(s_bit->As<bool>()) result->Set(ot->GetOutput(1));
    else result->Set(ot->GetOutput(0));
    // let other waiting routines know that this wire's value is now set
    result->SetFinished();
  });
  return result;
}

// knows only secret-shared bit s_bit
template<typename T>
ArithmeticGMWWirePtr<T> MultiplyClient(BooleanGMWWirePtr s_bit){
  auto& backend = s_bit->GetBackend();
  auto other_id = (backend.GetConfiguration().GetMyId() + 1) % 2;
  auto ot = backend.GetOTProvider(other_id).RegisterReceiveACOT<T>();
  auto result = std::make_shared<ArithmeticGMWWire<T>>(backend);

  backend.GetExecutor().EnqueueOnline(
  [s_bit, result, std::move(ot)]{
    s_bit->Wait();
    ot->SetCorrections(s_bit->As<bool>());
    ot->SendCorrections();
    ot->ComputeOutput();
    result->Set(ot->GetOutput());
    result->SetFinished();
  });
  return result;
}
```

Table 7. Run-times in nanoseconds in different test environments (cf. §8) for primitive operations for GMW (B) [33], arithmetic GMW (A), and the BMR protocol (Y) [8], and conversions between the protocols. For each entry, we specify the run-time of a single operation (amortized over one million operations for GMW and BMR operations, and conversions between them; over 100 thousand operations for arithmetic GMW operations, and over 1 thousand operations for the remaining conversions).

	LAN			WAN			AWS		
# Parties N	N=3	N=4	N=5	N=3	N=4	N=5	N=10	N=15	N=20
Rec_{8}^{A}	50.5	57.9	61.0	4 062.9	5 443.1	6 445.4	74.5	123.4	201.4
Rec_{16}^{A}	51.0	55.6	60.6	5 111.2	6 008.4	7 006.5	93.2	178.0	233.9
Rec_{32}^{A}	62.2	67.0	72.6	5 996.7	6 845.2	8 118.9	155.4	278.6	422.4
Rec_{64}^{A}	79.6	130.2	100.0	8 949.3	8 194.0	10 822.8	274.4	407.8	737.3
Rec^{B}	5.8	6.2	6.5	507.7	594.1	687.9	9.3	14.0	23.5
Rec^{Y}	7.9	7.9	8.6	508.0	596.6	687.1	9.9	15.9	20.1
Share_{8}^{A}	223.3	292.8	377.9	222.2	269.7	324.0	462.0	701.7	968.7
Share_{16}^{A}	180.3	240.7	302.6	203.8	240.6	279.5	467.6	730.4	980.8
Share_{32}^{A}	184.8	249.0	301.4	200.5	237.6	282.7	502.5	743.0	1 139.1
Share_{64}^{A}	161.0	219.1	272.6	184.2	222.1	255.1	524.5	568.9	1 239.6
Share^{B}	5.5	5.9	7.1	5.7	6.5	7.8	8.1	11.3	16.2
Share^{Y}	179.8	208.7	243.5	2 292.4	2 496.9	3 064.1	529.5	829.5	1 181.9
AND^{B}	545.5	747.8	829.0	3 116.9	3 725.3	4 640.8	1 729.1	2 777.0	4 743.0
AND^{Y}	5 022.4	7 690.1	10 809.1	18 218.3	21 852.6	27 557.8	50 727.0	111 913.3	237 933.2
XOR^{B}	3.6	3.6	3.3	3.9	4.1	4.1	3.9	4.2	9.3
XOR^{Y}	40.9	44.9	52.9	40.2	44.9	52.2	99.9	170.4	263.5
ADD_{8}^{A}	30.3	32.1	33.5	28.9	29.1	30.5	36.1	57.6	75.0
ADD_{16}^{A}	30.3	31.6	34.0	29.5	29.4	31.6	38.3	54.6	80.2
ADD_{32}^{A}	31.6	32.7	33.2	29.4	29.7	29.1	35.6	42.5	100.0
ADD_{64}^{A}	32.9	34.4	34.4	32.1	31.5	32.6	39.3	57.8	92.6
MUL_{8}^{A}	7 193.2	8 732.5	10 209.0	27 623.5	32 881.4	39 043.0	23 665.4	35 051.1	78 641.0
MUL_{16}^{A}	13 752.4	17 052.3	19 683.1	41 005.1	52 871.6	61 525.6	47 492.4	74 489.9	171 135.2
MUL_{32}^{A}	26 656.8	33 206.7	38 850.6	72 443.3	84 093.7	92 270.3	96 698.7	161 940.6	357 438.1
MUL_{64}^{A}	55 081.1	67 885.0	79 636.0	134 562.0	149 166.6	166 928.1	202 929.5	330 017.2	733 487.4
SQR_{8}^{A}	4 939.5	6 330.0	6 843.8	24 108.5	23 951.9	26 258.3	15 290.2	23 176.2	49 672.1
SQR_{16}^{A}	9 712.7	11 963.9	13 749.3	41 024.8	39 588.4	41 870.2	30 750.3	46 788.2	100 951.9
SQR_{32}^{A}	19 070.8	23 346.3	26 354.9	61 014.2	66 318.1	68 817.6	62 659.4	94 266.4	200 768.2
SQR_{64}^{A}	37 716.6	48 705.0	52 755.7	108 325.3	114 981.0	123 786.6	122 792.5	193 317.1	405 950.1
$A2B_8$	84 914.0	152 182.0	263 464.0	2 402 112.0	2 739 673.0	3 130 609.0	1 345 075.0	3 998 044.0	8 907 812.0
$A2B_{16}$	166 180.0	313 269.0	501 840.0	2 810 842.0	3 186 936.0	4 456 947.0	2 605 770.0	8 057 326.0	18 724 116.0
$A2B_{32}$	344 879.0	596 925.0	926 113.0	3 228 496.0	4 490 477.0	6 645 732.0	5 296 703.0	17 323 969.0	41 634 733.0
$A2B_{64}$	664 843.0	1 142 710.0	1 629 902.0	4 703 179.0	7 028 108.0	9 348 717.0	11 335 031.0	37 826 123.0	90 551 799.0
$A2Y_8$	85 514.0	260 872.0	260 897.0	2 399 031.0	2 671 459.0	3 170 296.0	1 356 501.0	3 966 832.0	9 148 202.0
$A2Y_{16}$	164 057.0	301 738.0	497 765.0	2 780 980.0	3 596 824.0	4 521 377.0	2 670 646.0	8 120 268.0	19 109 086.0
$A2Y_{32}$	330 767.0	584 865.0	894 217.0	3 207 824.0	4 453 959.0	6 369 496.0	5 347 239.0	17 262 721.0	41 456 717.0
$A2Y_{64}$	653 240.0	1 104 743.0	1 606 672.0	4 764 729.0	6 818 137.0	10 012 767.0	11 197 584.0	37 970 804.0	89 238 532.0
$B2A_8$	116 578.0	129 860.0	138 925.0	1 535 135.0	1 593 539.0	1 712 148.0	293 300.0	423 229.0	845 219.0
$B2A_{16}$	346 598.0	417 955.0	476 560.0	2 643 318.0	3 085 103.0	2 900 933.0	1 043 550.0	1 603 717.0	3 264 207.0
$B2A_{32}$	1 267 014.0	1 567 303.0	1 774 488.0	6 200 784.0	7 031 536.0	7 903 015.0	4 044 245.0	6 191 658.0	13 301 903.0
$B2A_{64}$	5 388 326.0	6 448 544.0	7 442 426.0	16 332 760.0	17 040 985.0	18 053 991.0	17 535 467.0	27 321 748.0	59 919 132.0
$B2Y_1$	149.3	194.5	210.1	1 909.6	1 956.3	2 258.3	498.7	795.9	1 111.7
$Y2A_8$	108 335.0	120 890.0	134 134.0	1 596 040.0	1 581 936.0	1 612 624.0	279 856.0	409 255.0	825 231.0
$Y2A_{16}$	330 551.0	405 120.0	458 771.0	2 689 310.0	3 009 141.0	3 318 748.0	1 057 533.0	1 557 023.0	3 243 724.0
$Y2A_{32}$	1 262 030.0	1 546 668.0	1 791 343.0	6 392 610.0	7 367 808.0	7 646 224.0	4 029 879.0	6 193 397.0	12 777 651.0
$Y2A_{64}$	5 375 035.0	6 356 610.0	7 399 503.0	16 266 624.0	17 033 616.0	18 124 902.0	17 699 223.0	27 253 770.0	58 012 547.0
$Y2B_1$	3.1	3.2	3.2	3.5	3.6	3.6	3.6	4.5	6.2

Table 8. Run-times in microseconds in different test environments (cf. §8) for a bit-string comparison (EQ) and integer operations for the GMW (B) [33] and BMR protocol (Y) [8]. For each entry, we specify the run-time of a single operation amortized over 1 000 SIMD values. We take the average over 100 protocol runs in the LAN and WAN environments and over 10 protocol runs in the AWS environment.

	LAN			WAN			AWS		
# Parties N	N=3	N=4	N=5	N=3	N=4	N=5	N=10	N=15	N=20
8-BIT INTEGERS									
EQ^B	97	114	128	995	1 086	1 106	219	240	421
EQ^Y	58	68	89	2 050	2 126	2 228	241	544	1 131
INT ADD^B	47	54	60	981	1 138	1 208	142	217	428
INT ADD^Y	52	66	93	1 461	2 051	2 274	257	525	1 113
INT DIV^B	552	1 006	1 316	11 637	12 229	12 872	2 121	2 791	3 274
INT DIV^Y	499	705	932	3 943	4 932	6 342	2 819	6 446	12 528
INT GT^B	81	94	109	1 273	1 298	1 361	190	226	411
INT GT^Y	59	69	91	1 921	2 011	2 236	272	589	1 166
INT MUL^B	100	121	131	1 899	2 218	1 966	259	379	683
INT MUL^Y	184	276	365	2 904	3 404	3 993	1 114	2 287	4 109
INT SUB^B	44	52	56	1 099	1 338	1 266	131	175	394
INT SUB^Y	53	61	84	2 039	2 125	2 213	243	496	1 080
16-BIT INTEGERS									
EQ^B	70	94	119	1 263	1 325	1 406	212	279	427
EQ^Y	97	123	172	2 178	2 389	2 557	490	1 076	2 265
INT ADD^B	75	88	97	1 447	1 605	1 789	194	294	486
INT ADD^Y	89	122	161	2 289	2 368	2 421	471	1 045	2 159
INT DIV^B	2 100	3 844	5 579	42 011	45 913	48 543	8 560	11 089	12 594
INT DIV^Y	1 659	2 302	3 007	8 678	11 092	14 292	9 795	22 840	47 875
INT GT^B	97	111	137	1 424	1 741	1 742	253	296	532
INT GT^Y	93	124	170	2 222	2 302	2 550	493	1 110	2 295
INT MUL^B	221	261	289	2 859	2 922	3 279	612	914	1 568
INT MUL^Y	650	868	1 090	4 961	6 472	7 966	4 023	8 595	15 629
INT SUB^B	82	88	92	1 284	1 460	2 050	187	290	511
INT SUB^Y	90	121	163	2 299	2 357	2 478	455	1 050	2 153
32-BIT INTEGERS									
EQ^B	103	136	164	1 540	1 640	2 113	312	551	529
EQ^Y	175	249	337	2 587	2 778	3 081	1 001	2 120	4 829
INT ADD^B	133	156	175	2 038	2 476	2 385	356	489	796
INT ADD^Y	163	245	340	2 469	2 761	3 070	947	2 196	4 410
INT DIV^B	5 596	9 671	13 044	100 048	108 743	113 700	20 950	26 834	32 313
INT DIV^Y	4 249	5 833	7 521	22 099	26 301	33 792	27 492	64 913	126 849
INT GT^B	120	151	167	1 956	2 343	2 325	316	426	633
INT GT^Y	171	263	372	2 646	2 862	3 219	993	2 252	4 445
INT MUL^B	703	826	911	4 001	5 280	6 574	1 953	2 902	5 609
INT MUL^Y	2 231	2 967	3 798	14 715	18 504	24 671	16 318	36 393	64 952
INT SUB^B	136	155	177	2 012	2 244	2 400	364	523	795
INT SUB^Y	160	236	337	2 553	2 701	3 002	981	2 109	4 277
64-BIT INTEGERS									
EQ^B	153	177	210	2 131	2 314	2 506	578	1 121	985
EQ^Y	336	475	640	3 121	3 399	4 022	2 017	4 593	12 323
INT ADD^B	269	305	353	2 209	2 563	2 613	699	998	1 965
INT ADD^Y	337	521	702	3 045	3 374	3 796	1 859	4 225	8 552
INT DIV^B	21 256	36 856	49 973	364 851	388 783	404 573	81 058	107 427	123 363
INT DIV^Y	14 806	20 042	26 360	66 339	83 557	115 663	100 945	238 141	461 013
INT GT^B	178	213	262	2 500	2 571	2 897	471	652	1 020
INT GT^Y	334	506	695	3 026	3 302	4 011	1 962	4 366	8 882
INT MUL^B	2 434	2 814	3 374	11 993	12 635	12 999	8 169	15 590	26 310
INT MUL^Y	8 588	11 618	14 485	53 813	67 671	92 458	64 278	144 322	252 703
INT SUB^B	272	309	347	2 213	2 303	2 662	725	1 004	2 029
INT SUB^Y	326	516	705	2 734	3 464	3 890	1 919	4 201	8 565

B Efficient Circuit-based PSI with Linear Communication (EUROCRYPT'19)

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Online: https://ia.cr/2019/241. Code: https://encrypto.de/code/OPPRF-PSI. Springer, 2019, pp. 122–153. CORE Rank A*. Appendix B.

https://doi.org/10.1007/978-3-030-17659-4_5

Efficient Circuit-Based PSI with Linear Communication

Benny Pinkas[1( )], Thomas Schneider[2], Oleksandr Tkachenko[2], and Avishay Yanai[1]

[1] Bar-Ilan University, Ramat Gan, Israel
benny@pinkas.net, ay.yanay@gmail.com
[2] TU Darmstadt, Darmstadt, Germany
{schneider,tkachenko}@encrypto.cs.tu-darmstadt.de

Abstract. We present a new protocol for computing a circuit which implements the private set intersection functionality (PSI). Using circuits for this task is advantageous over the usage of specific protocols for PSI, since many applications of PSI do not need to compute the intersection itself but rather functions based on the items in the intersection.

Our protocol is the *first circuit-based PSI protocol to achieve linear communication complexity*. It is also concretely more efficient than all previous circuit-based PSI protocols. For example, for sets of size 2^{20} it improves the communication of the recent work of Pinkas et al. (EUROCRYPT'18) by more than 10 times, and improves the run time by a factor of 2.8x in the LAN setting, and by a factor of 5.8x in the WAN setting.

Our protocol is based on the usage of a protocol for computing oblivious programmable pseudo-random functions (OPPRF), and more specifically on our technique to amortize the cost of batching together multiple invocations of OPPRF.

Keywords: Private Set Intersection · Secure computation

1 Introduction

The functionality of Private Set Intersection (PSI) enables two parties, P_1 and P_2, with respective input sets X and Y to compute the intersection $X \cap Y$ without revealing any information about the items which are not in the intersection. There exist multiple constructions of secure protocols for computing PSI, which can be split into two categories: (i) constructions that output the intersection itself and (ii) constructions that output the result of a function f computed on the intersection. In this work, we concentrate on the second type of constructions (see Sect. 1.2 for motivation). These constructions keep the intersection $X \cap Y$ secret from both parties and allow the function f to be securely computed on top of it, namely, yielding only $f(X \cap Y)$. Formally, denote by $\mathcal{F}_{\mathsf{PSI},f}$ the functionality $(X, Y) \mapsto (f(X \cap Y), f(X \cap Y))$.

Y. Ishai and V. Rijmen (Eds.): EUROCRYPT 2019, LNCS 11478, pp. 122–153, 2019.
https://doi.org/10.1007/978-3-030-17659-4_5

A functionality for computing $f(X \cap Y)$ can be naively implemented using generic MPC protocols by expressing the functionality as a circuit. However, naive protocols for computing $f(X \cap Y)$ have high communication complexity, which is of paramount importance for real-world applications. The difficulty in designing a circuit for computing the intersection is in deciding which pairs of items of the two parties need to be compared. We refer here to the number of comparisons computed by the circuit as the major indicator of the overhead, since it directly affects the amount of communication in the protocol (which is proportional to the number of comparisons, times the length of the representation of the items, times the security parameter). Since the latter factors (input length and security parameter) are typically given, and since the circuit computation mostly involves symmetric key operations, the goal is to minimize the communication overhead as a function of the input size. We typically state this goal as minimizing the number of comparisons computed in the circuit. The protocol presented in this paper is the first to achieve linear communication overhead, which is optimal.

Suppose that each party has an input set of n items. A naive circuit for this task compares all pairs and computes $O(n^2)$ comparisons. More efficient circuits are possible, assuming that the parties first order their respective inputs in specific ways. For example, if each party has sorted its input set then the intersection can be computed using a circuit which first computes, using a merge-sort network, a sorted list of the union of the two sets, and then compares adjacent items [HEK12]. This circuit computes only $O(n \log n)$ comparisons. The protocol of [PSSZ15] (denoted "Circuit-Phasing") has P_1 map its items to a table using Cuckoo hashing, and P_2 maps its items using simple hashing. The intersection is computed on top of these tables by a circuit with $O(n \log n / \log \log n)$ comparisons. This protocol is the starting point of our work.

A recent circuit-based PSI construction [PSWW18] is based on a new hashing algorithm, denoted "two-dimensional Cuckoo hashing", which uses a table of size $O(n)$ and a stash of size $\omega(1)$. Each party inserts its inputs to a separate table, and the hashing scheme assures that each value in the intersection is mapped by both parties to exactly one mutual bin. Hence, a circuit which compares the items that the two parties mapped to each bin, and also compares all stash items to all items of the other party, computes the intersection in only $\omega(n)$ comparisons (namely, the overhead is slightly more than linear, although it can be made arbitrarily close to being linear).

Our work is based on the usage of an oblivious programmable pseudo-random function (OPPRF), which is a new primitive that was introduced in [KMP+17]. An OPRF—oblivious pseudo-random function (note, this is different than an OPPRF)—is a two-party protocol where one party has a key to a PRF F and the other party can privately query F at specific locations. An OPPRF is an extension of the protocol which lets the key owner "program" F so that it has specific outputs for some specific input values (and is pseudo-random on all other values). The other party which evaluates the OPPRF does not learn whether it learns a "programmed" output of F or just a pseudo-random value.

1.1 Overview of Our Protocol

The starting point for our protocols is the Circuit-Phasing PSI protocol of [PSSZ15], in which $O(n)$ bins are considered and the circuit computes $O(n \log n / \log\log n)$ comparisons. Party P_1 uses Cuckoo hashing to map at most one item to each bin, whereas party P_2 maps its items to the bins using simple hashing (two times, once with each of the two functions used in the Cuckoo hashing of the first party). Thus, P_2 maps up to $S = O(\log n / \log\log n)$ items to each bin. Since the parties have to hide the number of items that are mapped to each bin, they pad the bins with "dummy" items to the maximum bin size. That is, P_1 pads all bins so they all contain exactly one item and P_2 pads all bins so they all contain S items.

Both parties use the same hash functions, and therefore for each input element x that is owned by both parties there is exactly one bin to which x is mapped by both parties. Thus, it is only needed to check whether the item that P_1 places in a bin is among the items that are placed in this bin by P_2. This is essentially a private set membership (PSM) problem: As input, P_1 has a single item x and P_2 has a set Σ with $|\Sigma|$ items, where $S = |\Sigma|$. As for the output, if $x \in \Sigma$ then both parties learn the same random output, otherwise they learn independent random outputs. These outputs can then be fed to a circuit, which computes the intersection. The Circuit-Phasing protocol [PSSZ15] essentially computes the PSM functionality using a sub-circuit of the overall circuit that it computes. Namely, let $S = O(\log n / \log\log n)$ be an upper bound on the number of items mapped by P_2 to a single bin. For each bin the sub-circuit receives one input from P_1 and S inputs from P_2, computes S comparisons, and feeds the result to the main part of the circuit which computes the intersection itself (and possibly some function on top of the intersection). Therefore the communication overhead is $O(nS) = O(n \log n / \log\log n)$. A very recent work in [CO18] uses the same hashing method and computes the PSM using a specific protocol whose output is fed to the circuit. The circuit there computes only $\omega(n)$ comparisons but the PSM protocol itself incurs a communication overhead of $O(\log n / \log\log n)$ and is run $O(n)$ times. Therefore, the communication overhead of [CO18] is also $O(n \log n / \log\log n)$.

We diverge from the protocol of [PSSZ15] in the method for comparing the items mapped to each bin. In our protocol, the parties run an oblivious programmable PRF (OPPRF) protocol for each bin i, such that party P_2 chooses the PRF key and the programmed values, and the first party learns the output. The function is "programmed" to give the same output β_i for each of the $O(\log n / \log\log n)$ items that P_2 mapped to this bin. Therefore, if there is any match in this bin then P_1 learns the same value β_i. Then, the parties evaluate a circuit, where for each bin i party P_1 inputs its output in the corresponding OPPRF protocol, and P_2 inputs β_i. This circuit therefore needs to compute only a *single* comparison per bin.

The communication overhead of an OPPRF is linear in the number of programmed values. Thus, a stand alone invocation of an OPPRF for every bin incurs an overall overhead of $O(n \log n / \log\log n)$. We achieve linear overhead

for comparing the items in all bins, by observing that although each bin is of maximal size $O(\log n / \log \log n)$ (and therefore naively requires to program this number of values in the OPPRF), the total number of items that need to be programmed in all bins is $O(n)$. We can amortize communication so that the total communication of computing all $O(n)$ OPPRFs is the same as the total number of items, which is $O(n)$.

In addition to comparing the items that are mapped to the hash tables, the protocol must also compare items that are mapped to the stash of the Cuckoo hashing scheme. Fixing a stash size $s = O(1)$, the probability that the stash does not overflow is $O(n^{-(s+1)})$ [KMW09]. It was shown in [GM11] that a stash of size $O(\log n)$ ensures a negligible failure probability (namely, a probability that is asymptotically smaller than any polynomial function). Each item that P_1 places in the stash must be compared to all items of P_2, and therefore a straightforward implementation of this step requires the circuit to compute $\omega(n)$ comparisons. However, we show an advanced variant of our protocol that computes all comparisons (including elements in the stash) with only $O(n)$ comparisons.

In addition to designing a generic $O(n)$ circuit-based PSI protocol, we also investigate an important and commonly used variant of the problem where each item is associated with some value ("payload"), and it is required to compute a function of the payloads of the items in the intersection. (For example, compute the sum of financial transactions associated with these items). The challenge is that each of the S items that the second party maps to a bin has a different payload and therefore it is hard to represent them using a single value. (The work in [PSSZ15, CO18], for example, did not consider payloads). We describe a variant of our PSI protocol which injects the correct payloads to the circuit while keeping the $O(n)$ overhead.

Overall, the work in this paper improves the state of the art in two dimensions:

- With regards to **asymptotic** performance, we show a protocol for circuit-based PSI which has only $O(n)$ communication. This cost is asymptotically smaller than that of all known circuit-based constructions of PSI, and matches the obvious lower bound on the number of comparisons that must be computed.
- With regards to **concrete** overhead, our most efficient protocols improve communication by a factor of 2.6x to 12.8x, and run faster by factor 2.8x to 5.8x compared to the previous best circuit-based PSI protocol of [PSWW18]. We demonstrate this both analytically and experimentally.

1.2 Motivation for Circuit-Based PSI

Most research on computing PSI focused on computing the intersection itself (see Sect. 1.4). On the other hand, many **applications** of PSI are based on computing arbitrary functions of the intersection. For example, Google reported a PSI-based application for measuring the revenues from *online* ad viewers who later perform a related *offline* transaction (namely, ad conversion

rates) [Yun15,Kre17]. This computation compares the set of people who were shown an ad with the set of people who have completed a transaction. These sets are held by the advertiser, and by merchants, respectively. A typical use case is where the merchant inputs pairs of the customer-identity and the value of the transactions made by this customer, and the computation calculates the total revenue from customers who have seen an ad, namely customers in the intersection of the sets known to the advertiser and the merchant. Google reported implementing this computation using a Diffie-Hellman-based PSI cardinality protocol (for computing the cardinality of the intersection) and Paillier encryption (for computing the total revenues) [IKN+17,Kre18]. In fact, it was recently reported that Google is using such a "double-blind encryption" protocol in a beta version of their ads tool.[1] However, their protocol reveals the size of the intersection, and has substantially higher runtimes than our protocol as it uses public key operations, rather than efficient symmetric cryptographic operations (cf. Sect. 7.4).

Another motivation for running circuit-based PSI is **adaptability.** A protocol that is specific for computing the intersection, or a specific function such as the cardinality of the intersection, cannot be easily changed to compute another function of the intersection (say, the cardinality plus some noise to preserve differential privacy). Any change to a specialized protocol will require considerable cryptographic know-how, and might not even be possible. On the other hand, the task of writing a new circuit component which computes a different function of the intersection is rather trivial.

Circuit-based protocols also benefit from the **existing code base** for generic secure computation. Users only need to design the circuit to be computed, and can use available libraries of optimized code for secure computation, such as [HEKM11,EFLL12,DSZ15,LWN+15].

1.3 Computing Symmetric Functions

We focus in this work on constructing a circuit which computes the intersection. On top of that circuit it is possible to compose a circuit for computing any function that is based on the intersection. In order to preserve privacy, that function must be a symmetric function of the items in the intersection. Namely, the output of the function must not depend on the *order* of its inputs.

If the function that needs to be computed is non-symmetric, then the circuit for computing the intersection must shuffle its output, in order to place each item of the intersection in a location which is independent of the other values. The result is used as the input to the function. The size of this "shuffle" step is $O(n \log n)$, as is described in [HEK12], and it dominates the $O(n)$ size of the intersection circuit. We therefore focus on the symmetric case.[2]

[1] https://www.bloomberg.com/news/articles/2018-08-30/google-and-mastercard-cut-a-secret-ad-deal-to-track-retail-sales.

[2] Note that outputting the intersection is a *non-symmetric* function. Therefore in that case the order of the elements must be shuffled. However, it is unclear why a circuit-based protocol should be used for computing the intersection, since there are specialized protocols for this which are much more efficient, e.g. [KKRT16,PSZ18].

Most interesting functions of the intersection (except for the intersection itself) are symmetric. Examples of symmetric functions include:

- The size of the intersection, i.e., PSI cardinality (PSI-CA).
- A threshold function that is based on the size of the intersection. For example identifying whether the size of the intersection is greater than some threshold (PSI-CAT). An extension of PSI-CAT, where the intersection is revealed only if the size of the intersection is greater than a threshold, can be used for privacy-preserving ridesharing [HOS17]. Other public-key based protocols for this functionality appear in [ZC17, ZC18].
- A differentially private [Dwo06] value of the size of the intersection, which is computed by adding some noise to the exact count.
- The sum of values associated with the items in the intersection. This is used for measuring ad-generated revenues (cf. Sect. 1.2).

The circuits for computing all these functions are of size $O(n)$. Therefore, with our new construction the total size of the circuits for applying these functions to the intersection is $O(n)$.

1.4 Related Work

We classify previous works into dedicated protocols for *PSI*, generic protocols for *circuit-based PSI*, and dedicated protocols for *PSI cardinality*.

PSI. The first PSI protocols were based on public-key cryptography, e.g., on the Diffie Hellman function (e.g. [Mea86], with an earlier mention in [Sha80]), oblivious polynomial evaluation [FNP04], or blind RSA [DT10]. More recent protocols are based on oblivious transfer (OT) which can be efficiently instantiated using symmetric key cryptography [IKNP03, ALSZ13]: these protocols use either Bloom filters [FNP04] or hashing to bins [PSZ14, PSSZ15, KKRT16, PSZ18]. All these PSI protocols have super-linear complexity and many of them were compared experimentally in [PSZ18]. PSI protocols have also been evaluated on mobile devices, e.g., in [HCE11, ADN+13, CADT14, KLS+17]. PSI protocols with input sets of different sizes were studied in [KLS+17, PSZ18, RA18].

Circuit-based PSI. These protocols use secure evaluation of circuits for PSI. A trivial circuit for PSI computes $O(n^2)$ comparisons which result in $O(\sigma n^2)$ gates, where σ is the bit-length of the elements.

The sort-compare-shuffle (SCS) PSI circuit of [HEK12] computes $O(n \log n)$ comparisons and is of size $O(\sigma n \log n)$ gates (even without the final shuffle layer).

The Circuit-Phasing PSI circuit of [PSSZ15] uses Cuckoo hashing to $O(n)$ bins by one party and simple hashing by the other party which maps at most $O(\log n / \log \log n)$ elements per bin. Therefore, the Circuit-Phasing circuit has a size of $O(\sigma n \log n / \log \log n)$ gates.

The recent circuit-based PSI protocol of [CO18] applies a protocol based on OT extension to compute private set membership in each bin. The outputs of the invocations of this protocol are input to a comparison circuit. The circuit

itself computes a linear number of comparisons, but the total communication complexity of the private set membership protocols is of the same order as that of the Circuit-Phasing circuit [PSSZ15] with $O(\sigma n \log n / \log\log n)$ gates.

Another recent circuit-based PSI protocol of [FNO18, Sect. 8] has communication complexity $O(\sigma n \log\log n)$. It uses hashing to $O(n)$ bins where each bin has multiple buckets and then runs the SCS circuit of [HEK12] to compute the intersection of the elements in the respective bins.

The two-dimensional Cuckoo hashing circuit of [PSWW18] uses a new variant of Cuckoo hashing in two dimensions and has an almost linear complexity of $\omega(\sigma n)$ gates.

In this work, we present the first circuit-based PSI protocol with a true linear complexity of $O(\sigma n)$ gates.

PSI Cardinality. Several protocols for securely computing the cardinality of the intersection, i.e., $|X \cap Y|$, were proposed in the literature. These protocols have linear complexity and are based on public-key cryptography, namely Diffie-Hellman [DGT12], the Goldwasser-Micali cryptosystem [DD15], or additively homomorphic encryption [DC17]. However, these protocols reveal the cardinality of the intersection to one of the parties. In contrast, circuit-based PSI protocols can easily be adapted to efficiently compute the cardinality and even functions of it using mostly symmetric cryptography.

1.5 Our Contributions

In summary, in this paper we present the following contributions:

- The first circuit-based PSI protocol with linear asymptotic communication overhead. We remark that achieving a linear overhead is technically hard since hashing to a table of linear size requires a stash of super-linear size in order to guarantee a negligible failure probability. It is hard to achieve linear overhead with objects of super-linear size.
- A circuit-based PSI protocol with small constants and an improved concrete overhead over the state of the art. As a special case, we consider a very common variant of PSI, namely threshold PSI, in which the intersection is revealed only if it is bigger/smaller than some threshold. Surprisingly, our protocol is 1–2 orders of magnitude more efficient than the state-of-the-art [ZC18] and has the same asymptotic communication complexity of $O(n)$, despite the fact that the protocol in [ZC18] is a special purpose protocol for threshold-PSI.
- Our protocol supports associating data ("payload") with each input (from both parties), and compute a function that depends on the data associated with the items in the intersection. This property was not supported by the Phasing circuit-based protocol in [PSSZ15]. It is important for applications that compute some function of data associated with the items in the intersection, e.g., aggregate revenues from common users (cf. Sect. 1.2).

– On a technical level, we present a new paradigm for handling $\omega(1)$ stash sizes and obtaining an overall overhead that is linear. This is achieved by running an extremely simple dual-execution variant of the protocol.
– Finally, with regards to concrete efficiency, we introduce a circuit-based PSI protocol with linear complexity. This is achieved by using Cuckoo hashing with $K = 3$ instead of $K = 2$ hash functions, and no stash. This protocol substantially reduces communication (by a factor of 2.6x to 12.8x) and runtime (by a factor of 2.8x to 5.8x) compared to the best previous circuit-based PSI protocol of [PSWW18].

2 Preliminaries

2.1 Setting

There are two parties, which we denote as P_1 (the "receiver") and P_2 (the "sender"). They have input sets, X and Y, respectively, each of which contains n items of bitlength λ. We assume that both parties agree on a function f and wish to securely compute $f(X \cap Y)$. They also agree on a circuit C that receives the items in the intersection as input and computes f. That is, C has $O(n\lambda)$ input wires if we consider a computation on the elements themselves or $O(n(\lambda + \rho))$ if we consider a computation on the elements and their associated payloads where the associated payload of each item has bitlength ρ. We denote the computational and statistical security parameters by κ and σ, respectively. Denote the set $1, \ldots, c$ by $[c]$. We use the notation $X(i)$ to denote the i-th element in the set X.

2.2 Security Model

This work, similar to most protocols for private set intersection, operates in the semi-honest model, where adversaries may try to learn as much information as possible from a given protocol execution but are not able to deviate from the protocol steps. This is in contrast to malicious adversaries which are able to deviate arbitrarily from the protocol. PSI protocols for the malicious setting exist, but they are less efficient than protocols for the semi-honest setting, e.g., [FNP04, DSMRY09, HN10, DKT10, FHNP16, RR17a, RR17b]. The only circuit-based PSI protocol that can be easily secured against malicious adversaries is the Sort-Compare-Shuffle protocol of [HEK12]: here a circuit of size $O(n)$ can be used to check that the inputs are sorted, resulting in an overall complexity of $O(n \log n)$. For the recent circuit-based PSI protocols that rely on Cuckoo hashing, ensuring that the hashing was done correctly remains the challenge.The semi-honest adversary model is appropriate for scenarios where execution of the intended software is guaranteed via software attestation or business restrictions, and yet an untrusted third party is able to obtain the transcript of the protocol after its execution, by stealing it or by legally enforcing its disclosure.

FUNCTIONALITY 1 (Two-Party Computation)

Parameters. The Boolean circuit C to be computed, with I_1, I_2 inputs and O_1, O_2 outputs associated with P_1 and P_2 resp.
Inputs. P_1 inputs bits $x_1, \ldots, x_{I_1}$ and P_2 inputs bits $y_1, \ldots, y_{I_2}$.
Outputs. The functionality computes the circuit C on the parties' inputs and returns the outputs to the parties.

2.3 Secure Two-Party Computation

There are two main approaches for generic secure two-party computation of Boolean circuits with security against semi-honest adversaries: (1) Yao's garbled circuit protocol [Yao86] has a constant round complexity and with today's most efficient optimizations provides free XOR gates [KS08], whereas securely evaluating an AND gate requires sending two ciphertexts [ZRE15]. (2) The GMW protocol [GMW87] also provides free XOR gates and also sends two ciphertexts per AND gate using OT extension [ALSZ13].

The main advantage of the GMW protocol is that *all* symmetric cryptographic operations can be pre-computed in a constant number of rounds in a setup phase, whereas the online phase is very efficient, but requires interaction for each layer of AND gates. In more detail, the setup phase is independent of the actual inputs and precomputes multiplication triples for each AND gate using OT extension in a constant number of rounds. The online phase begins when the inputs are provided and involves a communication round for each layer of AND gates. See [SZ13] for a detailed description and comparison between Yao and GMW.

In our protocol we make use of Functionality 1.

2.4 Cuckoo Hashing

Cuckoo hashing [PR01] uses two hash functions h_0, h_1 to map n elements to two tables T_0, T_1 which each contain $(1+\varepsilon)n$ bins. (It is also possible to use a single table T with $2(1+\varepsilon)n$ bins. The two versions are essentially equivalent). Each bin accommodates at most a single element. The scheme avoids collisions by relocating elements when a collision is found using the following procedure: Let $b \in \{0,1\}$. An element x is inserted into a bin $h_b(x)$ in table T_b. If a prior item y exists in that bin, it is evicted to bin $h_{1-b}(y)$ in T_{1-b}. The pointer b is then assigned the value $1-b$. The procedure is repeated until no more evictions are necessary, or until a threshold number of relocations has been reached. In the latter case, the last element is put in a special stash. It was shown in [KMW09] that for a stash of constant size s the probability that the stash overflows is at most $O(n^{-(s+1)})$. It was also shown in [GM11] that this failure probablity is negligilble when the stash is of size $O(\log n)$. An observation in [KM18] shows that this is also the case when $s = O(\omega(1) \cdot \frac{\log n}{\log \log n})$. After insertion, each item can be found in one of two locations or in the stash.

2.5 PSI Based on Hashing

Existing constructions for circuit-based PSI require the parties to reorder their inputs before inputting them to the circuit. In the sorting network-based circuit of [HEK12], the parties sort their inputs. In the hashing-based construction of [PSSZ15], the parties map their items to bins using a hashing scheme.

It was observed as early as [FNP04] that if the two parties agree on the same hash function and use it to assign their respective input to bins, then the items that one party maps to a specific bin only need to be compared to the items that the other party maps to the same bin. However, the parties must be careful not to reveal to each other the number of items that they mapped to each bin, since this leaks information about their input sets. Therefore, the parties agree beforehand on an upper bound m for the maximum number of items that can be mapped to a bin (such upper bounds are well known for common hashing algorithms, and can also be substantiated using simulation), and pad each bin with random dummy values until it has exactly m items in it. If both parties use the same hash algorithm, then this approach considerably reduces the overhead of the computation from $O(n^2)$ to $O(\beta \cdot m^2)$ where β is the number of bins.

When using a random hash function h to map n items to n bins such that x is mapped to bin $h(x)$, the most occupied bin has at most $m = \frac{\ln n}{\ln \ln n}(1 + o(1))$ items with high probability [Gon81]. For instance, for $n = 2^{20}$ and a desired error probability of 2^{-40}, a careful analysis shows that $m = 20$. Cuckoo hashing is much more promising, since it maps n items to $2n(1+\varepsilon)$ bins, where each bin stores at most $m = 1$ items.

It is tempting to let both parties, P_1 and P_2, map their items to bins using Cuckoo hashing, and then only compare the item that P_1 maps to a bin with the item that P_2 maps to the same bin. The problem is that P_1 might map x to $h_0(x)$ whereas P_2 might map it to $h_1(x)$. Unfortunately, they cannot use a protocol where P_1's value in bin $h_0(x)$ is compared to the two bins $h_0(x), h_1(x)$ in P_2's input, since this reveals that P_1 has an item which is mapped to these two locations. The solution used in [FHNP16, PSZ14, PSSZ15] is to let P_1 map its items to bins using Cuckoo hashing, and P_2 map its items using simple hashing. Namely, each item of P_2 is mapped to both bins $h_0(x), h_1(x)$. Therefore, P_2 needs to pad its bins to have exactly $m = O(\log n / \log \log n)$ items in each bin, and the total number of comparisons is $O(n \log n / \log \log n)$.

3 OPPRF – Oblivious Programmable PRF

Our protocol builds on a (batched) oblivious programmable pseudorandom function (OPPRF). In this section we gradually present the properties required by that kind of a primitive, by first describing simpler primitives, namely, Programmable PRF (and its batched version) and Oblivious PRF.

3.1 Oblivious PRF

An oblivious PRF (OPRF) [FIPR05] is a two-party protocol implementing a functionality between a sender and a receiver. Let F be a pseudo-random

function (PRF) such that $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$. The sender inputs a key k to F and the receiver inputs $q_1 \ldots, q_c$. The functionality outputs $F(k, q_1), \ldots, F(k, q_c)$ to the receiver and nothing to the sender. In another variant of oblivious PRF the sender is given a fresh random key k as an output from the functionality rather than choosing it on its own. In our protocol we will make use of a "one-time" OPRF functionality in which the receiver can query *a single query*, namely, the sender inputs nothing and the receiver inputs a query q; the functionality outputs to the sender a key k and to the receiver the result $F_k(q)$. Let us denote that functionality by $\mathcal{F}_{\mathsf{OPRF}}$.

3.2 (One-Time) Programmable PRF (PPRF)

A programmable PRF (PPRF) is similar to a PRF, with the additional property that on a certain "programmed" set of inputs the function outputs "programmed" values. Namely, for an arbitrary set X and a "target" multi-set T, where $|X| = |T|$ and each $t \in T$ is uniformly distributed[3], it is guaranteed that on input $X(i)$ the function outputs $T(i)$. Let $\mathcal{T}$ be a distribution of such multi-sets, which may be public to both parties.

The restriction of the PPRF to be only one-time comes from the fact that we allow the elements in T to be correlated. If the elements are indeed correlated then by querying it two times (on the correlated positions) it would be easy to distinguish it from a random function.

We capture the above notion by the following formal definition:

Definition 1. *An ℓ-bits PPRF is a pair of algorithms $\hat{F} = (\mathsf{Hint}, F)$ as follows:*

- $\mathsf{Hint}(k, X, T) \to \mathsf{hint}_{k,X,T}$*: Given a uniformly random key $k \in \{0,1\}^\kappa$, the set X where $|X(i)| = \ell$ for all $i \in [|X|]$ and a target multi-set T with $|T| = |X|$ and all elements in T are uniformly distributed (but may be correlated), output the hint $\mathsf{hint}_{k,X,T} \in \{0,1\}^{\kappa \cdot |X|}$.*
- $F(k, \mathsf{hint}, x) \to y^\star$*. Given a key $k \in \{0,1\}^\kappa$, a hint $\mathsf{hint} \in \{0,1\}^{\kappa \cdot |X|}$ and an input $x \in \{0,1\}^\ell$, output $y^\star \in \{0,1\}^\ell$.*

We consider two properties of a PPRF, correctness and security:

- **Correctness.** *For every k, T and X, and for every $i \in [|X|]$ we have:*

$$F\left(k, \mathsf{hint}, X(i)\right) = T(i).$$

- **Security.** *We say that an interactive machine M is a* PPRF oracle over $\hat{F}$ *if, when interacting with a "caller" $\mathcal{A}$, it works as follows:*
  1. *M is given a set X from $\mathcal{A}$.*
  2. *M samples a uniformly random $k \in \{0,1\}^\kappa$ and T from $\mathcal{T}$, invokes $\mathsf{hint} \leftarrow \mathsf{Hint}(k, X, T)$ and hands hint to $\mathcal{A}$.*

[3] We require that each element in T is uniformly random but the elements may be correlated.

CONSTRUCTION 2 (PPRF)

Let $F' : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a PRF.

- $\mathsf{Hint}(k, X, T)$. Interpolate a polynomial p over the points $\{(X(i)\ ,\ F'_k(X(i)) \oplus T(i))\}_{i\in[|X|]}$. Return p as the hint.
- $F(k, \mathsf{hint}, x)$. Interpret hint as a polynomial, denoted p. Return $F'_k(x) \oplus p(x)$.

3. M is given an input $x \in \{0,1\}^\ell$ from $\mathcal{A}$ and responds with $F(k, \mathsf{hint}, x)$.
4. M halts (any subsequent queries will be ignored).

The scheme $\hat{F}$ is said to be secure *if, for every X input by $\mathcal{A}$ (i.e. the caller), the interaction of $\mathcal{A}$ with M is computationally indistinguishable from the interaction with the* PPRF oracle *$\mathcal{S}$, where $\mathcal{S}$ outputs a uniformly random "hint" $\{0,1\}^{\kappa\cdot|X|}$ and a "PRF result" from $\{0,1\}^\ell$.*

The definition is reminiscent of a semantically secure encryption scheme. Informally, semantic security means that whatever is efficiently computable about the cleartext given the ciphertext, is also efficiently computable without the ciphertext. Also here, whatever can be efficiently computable given X is also efficiently computable given only $|X|$. That implicitely means that the interaction with a PPRF oracle M over (KeyGen, F) does not leak the elements in X.

Our security definition diverges from that of [KMP+17] in two aspects:

1. In [KMP+17], $\mathcal{A}$ has many queries to M in Step 3 of the interaction, whereas our definition allows only a single query. In the (n,t)-security definition in [KMP+17] this corresponds to setting $t = 1$. Our definition is weaker in this sense, but this is sufficient for our protocol as we invoke multiple instances of the one-time PPRF.
2. The definition in [KMP+17] compensates for the fact that $\mathcal{A}$ has many queries, by requiring that the function F outputs an *independent* target value for every $x \in X$. Our definition is stronger as it allows having correlated target elements in T. In the most extreme form of correlation all values in T are equal, which makes the task of the adversary "easier". We require the security property to hold even in this case.

We present in Construction 2 a polynomial-based PPRF scheme that is based on the construction in [KMP+17].

Theorem 3. *Construction 2 is a PPRF.*

Proof. It is easy to see that this construction is correct. For every k, X and T, let $p = \mathsf{Hint}(k, X, T)$, then for all $i \in |X|$ it holds that

$$\begin{aligned} F(k, p, X(i)) &= F'(k, X(i)) \oplus p(X(i)) \\ &= F'(k, X(i)) \oplus F'(k, X(i)) \oplus T(i) \\ &= T(i) \end{aligned}$$

as required.We now reduce the security of the scheme to the security of a PRF (i.e., to the standard PRF definition, with many oracle accesses). Let M be a PPRF oracle over $\hat{F}$ of Construction 2. Assume there exists a distinguisher $\mathcal{D}$ and a caller $\mathcal{A}$ such that $\mathcal{D}$ distinguishes between the output of M after interacting with $\mathcal{A}$, when $\mathcal{A}$ chooses X and x as its inputs, and the output of $\mathcal{S}(1^\kappa, |X|)$ (where $\mathcal{S}$ is the simulator described in Definition 1) with probability μ.

We present a distinguisher $\mathcal{D}'$ that has an oracle access to either a truly random function $R(\cdot)$ or a PRF $\tilde{F}(k, \cdot)$. The distinguisher $\mathcal{D}'$ runs as follows:

Given an oracle $\mathcal{O}$ to either $R(\cdot)$ or $\tilde{F}(k, \cdot)$, $\mathcal{D}'$ samples T from $\mathcal{T}$, then, for every $i \in [|X|]$ it queries the oracle on $X(i)$ and obtains $\mathcal{O}(X(i))$. It interpolates the polynomial p using the points $\{(X(i), \mathcal{O}(X(i)) \oplus T(i))\}_{i \in |X|}$ and provides p's coefficients to $\mathcal{D}$. For the query x, $\mathcal{D}'$ hands $\mathcal{D}$ the value $\mathcal{O}(x) \oplus p(x)$ and outputs whatever $\mathcal{D}$ outputs.

Observe that if $\mathcal{O}$ is truly random, then the values $\{R(X(i)) \oplus T(i))\}_{i \in [|X|]}$ are uniformly random and thus the polynomial p is uniformly random and independent of T. If $x \notin X$ then the value $R(x) \oplus p(x)$ is obviously random since $R(x)$ is independent of p. In addition, if $x = X(i)$ for some i, then the value $R(x) \oplus p(x)$ equals $T(i)$ for some $i \in [|X|]$, which is uniformly random since T is sampled from $\mathcal{T}$ and every $t \in T$ is distributed uniformly. Therefore, the pair $(p, R(x) \oplus p(x))$ is distributed identically to the output of $\mathcal{S}$. On the other hand, if $\mathcal{O}$ is a pseudorandom function, then the values $\{F_k(X(i)) \oplus T(i))\}_{i \in [|X|]}$ from which the polynomial p is interpolated, along with the second output $F_k(x) \oplus p(x)$, are distributed identically to the output of M upon an interaction with $\mathcal{A}$. This leads to the same distinguishing success probability μ, for both $\mathcal{D}$ and $\mathcal{D}'$, which must be negligible. □

3.3 Batch PPRF

Note that the size of the hint generated by algorithm KeyGen is $\kappa \cdot |X|$ (i.e., the polynomial is represented by $|X|$ coefficients, each of size κ bits). In our setting we use an independent PPRF per bin, where each bin contains at most $O(\log n / \log \log n)$ values. Therefore the hint for one bin is of size $O(\kappa \cdot \log n / \log \log n)$, and the size of all hints is $O(\kappa \cdot n \cdot \log n / \log \log n)$. However, we know that the total number of values in all P_2's bins is $2n$, since each value is stored in (at most) two locations of the table[4]. We next show that it is possible to combine the hints of *all* bins to a single hint of length $2n$, thus reducing the total communication for all hints to $O(n)$.

We first present a formal definition of the notion of batch PPRF.

Definition 2. *An ℓ-bits, β-bins PPRF (or (ℓ, β)-PPRF) is a pair of algorithms $\hat{F} = (\mathsf{KeyGen}, F)$ as follows:*

[4] In the actual implementation we use a more general variant of Cuckoo hashing with a parameter $K \in \{2, 3\}$ where each item is stored in K locations in the table. The size of the hint will be $K \cdot n$.

CONSTRUCTION 4 (Batched PPRF)

Let $F' : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a PRF.

- $\mathsf{Hint}(k, X, T)$.
  Given the keys $k = k_1, \ldots, k_\beta$, the sets $X = X_1, \ldots, X_\beta$ and the target multi-sets $T = T_1, \ldots, T_\beta$, interpolate the polynomial p using the points $\{(X_j(i), F'(k_j, X_j(i)) \oplus T_j(i))\}_{j \in \beta; i \in [|X_j|]}$. Return p as the hint.
- $F(k, \mathsf{hint}, x)$.
  Interpret hint as a polynomial, denoted p. Return $F'(k, x) \oplus p(x)$. (Same as in Construction 2.)

- $\mathsf{Hint}(k, X, T) \to \mathsf{hint}_{k,X,T}$. *Given a set of uniformly random and independent keys $k = k_1, \ldots, k_\beta \in \{0,1\}^\kappa$, the sets $X = X_1, \ldots, X_\beta$ where $|X_j(i)| = \ell$ for all $j \in [\beta]$ and $i \in [|X|]$ and a target multi-sets $T = T_1, \ldots, T_\beta$ where for every $j \in [\beta]$ it holds that $|T_j| = |X_j|$ and all elements in T_j are uniformly distributed (but, again, may be correlated), output the hint $\mathsf{hint}_{k,X,T} \in \{0,1\}^{\kappa \cdot N}$ where $N = \sum_{j=1}^{\beta} |X_j|$.*
- $F(k, \mathsf{hint}, x) \to y^\star$. *Given a key $k \in \{0,1\}^\kappa$, a hint $\mathsf{hint} \in \{0,1\}^{\kappa \cdot N}$ and an input $x \in \{0,1\}^\ell$, output $y^\star \in \{0,1\}^\ell$.*

As before, we want a batched PPRF to have the following properties:

- **Correctness.** *For every $k = k_1, \ldots, k_\beta$, $T = T_1, \ldots, T_\beta$ and $X = X_1, \ldots, X_\beta$ as above, we have*
$$F\left(k_j, \mathsf{hint}, X_j(i)\right) = T_j(i)$$
for every $j \in [\beta]$ and $i \in [|X_j|]$.
- **Security.** *We say that an interactive machine M is a* batched PPRF oracle *over $\hat{F}$ if, when interacting with a "caller" $\mathcal{A}$, it works as follows:*
  1. *M is given $X = X_1, \ldots, X_\beta$ from $\mathcal{A}$.*
  2. *M samples uniformly random keys $k = k_1, \ldots, k_\beta$ and target multi-sets $T = T_1, \ldots, T_\beta$ from $\mathcal{T}$, and invokes $\mathsf{hint} \leftarrow \mathsf{Hint}(k, X, T)$ hands hint to $\mathcal{A}$.*
  3. *M is given β queries $x_1, \ldots, x_\beta$ from $\mathcal{A}$ and responds with $y_1^\star, \ldots, y_\beta^\star$ where $y_j^\star = F(k_j, \mathsf{hint}, x_j)$.*
  4. *M halts.*

  The scheme $\hat{F}$ is said to be secure if for every disjoint sets $X_1, \ldots, X_\beta$ (where $N = \sum_{j \in [\beta]} |X_j|$) input by a PPT machine $\mathcal{A}$, the output of M is computationally indistinguishable from the output of $\mathcal{S}(1^\kappa, N)$, such that $\mathcal{S}$ outputs a uniformly random $\mathsf{hint} \in \{0,1\}^{\kappa \cdot N}$ and a set of β uniformly random values from $\{0,1\}^\ell$.

Construction 4 is a batched version of Construction 2.

Theorem 5. *Construction 4 is a secure (ℓ, β)-PPRF.*

Proof. For correctness, note that for every $j \in [\beta]$ and $i \in [|X_j|]$ it holds that

$$\begin{aligned} F(k_j, p, X_j(i)) &= F'(k_j, X_j(i)) \oplus p(X_j(i)) \\ &= F'(k_i, X_j(i)) \oplus F'(k_i, X_j(i)) \oplus T_j(i) \\ &= T_j(i). \end{aligned}$$

The security of the scheme is reduced to the security of a batch PRF $\tilde{F}$. Informally, a batch PRF works as follows: Sample uniform keys $k_1, \ldots, k_\beta \in \{0,1\}^\kappa$ and for a query (j, x) respond with $\tilde{F}(k_j, x)$. One can easily show that a batch PRF is indistinguishable from a set of β truly random functions $R_1, \ldots, R_\beta$ where on query (j, x) the output is $R_j(x)$.

Let M be a batched PPRF oracle over $\hat{F}$ of Construction 4. Assume there exists a distinguisher $\mathcal{D}$ and a caller $\mathcal{A}$ such that $\mathcal{D}$ distinguishes between the output of M after interacting with $\mathcal{A}$, when $\mathcal{A}$ chooses $X_1, \ldots, X_\beta$ and $x_1, \ldots, x_\beta$ as its inputs, and the outputs of $\mathcal{S}(1^\kappa, N)$, where $\mathcal{S}$ is the simulator described in Definition 2.

We present a distinguisher $\mathcal{D}'$ that has an oracle access $\mathcal{O}$, to either a batch PRF $\tilde{F}(k_j, \cdot)$ or a set of truly random functions $R_j(\cdot)$ (where $j \in [\beta]$). The distinguisher $\mathcal{D}'$ works as follows: sample $T_1, \ldots, T_\beta$ from $\mathcal{T}$, interpolate a polynomial p with the points $\{(X_j(i), \mathcal{O}(j, X_j(i)) \oplus T_j(i))\}_{j \in [\beta]; i \in [|X_j|]}$ and hand p's coefficients to $\mathcal{D}$ as the hint. Then, for query x_j of $\mathcal{D}$, respond with $y_j^\star = \mathcal{O}(x_j) \oplus p(x_j)$. Finally, $\mathcal{D}'$ outputs whatever $\mathcal{D}$ outputs.

First note that if $\mathcal{O}$ is a set of truly random functions then the polynomial p is uniformly random and independent of $y_1^\star, \ldots, y_\beta^\star$ because all interpolation points are uniformly random. Now, if $x_j \notin X_j$ then the result is obviously uniformly random. Otherwise, if $x_j = X_j(i)$ for some i then note that the result is $T_j(i)$ which is uniformly random as well, since the other elements in T_j are unknown. Thus, this is distributed identically to the output of $\mathcal{S}(1^\kappa, N)$. On the other hand, if $\mathcal{O}$ is a batch PRF then the interpolation points $\{(X_j(i), \mathcal{O}(j, X_j(i)) \oplus T_j(i))\}_{j \in [\beta]; i \in [|X_j|]}$ along with $y_1^\star, \ldots, y_\beta^\star$ are distributed identically to the output of M upon an interaction with $\mathcal{A}$. This leads to the same distinguishing success probability for both $\mathcal{D}$ and $\mathcal{D}'$, which must be negligible. □

3.4 Batch *Oblivious* Programmable Pseudorandom Functions

In this section we define a two-party functionality for batched oblivious programmable pseudorandom function (Functionality 6), which is the main building block in our PSI protocols. The functionality is parametrised by a (ℓ, β)-PPRF $\hat{F} = (\mathsf{Hint}, F)$ and interacts with a sender, who programs $\hat{F}$ with β sets, and a receiver who queries $\hat{F}$ with β queries. The functionality guarantees that the sender does not learn what are the receiver's queries and the receiver does not learn what are the programmed points.

Given a protocol that realizes $\mathcal{F}_{\mathsf{OPRF}}$ and a secure (ℓ, β)-PPRF, the realization of Functionality 6 is simple and described in Protocol 7.

Theorem 8. *Given an (ℓ, β)-PPRF, Protocol 7 securely realizes Functionality 6 in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model.*

FUNCTIONALITY 6 (Batch Oblivious PPRF)

Parameters. A (ℓ, β)-PPRF $\hat{F} = (\mathsf{Hint}, F)$.
Sender's inputs. These are the following values:
- Disjoint sets $X = X_1, \ldots, X_\beta$ where $|X_j(i)| \in \{0,1\}^\ell$ for every $j \in [\beta]$ and $i \in [|X_j|]$. Let the total number of elements in all sets be $N = \sum_j |X_j|$.
- The sets $T = T_1, \ldots, T_\beta$ sampled independently from $\mathcal{T}$.

Receiver's inputs. The queries $x_1, \ldots, x_\beta \in \{0,1\}^\ell$.

The functionality works as follows:
1. Sample uniformly random and independent keys $k = k_1, \ldots, k_\beta$.
2. Invoke $\mathsf{Hint}(k, X, T) \to \mathsf{hint}$.
3. Output hint to P_1 (P_2 can compute it on its own from k, X, T).
4. For every $j \in [\beta]$ output $F(k_j, \mathsf{hint}, x_j)$ to the receiver.

PROTOCOL 7 (Batch Oblivious PPRF)

The protocol is defined in the $\mathcal{F}_{\mathsf{OPRF}}$-hybrid model and receives an (ℓ, β)-PPRF $\hat{F} = (\mathsf{Hint}, F)$ as a parameter. The underlying PRF in both $\mathcal{F}_{\mathsf{OPRF}}$ and $\hat{F}$ is the same and denoted F'. The protocol proceeds as follows:
1. The parties invoke β instances of $\mathcal{F}_{\mathsf{OPRF}}$. In the $j \in [\beta]$ instance, P_2 inputs nothing and receives the key k_j, and P_1 inputs x_j and receives $F'(k_j, x_j)$.
2. Party P_2 invokes $p \leftarrow \mathsf{Hint}(k, X, T)$ and sends p to P_1.
3. For every $j \in [\beta]$, party P_1 outputs $F'(k_j, x_j) \oplus p(x_j)$.

Proof. Note that party P_2 receives nothing in the functionality but receives $k_1, \ldots, k_\beta$ in the real execution as output from $\mathcal{F}_{\mathsf{OPRF}}$. Therefore, P_2's view can be easily simulated with the simulator of $\mathcal{F}_{\mathsf{OPRF}}$.

As for the view of P_1, from the security of the PPRF it follows that it is indistinguishable from the output of $\mathcal{S}(1^\kappa, N)$ where $\mathcal{S}$ is the simulator from Definition 2. □

4 A Super-Linear Communication Protocol

4.1 The Basic Construction

Let $C_{a,b}$ be a Boolean circuit that has $2 \cdot a \cdot (b + \lambda)$ input wires, divided to a sections of $2b + \lambda$ inputs wires each. For each section, the first (resp. second) β input wires are associated with P_1 (resp. P_2). The last λ input wires are associated with P_1 as well. Denote the first (resp. second) β bits input to the i-th section by $u_{i,1}$ (resp. $v_{i,2}$) and the last λ bits by z_i. The circuit first compares $u_{i,1}$ to $v_{i,2}$ for every $i \in [\alpha]$ and produces $w_i = 1$ if $u_{i,1} = v_{i,2}$ and 0 otherwise.

PROTOCOL 9 (Private Set Intersection)

Inputs. P_1 has $X = \{x_1, \ldots, x_n\}$ and P_2 has $Y = \{y_1, \ldots, y_n\}$.
Protocol. The protocol proceeds in 3 steps as follows:

1. **Hashing.** The parties agree on hash functions $H_1, H_2 : \{0,1\}^\ell \to [\beta]$, which are used as follows:
   - P_1 uses H_1, H_2 in a Cuckoo hashing construction that maps $x_1, \ldots, x_n$ to a table Table_1 of $\beta = 2(1+\varepsilon)n$ entries, where input x_i is mapped to either entry $\mathsf{Table}_1[H_1(x_i)]$ or $\mathsf{Table}_1[H_2(x_i)]$ or the stash Stash (which is of size s)[a]. Since $\beta > n$, P_1 fills the empty entries in Table_1 with a uniformly random value.
   - P_2 maps $y_1, \ldots, y_n$ to Table_2 of β entries using both H_1 and H_2. That is, y_i is placed in both $\mathsf{Table}_2[H_1(y_i)]$ and $\mathsf{Table}_2[H_2(y_i)]$. (Obviously, some bins will have multiple items mapped to them. This is not an issue, and there is even no need to use a probabilistic upper bound on the occupancy of the bin.)
2. **Computing batch OPPRF.** P_2 samples uniformly random and independent target values $t_1, \ldots, t_\beta \in \{0,1\}^\kappa$. The parties invoke an (λ, β)-OPPRF (Functionality 6; recall that λ is the bit-length of the items). P_2 inputs $Y_1, \ldots, Y_\beta$ and $T_1, \ldots, T_\beta$ where $Y_j = \mathsf{Table}_2[j] = \{y || j \mid y \in Y \wedge j \in \{H_1(y), H_2(y)\}\}$ and T_j has $|Y_j|$ elements, all equal to t_j. If, $j = H_1(y) = H_2(y)$ for some $y \in Y$ then P_2 adds a uniformly random element to $\mathsf{Table}_2[j]$. P_1 inputs $\mathsf{Table}_1[1], \ldots, \mathsf{Table}_1[\beta]$ and receives $y_1^\star, \ldots, y_\beta^\star$. According to the definition of the OPPRF, if $\mathsf{Table}_1[j] \in \mathsf{Table}_2[j]$ then $y_j^\star = t_j$.
3. **Computing the circuit.** The parties use a two-party computation (Functionality 1) with the circuit $C_{\beta+s\cdot n,\gamma}$.[b] For section $j \in [\beta]$ of the circuit, party P_1 inputs the first γ bits of $y_j^\star$ and $\mathsf{Table}_1[j]$, and P_2 inputs the first γ bits t_j; for the $\beta + j$-th section P_1 inputs $\mathsf{Stash}[\lceil j/n \rceil + 1]$ and P_2 inputs $\mathsf{Table}[(j \bmod n) + 1]$.

[a] We discuss the value of s in Sect. 4.2 and the value of ε in Sect. 7.1.
[b] We discuss the value of γ in Sect. 4.2.

Then, the circuit computes and outputs $f(Z)$ where $Z = \{z_i \mid w_i = 1\}_{i \in [a]}$ and f is the function required to be computed in the $\mathcal{F}_{\mathsf{PSI},f}$ functionality.

Correctness. If $z \in X \cap Y$ then z is mapped to both $\mathsf{Table}_2[H_1(z)]$ and $\mathsf{Table}_2[H_2(z)]$ by P_2. There are two cases: (1) z is mapped to $\mathsf{Table}_1[H_b(z)]$ by P_1 for $b \in \{1,2\}$. (2) z is mapped to Stash by P_1. In the first case the match is found in section $H_b(z)$ of the circuit; in the second case the match is certainly found since every item in the Stash is compared to every item in Y.

Two items $x \in X$ and $y \in Y$ where $x \neq y$ will not be matched, since by the properties of the PPRF P_1 receives a pseudorandom output. Since the parties only input the first γ bits of the PPRF results, those values will be matched with probability $2^{-\gamma}$. See Sect. 4.2 for a discussion on limiting the failure probability.

Security. The security of the protocol follows immediately from the security of the OPPRF and the two-party computation functionalities.

4.2 Limiting the Failure Probability

Protocol 9 might fail due to two reasons:

- **Stash size.** For an actual implementation, one needs to fix s and ε so that the stash failure probability will be smaller than $2^{-\sigma}$. If the stash is overflowed (i.e., more than s items are mapped to it) then the protocol fails.[5] As discussed in Sect. 2, setting $s = O(\log n / \log\log n)$ makes the failure probability negligible.
- **Input encoding.** The circuit compares the first γ bits of $y_j^\star$ of P_1 to the first γ bits of t_j of P_2. Thus, the false positive error probability in each comparison equals $2^{-\gamma}$ (due to $F(x)$, for $x \notin Y$, being equal to the programmed output), and therefore the overall probability of a false positive is at most $\beta \cdot 2^{-\gamma} = 2(1+\varepsilon)n \cdot 2^{-\gamma}$.

4.3 Reducing Computation

A major computation task of the protocol is interpolating the polynomial which encodes the hint. If we use Cuckoo hashing with $K = O(1)$ hash functions then the polynomial encodes $O(n)$ items and is of degree $O(n)$. This section describes how to reduce the *asymptotic* overhead of *computing* the polynomial and therefore we will use asymptotic notation. The concrete overhead is discussed in Sect. 7.2.

The overhead of interpolating a polynomial of degree $O(n)$ over arbitrary points is $O(n^2)$ operations using Lagrange interpolation, or $O(n \log^2 n)$ operations using FFT. The overhead can be reduced by dividing the polynomial to several lower-degree polynomials. In particular, let us divide the $\beta = O(n)$ bins to B "mega-bins", each encompassing β/B bins. Suppose that we have an upper bound such that the number of items in a mega-bin is at most m, except with negligible probability. Then the protocol can invoke a batch OPPRF for each mega-bin, using a different hint polynomial. Each such polynomial is of degree m. Therefore the computation overhead is $O(B \cdot m \log^2 m)$. Ideally, the upper bound on the number of items in a mega-bin, m, is of the same order as the expected number of items in a mega-bin, $O(n/B)$. In this case the computation overhead is $O(n/B \cdot B \cdot \log^2(n/B)) = O(n \log^2(n/B))$ and will be minimized when the number of mega-bins B is maximal.

It is known that when mapping $O(n)$ items to $B = n/\log n$ (mega-)bins, then with high probability the most occupied bin has less than $m = O(n/B) = O(\log n)$ items. When interested in concrete efficiency we can use the analysis

[5] In that case either not all items are stored in the stash – resulting in the protocol ignoring part of the input and potentially computing the wrong output, or P_1 needs to inform P_2 that it uses a stash larger than s – resulting in a privacy breach.

in [PSZ18] to find the exact number of mega-bins to make the failure probability sufficiently small (see Sect. 7.2). When interested in asymptotic analysis, it is easy to deduce from the analysis in [PSZ18] that with $B = n/\log n$ mega-bins, the probability of having more than $\omega(\log n)$ items in a mega-bin is negligible. Therefore when using this number of mega-bins, the computation overhead is only $\omega((n/\log n) \cdot \log^2(n)) = \omega(n \log n)$ using Lagrange interpolation. Using FFT interpolation, the asymptotic overhead is reduced to $\omega((n/\log n) \log n(\log\log n)^2) = \omega(n \cdot (\log\log n)^2)$. But since we map relatively few items to each mega-bin the gain in practice of using FFT is marginal.

5 A Linear Communication Protocol

We describe here a protocol in which the circuit computes only $O(n)$ comparisons. This protocol outperforms the protocols in Sect. 4.1 or in [PSWW18, CO18] which have a circuit that computes $\omega(n)$ comparisons. A careful analysis reveals that those protocols require $O(n)$ comparisons to process all items that were mapped to the Cuckoo hash table, and an additional $s \cdot n$ comparisons to process the $s = \omega(1)$ items that were mapped to the stash. We note that the concurrent and independent work of [FNO18] proposes to use a PSI protocol for unbalanced set sizes, such as in the work of [KLS+17], to reduce the complexity of handling the stash from $\omega(n)$ to $O(n)$ in PSI protocols. However, their idea can only be applied when the output is the intersection itself. When the output is a function of the intersection then their protocol has communication complexity $O(n \log\log n)$, cf. Sect. 1.4). In contrast, we achieve $O(n)$ communication even when the output is a function of the intersection.

We present two different techniques to achieve a linear communication protocol with failure probability that is negligible in the statistical security parameter σ. The first technique (see Sect. 5.1) is implied by a mathematical analysis of the failure probability (as argued in Sect. 1.4). The second technique (see Sect. 5.2) is implied by the empirical analysis presented in [PSZ18].

5.1 Linear Communication via Dual Execution

We overcome the difficulty of handling the stash by running a modified version of the protocol in three phases. The first phase is similar to the basic protocol, but ignores the items that P_1 maps to the stash. Therefore this phase inputs to the circuit the $O(n)$ results of comparing P_1's input items (except those mapped to the stash) with all of P_2's items. The second phase reverses the roles of the parties, and in addition now P_1 inputs only the items that it previously mapped to the stash. In this phase P_2 uses Cuckoo hashing and might map some items to the stash. The last phase only compares the items that P_1 mapped to the stash in the first phase, to the items that P_2 mapped to the stash in the second phase, and therefore only needs to handle very few items. Below, we describe our protocol in more detail: In Protocol 10, we describe our protocol in more detail.

PROTOCOL 10 (PSI with Linear Communication)

Inputs. P_1 has $X = \{x_1, \ldots, x_n\}$ and P_2 has $Y = \{y_1, \ldots, y_n\}$.
Protocol. The protocol proceeds in 3 phases as follows:

1. Run steps 1-2 of Protocol 9. Denote the items mapped to P_1's table by X_T (i.e., excluding the items mapped to the stash). In the end of this phase, for every $j \in [\beta]$, P_1 holds the OPPRF result $y_j^\star$ and P_2 holds the target value t_j.
2. Reverse the roles of P_1 and P_2 and run steps 1-2 of Protocol 9 again, where P_1 inputs $X_S = X \setminus X_T$ (i.e., only the items that were previously mapped to the stash) and P_2 inputs Y. Since the roles are reversed then P_1 maps X_S using simple hashing and P_2 maps Y using Cuckoo hashing. Denote the items mapped to the table and stash of P_2 by Y_T and Y_S, respectively. In the end of this phase, for every $j \in [\beta]$, P_1 has the target value $\tilde{t}_j$ and P_2 has the OPPRF result $\tilde{y}_j^\star$.
3. The parties use secure two-party computation (Functionality 1) with the circuit $C_{2\beta+s^2,\gamma}$ (where s is the stash size). For section $j \in [\beta]$ of the circuit, P_1 and P_2 input the first γ bits of $y_j^\star$ and t_j resp. For section $j \in \{\beta+1, \ldots, 2\beta\}$ of the circuit P_1 and P_2 input the first γ bits of $\tilde{t}_j$ and $\tilde{y}_j^\star$, respectively. Finally, for the rest s^2 sections of the circuit, the parties input every combination of $X_S \times Y_S$ (padded with uniformly random items so that $|X_S| = |Y_S| = s$).

Correctness & Efficiency. The protocol compares every pair in $X \times Y$ and therefore every item in the intersection is input to the circuit exactly once: Sections $1, \ldots, \beta$ of the circuit cover all pairs in $X_T \times Y$, sections $\beta+1, \ldots, 2\beta$ cover all pairs in $X_S \times Y_T$ and sections $2\beta+1, \ldots, 2\beta+s^2$ covers all pairs in $X_S \times Y_S$. This implies that the result of the three-phase construction is exactly the intersection $X \cap Y$. The communication complexity in the first two steps of the protocol is $O(n \cdot \kappa)$ as they involve the execution of a OPPRF with at most $O(n)$ items to the parties. The communication complexity of the third step is $O(n \cdot \gamma)$ since it involves $2n + s^2$ comparisons of γ-bit elements. Since the stash size is $s = O(\log n)$, overall there are $O(n)$ comparisons.

Security. As in the basic protocol (see Sect. 4.1), the security of this protocol is implied by the security of the OPPRF and secure two-party computation.

5.2 Linear Communication via Stash-Less Cuckoo Hashing

The largest communication cost factor in our protocols is the secure evaluation of the circuit. The asymptotically efficient Protocol 10 requires computing at least two copies of the basic circuit (for Phases 1 and 2), and it is therefore preferable to implement a protocol which has better *concrete* efficiency. We design a protocol that requires no stash (while achieving a small failure probability of less than 2^{-40}), and hence uses no dual execution.

In order to be able to not use the stash, hashing is done with $K > 2$ hash functions. We take into account the results of [PSZ18], which ran an empirical evaluation for the failure probability of Cuckoo hashing (failure is defined as the event where an item cannot be stored in the table and must be stored in the stash). They run experiments for a failure probability of 2^{-30} with $K = 3, 4$ and 5 hash functions, and extrapolated the results to yield the minimum number of bins for achieving a failure probability of less than 2^{-40}. The results showed that $\beta = 1.27n, 1.09n$, and $1.05n$ bins are required for $K = 3, 4$, and 5, respectively.

The main obstacle in using more than two hash functions in previous works on PSI was that the communication was still linear in $O(\max_b \cdot \beta)$, where $\max_b$ is the maximal number of elements in a bin of the simple hash table. The value of $\max_b$ increases with K since each item is stored K times in the simple hash table. In our protocol the communication for the circuit is independent of $\max_b$, as it only depends on the number of bins β. The communication for sending the polynomials, whose size is $O(K \cdot n \cdot \kappa)$, is just a small fraction of the overall communication and was in our experiments always smaller than 3%. In this paper, we therefore use $K = 3$ hash functions for our stash-less protocol.

6 PSI with Associated Payload

In many cases, each input item of the parties has some "payload" data associated with it. For example, an input item might include an id which is a credit card number, and a payload which is a transaction that was paid using this credit card. The parties might wish to compute some function of the *payloads* of the items in the intersection (for example, the sum of the financial transactions associated with these items). However, a straightforward application of our techniques does not seem to support this type of computation: Recall that P_2 might map multiple items to each bin. The OPPRF associates a single output β to all these items, and this value is compared in the circuit with the output α of P_1. But if P_2 inserts a single item to the circuit, it seems that this item cannot encode the payloads of all items mapped to this bin.

The 2D Cuckoo hashing circuit-based PSI protocol of [PSWW18] handles payloads well, since each comparison involves only a single item from each party. While our basic protocol cannot handle payloads, we show here how it can be adapted to efficiently encode payloads in the input to the circuit.

Let Table_1 and Stash be P_1's table and stash after mapping its items using Cuckoo hashing and let Table_2 be P_2's table after mapping its items using simple hashing. In addition, denote by $U(x)$ and $V(y)$ the payloads associated with $x \in X$ and $y \in Y$ respectively and assume that all payloads have the same length δ. The parties invoke two instances of batch OPPRF as follows:

1. A batch OPPRF where P_1 inputs $\mathsf{Table}_1[1], \ldots, \mathsf{Table}_1[\beta]$ and P_2 inputs $\mathsf{Table}_2[1], \ldots, \mathsf{Table}_2[\beta]$ and $T_1, \ldots, T_\beta$ where T_j has $|\mathsf{Table}_2[j]|$ elements, all equal to a uniformly random and independent value $t_j \in \{0,1\}^\lambda$. This is the same invocation of a batch OPPRF as in Protocol 9. At the end, P_1 has the OPPRF results $y_1^\star, \ldots, y_\beta^\star$ and P_2 has the target values $t_1, \ldots, t_j$.

2. In the second batch OPPRF, P_2 chooses the target values such that the elements in the set T_j are not equal. Specifically, P_1 inputs $\mathsf{Table}_1[1], \ldots, \mathsf{Table}_1[\beta]$ and P_2 samples $\tilde{t}_1, \ldots, \tilde{t}_\beta$ uniformly, and inputs $\mathsf{Table}_2[1], \ldots, \mathsf{Table}_2[\beta]$ and $T_1, \ldots, T_\beta$ where $T_j(i) = \tilde{t}_j \oplus V(\mathsf{Table}_2[j](i))$. Denote the OPPRF results that P_1 obtains by $\tilde{y}_1^\star, \ldots, \tilde{y}_\beta^\star$.

Then, the circuit operates in the following way: For the j-th section, P_1 inputs $\mathsf{Table}_1[j], y_j^\star, \tilde{y}_j^\star$ and $U(\mathsf{Table}_1[j])$, and P_2 inputs t_j and $\tilde{t}_j$. The circuit compares $y_j^\star$ to t_j. If they are equal then it forwards to the sub-circuit that computes f the item $\mathsf{Table}_1[j]$ itself, P_1's payload $U(\mathsf{Table}_1[j])$ and P_2's payload $\tilde{y}_j^\star \oplus \tilde{t}_j$. This holds since if $\mathsf{Table}_1[j]$ is the i-th item in P_2's table, namely, $\mathsf{Table}_2[j](i)$, then the value $\tilde{y}_j^\star$ received by P_1 is $\tilde{y}_j^\star = \tilde{t}_j \oplus V(\mathsf{Table}_2[j](i))$. Thus, $\tilde{y}_j^\star \oplus \tilde{t}_j = V(\mathsf{Table}_2[j](i))$ as required.

Efficiency. The resulting protocol has the same asymptotic complexity as our initial protocols without payloads. The number of comparisons in the circuit is the same as in the basic circuit.

Table 1. The results of [PSZ18] for the required stash sizes s for $K = 2$ hash functions and $\beta = 2.4n$ bins, and the minimum OPPRF output bitlength γ to achieve failure probability $< 2^{-40}$ when mapping n elements into β bins with Cuckoo hashing. For $K > 2$ hash functions we choose a large enough number of bins β to achieve stash failure probability $< 2^{-40}$.

# Elements n		2^8	2^{12}	2^{16}	2^{20}	2^{24}
Stash size s for $K = 2$		12	6	4	3	2
OPPRF output length γ	$K = 2$, $\beta = 2.4n$	50	54	58	62	66
	$K = 3$, $\beta = 1.27n$, $s = 0$	49	53	57	61	65
	$K = 4$, $\beta = 1.09n$, $s = 0$	49	53	57	61	65
	$K = 5$, $\beta = 1.05n$, $s = 0$	49	53	57	61	65

7 Concrete Costs

In this section we evaluate the concrete costs of our protocol for concrete values of the security parameters. We set the computational security parameter to $\kappa = 128$, and the statistical security parameter to $\sigma = 40$.

7.1 Parameter Choices for Sufficiently Small Failure Probability

For $K = 2$ hash functions, following previous works on PSI (e.g., [PSSZ15, PSWW18]), we set the table size parameter for Cuckoo hashing to $\epsilon = 0.2$, and use a Cuckoo table with $\beta = 2n(1+\epsilon) = 2.4n$ bins. The resulting stash sizes for mapping n elements into $\beta = 2.4n$ bins, as determined by the experiments in

[PSZ18], are summarized in Table 1. Note that we use here concrete values for the stash size, and are aiming for a failure probability smaller than 2^{-40}. This can either be achieved using the basic protocol of Sect. 4.1 with the right choice of the stash size, or by running the three rounds $O(n)$ complexity protocol of Sect. 5.

Another option is described in Sect. 5.2, where we use more than two hash functions (specifically, use $K = 3, 4$, or 5 functions), with the hash table being of size $\beta = 1.27n, 1.09n$, or $1.05n$, respectively. These parameters achieve a failure probability smaller than 2^{-40} according to the experimental analysis in [PSZ18].

As described in Sect. 4.2, even if there are no stash failures, the scheme can fail due to collisions in the output of the PRF, with probability $\beta \cdot 2^{-\gamma}$, where γ is the output bitlength of the OPPRF. To make this failure probability smaller than the statistical security parameter (which we set to 40), the output bitlength of the OPPRF must be $\gamma = 40 + \log_2 \beta$ bits.

7.2 Computing Polynomial Interpolation

We implemented interpolation of polynomials of degree d using an $O(d^2)$ algorithm based on Lagrange interpolation in a prime field where the prime is the Mersenne prime $2^{61} - 1$. The runtime for interpolating a polynomial of degree $d = 1024$ was 7 ms, measured on an Intel Core i7-4770K CPU with a single thread. The runtimefor different values of d behaved (very accurately) as a quadratic function of d. The actual algorithmsare those implemented in NTL v10.0 with field arithmetics replaced with our customized arithmetic operationsover the Mersenne prime $2^{61} - 1$. Most importantly, this field enables an order of magnitude faster multiplication of field elements: multiplying $x \cdot y$ with $|x|, |y| \leq 61$ is implemented by multiplying x and y over $\mathbb{Z}$ to obtain $z = xy$ with $|z| \leq 122$. Then the result is the sum of the element represented by the lower 61 bits of z with the element represented by the higher 61 bits of z (and therefore no expensive modular reduction is required). The Mersenne prime $2^{61} - 1$ allows the use of at least 40-bit statistical security for up to $n = 2^{20}$ elements for all our algorithms using permutation-based hashing (cf. [PSSZ15]). To use larger sets, we see two possible solutions: (i) using a larger Mersenne prime or (ii) reducing the statistical security parameter σ (e.g., using $\sigma = 38$ for achieving less than $2^{-\sigma}$ failure probability for $n = 2^{22}$ elements, $K = 3$ hash functions, and $\beta = 1.27n$ bins). The required minimum bit-length of the elements using permutation-based hashing with failure probability $2^{-\sigma}$ is computed as $\ell = \sigma + 2\log_2 n - \log_2 \beta$. The OPPRF output is also ≤ 61 bits in most cases as shown in Table 1.

For reducing the computation complexity of our protocol, we use the approach described in Sect. 4.3, where instead of interpolating a polynomial of degree $K \cdot n$, where K is the number of hash functions and n is the number of elements for PSI, we interpolate multiple smaller polynomials of degree at most $d = 1024$. We therefore have to determine the minimum number of mega-bins B such that when mapping $N = K \cdot n$ elements to B bins, the probability of having a bin with more than $\max_b = 1024$ elements is smaller than 2^{-40}. As in the analysis for simple hashing in [PSZ18], we use the formula from [MR95]:

$$P(\text{“}\exists \text{ bin with } \geq \max_b \text{ elements”}) \leq \sum_{i=1}^{B} P(\text{“bin } i \text{ has } \geq \max_b \text{ elements”})$$
$$= B \cdot \sum_{i=\max_b}^{N} \binom{N}{i} \cdot \left(\frac{1}{B}\right)^i \cdot \left(1 - \frac{1}{B}\right)^{N-i}.$$

We depict the corresponding numbers in Table 2. With these numbers and our experiments for polynomial interpolation described above, the estimated runtimes for the polynomial interpolation are $B \cdot 7$ ms. The hints (polynomials) that need to be sent have size $B \cdot \max_b \cdot \gamma$ bits which is only slightly larger than the ideal communication of $K \cdot n \cdot \gamma$ bits when using one large polynomial as shown in Table 2.

Note that in contrast to many PSI solutions whose main run-time bottleneck is already network bandwidth (which cannot be easily improved in many settings such as over the Internet), the run-time of our protocols can be improved by using multiple threads instead of one thread. Since the interpolation of polynomials for different mega-bins is independent of each other, the computation scales linearly in the number of physical cores and thus can be efficiently parallelized.

Table 2. Parameters for mapping $N = K \cdot n$ elements to B mega-bins s.t. each mega-bin has at most $\max_b \leq 1024$ elements with probability smaller than 2^{-40}. The lower half of the table contains the expected costs for the polynomial interpolations.

# hash functions	$K = 2$			$K = 3$		
Set size	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$
# mega-bins B	11	165	2663	16	248	4002
Maximum number of elements $\max_b$	944	1021	1024	975	1021	1024
Polynomial interpolation [in milliseconds]	126	1815	29293	183	2809	45335
Size of hints [in bits]	560736	9770970	169068544	826800	14432856	249980928
Ideal size of hints for one polynomial [in bits]	436330	7505580	128477895	651264	11206656	191889408

7.3 Communication and Depth Comparison

We first compute the communication complexity of our basic construction from Sect. 4.1. The communication is composed of (a) the OPRF evaluations for each of the B bins, (b) the hints consisting of the polynomials, (c) the circuit for comparing the outputs of the OPPRFs in each bin, and (d) the circuit for comparing the s elements on the stash with the n elements of P_2.

With regards to (a), the OPRF protocol of [KKRT16], which was also used in [KMP+17], has an amortized communication of at most 450 bits per OPRF

evaluation for set sizes up to $n = 2^{24}$ elements (cf. [KKRT16, Table 1]). This amounts to $B \cdot 450$ bits of communication.

With regards to (b), for the size of the hints in the OPPRF construction we use the values given in Table 2. These numbers represent the communication when using mega-bins, and are slightly larger than the ideal communication of $K \cdot n$ coefficients of size γ bits each, that would have been achieved by using a single polynomial for all values. However, it is preferable to use mega-bins since their usage substantially improves the computation complexity as described in Sect. 4.3, while the total communication for the hints is at most 3% of the total communication. (This also shows that any improvements of the size of the hints will have only a negligible effect on the total communication).

With regards to (c), the circuit compares B elements of bitlength γ, and hence requires $B \cdot (\gamma - 1)$ AND gates. With 256 bits per AND gate [ALSZ13, ZRE15] this yields $B \cdot (\gamma - 1) \cdot 256$ bits of communication.

With regards to (d), the final circuit consists of $s \cdot n$ comparisons of bitlength σ. This requires $sn(\sigma - 1) \cdot 256$ bits of communication.

We now analyze the communication complexity of our $O(n)$ protocol described in Sect. 5. The main difference compared to the basic protocol analyzed above is that a different method is used for comparing the elements of the stash, i.e., replacing step (d) above. The new method replaces this step by letting P_2 use Cuckoo hashing of its n elements into B bins and then evaluating OPRF for each of these bins. This requires $B \cdot 450$ bits of communication plus B comparisons of γ bit values. Overall, this amounts to $B \cdot (450 + (\gamma - 1) \cdot 256)$ bits of communication. For simplicity, we omit the communication for comparing the elements for phase 3 which compares the elements on the two stashes, as it is negligible.

Comparison to Previous Work. In Table 3, we compare the resulting communication of our protocols to those of previous circuit-based PSI protocols of [HEK12, PSSZ15, PSWW18, FNO18]. As can be seen from this table, our protocols improve communication by an integer factor, where the main advantage of our protocols is that their communication complexity is *independent* of the bitlength of the input elements. Namely, for arbitrary input bitlengths, our no-stash protocol improves the communication over the previous best protocol of [PSWW18] *by a factor of 12.8x for* $n = 2^{12}$ *to a factor of 10.1x for* $n = 2^{20}$. For fixed bitlength of $\sigma = 32$ bits, our no-stash protocol improves communication over [PSWW18] *by a factor of 5.7x for* $n = 2^{12}$ *to a factor of 2.6x for* $n = 2^{20}$.

Circuit Depth. For some secure circuit evaluation protocols like GMW [GMW87] the round complexity depends on the depth of the circuit. In Table 4, we depict the circuit depths for concrete parameters of our protocols and previous work, and show that our circuits have about the same low depth as the best previous works [PSSZ15, PSWW18]. In more detail, the Sort-Compare-Shuffle (SCS) circuit of [HEK12] has depth $\log_2 \sigma \cdot \log_2 n$ when using depth-optimized comparison circuits. The protocols of [PSSZ15, PSWW18] have depth

Table 3. Communication in MB for circuit-based PSI on n elements of fixed bitlength $\sigma = 32$ (left) and arbitrary bitlength hashed to $\sigma = 40 + 2\log_2(n) - 1$ bits (right). The numbers for previous protocols are based on the circuit sizes given in [PSWW18, Table 3] with 256 bit communication per AND gate. The best values are marked in bold.

Protocol $n =$	$\sigma = 32$			Arbitrary σ		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
SCS [HEK12]	104	2 174	42 976	205	4 826	106 144
Circuit-Phasing [PSSZ15]	130	1 683	21 004	320	5 552	97 708
Hashing + SCS [FNO18]	-	1 537	21 207	-	3 998	72 140
2D CH [PSWW18]	51	612	6 582	115	1 751	25 532
Ours Basic Sect. 4.1	41	550	8 123	65	870	12 731
Ours Advanced Sect. 5	35	604	10 277	35	604	10 277
Ours No-Stash Sect. 5.2	**9**	**149**	**2 540**	**9**	**149**	**2 540**
Breakdown:						
OPRF	0.3 (3%)	5 (3%)	72 (3%)	0.3 (3%)	5 (3%)	72 (3%)
Sending polynomials	0.1 (1%)	2 (1%)	30 (1%)	0.1 (1%)	2 (1%)	30 (1%)
Circuit	9 (96%)	142 (96%)	2438 (96%)	9 (96%)	142 (96%)	2438 (96%)
Improvement factor	5.7x	4.1x	2.6x	12.8x	11.8x	10.1x

$\log_2 \sigma$. A depth-optimized SCS circuit for the construction in [FNO18] has depth $\log_2(\sigma - \log_2(n/b)) \cdot \log_2((1+\delta)b)$, where concrete parameters for n, δ, b are given in [FNO18, Table 1]. Our protocols consist of circuits for comparing the elements on the stash of bitlength σ and the outputs of the OPPRFs of length γ and therefore have depth $\max(\log \sigma, \log \gamma) = \max(\log \sigma, \log_2(40 + 2\log_2(n) - 1))$.

Table 4. Circuit depth for circuit-based PSI on n elements of fixed bitlength $\sigma = 32$ (left) and arbitrary bitlength hashed to $\sigma = 40 + 2\log_2(n) - 1$ bits (right).

Protocol $n =$	$\sigma = 32$			Arbitrary σ		
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
SCS [HEK12]	60	80	100	72	98	126
Circuit-Phasing [PSSZ15]	5	5	5	6	7	7
Hashing + SCS [FNO18]	-	42	36	-	54	51
2D CH [PSWW18]	5	5	5	6	7	7
Our Protocols	6	6	6	6	7	7

7.4 Runtime Comparison

In this section we compare the runtimes of different PSI protocols. In Sect. 7.2 we conducted experiments for polynomial interpolation, the main new part of our protocol, and we show below that this step takes only a small fraction of the total runtime. We also implemented our most efficient protocol (see Sect. 5.2).[6] In addition, we estimate the runtime of our less efficient basic protocol (see

[6] Our implementation is available at https://github.com/encryptogroup/OPPRF-PSI.

Sect. 4.1) and the protocol with linear communication overhead (see Sect. 5) based on the experiments of the interpolation procedure and rigorous estimations from previous works.

Previous Work. As we have seen in the analysis of the communication overhead in Sect. 7.3, our protocols provide better improvements to performance in the case of arbitrary bitlengths. The previous work of [PSWW18] gave runtimes only for fixed bitlength of 32 bits in [PSWW18, Table 4]. Therefore, we extrapolate the runtimes of the previous protocols from fixed bitlength to arbitrary bitlength based on the circuit sizes given in [PSWW18, Table 3]. The estimated runtimes are given in Table 5. The LAN setting is a 1 Gbit/s network with round-trip time of 1 ms and the WAN setting is a 100 Mbit/s network with round-trip time of 100 ms. Runtimes were not presented in [FNO18], but since their circuit sizes and depths are substantially larger than those of [PSWW18] (cf. Tables 3 and 4), their runtimes will also be substantially higher than those of [PSWW18].

Our Implementation. We implemented and benchmarked our most efficient no-stash OPPRF-based PSI protocol (see Sect. 5.2) on two commodity PCs with an Intel Core i7-4770K CPU. We instantiated our protocol with security parameter $\kappa = 128$ bits, $K = 3$ hash functions, $B = 1.27n$ bins, and no stash (see Sect. 5.2). Our OPPRF implementation is based on the OPRF protocol of [PSZ18].[7] For the secure circuit evaluation, we used the ABY framework [DSZ15]. The run-times are averaged over 50 executions. The results are described in Table 5.

Comparison with PSI Protocols. As a baseline, we compare our performance with specific protocols for computing the intersection itself. (However, as is detailed in Sect. 1.2, our protocol is circuit-based and therefore has multiple advantages compared to specific PSI protocols). Our best protocol is slower by a factor of 41x than today's fastest PSI protocol of [KKRT16] for $n = 2^{20}$ elements in the WAN setting (cf. Table 5).

Comparison with Public Key-Based PSI Variant Protocols. Our circuit-based protocol is substantially faster than previous public key-based protocols for computing variants of PSI, although they have similar asymptotic linear complexity. As an example, consider comparing whether the size of the intersection is greater than a threshold (PSI-CAT). In our protocol, we can compute the PSI-CAT functionality by extending the PSI circuit of Table 5 with a Hamming distance circuit (which, using the size-optimal construction of [BP06], adds less than n AND gates). The final comparison with the threshold adds another $\log_2 n$

[7] This OPRF protocol has communication that is higher by 10% to 15% than the communication of the OPRF protocol of [KKRT16]. But since OPRF requires less than 3% of the total communication, this additional cost is negligible in our protocol.

Table 5. Total run-times in ms for PSI variant protocols on n elements of arbitrary bitlength using GMW [GMW87] for secure circuit evaluation and one thread. Numbers for all but our protocols are based on [PSWW18]. The best values for generic circuit-based PSI protocols are marked in bold.

Network	LAN			WAN		
Protocol $n =$	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
Special-purpose PSI protocols (as baseline)						
DH/ECC PSI [Sha80, Mea86, DGT12]	3296	49010	7904054	4082	51866	8008771
BaRK-OPRF [KKRT16]	113	295	3882	540	1247	14604
Generic circuit-based PSI protocols						
Circuit-Phasing [PSSZ15]	7825	67292	1126848	37380	327976	4850571
2D CH [PSWW18]	5031	25960	336134	22796	129436	1512505
Ours Basic Sect. 4.1 (estimated)	2908	13767	182204	12934	63861	752695
Ours Advanced Sect. 5 (estimated)	1674	9763	148436	7372	43675	597885
Ours No-Stash Sect. 5.2, Total	**1 199**	**8 486**	**120 731**	**5 910**	**22 134**	**261 481**
Breakdown:						
OPRF	724 (60%)	1097 (13%)	5844 (5%)	2867 (49%)	4164 (19%)	26121 (10%)
Polynomial interpolation	183 (15%)	2809 (33%)	45335 (38%)	183 (3%)	2809 (13%)	45335 (17%)
Polynomial transmission	16 (1%)	145 (2%)	667 (0%)	816 (13%)	1079 (5%)	4012 (2%)
Polynomial evaluation	58 (5%)	1344 (16%)	21768 (18%)	58 (1%)	1344 (6%)	21768 (8%)
Circuit	218 (18%)	3091 (36%)	47117 (39%)	1986 (34%)	12738 (57%)	164245 (63%)
Improvement over [PSWW18]	4.2x	3.1x	2.8x	3.9x	5.8x	5.8x

AND gates [BPP00] which are negligible as well. For the PSI-CAT functionality, [ZC17] report runtimes of 779 s for $n = 2^{11}$ elements, [HOS17] report runtimes of 728 s for $n = 2^{11}$ elements, and [ZC18] report runtimes of at least 138 s for $n = 100$ elements, whereas our protocol requires 0.52 s for $n = 2^{11}$ elements and 0.34 s for $n = 100$ elements. Hence, we improve over [ZC17] by a factor of 1 498x, over [HOS17] by a factor of 1 400x, and over [ZC18] by a factor of 405x. As an example for computing PSI-CAT with larger set sizes, our protocol requires 124 s for $n = 2^{20}$ elements.

The protocol described by Google for computing ad revenues [Yun15, Kre17] (see Sect. 1.2) is based on the DH-based PSI protocol which is already 65x slower than our protocol for $n = 2^{20}$ elements over a LAN (cf. Table 5) and leaks the intersection cardinality as an intermediate result. Here, too, our circuit would be only slightly larger than the PSI circuit of Table 5.

Comparison with Circuit-Based PSI Protocols. As can be seen from Table 5, our no-stash protocol from Sect. 5.2 is substantially more efficient than our basic protocol and our linear asymptotic overhead protocol from Sect. 4.1 and Sect. 5, respectively. It improves over the best previous circuit-based PSI protocol from [PSWW18] by factors of 4.2x to 2.8x in the LAN setting, and by factors of 5.8x to 3.9x in the WAN setting. From the micro-benchmarks in Table 5, we also observe that the runtimes for the polynomial interpolation are a significant fraction of the total runtimes of our protocols (3% to 33% for the interpolation and 1% to 18% for the evaluation). Since polynomials are independent of each other, the interpolation and evaluation can be trivially parallelized

for running with multiple threads, which would give this part of the computation a speed-up that is linear in the number of physical cores of the processor.

Acknowledgements. We thank Ben Riva and Udi Wieder for valuable discussions about this work. This work has been co-funded by the DFG within project E4 of the CRC CROSSING and by the BMBF and the HMWK within CRISP, by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Ministers Office, and by a grant from the Israel Science Foundation.

References

[ADN+13] Asokan, N., et al.: CrowdShare: secure mobile resource sharing. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 432–440. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_27

[ALSZ13] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS (2013)

[BP06] Boyar, J., Peralta, R.: Concrete multiplicative complexity of symmetric functions. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 179–189. Springer, Heidelberg (2006). https://doi.org/10.1007/11821069_16

[BPP00] Boyar, J., Peralta, R., Pochuev, D.: On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. TCS **235**(1), 43–57 (2000)

[CADT14] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. For your phone only: custom protocols for efficient secure functionevaluation on mobile devices. Secur. Commun. Netw. **7**(7) (2014)

[CO18] Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 464–482. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_25

[DC17] Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10343, pp. 261–278. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59870-3_15

[DD15] Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 209–226. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23318-5_12

[DGT12] De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 218–231. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35404-5_17

[DKT10] De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_13

[DSMRY09] Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_8

[DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)

[DT10] De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_13

[Dwo06] Dwork, C.: Differential privacy. In: ICALP (2006)

[EFLL12] Ejgenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: SCAPI: the secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629 (2012)

[FHNP16] Freedman, M.J., Hazay, C., Nissim, K., Pinkas, B.: Efficient set intersection with simulation-based security. J. Cryptol. **29**(1), 115–155 (2016)

[FIPR05] Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_17

[FNO18] Falk, B.H., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. Cryptology ePrint Archive, Report 2018/238 (2018)

[FNP04] Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_1

[GM11] Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious RAM simulation. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 576–587. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_46

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC (1987)

[Gon81] Gonnet, G.H.: Expected length of the longest probe sequence in hash code searching. J. ACM **28**(2), 289–304 (1981)

[HCE11] Huang, Y., Chapman, P., Evans, D.: Privacy-preserving applications on smartphones. In: Hot Topics in Security (HotSec) (2011)

[HEK12] Huang, Y., Evans, D., Katz, J.: Private set intersection: are garbled circuits better than custom protocols? In: NDSS (2012)

[HEKM11] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security (2011)

[HN10] Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 312–331. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_19

[HOS17] Hallgren, P., Orlandi, C., Sabelfeld, A.: PrivatePool: privacy-preserving ridesharing. In: Computer Security Foundations Symposium (CSF) (2017)

[IKN+17] Ion, M., et al.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738 (2017)

[IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_9

[KKRT16] Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS (2016)

[KLS+17] Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. PoPETs **2017**(4), 177–197 (2017)

[KM18] Kushilevitz, E., Mour, T.: Sub-logarithmic distributed oblivious RAM with small block size. CoRR, abs/1802.05145 (2018)

[KMP+17] Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: CCS (2017)

[KMW09] Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: cuckoo hashing with a stash. SIAM J. Comput. **39**(4), 1543–1561 (2009)

[Kre17] Kreuter, B.: Secure multiparty computation at Google. In: RWC (2017)

[Kre18] Kreuter, B.: Techniques for Scalable Secure Computation Systems. Ph.D. thesis, Northeastern University (2018)

[KS08] 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

[LWN+15] Liu, C., Wang, X. S., Nayak, K., Huang, Y., Shi, E.: ObliVM: a programming framework for secure computation. In: S&P (2015)

[Mea86] Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: S&P (1986)

[MR95] Motwani, R., Raghavan, P.: Randomized Algorithms (1995)

[PR01] Pagh, R., Rodler, F.F.: Cuckoo hashing. In: auf der Heide, F.M. (ed.) ESA 2001. LNCS, vol. 2161, pp. 121–133. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44676-1_10

[PSSZ15] Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: USENIX Security (2015)

[PSWW18] Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 125–157. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_5

[PSZ14] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security (2014)

[PSZ18] Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. TOPS **21**(2), 7 (2018)

[RA18] Resende, A.C.D., Aranha, D.F.: Faster unbalanced private set intersection. In: FC (2018)

[RR17a] Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 235–259. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_9

[RR17b] Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: CCS (2017)

[Sha80] Shamir, A.: On the power of commutativity in cryptography. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 582–595. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_100

[SZ13] Schneider, T., Zohner, M.: GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 275–292. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_23

[Yao86] Yao, A.C.: How to generate and exchange secrets. In: FOCS (1986)

[Yun15] Yung, M.: From mental poker to core business: why and how to deploy secure computation protocols? In: CCS (2015)

[ZC17] Zhao, Y., Chow, S.S.M.: Are you the one to share? Secret transfer with access structure. PoPETs **2017**(1), 149–169 (2017)

[ZC18] Zhao, Y., Chow, S.S.M.: Can you find the one for me? Privacy-preserving matchmaking via threshold PSI. In: WPES (2018)

[ZRE15] 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

C PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing (EuroS&P'19)

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E. S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing"**. In: *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 448–463. Appendix C.

https://doi.org/10.1109/EuroSP.2019.00040

2019 IEEE European Symposium on Security and Privacy (EuroS&P)

PILOT: Practical Privacy-Preserving Indoor Localization using OuTsourcing

Kimmo Järvinen
University of Helsinki
kimmo.u.jarvinen@helsinki.fi

Helena Leppäkoski
Tampere University
helena.leppakoski@tuni.fi

Elena-Simona Lohan
Tampere University
elena-simona.lohan@tuni.fi

Philipp Richter
Tampere University
philipp.richter@tuni.fi

Thomas Schneider
TU Darmstadt
schneider@encrypto.cs.tu-darmstadt.de

Oleksandr Tkachenko
TU Darmstadt
tkachenko@encrypto.cs.tu-darmstadt.de

Zheng Yang
Singapore University of Technology and Design
zheng_yang@sutd.edu.sg

***Abstract*—In the last decade, we observed a constantly growing number of Location-Based Services (LBSs) used in indoor environments, such as for targeted advertising in shopping malls or finding nearby friends. Although privacy-preserving LBSs were addressed in the literature, there was a lack of attention to the problem of enhancing privacy of indoor localization, i.e., the process of obtaining the users' locations indoors and, thus, a prerequisite for any indoor LBS.**

In this work we present PILOT, the first practically efficient solution for Privacy-Preserving Indoor Localization (PPIL) that was obtained by a synergy of the research areas indoor localization and applied cryptography. We design, implement, and evaluate protocols for Wi-Fi fingerprint-based PPIL that rely on 4 different distance metrics. To save energy and network bandwidth for the mobile end devices in PPIL, we securely outsource the computations to two non-colluding semi-honest parties. Our solution mixes different secure two-party computation protocols and we design size- and depth-optimized circuits for PPIL. We construct efficient circuit building blocks that are of independent interest: Single Instruction Multiple Data (SIMD) capable oblivious access to an array with low circuit depth and selection of the k-Nearest Neighbors with small circuit size. Additionally, we reduce Received Signal Strength (RSS) values from 8 bits to 4 bits without any significant accuracy reduction. Our most efficient PPIL protocol is 553x faster than that of Li et al. (INFOCOM'14) and 500x faster than that of Ziegeldorf et al. (WiSec'14). Our implementation on commodity hardware has practical run-times of less than 1 second even for the most accurate distance metrics that we consider, and it can process more than half a million PPIL queries per day.

I. Introduction

A Location-Based Service (LBS) is a service that relies on a user's physical location to provide the user with additional value. Pin-pointing the user's location on a map and offering routes to a given destination, navigation, targeted advertising, or finding nearby friends [1, 2, 3] are such add-on values that have already become a reality outdoors, but as well indoors. Privacy is a major concern in LBSs because the location history allows very accurate user profiling and even predicting users' future movements [4]. A lot of literature exists on privacy-preserving LBSs, e.g., [5, 6, 7, 8, 9]. Typically in these works, the user's location is assumed to be known by the user and the privacy concerns are about how this location information is shared and used in an LBS. In this paper, we focus on privacy issues of localization, i.e., of techniques for obtaining a user's location. We emphasize that privacy-preserving localization is the prerequisite for a privacy-preserving LBS because if privacy is violated already in the localization phase, then all further efforts for privacy protection are in vain.

Global Navigation Satellite System (GNSS) such as GPS or Galileo is the primary technology for localization in many LBSs [10]. Here, privacy of localization is not an issue because it is performed in a stand-alone GNSS receiver (e.g., in-car navigator) which calculates the location locally without any communication with other parties [11, 12]. Assisted or cloud GNSS receivers using cellular networks or the cloud for improving localization are exceptions where privacy of localization becomes an issue. Even though GNSS privacy is good, GNSS is unavailable or has poor services in indoor environments on which we focus in this work.

Thus, other means for localization must be used indoors. Proposed Indoor Localization (IL) techniques include Wi-Fi [13, 14, 15, 16], magnetic-field positioning, cellular, Bluetooth Low Energy (BLE) [17], and Ultra Wide Band (UWB) (see, e.g., [18, 19] for good surveys on indoor positioning technologies). Arguably, Wi-Fi fingerprinting is the most prominent and widely-adopted solution for IL [20, 21, 22, 23], as Wi-Fi networks and Wi-Fi capable devices are ubiquitous and therefore virtually free for the IL provider.

More recent Wi-Fi standards allow to measure additional parameters of the Wi-Fi signals such that more accurate propagation times [24] and angles of arrival [25, 26] can be obtained, enabling lateration or angulation localization methods. In multipath-free scenarios, where a signal can reach the user only once, these methods yield accuracies compareable with fingerprinting approaches. However, they do not preserve the users' privacy, require knowledge about the position of the Wi-Fi Access Points (APs), and the problems due to multipath propagation are not solved yet. Solutions that do not build upon Wi-Fi, such as cellular or UWB, either share these drawbacks or require additional network infrastructure. On the other hand, Wi-Fi fingerprinting is not limited to IL and can be used outdoors as well [27, 28].

DOI 10.1109/EuroSP.2019.00040

Wi-Fi fingerprinting relies on the spatial distribution of the Received Signal Strength (RSS) of Wi-Fi APs. A user reads the RSSs of nearby APs and sends them to a server that hosts a database of pre-measured sets of RSS signatures and associated positions, so called Reference Points (RPs), encoding the measurement locations. The server then calculates the distances between the user's RSS measurements and the RSSs in the database under some distance metric. After that, the server finds the user's location as the RP associated to the RSSs that yielded the smallest distance. While Wi-Fi fingerprinting provides good localization accuracy and practical feasibility (e.g., cheap installation costs thanks to the use of the existing Wi-Fi infrastructure), there is the prevailing problem of privacy: the server, who calculates the user's location, can track the user's movements inside the building covered by IL. Also, the server has incentives not to publish its database and, hence, calculating locations locally in the user's device is typically not an option either as described next.

A. Motivation

Privacy-Preserving Indoor Localization (PPIL) is a very topical problem given the constantly increasing use of IL [29, 30, 31] in environments including shopping malls [21], exhibition centers, airports, hospitals, university campuses [22, 32], and museums. Examples of deployed IL services (mobile apps) include AnyPlace[1], HERE Indoor Positioning[2], IndoorAtlas[3], and Point Inside[4]. At the same time, with the constantly growing use of IL also the IL accuracy is constantly increasing, with recent papers pointing out to cm-level accuracy [30, 33]. Al Altogether, this means that the threats to the users' privacy are significant and growing [4, 11, 12, 34].

In the following, we consider the use of IL for customers of a shopping mall as a motivating example for PPIL. The customers of the shopping mall would greatly benefit from the availability of IL because it would help them to easily find shops, restaurants and other places of interest in the mall, and would also allow tailored flash discounts for nearby customers benefiting both the customers and retailers. However, the customers' movements in the mall and visits to specific places may reveal personal details and preferences that they are reluctant to share with anyone. Hence, privacy issues may hinder the adaptation of IL among privacy-aware customers. A trivial solution for PPIL would be to let the customers download local copies of the database, but the service provider has many incentives to keep the database secret. The database is a result of laborious measurements. An accurate database can be constructed only by making several measurements with different devices in exact locations, which typically requires special equipment, and/or time-consuming crowdsourced collection of fingerprints. Hence, the database is a central business secret of the LBS. Furthermore, the database may reveal sensitive information about the infrastructure (at least approximate locations of APs) and it is also not trivial to update a distributed database. Consequently, a PPIL scheme should prevent the leakage of both the customers' location and the LBS's database. Additionally, a PPIL scheme should permit extending the PPIL with arbitrary privacy-preserving functionalities (e.g., with privacy-preserving location-based marketing) in order to maintain the service provider's financial motivation to provide the service to its customers. Finally, a PPIL scheme should be efficient enough for practical deployment for large-scale use. In this paper, we describe PILOT, a PPIL scheme that fulfills the above requirements.

[1] https://anyplace.cs.ucy.ac.cy

[2] https://www.here.com/en/products-services/products/here-positioning/here-indoor-positioning

[3] http://www.indooratlas.com

[4] https://www.pointinside.com

B. Our Contributions

We provide the following contributions in this paper:

The First Practical PPIL Scheme. Our main contribution is the design and implementation of PILOT, the first Privacy-Preserving Indoor Localization (PPIL) scheme that (i) preserves the privacy of users' locations, (ii) protects the server's database, (iii) can be extended with arbitrary privacy-preserving functionalities such as location-based advertising, and (iv) is efficient enough for large-scale use in practice. PILOT is not just engineering, but a new synergy between the research areas of indoor localization and applied cryptography that leads to the *first* practically efficient and secure solution for PPIL. This main contribution is based on the following specific contributions.

Outsourcing Scenario and Impracticality of the Client/Server Scenario for PPIL. We use an efficient outsourcing protocol for computing PPIL using two Semi-Trusted Third Parties (STTPs). This protocol is very efficient for mobile clients, because it (i) increases the communication of the mobile client only by about factor $2\times$ compared to the non-private protocol, and (ii) enables cheap and energy-efficient computations for the mobile clients. In contrast, we show that even optimized PPIL in the setting with a single server using Secure Two-Party Computation (STPC) currently requires several hundred megabytes of communication and cryptographic operations which makes it impractical for mobile clients.

Quantization of the Received Signal Strength Values. To decrease the overhead of STPC evaluation, we quantize the RSS values with reduced bit-length and show the impact on the IL accuracy. As a result of a field experiment, we can conclude that the reduction from 8 to 4 bits has very limited impact on the accuracy of IL (0.1–5 %), and to 2 and 1 bit still achieves meaningful accuracy.

Well-Suited Distance Metrics. We design, implement, and evaluate a variety of privacy-preserving algorithms for computing conventional and more advanced distance metrics. We consider the Manhattan, Euclidean, Sørensen, and Kumar-Hassebrook distances, which differ in accuracy and efficiency.

Improved Building Blocks. Some of our newly introduced circuit building blocks are generic and thus of independent interest. We develop a circuit for

finding k-Nearest Neighbors (k-NN) out of N elements with $Nk(2\ell + \lceil \log_2 N \rceil)$ AND gates, and a Single Instruction Multiple Data (SIMD) capable oblivious access to an N-element array with $\lceil \log_2 \lceil \log_2 N \rceil \rceil + 1$ multiplicative depth.

Comprehensive Performance Analysis. We provide detailed benchmarking results for our best-performing and most precise algorithms for ranges of Reference Points (RPs) and Wi-Fi Access Points (APs). In addition, we provide even more detailed benchmarks for the same practical setting as in [35] with M=505 RPs and N=241 APs. Our most efficient algorithm has an online run-time of 0.15 s on two commodity servers which is orders of magnitude faster than prior work. Furthermore, the servers can process up to half a million PPIL queries per day.

II. Related Work

Many proposed solutions for Privacy-Preserving Indoor Localization (PPIL) use Homomorphic Encryption (HE) such as the Paillier cryptosystem [36]. Generic shortcomings of HE-based PPIL schemes are that they rely on computationally demanding Public-Key Encryption (PKE), and that a user would require to generate a new keypair at least for each visit of the building in order to prevent possible information leakage. Ideally, he or she would need to generate a new keypair for each query. Consequently, key generation and any setup phase precomputation would cause not only a large computational overhead, but also storage problems because of the very limited storage capacity in mobile devices. In contrast, our solutions perform efficient precomputation of the setup phase by the Semi-Trusted Third Parties (STTPs), i.e., only the online phase must be evaluated in real-time, and relies on symmetric-key cryptography. Moreover, the PPIL server shares its database only once in the STTP initialization phase. The client transfers as little as about twice the amount of communication of the currently used non-private protocol, it is not involved in the precomputation, and it performs only plaintext operations.

Li et al. [37] design algorithms for Privacy-Preserving Indoor Localization (PPIL) based on the k-Nearest Neighbors (k-NN) algorithm relying on Paillier encryption. Their work has several shortcomings: Yang and Järvinen [35] very recently presented an attack against this system that reveals the server's database under a realistic attack model. Our most efficient Euclidean distance-based PPIL protocol performs 121× faster with respect to the total run-time and 553× faster in the online phase than the protocol of [37] (see §VI-E for details).

Shu et al. [38] make use of the Paillier cryptosystem and Oblivious Transfer (OT) to construct a localization protocol that involves other mobile devices knowing their locations for determining the user's location. Our protocol requires only access to the mobile Internet by the mobile client. In [38], the weak battery-powered mobile client performs computational heavy public-key crypto, whereas in our protocol it performs only very cheap operations (Pseudorandom Generator (PRG) evaluations and XORs).

In [39], Ziegeldorf et al. apply HE for computing Hidden Markov Model-based PPIL. Our most efficient Euclidean distance-based PPIL protocol has 100× faster total run-time and 500× faster online run-time than the protocols in [39], even though we benchmark on a real mobile client and a cellular network, whereas they benchmark on two servers connected via a Gigabit LAN (see §VI-E for details).

Konstantidis et al. [40] use *k*-anonymity [41] for PPIL. In PPIL, the *k*-anonymity approach guarantees that the client is hard to identify in a set of *k* clients, and it generally reduces the utility of the data by adding noise.

Zhang et al. [42] propose Support Vector Machine (SVM)-based solutions for PPIL using HE. This is, however, insecure because the client can easily reconstruct the server's model using specially crafted queries. For example, the attack on a linear SVM with the kernel function $f(\vec{x}) = \vec{w} \cdot \vec{x} + b$ can be conducted as follows: (i) determine b by sending all-zero $\vec{x}$ and (ii) determine each element of $\vec{w}$ by sending a unit vector with a single one entry for each of the M elements in $\vec{x}$ with all other elements being zero.

Yang and Järvinen [35] present four high-level proposals for implementing secure PPIL: fully homomorphic encryption, Garbled Circuits (GCs), Paillier encryption only, and Paillier encryption combined with GCs. They conclude that while some of them are clearly unpractical (especially, fully homomorphic encryption), at least the one combining Paillier encryption with GCs should be feasible in practice. Their best scheme would require more than 3 MB communication per query (i.e., more than 1 GB per hour with 1 query every 10 s) vs. less than 2 kB per query in our work (less than 720 kB per hour with 1 query every 10 s). Unfortunately, the paper does not provide enough details to make a meaningful run-time comparison with our protocols possible. However, even their most efficient proposal requires expensive Paillier encryption operations, for computing Euclidean distances under encryption, and evaluation of a GC for k-NN computation (62 thousand AND gates in our setting), which introduce significant computational overheads for the server and the mobile client [43], and we expect their protocol to have run-times in the order of tens of seconds. Although the computation of a GC can be outsourced [44, 45], which decreases the overhead of the mobile client, it requires an additional party for the computation and hence they would be in the same setting as we are.

To summarize, the existing literature still lacks a PPIL solution that is both secure and efficient enough for practical deployment.

III. Preliminaries

In this section, we describe the distance metrics used for fingerprint-based Indoor Localization (IL) and the application of the k-Nearest Neighbors (k-NN) algorithm for IL. Afterwards, we give a summary of Secure Multi-Party Computation (SMPC). Table I depicts notation that we use throughout the paper.

Table I: Notation used throughout this paper.

P_0, P_1	Parties performing secure computation
$t \in \{A, B, Y\}$	Type of sharing: **A**rithmetic, **B**oolean, or **Y**ao
$\langle s \rangle^t$	Share s in sharing t held by party P_i
$\langle z \rangle^t = \langle x \rangle^t \odot \langle y \rangle^t$	Operation $\odot$ on shares $\langle x \rangle^t$ and $\langle y \rangle^t$
$\langle x \rangle^s = s2t(\langle x \rangle^t)$	Sharing conversion from source s to target t
$\langle 0 \rangle$, $\langle 1 \rangle$, $\langle 2 \rangle$	Secret-shared constants 0, 1, and 2
$\langle \mathbf{F}(\cdot) \rangle$	Secret-shared output of a local function **F**
$\mathbf{GT}(\langle x \rangle, \langle y \rangle)$	Greater Than gate yielding 1 if $x > y$, and 0 otherwise
$\mathbf{MUX}(\langle x \rangle, \langle y \rangle, \langle s \rangle)$	Multiplexer gate yielding $\langle x \rangle$ if $s = 0$, and $\langle y \rangle$ otherwise
$x \oplus y$ and $x \wedge y$	Bit-wise XOR and AND operation
$L[i]$	i-th element in list L
κ	Symmetric security parameter; κ=128 in this work

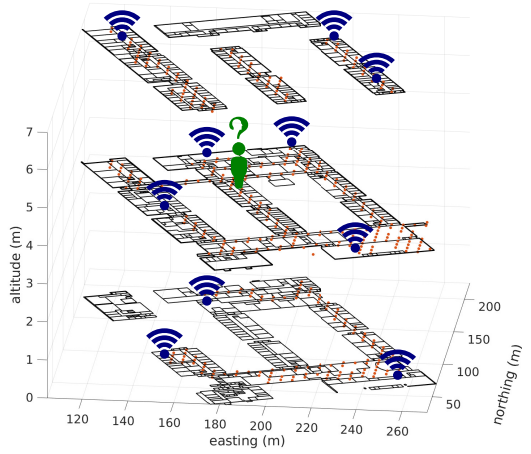

Figure 1: Exemplified setting for IL in a three-floor building with several Access Points (APs) (blue), pre-measured fingerprints at the Reference Points (RPs) (orange), and a client/user carrying a mobile device (green).

A. Indoor Localization

Location information already drives many services that are part of our daily routines. The spectrum of the services that rely on location information will broaden even further as the Internet of Things and 5G cellular networks emerge. The location information for these services is derived from the signal's propagation time, bearing or signal strength. We will focus here on Received Signal Strength (RSS)-based fingerprint localization, exemplified for Wi-Fi networks, as Wi-Fi fingerprinting is frequently used for IL due to its balance of complexity, availability of infrastructure, and accuracy [46]. However, our techniques can also be applied to fingerprint localization systems in other networks, e.g., cellular or Bluetooth networks. Fig. 1 presents a typical IL setting: a three-floor building with APs, RPs, and a user who wants to know his or her location.

Wi-Fi fingerprinting is a technique based on the spatial distribution of signal strengths. A location (RP) is assigned a signal strength signature, so-called fingerprint, that is ideally unique and therefore recognizable at a later point. It consists of the RSSs of the surrounding APs and identifiers relating the RSSs to their APs. To use Wi-Fi fingerprinting for localization, first, the service provider establishes an RSS map of the environment of interest. In a second step, a user records his or her RSS signature at a certain location, and the service provider can localize the user by comparing this RSS signature with the ones stored in the database holding the RSS map. The most likely location is the one whose fingerprint matches best the user's RSS signature under some distance metric, i.e., the Nearest Neighbor (NN) in RSS space.

Table II: Positioning performance of selected Received Signal Strength (RSS) distance metrics from [47] that are well-suited for Secure Multi-Party Computation (SMPC). The least accurate but most efficient metric is at the top, and the most accurate but least efficient metric is at the bottom.

Distance Metric	Success Rate (%)	Error (m)
Manhattan	90.73	7.06
Euclidean	92.71	7.40
Kumar-Hassebrook	94.33	7.00
Sørensen	94.78	6.86

The comparison of the user's RSS signature and the RSS signatures in the RSS map is done with distance measures which we detail below. The notation x_i and y_i represents the RSS of the i-th AP found in the RSS map and the one measured by the user, respectively. N is the number of APs. A comparison of the success and error rates using the different metrics is given in Tab. II. Success rate is the percentage of test points for which the building and the floor were estimated correctly. The error is the two-dimensional, horizontal distance between the estimated position and the ground truth position of test points; it was only computed for test points whose building and floor was estimated correctly.

1) Manhattan Distance: The Manhattan (also City Block) distance is a distance metric where only horizontal and vertical paths are allowed. It is calculated as shown in Eq. 1.

$$d_m = \sum_{i=1}^{N} |x_i - y_i| \tag{1}$$

2) Euclidean Distance: The Euclidean distance is the diagonal distance between two points. For computing the diagonal between two points, we apply the Pythagorean theorem to the distances between particular dimensions as shown in Eq. 2.

$$d_e = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{2}$$

3) Sørensen Distance: The Sørensen distance is based on the Manhattan distance, which is normalized by the sum of the values. The calculation of this metric is shown in Eq. 3.

$$d_s = \frac{\sum_{i=1}^{N} |x_i - y_i|}{\sum_{i=1}^{N} (x_i + y_i)} \tag{3}$$

According to [47], the Sørensen Distance outperforms most other metrics for IL in terms of accuracy, and in particular the two aforementioned metrics (see Tab. II).

4) Kumar-Hassebrook Distance: The Kumar-Hassebrook distance is almost as accurate as the Sørensen distance, and it requires similar operations. It is computed as shown in Eq. 4.

$$d_{kh} = \frac{\sum_{i=1}^{N}(x_i \cdot y_i)}{\sum_{i=1}^{N} x_i^2 + \sum_{i=1}^{N} y_i^2 - \sum_{i=1}^{N}(x_i \cdot y_i)} \quad (4)$$

This distance metric performs better than the Euclidean and Manhattan distance, but marginally worse than the Sørensen distance (see Tab. II). However, as we will show in §V-B4, the Kumar-Hassebrook distance can be computed more efficiently in general-purpose SMPC than the Sørensen distance.

Choice of a distance metric: In this work, we aim at designing, implementing, and benchmarking a practically efficient PPIL system and PPIL protocols for different distance metrics. We select the distance metrics that either can be implemented very efficiently using SMPC or are one of the most accurate among others (see Tab. II) but can still be efficiently implemented. Based on the benchmarks, we show and discuss the trade-off between the more accurate but less efficient metrics and more efficient but less accurate metrics. However, the choice of the best-suited metric for a specific application is out of the scope of this paper.

Indoor Localization using k-Nearest Neighbors: To increase the accuracy of IL, the k-Nearest Neighbors (k-NN) algorithm can be used. In that case, not only one entry of the database is retrieved, but instead the positions corresponding to the k smallest RSS distances are averaged to estimate the user location. The value k is chosen to yield good performance [48]. Typical values are k=3 or k=4 [49, 50].

B. Secure Multi-Party Computation

A Secure Multi-Party Computation (SMPC) protocol enables n parties, each of whom holding a secret input, to compute a public function on their inputs without revealing anything except the result. In the following, we briefly review the main cryptographic tools and implementation of SMPC used in our solutions.

ABY Framework [51]: Among the variety of Secure Multi-Party Computation (SMPC) frameworks, e.g., [52, 51, 53, 54], we choose the ABY framework [51] for its efficiency and the ability to mix multiple SMPC protocols. ABY is a state-of-the-art framework for Secure Two-Party Computation (STPC). It supports three sharing types with state-of-the-art optimizations: Yao sharing based on Yao's Garbled Circuits (GCs) [55], Boolean sharing based on the GMW protocol [56], and Arithmetic sharing based on Arithmetic GMW. ABY guarantees passive security which means that an adversary is assumed to follow the protocol, but he or she may try to learn additional information from the received messages.

The STPC protocols have different benefits and drawbacks. GCs require a constant number of communication rounds, but need substantial communication and computation per AND gate. In low-latency networks, securely evaluating circuits with low depth using the GMW protocol is often faster than using GCs, but requires multiple rounds of interaction (one message for each layer of AND gates in the circuit). Arithmetic GMW allows computation of addition and subtraction without any interaction, and multiplication utilizes one Multiplication Triple (MT) [57] which can be efficiently precomputed using Oblivious Transfer (OT) extensions [58, 59] as shown in [51]. The drawback of Arithmetic GMW is that these are the only operations that can be computed efficiently.

The ABY framework allows to efficiently convert between these different types of sharing and benefit from their respective advantages. Moreover, it supports outsourcing of the computation to Semi-Trusted Third Parties (STTPs) and Single Instruction Multiple Data (SIMD) gates [60]. In principle, ABY also allows to compute very complex distance metrics that require logarithm or exponentiation operations using IEEE 754 Floating Point numbers [61], but this will slow down the protocols significantly and thus will not be considered in this paper.

In this work, we use Arithmetic sharing to improve the efficiency of distance computation, Boolean sharing to compute the depth-optimized Bitonic sort (for finding k nearest RPs), and Yao sharing to compute the size-optimized k-NN algorithm (instead of Bitonic sort). The choice between Boolean and Yao sharing depends on the network latency and the value of nearest neighbors k (see §VI).

Arithmetic Sharing: Arithmetic sharing is a protocol that enables two parties to secret-share an ℓ-bit integer x, and to evaluate a function which is expressed as an Arithmetic circuit consisting of addition and multiplication gates based on secret-shared values. Namely, for an integer x, each party has a random share $\langle x \rangle_i^A$, $i \in \{0,1\}$, such that $\langle x \rangle^A = \langle x \rangle_0^A + \langle x \rangle_1^A \pmod{2^\ell}$ and multiplication using the OT extension-based multiplication protocol of [51] requires O(ℓ) bits communication and one round of interaction, whereas addition and subtraction can be evaluated locally. However, XOR and AND operation are expensive in Arithmetic sharing as they would require expensive bit decomposition.

Boolean Sharing: In [56], Goldreich, Micali, and Wigderson (GMW) introduced a protocol which allows multiple parties to securely evaluate a function f expressed as Boolean circuit consisting of XOR and AND gates. We use the two-party variant of GMW, where each input and intermediate wire is shared by both parties with a 2-out-of-2 secret sharing scheme, i.e., each party P_i, $i \in \{0,1\}$, holds a Boolean share $\langle x \rangle_i^B$ for a secret-shared bit $\langle x \rangle^B$ such that $\langle x \rangle^B = \langle x \rangle_0^B \oplus \langle x \rangle_1^B$. While a XOR gate can be evaluated locally, an AND gate requires 2κ bits of communication in the setup phase and 4 bits in the online phase, as described in [59]. The communication of GMW can be further reduced at the cost of a higher computation complexity [62, 63].

Yao Sharing: Another approach for securely evaluating a Boolean circuit between two parties are Yao's Garbled Circuits (GCs) [55]. A GC can be generated as follows: the sender encrypts the Boolean gates of the circuit using randomly chosen symmetric keys for the wires and sends the encrypted function, called garbled circuit, together

with the keys representing its input bits to the receiver. The receiver obliviously obtains the keys corresponding to its inputs by running an Oblivious Transfer (OT) protocol with the sender which can be efficiently implemented using OT extension [58, 59]. Now, the receiver can evaluate the garbled circuit gate-by-gate using the keys and obtains the result. ABY implements state-of-the-art GC improvements such as point-and-permute [64], free-XOR [65], fixed-key AES garbling [66], and half-gates [67]. With this, XOR gates are free, whereas an AND gate costs 2κ bits of communication in the setup phase and no communication in the online phase, i.e., no additional rounds of communication.

Conversions: The cost of one Arithmetic to Yao (A2Y) or Arithmetic to Boolean (A2B) conversion is $6\ell\kappa$ bits of communication within 2 messages and 12ℓ AES operations, where ℓ is the bit-length of the Arithmetic share [51].

Hardware-Enhanced Secure Multi-Party Computation: Several works propose to use additional assumptions in form of trusted hardware to improve generic SMPC which could be used to improve PPIL schemes on mobile devices. Demmler et al. [68] use a smartcard in a mobile device for locally generating correlated randomness for the GMW protocol. Gupta et al. [69] design STPC protocols that use Intel SGX[5] for improving the efficiency of STPC. However, all these protocols require substantial trust in the hardware manufacturer.

IV. System Details

In this section, we describe our system PILOT for Privacy-Preserving Indoor Localization (PPIL) using outsourcing and detail the underlying security assumptions.

A. Outsourcing Scheme

Motivation: The best known solutions to PPIL without the non-colluding STTPs, i.e., PPIL in the client/single server setting, are very communication intensive. It is important to note that by using the outsourcing setting we achieve practical PPIL with security against malicious PPIL clients and malicious PPIL servers, whereas in the client/server setting even solutions with substantially weaker security against a semi-honest client currently have impractical runtimes and communication. Furthermore, the outsourcing model with multiple STTPs is widely adopted not only in recent academic papers, e.g., for private machine learning [70, 71], and genomic privacy [72, 73, 74, 75, 76, 77], to name just a few, but also deployed in industrial products, see [78] for a few examples.

Implementation: The outsourcing scheme we use is depicted in Fig. 2. The scenario involves three parties: a client who is interested in finding his or her coordinates, and two non-colluding Semi-Trusted Third Parties (STTPs), denoted as T_0 and T_1, that obliviously compute coordinates of the client using STPC. An additional entity in our scenario is the PPIL server (IL provider), which is not depicted in the figure because its overhead is one-time, and it is involved in the initialization of the STTPs only. On the initialization of the system or whenever updates occur (which in practice happens rarely), the PPIL server secret-shares its database of $N \cdot M$ Received Signal Strength (RSS) values between T_0 and T_1, where N is the number of APs and M is the number of RPs.

[5]https://software.intel.com/en-us/sgx

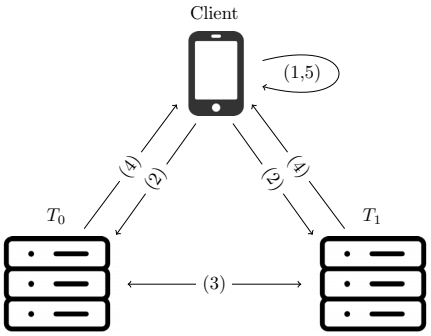

Figure 2: System model of our outsourcing scenario for efficient Privacy-Preserving Indoor Localization (PPIL). The PPIL server is not depicted in the figure, because it shares its database with the Semi-Trusted Third Parties T_0 and T_1 only once for the initialization of the protocol.

The STTPs are assumed to not collude. The assumption for STTPs not colluding is supported also by the facts that (i) a privacy-preserving service is a selling-point by itself to privacy-aware customers and getting caught from colluding would seriously damage reputation, and (ii) modern legislation, e.g., the EU General Data Protection Regulation (GDPR), emphasizes users' privacy and services have an obligation to protect their customers' data.

The PPIL works as follows (see Fig. 2):

1) The client collects RSS data from the APs (predefined by the PPIL server) to create a list $(\text{RSS}_1, \ldots, \text{RSS}_N)$, and secret-shares these values into $(\langle \text{RSS}_1 \rangle, \ldots, \langle \text{RSS}_N \rangle)$.
2) The client sends $(\langle \text{RSS}_1 \rangle_0, \ldots, \langle \text{RSS}_N \rangle_0)$ to T_0 and $(\langle \text{RSS}_1 \rangle_1, \ldots, \langle \text{RSS}_N \rangle_1)$ to T_1 over two secure channels.
3) The STTPs execute a PPIL protocol using STPC. As the result of the computation, they get the secret-shared coordinate $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$.
4) T_0 sends $(\langle x \rangle_0, \langle y \rangle_0, \langle z \rangle_0)$ to the client and T_1 sends $(\langle x \rangle_1, \langle y \rangle_1, \langle z \rangle_1)$ to the client over two secure channels.
5) The client recombines the shares and reconstructs the cleartext coordinate (x, y, z).

B. Deployment

In a practical deployment, the parties in our system can be run by the following entities:

Client can be run by any visitor of the building on his/her mobile device.

Each of the STTPs can be operated by any of the following entities: the building owner, the mobile network operator, or a cloud provider.

PPIL server can be run by the mall or building owner. Real-world examples of IL providers are given in §I-A.

For the number of STTPs, we see three possible real-world settings: one, two, and three STTPs.

One STTP has the advantage that it requires only one server and one entity to operate the STTP. However, the computational overhead is substantial (cf. §II and §VI-E), and here the single server learns the database in the clear.

Two STTPs is the best model for real-world applications regarding the costs and security, since it is substantially more efficient than the model with one STTP, and the support and maintenance costs remain moderate compared to the setting with three STTPs. Furthermore, the STTPs obtain only a secret-shared database which preserves its privacy. Due to this, we choose the two-STTP setting in this paper.

Three STTPs can be more efficient than both previous models in an honest majority setting where at most one party can be corrupted, e.g., using the protocols of [79, 80]. However, it (i) guarantees a weaker level of security (it is easier to corrupt any two out of three parties than two out of two), and (ii) has much higher costs for the support and maintenance (to mitigate attacks, each STTP would need a different software stack and its own administration team).

C. Security Assumptions

In this work, we concentrate on a realistic scenario where the STTPs can only be passively corrupted, and the client and PPIL server can be actively corrupted. The latter is due to the fact that an actively corrupted PPIL server can only affect the correctness of the protocol by changing its inputs, and an actively corrupted client can only change the single message sent which corresponds to choosing a different input to the ideal functionality. The two STTPs have no inputs to the protocol (the inputs are already shared by the PPIL server and client) and thus can only try to learn more information from the protocol. Using standard techniques for outsourcing with active security [81, 82, 83], our protocols can be extended to full active security at the expense of higher communication and run-times. In this case, the system will detect one actively corrupted STTP, which can arbitrarily deviate from the protocol.

In order to guarantee privacy of the PPIL users, we make the following assumptions:

- When a device connects to a Wi-Fi Access Point (AP), it transmits to the AP its global Media Access Control (MAC) address, which is a unique, persistent identifier. Therefore, the client should be aware that there exists the possibility of being tracked by the AP owner, who could collude with the PPIL server or even be the same entity, e.g., the building owner provides a PPIL service and free Wi-Fi in the building. The use of such an AP would allow its owner to track the potential presence of the user in particular parts of the building. To mitigate this, we propose to use mobile Internet for PPIL, since even when a Wi-Fi enabled device is not connected to other Wi-Fi devices, it periodically scans its environment by sending probe requests. These requests also include the sender's MAC address and can be used to track the user [84]. However, in these requests the device is allowed to use a local MAC instead of its global MAC. Therefore, the client must use a randomized, changing local MAC address in the probe requests. This became possible in iOS version 8 and in Android version 6 [85].[6] One can root older Android devices or jailbreak older iOS devices that do not support randomized MACs by default, which enables randomized MACs and monitor mode.
- The STTPs must be chosen s.t. there is a strong incentive to not collude. One possibility is that the building operator operates T_0 and the user's mobile Internet provider operates T_1.
- For preventing an attacker from restoring the secret-shared inputs or coordinates by intercepting both shares, secure channels between the client and the STTPs must be used. However, a persistent TLS channel would leak information about the client. We avoid this potential leakage by establishing a new TLS connection for each new dynamically assigned IP address of the client. If the client's IP address is hidden behind a Network Address Translation (NAT) router getting a new network port for each query, he or she could establish a new TLS channel for each PPIL query to prevent tracking of his or her session.

D. Security Analysis

Here, we briefly analyze the security of our outsourcing scheme based on the SMPC model defined in [88] with concrete participants, i.e., client, PPIL server, and two outsourcing parties: STTPs T_0 and T_1. Kamara and Raykova [88] define and prove secure a generic construction that turns an n-party SMPC protocol into a protocol, where computation is outsourced to n non-colluding outsourcing parties. For our PPIL protocol, we instantiate the outsourcing scheme of [88] with the secure two-party (n=2) ABY framework. Our PPIL scheme is secure against semi-honest non-colluding STTPs, malicious clients, and malicious PPIL servers. Our threat model does not cover malicious Received Signal Strength (RSS) values and the leakage from the protocol output (i.e., client's coordinate), but in practice it would be hard to learn the PPIL server's database by fabricating RSS values.

Theorem IV.1. Suppose that the SMPC building blocks implemented in ABY [51] are secure, then our PPIL scheme is secure against passively corrupted non-colluding STTPs, and malicious clients and malicious PPIL servers.

Proof (sketch). Each secret-share $\langle\phi\rangle_i^t$ for $i \in \{0,1\}$ and some value ϕ is blinded with a one-time random value (determined by the sharing type t of ABY). Namely, each uncompromised secret-share is statistically close to a truly

[6] Recent research, e.g., [86], have shown that MAC randomization is vulnerable to different types of attacks. As additional privacy measure, the client could set its Wi-Fi card into monitor mode which disables the probe requests but allows listening to the beacons from APs and measuring their RSS. In the current Android implementations, the Wi-Fi monitor mode requires rooting the Android device and installing the required application packages [87].

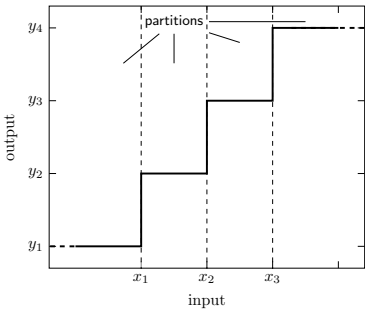

Figure 3: Uniform 2-bit quantizer showing the partitions.

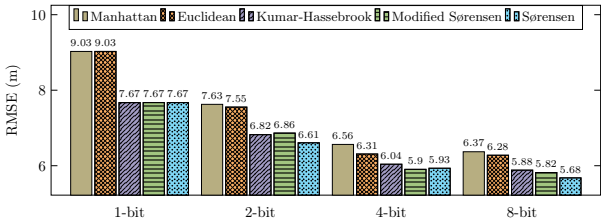

Figure 4: Positioning Root Mean Square Error (RMSE) in meters of k-NN for 1-, 2-, 4-, and 8-bit quantization of Received Signal Strength and different distance metrics obtained in a field experiment.

random value. Hence, a corrupted party obtaining a secret-share from its honest partner cannot obtain any useful information from it with overwhelming probability. Since two non-colluding STTPs cannot be both corrupted by definition, they will not collude in the security experiment. In this sense, at least one of the inputs of the subsequent STPC protocol is from an uncorrupted STTP (which received it from the corresponding uncorrupted party). Thus, we can reduce the security of our protocol to that of the STPC protocol (executed between STTPs) that uses the secure building blocks from the ABY framework. Namely, the security of the ABY-based STPC protocol ensures that the intermediate secret-shares (i.e., the outcome of any operation between secret shares), share conversions, and the final target location coordinate shares are secure as well. If the client and at least one STTP are not corrupted, then no party except the client can get the final location since the location coordinate shares are only reconstructed by the client. Analogously, if the PPIL server and at least one STTP are not corrupted, all secret shares leak no information about the RSSs of the server's database. PPIL server's malicious input can only affect the correctness of the protocol. Each malicious input from the client corresponds to a different input to the ideal functionality.

V. Privacy-Preserving Indoor Localization

In this section, we show our practical protocols for Privacy-Preserving Indoor Localization (PPIL) and optimizations.

A. Quantization of Received Signal Strength

We rely on the concept of a uniform quantizer to quantize the Received Signal Strength (RSS) values. A uniform quantizer is characterized by equally spaced boundary points [89] forming 2^ℓ non-overlapping partitions, where ℓ is the number of available bits. The midpoints of these partitions (except the two outermost partitions) correspond to the output levels, such that an input to the quantizer that lies within a partition is rounded off to the partition's midpoint. Each output level can be represented arbitrarily according to the application. Fig. 3 illustrates a uniform quantizer for ℓ=2.

To construct the quantizer for the RSSs, we specify the midpoints of the partitions by dividing the range between the lower and upper limit of the RSS values (of all fingerprints) by $2^\ell - 1$. To allow the localization algorithm to benefit from the information about the approximate coverage areas of the APs, we want to distinguish the cases when a signal is received with low RSS from the cases when a signal is not received at all. Therefore, we reserve one quantization partition for recording these signal-not-received cases and use this special partition during both the collection of the fingerprint database and the positioning phase. The boundary points for the RSS-quantizer are given by the midpoints plus half the size of a partition. The reserved partition for the zero-bit is added to the uniform quantizer, making it actually non-uniform. Its midpoint is chosen to be smaller than the lower RSS limit. For the purpose of RSS-based Indoor Localization (IL) via fingerprinting it suffices to represent the output levels of the quantizer simply by ℓ-bit codewords. The fingerprints of the Reference Points (RPs) can be quantized off-line on the server side. The fingerprints measured by the user must be quantized on the user device, thus, the quantizer must be made available for the clients.

Localization Performance: We compare the positioning accuracy, in terms of Root Mean Square Error (RMSE), of the k-NN algorithm for the four distance metrics specified in §III and consider additionally the modified Sørensen metric (see Eq. (5) in §V-B3). The data for that experiment was collected in a three-floor building at Tampere University of Technology, which is shown in Fig. 1. The corresponding database consists of M=446 RPs and N=449 APs; further details and the database itself are available at [90].

Fig. 4 shows the RMSE (averaged over a path through the building) for these four distances for several bit-lengths ℓ. The number of neighbors of the k-Nearest Neighbors (k-NN) was set to k=3. As shown in the figure, the accuracy of 4-bit quantization is similar to that of 8-bit with only 0.1–5 % loss of accuracy. Even with 1-bit quantization, a positioning accuracy of about 7–9 meters is achievable. This confirms the benefit of the information about the approximate coverage of the APs. The positioning error reduces with the number of bits spent to quantize the RSS, as expected. The Manhattan distance shows the poorest accuracy, whereas the (Modified) Sørensen distance is the most accurate one. We further investigate the effects of RSS quantization in [91].

Table III: AND-sizes and minimum bit-length of the circuits for the distance algorithms with N=241 Access Points, bit-length of Received Signal Strength ℓ, and the scaling factor of γ=16 bits.

Distance metric	Naive: Boolean / Yao Sharing	Optimized: Arithmetic, Boolean, Yao Sharing, OT				Min. bit-length	Min. bit-length with fixed ℓ			
	AND-size	AND-size	A_*	A2B / A2Y	C-OTs		ℓ=1	ℓ=2	ℓ=4	ℓ=8
Manhattan	$4(N \cdot \ell) - \ell$	$N \cdot (3\ell - \lceil \log_2 \ell \rceil - 2)$	0	0	$2N$	$\log_2 N + \ell$	8.9	9.9	11.9	15.9
Euclidean[7]	$N(2\ell^2 + 2\ell)$	0	N	0	0	$\log_2 N + 2\ell$	9.9	11.9	15.9	23.9
Sørensen	$6(N \cdot \ell) - 2\ell + 7\,079$	$N \cdot (3\ell - \lceil \log_2 \ell \rceil - 2) + 7\,079$	0	2	$2N$	$\log_2 N + \ell + \gamma$	24.9	25.9	27.9	31.9
Modified Sørensen	$N(2\ell^2 + 3\ell) + 7\,079$	7 079	N	2	0	$\log_2 N + 2\ell + \gamma$	25.9	27.9	31.9	39.9
Kumar-Hassebrook	$2N(3\ell^2 - \ell) + 2\ell + 7\,079$	7 079	N	2	0	$\log_2 N + 2\ell + \gamma$	25.9	27.9	31.9	39.9

B. Secure Multi-Party Computation of Distance Metrics

In this section, we describe in detail our privacy-preserving evaluation of the aforementioned distance metrics using the ABY framework [51]. These can be split into four categories: pure Arithmetic sharing (A), pure Boolean sharing (B), pure Yao sharing (Y), and mixed sharing (AB, AY, ABOY where O denotes OT) implementations. The latter use Arithmetic sharing first and then convert into Boolean or Yao sharing, respectively. In the following, we give the complexities for computing one distance, but for Privacy-Preserving Indoor Localization (PPIL), M instances need to be computed in parallel, one for each of the M Reference Points (RPs).

For Yao sharing we use size-optimized circuits of [92], and for Boolean sharing we use depth-optimized circuits of [60], as implemented in ABY [51], see [60, Tab. 7] for the complexities of the building blocks. We give an overview of size and depth complexities of the described algorithms as well as the minimum bit-length of the output to guarantee the correctness in Tab. III. In all our algorithms for securely computing distance metrics, we use sufficient bit-lengths which guarantees that the accuracy of the metrics does not decrease when compared with the cleartext protocols (denoted as minimum bit-length in Tab. III).

1) Manhattan Distance (cf. Eq. 1 on p. 4): The Manhattan distance requires computation of absolute values. This operation is nonlinear and hence cannot be efficiently computed in Arithmetic sharing. This is why we propose two workarounds to mitigate this issue: (i) enhancing the "naive" computation of the Manhattan distance without the loss of predicting accuracy by using Correlated OT (C-OT)-based multiplication of Arithmetic shares by a secret-shared bit (inspired by [93, 51]), and (ii) reducing the bit-length of the RSSs to 1 bit, which allows for the use of more efficient building blocks while guaranteeing accuracy acceptable in many scenarios.

To compute the Manhattan distance in a "naive" way, we use the size-optimized circuit of [92] that requires 1 Greater Than (GT), 2 Multiplexer (MUX), and 1 Subtraction (SUB) circuit which results in $4N\ell - \ell$ AND gates, where N is the number of APs and ℓ is the bit-length of the RSS values.

In our first improvement, we use very efficient C-OT-based multiplication of secret-shared values to choose either the initial result of $\langle \text{RSS}_C \rangle^A - \langle \text{RSS}_S \rangle^A$ or its two's complement [94] based on the outcome of the comparison of both in Boolean sharing (the comparison can also be performed by using the underflow bit of the result, but this also incurs some overhead because of the conversion to Boolean sharing). This does not violate the security, since the multiplication protocol is performed obliviously on random-looking secret-shared values and yields random-looking secret-shared values as the result. A similar technique was used, for example, in the ABY framework for precomputing MTs [51]. This protocol is not only much more efficient than doing this using the "naive" solution, but also allows to sum up the distances of all dimensions locally, since the computation yields Arithmetic shares for each dimension. This protocol requires only 1 Greater Than (GT) gate and 2 C-OTs for each dimensions resulting in $N(3\ell - \lceil \log_2 \ell \rceil - 2)$ AND gates and $2N$ parallel C-OTs of ℓ-bit strings. We depict our C-OT-based implementation of the Manhattan distance in Alg. 1 in §A.

In our second improvement, which we denote as 1-bit Manhattan distance, we use 1-bit RSS values and compute the distance in pure Yao sharing. Since the Manhattan distance of 1-bit values corresponds to their XOR, we perform the computation non-interactively. To compute the sum of the computed distances, we utilize the AND-optimized Hamming Weight (HW) circuit of Boyar and Peralta [95] with $\ell - d_H(\ell)$ AND gates, where $d_H(\ell)$ denotes the HW of ℓ. This protocol comes at the price of accuracy reduction, which, however, remains meaningful for many scenarios, i.e., 9 m RMSE in our experiments (see Fig. 4). By replacing the HW circuit with the custom C-OT-based protocol of [96] for securely computing an Arithmetic sharing of the HW, the communication of this protocol can be improved by a factor of 3.7× from 5.2 MB to 1.4 MB. Overall, this improves communication over the naive approach by factor 215×.

2) Euclidean Distance (cf. Eq. 2 on p. 4): Since computing the square root is very expensive in SMPC, we replace the Euclidean distance with its squared version, which only omits the last operation of computing the square root. Note that this does not affect the outcome of finding the nearest RPs because all distances are squared. The Boolean circuit-based Euclidean distance requires $N(2\ell^2 + 2\ell)$ AND gates, whereas the Arithmetic sharing optimized algorithm can completely be computed using Arithmetic sharing only. This algorithm utilizes N multiplication gates in Arithmetic sharing. An important difference to the other securely computed distance metrics is that the result of the computation has to be converted to another sharing type for efficient computation of k nearest RPs, which for other metrics either happens during the distance computation (if required) or does not happen at all (if not required). We give our Arithmetic sharing-based implementation of the Euclidean distance in Alg. 2 in §A.

3) Sørensen Distance (cf. Eq. 3 on p. 4): The Boolean circuit-based Sørensen Distance can be computed using $6(N \cdot \ell) - 2\ell + 7\,079$ AND gates. To reduce the overhead for computing the Manhattan distance in the numerator, we utilize the idea of the C-OT-based Manhattan distance protocol. Since the addition of the values in the denominator is performed on the Arithmetic shares, the difference in the interactive operations to the Manhattance distance protocol comes from the 32-bit division operation, which costs 7079 AND gates.

Modified Sørensen Distance: Since the computation of absolute values cannot be efficiently done in Arithmetic sharing, we introduce a more SMPC-friendly variant of the Sørensen Distance where we replace the Manhattan distance by the Euclidean distance (cf. Eq. 5) and evaluate its IL accuracy. We denote this distance metric as modified Sørensen distance.

$$d'_s = \frac{\sum_{i=1}^{D}(x_i - y_i)^2}{\sum_{i=1}^{D}(x_i + y_i)} \tag{5}$$

The Root Mean Square Error (RMSE) of the modified Sørensen distance is depicted in Fig. 4, and in particular settings this metric is even more accurate than the original Sørensen distance. The Boolean circuit-based Modified Sørensen distance can be computed using $N(2\ell^2+3\ell)+7\,079$ AND gates. As a result, this protocol requires 7 079 AND gates, N multiplication gates in Arithmetic sharing, and 2 A2B or A2Y conversions, respectively.

In Alg. 4 in §A, we show the AY implementation of the modified Sørensen distance. We use a γ=16-bit scaling factor for both Sørensen and Kumar-Hassebrook distance to achieve a precision comparable to that of floating point arithmetic, i.e., we perform a bit-shift on the distance by γ bits before dividing it by the denominator.

4) Kumar-Hassebrook Distance (cf. Eq. 4 on p. 5): The Kumar-Hassebrook distance, unlike the Sørensen distance, requires multiplications in the numerator. Therefore, its Boolean circuit-based algorithm requires $2N(3\ell^2 - \ell) + 2\ell + 7\,079$ AND gates. However, it can naturally be optimized in Arithmetic sharing. The complexities of the optimized algorithm are as follows: 7 079 AND gates, N multiplication gates in Arithmetic sharing, and two A2B or A2Y conversions, respectively. The implementation of the optimized Kumar-Hassebrook distance is shown in Alg. 3 in §A.

C. k-Nearest Neighbors

Songhori et al. [97] propose a circuit for k-Nearest Neighbors (k-NN) with size $M(2k-1)(2\ell+\lceil\log_2 M\rceil)$ AND gates, where M is the number of elements (here, the number of Reference Points) and ℓ is their bit-length.

Optimized k-Nearest Neighbors: We improve the k-NN circuit of [97] by using more efficient conditional swap circuits of [65]. This results in $Mk(2\ell + \lceil\log_2 M\rceil)$ AND gates which for k=3 is an improvement by factor $1.6\times$ over [97]. Our modified k-NN implementation is given in Alg. 6 in §A.

k-Nearest Neighbors via Oblivious Sorting: An alternative approach for computing k-NN is to use Oblivious Sorting. This improves the efficiency of the non-constant round SMPC protocols such as GMW because the k-NN algorithm of [97] has high depth of $\mathrm{O}(Mk\ell)$. Instead, we use the Bitonic Sort algorithm of [98] that is summarized in Alg. 5 in §A. As a baseline for our implementation, we use the algorithm of [99]. The algorithm requires $0.5 \cdot \lceil\log_2 M\rceil(\lceil\log_2 M\rceil + 1)$ stages with $M(4\ell-\lceil\log_2 \ell\rceil-2+\lceil\log_2 M\rceil)$ AND gates each. This is by factor $\sim$12x larger than for our optimized k-NN algorithm for M=505 and k=3. However, this algorithm produces a Boolean circuit that is by factor $\sim$10x shallower than that of our improved k-NN already for k=3, and its AND-depth does not grow with k.

D. Oblivious Array Access

In order to find the user's coordinates in Privacy-Preserving Indoor Localization (PPIL), we have to find the coordinates that correspond to the nearest Reference Points (RPs) by their secret-shared indices computed by the k-NN algorithm. We use an approach similar to the one proposed by Keller and Scholl [100] for our size-optimized implementation, and design a new Single Instruction Multiple Data (SIMD) capable circuit for the depth-optimized implementation with the same AND-depth and slightly larger AND-size than the circuit proposed by Buescher et al. [101] (the latter cannot be parallelized easily with SIMD).

The size-optimized circuit is built as an inverted binary tree, i.e., initial elements represent leaves, and the selected element represents the root. Each intermediate node between the root and the leaves is a MUX-gate which has two elements and a choice bit as input. The tree has $\lceil\log_2 M\rceil$ depth, where M is the length of the array, and all MUX-gates on the layer i have the same choice bit $\langle idx\rangle^b[i]$, i.e., the i-th bit of the secret-shared index $\langle idx\rangle^b$ with $b \in \{B, Y\}$. This requires $M-1$ MUX-gates which yields $\ell(M-1)$ AND gates (8 064 for M=505 and ℓ=16), where ℓ is the bit-length of the coordinate. The AND-depth is hereby $\lceil\log_2 M\rceil$ (9 for M=505). The size-optimized circuit is depicted in Alg. 7 in §A.

The depth-optimized circuit is designed as follows: the server provides a list l which contains the secret-shared coordinates corresponding to the APs. For each index $\langle idx\rangle^b$ which corresponds to one of the k nearest APs, we compare $\langle idx\rangle^b$ with $(\langle 1\rangle^b, \ldots, \langle M\rangle^b)$ using a comparison circuit. This yields a list of secret-shared zeros with a single secret-shared one entry. Afterwards, we apply a MUX gate to each set of coordinates which results in all coordinates being zeros except of exactly one set. To get the final result, we perform the XOR operation on all the coordinates. This circuit has AND-depth $\lceil\log_2\lceil\log_2 M\rceil\rceil + 1$ (5 for M=505, 44 % shallower than the size-optimized) and $M(\lceil\log_2 M\rceil - 1 + \ell)$ AND-size (12 120 for M=505 and ℓ=16, by 50 % larger than the size-optimized). The depth-optimized circuit is shown in Alg. 8 in §A. In contrast to the circuits of Buescher et al. [101],

[7]The optimized Euclidean distance uses only Arithmetic sharing.

our Oblivious Array Access (OA) circuit is slightly larger, but it has a much simpler structure and therefore the same sub-circuits can be applied in parallel using SIMD.

For PPIL, the circuits are applied kD times to find all D dimensions of the k coordinates. Note that for both Oblivious Array Access circuits the AND-depth is independent of k and D because the circuits are evaluated in parallel.

E. Extended Functionalities

A service provider runs the Indoor Localization (IL) service in order to get financial profit, e.g., by using customers' location information for location-based advertising. Hence, it is essential that Privacy-Preserving Indoor Localization (PPIL) can be extended with other Location-Based Services (LBSs), because otherwise the service provider could lose its incentive to offer the IL service in the first place. We emphasize that this requirement is met by PILOT. The extended functionalities can be implemented either (i) as an additional layer within PILOT (which would even not leak the location as intermediate result) or (ii) as an additional privacy-preserving protocol after localization with PILOT (which leaks the location).

VI. Evaluation

We implement and benchmark our protocols on two servers acting as Semi-Trusted Third Parties (STTPs) T_0 and T_1, respectively, each equipped with an Intel Core i7-4770K 3.5 GHz processor and 16 GB RAM. The servers are located in a local Gigabit network with average network latency of about 0.1 ms. The client is run on a Samsung Galaxy S7 mobile device equipped with a Qualcomm MSM8996 Snapdragon 820 processor and 4 GB RAM and is connected to the servers via LTE mobile Internet with 229 ms average Round Trip Time (RTT) and 163 ms median RTT.

Here, we describe the benchmarking results of our algorithms in the outsourcing scenario. We instantiate all primitives with a security level of κ=128-bit and report run-times averaged over 10 executions. We use 4-bit Received Signal Strength (RSS) values (see §V-A) in all but one (1-bit Yao sharing-based Manhattan distance) of our benchmarks.

For establishing a TLS 1.2[8] connection between the client and the STTPs given an established TCP connection (establishing a TCP connection adds a delay of $\sim$1 RTT), we measured 468 ms average run-time, which is approximately equal to 2 RTTs. The time for transferring the data was approximately 1 RTT. These additional time demands depend almost only on the network latency which significantly varies depending on the environment. Note that with broader use of TLS 1.3[9], the latency for establishing a secure channel will be reduced by half due to the 1-RTT handshake protocol. Furthermore, in TLS 1.3 it is possible to resume the connection at very low additional costs due to the 0-RTT connection resumption protocol.

[8]https://tools.ietf.org/html/rfc5246
[9]https://tools.ietf.org/html/rfc8446

A. Input Sharing of Mobile Client

As shown in Tab. IV, the overhead for sharing the RSS values ranges from 0.2 ms to 6.1 ms depending on the number of Access Points (APs) N and hence is negligible compared to the run-time of the STPC protocols, e.g., about 2 % overhead compared to the run-time in the online phase of the most efficient Euclidean distance-based PPIL. In addition, both the bit-length of the shared value and the sharing type only slightly influence the run-times because of the high level of abstraction for the Java code that runs on the mobile client. The shares are created using a modified `java.security.SecureRandom` class, which is extended for generating also values of small byte-lengths.

B. Precomputation in the Setup Phase

As shown in Tab. V, the setup phase of the 4-bit protocols has between 164 MB and 6.1 GB communication and a run-time between 0.7 s and 66.3 s, depending on which distance metric and protocol is used. Since the setup phase is independent of the inputs (only correlated randomness is produced), it can efficiently be precomputed by the STTPs while they are idling, e.g., over night. The most important advantage of our outsourcing scheme is that this setup phase is run only between the STTPs, i.e., it is independent of the PPIL server and client. Moreover, the PPIL computation is independent of the user's identity, i.e., neither the identity of the user influences the setup phase nor the precomputation of the setup phase violates the privacy of the user. Due to the fact that the STTPs are easy to equip with hard disk drives of large capacity for a reasonable price, the STTP servers can precompute thousands of PPIL queries. If this throughput is not sufficient, we can think of the following: one can either upgrade the available hardware or continue the precomputation on the days while the building is closed (weekends, holidays). While the first gives many degrees of freedom for possible efficiency improvements, the second allows the precomputation of more than 100 000 queries during only one weekend.

C. k-Nearest Reference Points

We show the performance differences between the k-NN and Bitonic Sort algorithm on the left of Tab. IV. As it can be seen, k-NN in Yao sharing significantly outperforms Bitonic Sort in Boolean sharing for k=3. On the other hand, Bitonic Sort becomes more efficient with increasing k and would be more efficient than k-NN when k is greater than 40 or 50 because its performance is independent of k. However, for IL k is usually small, e.g., k=3 or k=4. Our k-NN protocol achieves a $1.4\times$ to $1.6\times$ run-time improvement over [97].

D. Oblivious Array Access

The run-times of the Yao-based size-optimized and GMW-based depth-optimized Oblivious Array Access (OA) algorithm for the ranges of RPs are given in the middle of Tab. IV. As it turns out, the size-optimized implementation of OA slightly outperforms the depth-optimized implementation in our standard setting with M=505 RPs. While the first

Table IV: The run-times in milliseconds for sharing client's inputs and for the online phase of our most promising PPIL protocols for N Access Points (APs) and M Reference Points (RPs). In the right part of the table, we depict the best-performing protocols for PPIL using the corresponding distance metrics. k-Nearest Neighbors (k-NN) is implemented in Yao and Bitonic Sort in Boolean sharing.

M	Input sharing	k nearest RPs				Oblivious Array		Manhattan (ABOY)		Euclidean (AY)		Kumar-Hassebrook (AY)		Sørensen (ABOY)		Mod. Sørensen (AY)	
		3-NN	3-NN [97]	30-NN	Bitonic Sort	Size-opt.	Depth-opt.	N=256	N=512	N=256	N=512	N=256	N=512	N=256	N=512	N=256	N=512
32	0.2	0.7	1.1	8.3	26.5	1.2	2.1	47.2	75.0	11.7	18.9	57.6	61.1	66.2	80.9	58.8	61.6
64	0.4	1.5	2.3	16.3	51.9	2.3	4.1	85.6	151.9	20.6	35.8	109.7	114.5	123.0	151.2	112.0	119.3
128	0.8	3.1	4.5	32.7	106.1	4.6	7.3	159.8	289.7	38.3	62.0	213.6	230.5	234.9	293.6	215.4	228.9
256	1.5	5.9	10.3	59.0	186.9	9.3	15.6	304.1	570.9	74.5	124.1	420.3	455.2	476.7	575.0	424.9	455.2
512	3.1	12.3	19.7	120.4	332.4	20.1	31.0	599.7	1 137.4	162.5	270.9	843.8	922.8	948.7	1 144.4	874.6	948.1
1 024	6.2	24.7	40.4	239.8	681.3	39.5	63.8	1 184.6	2 167.8	363.7	594.3	1 646.4	1 969.4	1 873.4	2 179.8	1 710.1	1 979.9

Table V: Benchmarking results for Privacy-Preserving Indoor Localization run between two Semi-Trusted Third Parties (STTPs) based on the introduced distance metrics with N=241 Access Points and M=505 Reference Points. The most efficient solutions for each metric are marked in bold.

Distance metric (Protocol)	Comm. across STTPs in MB			Runtimes across STTPs in s		
	Setup	Online	Total	Setup	Online	Total
Manhattan (B)	974	15	990	4.2	15.0	20.6
Manhattan (Y)	301	51 B	301	3.9	2.2	6.1
Manhattan (ABOY)	167	11	178	0.7	0.4	1.1
Manhattan (1-bit, Y)	5.2	**47 B**	**5.2**	0.3	**0.2**	**0.5**
Euclidean (Y)	6 100	**50 B**	6 100	66.3	16.9	82.3
Euclidean (AB)	203	4.0	207	1.0	0.9	1.9
Euclidean (AY)	164	2.7	**167**	0.8	**0.15**	**1.0**
Kumar-Hassebrook (AB)	319	6.6	326	1.6	2.5	4.1
Kumar-Hassebrook (AY)	280	**2.5**	**283**	2.2	**0.8**	**3.0**
Sørensen (ABO)	196	9.2	205	1.1	2.0	3.1
Sørensen (ABOY)	148	**5.7**	**154**	1.5	**0.9**	**2.6**
Modified Sørensen (AB)	321	6.6	328	1.6	2.4	4.0
Modified Sørensen (AY)	282	**2.6**	**285**	2.2	**0.8**	**3.0**

requires 34 ms run-time in the online phase, the second runs in 39 ms which is by ∼15 % slower. However, in total the depth-optimized algorithm runs by factor ∼1.5× faster than the size-optimized (46 ms vs. 72 ms) and requires almost 2× less communication (1.3 MB vs. 2.3 MB).

E. Performance of Privacy-Preserving Indoor Localization

Concrete parameters for Indoor Localization (IL) depend on the specific application:

- The accuracy of IL depends on the number of RPs, the demands of which depend on the particular scenario, e.g., in one scenario we need to determine whether a user is in a room and in another scenario whether the user is in a particular part of the room.
- One will not always require all possible RPs in the building, e.g., only one floor, or all floors, but only one wing of the building.
- The APs often have many MAC addresses which causes duplicating the same APs in IL. Duplicates can be filtered out by creating the public list of single APs. Moreover, the APs that provide unreliable data can also be filtered out using machine learning techniques, such as [102]. Using these techniques, also the most reliable APs can be found.

In this section, we present the performance results for our PPIL protocols: (i) for a real-world dataset with N=241 APs and M=505 RPs, (ii) for the settings used in the existing literature for comparing our solutions with those proposed in the previous works, and (iii) for ranges of APs and RPs.

Real-World Setting: In Tab. V we show the benchmarking results of our algorithms for N=241 APs, M=505 RPs, and k=3 Nearest Neighbors. The setting is the same as in [35] and represents a typical size for a single building database. Examples on databases with similar sizes can be found, e.g., in [103, 104, 105]. Note that only algorithms fitting into 16 GB RAM on the STTPs are depicted in the table. Since the setup phase can efficiently be precomputed, the most important information in this table are the online run-times and communication.

We can deduce the following from Tab. V:

- Even with all our optimizations, 4-bit PPIL in the client/server setting with a single server would require transferring at least 100 MB between the mobile client and the server over the mobile Internet per query. This is highly unpractical, especially given today's mobile dataplans of usually a few GB per month. Therefore, outsourcing to two or more non-colluding servers as in the model investigated in this work is the only realistic model because only a few kB need to be sent over the mobile Internet.
- Our most efficient Euclidean distance-based PPIL protocol using Arithmetic and Yao sharing achieves 1.0 s total and 0.15 s online time, which is a reasonable delay in practice.
- Many optimized algorithms require less than 1 second online time even for the most precise metrics which is also a reasonable delay in practice.
- The optimized algorithms outperform the non-optimized ones in terms of online run-time by up to factor 100×.
- The optimized algorithms in Yao sharing outperform the ones in Boolean sharing in terms of online time by factor 3–4×. This is due to the fact that although the circuits in Boolean sharing have reasonable depth (298–596), they are up to factor 6× larger than the circuits in Yao sharing.
- For the Manhattan/Euclidean/Kumar-Hassebrook/Sørensen/Modified Sørensen-based PPIL in Arithmetic and Yao sharing (AY), our two commodity hardware-based servers can daily precompute 172k/86k/28k/33k/28k queries, and process 432k/576k/108k/96k/108k queries, respectively.
- The 1-bit Manhattan distance PPIL protocol yields 57× less communication than the "naive" solution (only 5.2 MB in total), it requires only 0.5 s of total run-time, 0.2 s online time, and its online communication is in the order of a few Bytes, which makes this protocol an excellent candidate also for the client-server scenario without outsourcing.

Comparison with Related Work: In the following, we compare our most efficient 4-bit Euclidean distance-based PPIL protocol with the most recent related works of Li et al. [37] and Ziegeldorf et al. [39].

[37]: The experiments of [37] have very high run-times of 60.9 s for very small IL parameters (N=15 APs and M=1 000 RPs) and outdated security parameters (1024-bit Paillier modulus which is considered insecure today [106]). For these values of N and M and state-of-the-art 128-bit security, our solution has an online run-time of 0.11 s (553× faster) and a total run-time of 0.5 s (121× faster).

[39]: Evaluation of the prototype implementation of [39] on two servers with 1 ms network latency resulted in a 10 s run-time for only N=20 APs and 160 possible states. For the same setting (we set the number of RPs to M=160), we achieve an online run-time of 0.02 s (500× faster) and a total run-time of 0.1 s (100× faster).

Ranges of APs and RPs: An intuitive solution for representing the real-world scalability of our algorithms is to perform benchmarks for different real-world buildings. This, however, has the drawback that the benchmarks for a few buildings are valid only for these buildings and say nothing about other buildings. To solve this problem, we benchmark our algorithms on ranges of APs and RPs in constant intervals. This allows to interpolate the benchmarking results for determining overhead for any numbers of APs and RPs in the measured range. Since the function or shape of a building does not affect the performance of our algorithms, the benchmarking results will apply to all buildings with the same number of APs and RPs. We give the online times for our most efficient protocols for ranges of APs and RPs in Tab. IV.

VII. Conclusion

In this work, we designed, implemented, and evaluated PILOT, a system for practical Privacy-Preserving Indoor Localization (PPIL). To the best of our knowledge, PILOT is the first PPIL system with practical online run-times of less than 1 s and is by several orders of magnitude faster than previous works. We applied quantization to reduce the Received Signal Strength bit-length without significant loss of accuracy. We designed efficient building blocks for PPIL, which are of independent interest: a size-efficient k-Nearest Neighbors circuit, and a Single Instruction Multiple Data-capable and easy-to-implement Oblivious Array Access circuit. We used an outsourcing scenario that shifts most of the communication and computation away from the weak and battery-powered mobile client. Overall, our PPIL system PILOT can process hundreds of thousands of queries per day on commodity hardware.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) and by the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, by the DFG as part of project E4 within the CRC 1119 CROSSING, by the Academy of Finland under grant number 303576 and 303578, and by the National Natural Science Foundation of China under Grant No. 61872051.

References

[1] A. Kushki, K. Plataniotis, and A. Venetsanopoulos, *WLAN positioning systems*. Cambridge University Press, 2012.

[2] S. Alletto, R. Cucchiara, G. D. Fiore, L. Mainetti, V. Mighali, L. Patrono, and G. Serra, "An indoor location-aware system for an IoT-based smart museum," *IEEE IoT-J*, 2016.

[3] R. Li, T. Song, N. Capurso, J. Yu, J. Couture, and X. Cheng, "IoT applications on secure smart shopping system," *IEEE IoT-J*, 2017.

[4] S. M. Bellovin, R. M. Hutchins, T. Jebara, and S. Zimmeck, "When enough is enough: Location tracking, mosaic theory, and machine learning," *NYU Journal of Law & Liberty*, 2013.

[5] P. Hallgren, M. Ochoa, and A. Sabelfeld, "Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol," in *Privacy, Security and Trust (PST)*, 2015.

[6] ——, "Maxpace: Speed-constrained location queries," in *Communications and Network Security (CNS)*, 2016.

[7] S. Stirbys, O. A. Nabah, P. Hallgren, and A. Sabelfeld, "Privacy-preserving location-proximity for mobile apps," in *Parallel, Distributed and Network-based Processing (PDP)*, 2017.

[8] P. Hallgren, C. Orlandi, and A. Sabelfeld, "PrivatePool: Privacy-preserving ridesharing," in *Computer Security Foundations Symposium (CSF)*, 2017.

[9] K. Järvinen, Á. Kiss, T. Schneider, O. Tkachenko, and Z. Yang, "Faster privacy-preserving location proximity schemes," in *CANS*, 2018.

[10] J. Einsiedler, I. Radusch, and K. Wolter, "Vehicle indoor positioning: A survey," in *IEEE Workshop on Positioning, Navigation and Communications (WPNC)*, 2017.

[11] L. Chen, S. Thombre, K. Järvinen, E. S. Lohan, A. Alén-Savikko, H. Leppäkoski, M. Z. H. Bhuiyan, S. Bu-Pasha, G. N. Ferrara, S. Honkala, J. Lindqvist, L. Ruotsalainen, P. Korpisaari, and H. Kuusniemi, "Robustness, security and privacy in location-based services for future IoT: A survey," *IEEE Access*, 2017.

[12] E. S. Lohan, P. Richter, V. Lucas-Sabola, J. A. López-Salcedo, G. Seco-Granados, H. Leppäkoski, and E. S. Santiago, "Location privacy challenges and solutions - parts I and II," *Inside GNSS*, 2017.

[13] A. M. Ladd, K. E. Bekris, G. Marceau, A. Rudys, D. S. Wallach, and L. E. Kavraki, "Using wireless Ethernet for localization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[14] P. Tao, A. Rudys, A. M. Ladd, and D. S. Wallach, "Wireless LAN location-sensing for security applications," in *WiSec*, 2003.

[15] A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki, "Practical robust localization over large-scale 802.11 wireless networks," in *MobiCom*, 2004.

[16] A. M. Ladd, K. E. Bekris, A. Rudys, L. E. Kavraki, and D. S. Wallach, "Robotics-based location sensing using wireless Ethernet," *Wireless Networks*, 2005.

[17] X.-Y. Lin, T.-W. Ho, C.-C. Fang, Z.-S. Yen, B.-J. Yang, and F. Lai, "A mobile indoor positioning system based on ibeacon technology," in *Engineering in Medicine and Biology Society (EMBC)*, 2015.

[18] C. Langlois, S. Tiku, and S. Pasricha, "Indoor localization with smartphones: Harnessing the sensor suite in your pocket," *IEEE CEM*, 2017.

[19] A. Yassin, Y. Nasser, M. Awad, A. Al-Dubai, R. Liu, C. Yuen, R. Raulefs, and E. Aboutanios, "Recent advances in indoor localization: A survey on theoretical approaches and applications," *IEEE Communications Surveys and Tutorials*, 2017.

[20] S. Capkun, S. Ganeriwal, F. Anjum, and M. Srivastava, "Secure RSS-based localization in sensor networks," *Technical Report/ETH Zurich*, 2011.

[21] C. K. Chung, I. Q. Chung, Y. H. Wang, and C. T. Chang, "The integrated applications of WIFI and APP used in the shopping mall environment for menber card E-marketing," in *International Conference on Machine Learning and Computing*, 2016.

[22] T. Guan, L. Fang, W. Dong, Y. Hou, and C. Qiao, "Indoor localization with asymmetric grid-based filters in large areas utilizing smartphones," in *IEEE International Conference on Communications (ICC)*, 2017.

[23] Z. Yin, C. Wu, Z. Yang, and Y. Liu, "Peer-to-peer indoor navigation using smartphones," *IEEE Journal on Selected Areas in Communications*, 2017.

[24] M. Ibrahim, H. Liu, M. Jawahar, V. Nguyen, M. Gruteser, R. Howard, B. Yu, and F. Bai, "Verification: Accuracy evaluation of WiFi fine time measurements on an open platform," in *Mobile Computing and Networking*, 2018.

[25] S. Sen, J. Lee, K.-H. Kim, and P. Congdon, "Avoiding multipath to revive inbuilding WiFi localization," in *Mobile Systems, Applications, and Services*, 2013.

[26] W. Gong and J. Liu, "SiFi: Pushing the limit of time-based WiFi localization using a single commodity access point," *Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2018.

[27] Z. Zhang, X. Zhou, W. Zhang, Y. Zhang, G. Wang, B. Y. Zhao, and H. Zheng, "I am the antenna: Accurate outdoor AP location using smartphones," in *MobiCom*, 2011.

[28] P. Richter and M. Toledano-Ayala, "Revisiting gaussian process regression modeling for localization in wireless sensor networks," *Sensors*, 2015.

[29] K. Sheng, Z. Gu, X. Mao, X. Tian, W. Wu, X. Gan, and X. Wang, "The collocation of measurement points in large open indoor environment," in *INFOCOM*, 2015.

[30] M. Wang, Z. Zhang, X. Tian, and X. Wang, "Temporal correlation of the RSS improves accuracy of fingerprinting localization," in *INFOCOM*, 2016.

[31] G. Bielsa, J. Palacios, A. Loch, D. Steinmetzer, P. Casari, and J. Widmer, "Indoor localization using commercial off-the-shelf 60 GHz access points," in *INFOCOM*, 2018.

[32] S. He, W. Lin, and S.-H. G. Chan, "Indoor localization and automatic fingerprint update with altered AP signals," *IEEE Transactions on Mobile Computing*, 2017.

[33] C. Chen, Y. Chen, Y. Han, H. Q. Lai, F. Zhang, and K. J. R. Liu, "Achieving centimeter-accuracy indoor localization on WiFi platforms: A multi-antenna approach," *IEEE IoT Journal*, 2017.

[34] X. Li, T. Zhang, D. Ma, B. Cao, and Q. Zhang, "Location privacy protection in asynchronous localization networks by resource allocation approaches," in *IEEE International Conference on Communications (ICC) Workshops*, 2017.

[35] Z. Yang and K. Järvinen, "The death and rebirth of privacy-preserving Wifi fingerprint localization with Paillier encryption," in *INFOCOM*, 2018.

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

[37] H. Li, L. Sun, H. Zhu, X. Lu, and X. Cheng, "Achieving privacy preservation in WiFi fingerprint-based localization," in *INFOCOM*, 2014.

[38] T. Shu, Y. Chen, J. Yang, and A. Williams, "Multi-lateral privacy-preserving localization in pervasive environments," in *INFOCOM*, 2014.

[39] J. H. Ziegeldorf, N. Viol, M. Henze, and K. Wehrle, "Poster: Privacy-preserving indoor localization," *WiSec*, 2014.

[40] A. Konstantinidis, G. Chatzimilioudis, D. Zeinalipour-Yazti, P. Mpeis, N. Pelekis, and Y. Theodoridis, "Privacy-preserving indoor localization on smartphones," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2015.

[41] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems (IJUFKS)*, 2002.

[42] T. Zhang, S. S. M. Chow, Z. Zhou, and M. Li, "Privacy-preserving Wi-Fi fingerprinting indoor localization," in *IWSEC*, 2016.

[43] Y. Huang, P. Chapman, and D. Evans, "Privacy-preserving applications on smartphones," in *HotSec*, 2011.

[44] H. Carter, C. Lever, and P. Traynor, "Whitewash: Outsourcing garbled circuit generation for mobile devices," in *ACSAC*, 2014.

[45] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, 2016.

[46] P. Richter and M. Toledano-Ayala, "Ubiquitous and seamless localization: fusing GNSS pseudoranges and WLAN signal strengths," *Mobile Information Systems*, 2017.

[47] J. Torres-Sospedra, R. Montoliu, S. Trilles, Ó. Belmonte, and J. Huerta, "Comprehensive analysis of distance and similarity measures for Wi-Fi fingerprinting indoor positioning systems," *Expert Systems with Applications*, 2015.

[48] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *Systems, Man, and Cybernetics*, 2007.

[49] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM*, 2000.

[50] B. Li, Y. Wang, H. K. Lee, A. Dempster, and C. Rizos, "Method for yielding a database of location fingerprints in WLAN," *IEEE Proceedings - Communications*, 2005.

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

[52] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *USENIX Security*, 2011.

[53] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *S&P*, 2015.

[54] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, "Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation," in *EuroS&P*, 2016.

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

[56] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with an honest majority," in *STOC*, 1987.

[57] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[58] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, 2003.

[59] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *CCS*, 2013.

[60] T. Schneider and M. Zohner, "GMW vs. Yao? Efficient secure two-party computation with low depth circuits," in *FC*, 2013.

[61] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Automated synthesis of optimized circuits for secure computation," in *CCS*, 2015.

[62] V. Kolesnikov and R. Kumaresan, "Improved OT extension for transferring short secrets," in *CRYPTO*, 2013.

[63] 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.

[64] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay – a secure two-party computation system," in *USENIX Security*, 2004.

[65] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008.

[66] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *S&P*, 2013.

[67] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT*, 2015.

[68] D. Demmler, T. Schneider, and M. Zohner, "Ad-hoc secure two-party computation on mobile devices using hardware tokens," in *USENIX Security*, 2014.

[69] D. Gupta, B. Mood, J. Feigenbaum, K. Butler, and P. Traynor, "Using Intel software guard extensions for efficient two-party secure function evaluation," in *FC*, 2016.

[70] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-preserving distributed linear regression on high-dimensional data," *PETS*, 2017.

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

[72] D. Demmler, K. Hamacher, T. Schneider, and S. Stammler, "Privacy-preserving whole-genome variant queries," in *CANS*, 2017.

[73] C. Bonte, E. Makri, A. Ardeshirdavani, J. Simm, Y. Moreau, and F. Vercauteren, "Towards practical privacy-preserving genome-wide association study," *BMC bioinformatics*, 2018.

[74] O. Tkachenko, C. Weinert, T. Schneider, and K. Hamacher, "Large-scale privacy-preserving statistical computations for distributed genome-wide association studies," in *ASIACCS*, 2018.

[75] T. Schneider and O. Tkachenko, "Towards efficient privacy-preserving similar sequence queries on outsourced genomic databases," in *WPES*, 2018.

[76] H. Cho, D. J. Wu, and B. Berger, "Secure genome-wide association analysis using multiparty computation," *Nature biotechnology*, 2018.

[77] T. Schneider and O. Tkachenko, "EPISODE: Efficient Privacy-preservIng Similar sequence queries on Outsourced genomic DatabasEs," in *ASIACCS*, 2019, to appear.

[78] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases—real-world applications of secure multi-party computation," *The Computer Journal*, 2018.
[79] T. Araki, J. Furukawa, Y. Lindell, A. N., and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *CCS*, 2016.
[80] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *CCS*, 2018.
[81] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi, "A framework for outsourcing of secure computation," in *CCSW*, 2014.
[82] H. Carter, B. Mood, P. Traynor, and K. Butler, "Outsourcing secure two-party computation as a black box," in *CANS*, 2015.
[83] D. Rotaru and T. Wood, "MArBled circuits: Mixing arithmetic and boolean circuits with active security," Cryptology ePrint Archive, Report 2019/207, 2019, https://eprint.iacr.org/2019/207.
[84] A. B. M. Musa and J. Eriksson, "Tracking unmodified smartphones using Wi-Fi monitors," in *Embedded Network Sensor Systems*, 2012.
[85] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens, "Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms," in *ASIACCS*, 2016.
[86] J. Martin, T. Mayberry, C. Donahue, L. Foppe, L. Brown, C. Riggins, E. C. Rye, and D. Brown, "A study of MAC address randomization in mobile devices and when it fails," *PETS*, 2017.
[87] L. Schauer, F. Dorfmeister, and F. Wirth, "Analyzing passive Wi-Fi fingerprinting for privacy-preserving indoor-positioning," in *International Conference on Localization and GNSS (ICL-GNSS)*, 2016.
[88] S. Kamara and M. Raykova, "Secure outsourced computation in a multi-tenant cloud," in *IBM Workshop on Cryptography and Security in Clouds*, 2011.
[89] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Kluwer Academic Publishers, 1992.
[90] P. Richter, E. S. Lohan, and J. Talvitie, "WLAN (WiFi) RSS database for fingerprinting positioning," 2018, https://zenodo.org/record/1161525.
[91] P. Richter, H. Leppäkoski, E. S. Lohan, Z. Yang, K. Järvinen, O. Tkachenko, and T. Schneider, "Received signal strength quantization for secure indoor positioning via fingerprinting," in *International Conference on Localization and GNSS (ICL-GNSS)*, 2018.
[92] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, 2009.
[93] N. Gilboa, "Two party RSA key generation," in *CRYPTO*, 1999.
[94] J. von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, 1993.
[95] J. Boyar and R. Peralta, "Tight bounds for the multiplicative complexity of symmetric functions," *Theoretical Computer Science*, 2008.
[96] 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.
[97] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar, "Compacting privacy-preserving k-nearest neighbor search using logic synthesis," in *DAC*, 2015.
[98] K. E. Batcher, "Sorting networks and their applications," in *Spring Joint Computer Conference*, 1968.
[99] H. W. Lang, "Bitonic sorting network for n not a power of 2," http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm, 2017, accessed: 2018-01-16.
[100] M. Keller and P. Scholl, "Efficient, oblivious data structures for MPC," in *ASIACRYPT*, 2014.
[101] N. Buescher, A. Holzer, A. Weber, and S. Katzenbeisser, "Compiling low depth circuits for practical secure computation," in *ESORICS*, 2016.
[102] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, 1997.
[103] E. Sansano Sansano, R. Montoliu Colás, O. Belmonte Fernández, and J. Torres-Sospedra, "IndoorLoc Platform, a public repository for comparing and evaluating indoor positioning and navigation databases and algorithms," http://indoorlocplatform.uji.es/, 2017, accessed: 2018-05-07.
[104] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," https://archive.ics.uci.edu/ml/, 2017, accessed: 2018-05-07.
[105] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, "CRAWDAD, a community resource for archiving wireless data at Dartmouth," https://crawdad.org/index.html, 2009, accessed: 2018-05-07.
[106] E. Barker, "Recommendation for key management part 1: General (revision 4)," *NIST*, 2016.

APPENDIX

A. Algorithms

In this appendix, we give details on our algorithm implementations in the ABY framework [51].

$distances \leftarrow d_m^{ABOY}(RSS_C, RefPoints, N)$

1: $distances \leftarrow \emptyset$
2: **foreach** RSS_S **in** $RefPoints$ **do**
3: $\langle dist \rangle^A \leftarrow 0$
4: **for** $i = 1 : N$ **do**
5: $\langle gt \rangle^B \leftarrow \mathbf{Y2B}(\mathbf{GT}(\langle RSS_S[i] \rangle^Y, \langle RSS_C[i] \rangle^Y))$
6: $\langle d \rangle^A \leftarrow \langle RSS_S - RSS_C \rangle^A$
7: $\langle d' \rangle^A \leftarrow \mathbf{Two'sComplement}(\langle d \rangle^A)$
8: $\langle d \rangle^A \leftarrow \langle d \rangle^A \cdot \langle gt \rangle^B$ //using C-OT
9: $\langle d' \rangle^A \leftarrow \langle d' \rangle^A \cdot \mathbf{INV}(\langle gt \rangle^B)$ //using C-OT
10: $\langle dist \rangle^A \leftarrow \langle dist \rangle^A + \langle d \rangle^A + \langle d' \rangle^A$
11: $distances.\mathbf{append}(\langle dist \rangle^A)$
12: **return** $distances$

Algorithm 1: Manhattan distance using Arithmetic, Boolean, and Yao sharing, and Correlated OT (C-OT).

$distances \leftarrow d_e^A(RSS_C, \langle RSS_C^2 \rangle^A, RefPoints, N)$

1: $distances \leftarrow \emptyset$
2: **foreach** RSS_S **in** $RefPoints$ **do**
3: $\langle dist \rangle^A \leftarrow \langle RSS_C^2 \rangle^A + \langle RSS_S^2 \rangle^A$ //precomputed
4: **for** $i = 1 : N$ **do**
5: $\langle product \rangle^A \leftarrow \langle RSS_C[i] \rangle^A \cdot \langle RSS_S[i] \rangle^A$
6: $\langle product \rangle^A \leftarrow \langle product \rangle^A + \langle product \rangle^A$
7: $\langle dist \rangle^A \leftarrow \langle dist \rangle^A - \langle product \rangle^A$
8: $distances.\mathbf{append}(\langle dist \rangle^A)$
9: **return** $distances$

Algorithm 2: Euclidean distance using Arithmetic sharing.

$distances \leftarrow d_{KH}^{AY}(RSS_C, \langle RSS_C^2 \rangle^A, RefPoints, N, ScalingFactor)$

1: $distances \leftarrow \emptyset$
2: **foreach** RSS_S **in** $RefPoints$ **do**
3: $\langle nom \rangle^A \leftarrow 0$
4: $\langle denom \rangle^A \leftarrow \langle RSS_C^2 \rangle^A + \langle RSS_S^2 \rangle^A$ //precomputed
5: **for** $i = 1 : N$ **do**
6: $\langle product \rangle^A \leftarrow \langle RSS_S[i] \rangle^A \cdot \langle RSS[i]_C \rangle^A$
7: $\langle nom \rangle^A \leftarrow \langle product \rangle^A$
8: $\langle denom \rangle^A \leftarrow \langle denom \rangle^A - \langle product \rangle^A$
9: $\langle nom \rangle^Y \leftarrow \mathbf{A2Y}(\langle nom \rangle^A)$
10: $\langle nom \rangle^Y \leftarrow \langle nom \rangle^Y \ll ScalingFactor$ //bit-shift
11: $\langle denom \rangle^Y \leftarrow \mathbf{A2Y}(\langle denom \rangle^A)$
12: $distances.\mathbf{append}(\langle nom \rangle^Y / \langle nom \rangle^Y)$
13: **return** $distances$

Algorithm 3: Kumar-Hassebrook distance using Arithmetic sharing and Yao sharing.

$distances \leftarrow d_s'^{AY}(RSS_C, RefPoints, N, \langle ScalingFactor \rangle^A)$

```
1 :  distances ← ∅
2 :  foreach RSS_S in RefPoints do
3 :      ⟨num⟩^A ← 0
4 :      ⟨denom⟩^A ← 0
5 :      for i = 1 : N do
6 :          ⟨eucl⟩^A ← d_e^A(⟨RSS_S[i]⟩^A, ⟨RSS_C[i]⟩^A)
7 :          ⟨sum⟩^A ← ⟨RSS_S[i]⟩^A + ⟨RSS_C[i]⟩^A
8 :          ⟨num⟩^A ← ⟨num⟩^A + ⟨eucl⟩^A
9 :          ⟨denom⟩^A ← ⟨denom⟩^A + ⟨sum⟩^A
10 :     ⟨num⟩^Y ← A2Y(⟨num⟩^A)
11 :     ⟨num⟩^Y ← ⟨num⟩^Y ≪ ScalingFactor        //bit-shift
12 :     ⟨denom⟩^Y ← A2Y(⟨denom⟩^A)
13 :     distances.append(⟨num⟩^Y / ⟨denom⟩^Y)
14 : return distances
```

Algorithm 4: Modified Sorensen distance using Arithmetic sharing and Yao sharing. Euclidean distance is computed similar to Algorithm 2.

BitonicSort$(dists, ids, start, M)$

```
1 :  for i = 1 : M do
2 :      m = M/2
3 :      BitonicSort(dists, ids, start, m)
4 :      BitonicSort(dists, ids, start + m, M − m)
5 :      BitonicMerge(dists, ids, start, M)
```

BitonicMerge$(dists, ids, start, M)$

```
1 :  m = PO2LT(M)
2 :  for i = begin : begin − M + m
3 :      ⟨gt⟩^B = GT(⟨dists[i]⟩^B, ⟨dists[i + m]⟩^B)
4 :      CondSwap(⟨dists[i]⟩^B, ⟨dists[i + m]⟩^B, ⟨gt⟩^B)
5 :      CondSwap(⟨ids[i]⟩^B, ⟨ids[i + m]⟩^B, ⟨gt⟩^B)
6 :  BitonicMerge(dists, ids, start, m)
7 :  BitonicMerge(dists, ids, start + m, M − m)
```

Algorithm 5: Bitonic Sort algorithm using Boolean sharing. The function $y \leftarrow \mathbf{PO2LT}(x)$ finds an y such that y is the biggest power of two less than x.

$minIds \leftarrow k\text{-}NN(dists, M, k)$

```
1 :  minDists, minIds ← ∅
2 :  for i = 1 : k + 1 do
3 :      minDists.append(⟨MAX_VALUE⟩^Y)
4 :      minIds.append(⟨0⟩^Y)
5 :  for i = 1 : M do
6 :      minDists[k + 1] ← dists[i]
7 :      minIds[k + 1] ← ⟨i⟩^Y
8 :      for j = k + 1 : 2 do
9 :          ⟨gt⟩^Y ← GT(⟨minDists[j − 1]⟩^Y, ⟨minDists[j]⟩^Y)
10 :         CondSwap(⟨minDists[j]⟩^Y, ⟨minDists[j − 1]⟩^Y, ⟨gt⟩^Y)
11 :         CondSwap(⟨minIds[j]⟩^Y, ⟨minIds[j − 1]⟩^Y, ⟨gt⟩^Y)
12 : return minIds[1 : k]
```

Algorithm 6: k-Nearest Neighbors (k-NN) circuit using Yao sharing.

$\langle coord \rangle^Y \leftarrow \mathbf{OA}^s(coords, \langle idx \rangle^Y, M)$

```
1 :  poTwo ← PO2LT(M)
2 :  for i = M − 1 : 2 · poTwo do
3 :      coords.append(NULL)
4 :  i ← 1
5 :  bitNum ← 0
6 :  while (i < 2 · poTwo) do
7 :      ⟨sel⟩^Y ← ⟨idx⟩^Y[bitNum]
8 :      j ← 0
9 :      while (j < coords.size()) do
10 :         if (⟨coords[j]⟩^Y == NULL)
11 :             j ← j + 2 · i
12 :             continue
13 :         else if (⟨coords[j + i]⟩^Y == NULL)
14 :             j ← j + 2 · i
15 :             continue
16 :         else
17 :             ⟨coords[j]⟩^Y ← MUX(
18 :                 ⟨coords[j + i]⟩^Y, ⟨coords[j]⟩^Y, ⟨sel⟩^Y)
19 :         j ← j + 2 · i
20 :     i ← 2 · i
21 :     bitNum ← bitNum + 1
22 : return ⟨res[0]⟩^Y
```

Algorithm 7: Size-optimized Oblivious Array Access (OA) circuit using Yao sharing.

$\langle coord \rangle^B \leftarrow \mathbf{OA}^d(coords, ids, \langle idx \rangle^B, M)$

```
1 :  res ← ∅
2 :  for i in 1 : M do
3 :      res.append(EQ(⟨ids[i]⟩^B, ⟨idx⟩^B))
4 :      ⟨res[i]⟩^B ← MUX(⟨0⟩^B, ⟨coords[i]⟩^B, ⟨res[i]⟩^B)
5 :  for i in 2 : M do
6 :      ⟨res[0]⟩^B ← ⟨res[0]⟩^B ⊕ ⟨res[i]⟩^B
7 :  return ⟨res[0]⟩^B
```

Algorithm 8: Depth-optimized Oblivious Array Access (OA) circuit using Boolean sharing.

D EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs (ASIACCS'19)

[ST19] T. SCHNEIDER, O. TKACHENKO. **"EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs"**. In: *ASIA Conference on Computer and Communications Security (ASIACCS)*. Full version: https://ia.cr/2021/029. ACM, 2019, pp. 315–327. CORE Rank A. Appendix D.

https://doi.org/10.1145/3321705.3329800

EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs*

Thomas Schneider
TU Darmstadt
schneider@encrypto.cs.tu-darmstadt.de

Oleksandr Tkachenko
TU Darmstadt
tkachenko@encrypto.cs.tu-darmstadt.de

ABSTRACT

Nowadays, genomic sequencing has become much more affordable for many people and, thus, many people own their genomic data in a digital format. Having paid for genomic sequencing, they want to make use of their data for different tasks that are possible only using genomics, and they share their data with third parties to achieve these tasks, e.g., to find their relatives in a genomic database. As a consequence, more genomic data get collected worldwide. The upside of the data collection is that unique analyses on these data become possible. However, this raises privacy concerns because the genomic data uniquely identify their owner, contain sensitive data about his/her risk for getting particular diseases, and even sensitive information about his/her family members.

In this paper, we introduce EPISODE — a highly efficient privacy-preserving protocol for Similar Sequence Queries (SSQs), which can be used for finding genetically similar individuals in an outsourced genomic database, i.e., securely aggregated from data of multiple institutions. Our SSQ protocol is based on the edit distance approximation by Asharov et al. (PETS'18), which we further optimize and extend to the outsourcing scenario. We improve their protocol by using more efficient building blocks and achieve a 5–6× run-time improvement compared to their work in the same two-party scenario.

Recently, Cheng et al. (ASIACCS'18) introduced protocols for outsourced SSQs that rely on homomorphic encryption. Our new protocol outperforms theirs by more than factor 24 000× in terms of run-time in the same setting and guarantees the same level of security. In addition, we show that our algorithm scales for practical database sizes by querying a database that contains up to a million short sequences within a few minutes, and a database with hundreds of whole-genome sequences containing 75 million alleles each within a few hours.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Management and querying of encrypted data**; **Privacy protections**;

KEYWORDS

medical privacy, privacy-enhancing technologies, genomic research, edit distance, secure computation, outsourcing

1 INTRODUCTION

Numerous efforts by the research community, industries, and governments of different countries substantially reduced the costs of genome sequencing: the costs for sequencing a whole genome have fallen from 10 million USD to less than 1 000 USD in the last ten years [Wet17]. This leads to more genome data being collected, and services that use genome data are becoming increasingly popular, e.g., 23andMe[1], MyHeritage[2], and ancestry[3]. Common use cases for genome data are: (i) Similar Sequence Queries (SSQs) for finding genome sequences that are similar to the sequence of the analyzed person, (ii) Genome-Wide Association Studies (GWAS) for finding associations between diseases and genetic variants, and (iii) genealogical tests for determining ancestral ethnicity of the person.

In this work, we focus on SSQs. They can be used for finding (up to that time unknown) relatives, and for making better diagnoses and prescribing the most promising treatments using the medical history of people that are genetically similar to the patient [FPI16]. However, a data provider (e.g., a medical institution) commonly has a limited number of collected genome sequences which prevents a high-quality similar patient analysis, since the diversity and completeness of the database is crucial in genome analyses [PF16].

A further use case for SSQs is crime solving where only the DNA of the suspect is known. It has been shown in the past that some very complex criminal investigations can be solved using solely the DNA information [Jon10], also even if only the suspect's second-degree relatives are contained in the database, e.g., by reconstructing the family tree [Cor18]. However, no global DNA databases exist at the moment that would facilitate such investigations, since this would raise concerns about the privacy of the DNA donors. As a solution, we consider privacy-preserving aggregation of the DNA databases of multiple parties and privacy-preserving queries on the aggregated database for ensuring privacy of the DNA donors.

Despite the good uses of genomic data, their leakage causes severe privacy violations for the genomic data donors. This is due to the fact that genome data are unique for each individual and contain sensitive information about him/her and his/her relatives [HAHT15, Ayd16], e.g., ethnicity information and predispositions to particular diseases. The possession of this information by third parties can give rise to genetic discrimination, e.g., if a health insurance company would increase the client's fee based on his/her predispositions to diseases, or if an employer would reject the candidate's application based on the aforementioned reasons. To address this, we employ Secure Multi-Party Computation (SMPC)

*A summary of preliminary results of this paper has been published as short paper at WPES'18 [ST18] and the full results have been published at ASIACCS'19 [ST19]. This version contains improvements to the description of the protocols in §3.3.

[1]https://www.23andme.com
[2]https://myheritage.com
[3]https://www.ancestry.com

techniques for constructing highly efficient privacy-preserving protocols for distributed SSQs. Although there already exist solutions for privacy-preserving SSQs, the solutions are either inefficient or custom-tailored, i.e., it is far from trivial to extend these protocols to privacy-preserving aggregation of databases or thresholding distances to similar sequences.

1.1 Our Contributions

We design *EPISODE*, an efficient SSQ protocol that is by orders of magnitude more efficient than previous related works. EPISODE is designed for outsourcing but can be used for the two-party client/server model as well, e.g., for the same setting as in [AHLR18] where one party has a database and the other party has a query, or when each party possesses one or more databases (e.g., for crowd-sourced SSQs). We also show how multiple databases can be aggregated in the outsourcing scenario and describe related costs for each scenario. In addition, we describe a thresholding protocol for finding relatives using EPISODE.

Large-Scale Experiments. We conduct large-scale experiments on an outsourced database with up to one million genome sequences of small and medium lengths, and we show that our implementation has practical run-times on commodity hardware, e.g., secure evaluation of an SSQ protocol on one million sequences of length one thousand took only 8.3 minutes.

Whole-Genome Experiments. To the best of our knowledge, we are the first to conduct experiments on whole-genome sequences (sequence length n=75 million alleles) using the Edit Distance (ED) approximation of [AHLR18] that, unlike [WHZ+15], can handle high-divergence data and show practical run-times of just a few hours.

1.2 Outline

Section 2 describes related work in the field of privacy-preserving SSQs. In Section 3, we explain the necessary basics of genetics, SMPC, and the SMPC framework used in this work. Afterwards, Section 4 gives the system model and details the designed algorithms. Finally, we show the benchmarking results of our algorithms in different scenarios in Section 5 and conclude in Section 6.

2 RELATED WORK

In this section, we compare our Edit Distance (ED) and Similar Sequence Query (SSQ) algorithm with that in previous related work. Our SSQ protocol is benchmarked in the same system model of the papers we compare to unless stated otherwise.

Asharov et al. [AHLR18] introduced an approximation technique for ED that can handle high divergence data. Their contribution is twofold: (i) they construct an efficient and precise approximation for ED, and (ii) they use Look-Up Tables (LUTs) instead of direct computation of the ED, thus precomputing the most expensive parts of the computation in the clear.

The first contribution works as follows: the genome sequences in the database and query are split into blocks of small size (e.g., b=5) and padded to a somewhat greater size (e.g., b'=16). Because of the much smaller size of the blocks compared to a full sequence, the overhead for computing ED is also much smaller (ED requires $O(n^2)$ computation in the sequence length n).

For their second contribution, they utilize the fact that genes in the blocks are naturally distributed highly non-uniformly. For all sequences available in the clear, this allows to compute cross-sequence LUTs block-wise for all observed block values. Using this approach, the value of the block is compared with each element of the corresponding LUT instead of computing the ED directly. If the value is equal to one of the values in the LUT, the corresponding distance is selected. Asharov et al. empirically show that the probability of an element not being in the LUT is small, and the absence of a single element influences the overall distance only slightly. Moreover, the authors design a custom protocol for computing the k nearest edit distances between a client's query and a server's database containing parts of genome sequences. However, their protocol has the drawback that it is custom which makes it non-trivial to extend to other functionalities such as aggregation of databases. Their protocol works in a setting with two semi-honest parties, where the client inputs the query and the server inputs a database into a Secure Two-Party Computation (STPC) protocol. EPISODE runs by factor 5–6× faster than their protocol in the same two-party setting (see Section 5.2 for details).

Atallah et al. [AKD03] developed protocols for secure sequence comparison, and Atallah and Li [AL05] moved these protocols to the outsourcing scenario. In both works, the authors compute the ED, i.e., the number of additions, deletions, and substitutions needed to transform one string into another, with quadratic computation and communication overhead in the sequence length n, i.e., $O(n^2)$, which is a much larger overhead than ours of $O(n\omega)$, where ω is the LUT width, usually 20 or 30.

Jha et al. [JKS08] designed algorithms for privacy-preserving ED using Garbled Circuits (GCs). Their construction scales much worse than the recent solutions including ours, e.g., their algorithm runs in 658 s and requires 364 MB communication, whereas ours requires only 5.7 ms run-time (more than 100 000× faster) and 397 kB communication (more than 900× less) for the same sequence length n=200, and we set the width of the Look-Up Table (LUT) to ω=20.

Wang et al. [WHZ+15] propose an extremely efficient approach for approximating the ED using a set size difference metric. Their approach can process a genome-wide query over one million patients in about 3 hours, but, unfortunately, it works only for data with very small divergence (less than 0.5 % variability between individuals), which is not always true for genome data.

The authors of [AAAM17] designed two approximations for ED: a set intersection method based on [PSSZ15] and a banded alignment-based algorithm that relies on GCs. The drawbacks of these methods are: (i) neither algorithm achieves good accuracy on long genome sequences, and (ii) the authors do not show which security parameters are used and do not detail communication requirements of their algorithms.

Zhu and Huang [ZH17] design efficient algorithms for ED, which are, however, much slower than the approximations of ED. Their benchmarks of ED on two sequences of 4 000 nucleotides with a security parameter of 127 bits took 7.08 s run-time and 2.04 GB communication. In contrast, our algorithm requires only 65 ms run-time (108× faster) and 8 MB communication (261× less) for the same setting and the width of the LUT ω=20.

Table 1: Notation used in our paper.

Parameters	
ω	Look-up table width
N	Number of sequences
n	Sequence length
b	Block size
b'	Padded block size
t	Number of blocks
ℓ	Block bit-length
ψ	Number of data providers
β	Bit-length for the distance values s.t. no overflow occurs
A, C, G, T	Nucleotides: Adenine, Cytosine, Guanine, Thymine
K, M, G, T	Powers of 10: kilo (10^3), mega (10^6), giga (10^9), tera (10^{12})
Notation from [DSZ15]	
$l[i]$	Operator for referencing element #i in list l
$l.e$	Operator for accessing element e in list l
$x \wedge y$ and $x \oplus y$	Bit-wise *AND* and *XOR* operation
A, B, Y	Sharing types: **A**rithmetic, **B**oolean, **Y**ao
$\langle x \rangle_i^t$	Share of value x in sharing type t held by party i
$\mathsf{Shr}_i^t(\mathsf{x})$	Sharing function for value x by party i in sharing type t
$\mathsf{Rec}(\langle \mathsf{x} \rangle_0^t, \langle \mathsf{x} \rangle_1^t)$	Reconstruction function for value x from both shares
$\langle z \rangle^t = \langle x \rangle^t \odot \langle y \rangle^t$	Operations on shares, $\odot \colon \langle x \rangle^t \times \langle y \rangle^t \mapsto \langle z \rangle^t$
$\langle x \rangle^t = s2t(\langle x \rangle^s)$	Conversion from sharing type s to sharing type t
$\langle 0 \rangle^t$, $\langle 1 \rangle^t$, $\langle n \rangle^t$	Secret-shared constant 0, 1, and n, respectively
$\langle \mathbf{F}(\cdot) \rangle^t$	Secret-shared constant of locally computed function $\mathbf{F}$
System Model	
T_0, T_1	Semi-trusted third parties that perform SMPC
$P_1, \ldots, P_\psi$	Data providers that contribute genomic data
C	Client

Mahdi et al. [MHM17] securely computed the Hamming distance for SSQ, which is an error-prone metric for measuring the distance between genome sequences, because the sequences are compared bit-wise and, thus, any deletions and additions, which cause shifts in the genome sequence, result in severe errors.

The authors of [CHW18] design a protocol for privacy-preserving SSQs based on [AHLR18] using homomorphic encryption in an outsourcing scenario with two non-colluding, semi-honest parties, which provides the same security guarantees as our model. As we show in Section 5.1, our protocol outperforms theirs by more than factor 24 000× in terms of run-time and by at least factor 16× in terms of communication in the same setting.

The privacy of other applications of genomics were also addressed in the literature, e.g., outsourcing of genome data storing [SLH+17], pattern matching [WSLH17], genome sequence queries [DHSS17], and Genome-Wide Association Studies (GWAS) [BMA+18, TWSH18, BKLS18, CWB18], see [MMDC19] for a good survey on genomic privacy.

3 PRELIMINARIES

In this section, we explain the basics underlying our constructions. Our notation is summarized in Table 1.

3.1 Genomic Primer

Deoxyribonucleic Acid (DNA) is contained in the cells of each living individual and encodes genome information. Based on DNA, individuals develop different phenotype traits — observable differences between individuals, e.g., hair or eye color. The basic components that form DNA are called nucleotides. There exist four of them: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T), which can be encoded in $\log_2 4 = 2$ bits. In our model, however, we require a dummy character for padding blocks of alleles to the predefined global block size, which yields $\lceil \log_2 5 \rceil = 3$ bits. DNA consists of multiple long sequences called chromosomes, which are built as sequences of pairs of nucleotides (e.g., "AT CG AA ...").

The human DNA consists of 3.5 billion base pairs from which only 0.1 % vary among individuals [Eur17]. The variations of single nucleotides in specific regions (called loci, singular locus) of a chromosome are called alleles. The genes, each allele of which is represented to some minimal degree in the population (e.g., more than 1 %), are called Single-Nucleotide Polymorphisms (SNPs).

3.2 Similar Sequence Queries (SSQs)

The approach of SSQs is used for finding the sequences that are most similar to the analyzed query. This approach can be used, for example, for finding individuals that are genetically very similar to a patient in order to better analyze the health conditions of the patient. This leads to more precise medical diagnoses based on the additional information provided by genetically similar individuals.

A precise SSQ algorithm requires the computation of the Edit Distance (ED) [Lev66], which measures how different two sequences are by finding the minimum number of deletions, additions, and substitutions that are required to transform one string into another. ED has $O(n^2)$ computation complexity, where $n = \mathbf{max}(n_s, n_q)$, n_s is the length of the sequence, and n_q is the length of the query. There exist other distance metrics for measuring the similarity of the sequences, but they are generally suboptimal, e.g., Hamming distance is not a good choice because it compares sequences bit by bit and thus any additions and deletions in the genome lead to large errors. To avoid heavy computations of ED, a few approximations have been developed, e.g., [WHZ+15, AHLR18]. For more details see Section 2.

This work focuses on the ED approximation of Asharov et al. [AHLR18], since, in contrast to [WHZ+15], it can handle high-divergence data (the authors of [AHLR18] empirically show this for up to 10 % variability between individual genomes). An example of computing this ED approximation is given in Figure 1. It works as follows: first, the sequences in the database are aligned to a public reference genome and split into blocks of predefined size. Then, the statistical distribution of the sequences in the database is used to construct a Look-Up Table (LUT) containing the most frequently observed block values and their distances to each other. Afterwards, the value of the block i in the query is compared to each entry of the i-th row of the LUT (the comparison is performed only once for one database), and based on the comparison result pre-computed distances are either selected as output or set to 0. Here, the sum of all outputs yields either the correct distance or 0 in the case of an error (this outcome is rare and influences the overall result only very slightly [AHLR18]). The last step is to sum up the distances of all blocks which results in the approximated distance between the query and the sequence. After computing the ED to all sequences in the database, they are used to find the indices of the k most similar sequences.

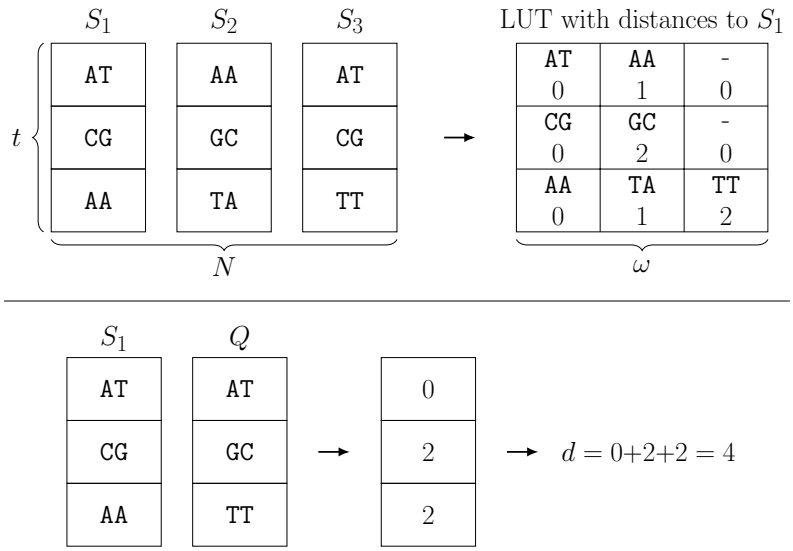

Figure 1: Example for computation of the Edit Distance approximation of [AHLR18]. Here, a Look-Up Table (LUT) for Sequence S_1 is precomputed in the clear based on the distribution of values in all sequences S_1, S_2, and S_3 each containing t blocks of size b=b'=2 alleles (top). In more detail, a LUT contains precomputed distances to all observed block values, e.g., in the third row block AA in S_1 has distances 0 to itself, 1 to TA in S_2, and 2 to TT in S_3. After the LUT construction, the LUT and the pre-computed distances for S_1 are used for computing the Edit Distance d between query Q and S_1 (bottom).

For managing LUTs in the outsourcing scenario, we store the LUTs of all data providers and use them in the ED computation of the corresponding genome sequences in the respective databases, which is a very promising approach in terms of efficiency. The efficiency of this approach grows with N/ψ (the number of sequences N divided by the number of institutions ψ), and in a real-world scenario we expect a small to medium number of data providers ψ that contribute a large number of sequences N. We discuss further LUT management options in Section 4.5.

Family Search from the Similar Sequence Query Protocol. Our SSQ protocol can also be extended for finding one's family. Consider a scenario where a large number of individuals possess their digital genome sequences. They are willing to contribute their data to a common database that can be used to perform family search. For this, they secret-share their genome sequences and compute distances to a public LUT (this can be prepared by a public authority and is the same as the reference genome), which are then used in the SSQ protocol. However, instead of computing k-Nearest Neighbors (k-NN), we can blind the indices that correspond to a big distance to the query (greater than some threshold T, e.g., at most 5 % difference). The result of this protocol is the set of indices of all similar sequences in the database.

3.3 Secure Multi-Party Computation (SMPC)

SMPC allows parties $P_1, \ldots, P_n$ to securely compute a function $f(x_1, \ldots, x_n)$ on their respective inputs without revealing the inputs to each other, i.e., one or more parties learn the result of f, but no intermediate values.

The first approaches to SMPC were proposed in the late 1980s (see, e.g., [Yao86, GMW87]). Although SMPC was first believed to be impractical, with the further progress on SMPC optimization and computer hardware improvements it is nowadays possible to solve complex problems using SMPC within seconds or minutes. SMPC can be conducted considering different adversary models. The two most common adversary models are passive (honest-but-curious) and active (malicious) adversaries. Whereas passive adversaries follow the protocol specification but try to learn as much information as possible from the information they obtain, active adversaries can arbitrarily deviate from the protocol. In this work, as in most previous works in this area [WHZ+15, AAAM17, AHLR18, ZH17, MHM17, CHW18] — to name just a few — we concentrate on protocols with security against passive adversaries that are much more efficient than actively secure protocols and provide sufficient security for settings where curious insiders want to learn additional information from the protocol runs without actively interfering with it. The major difficulty in the use of SMPC is the need of extensive knowledge of cryptography, circuit design, and algorithm complexity for constructing efficient privacy-preserving protocols.

3.3.1 Oblivious Transfer. Oblivious Transfer (OT) is an important building block of many SMPC protocols. In 1-out-of-2 OT, the sender has two messages m_0 and m_1 as input, and the receiver inputs a choice bit c. As output, the receiver receives the message of its choice m_c without learning m_{1-c}, and the sender does not learn c.

Public key-based OT protocols, e.g., [NP01], achieve thousands of OTs per second and OT extension allows using mainly symmetric key primitives resulting in millions of OTs per second [IKNP03,

ALSZ13]. There also exist other variants of OT, such as Random Oblivious Transfer (R-OT) [NNOB12, ALSZ13] and Correlated Oblivious Transfer (C-OT) [ALSZ13]. In C-OT, the sender has m as input, and the receiver has b as input. The outputs of the parties are as follows: the sender receives a random m_0 as output, and the receiver receives $m_0 + bm$ as output, where $m = m_1 - m_0$ in $\mathbb{Z}_{2^\ell}$ and ℓ is the bit-length of the values. This variant of OT improves the communication of the protocol (especially for large ℓ), where instead of $\kappa + 2\ell$ bits only $\kappa + \ell$ bits are sent in the C-OT extension, where κ is the symmetric security parameter, see [ALSZ13].

3.3.2 ABY Framework. We use the ABY framework [DSZ15], which implements state-of-the-art optimizations for Secure Two-Party Computation (STPC), i.e., SMPC with n=2 parties, and is secure against passive adversaries. It enables privacy-preserving algorithms using three different STPC protocols called sharings: Arithmetic sharing (a generalization of the GMW protocol [GMW87] to unsigned integers), Boolean sharing (the Boolean GMW protocol [GMW87]), and Yao sharing (Yao's Garbled Circuits (GCs) [Yao86]). It also enables mixing these protocols to use particular STPC protocols for the parts of the computed function where they perform best. For notation used in this paper, please refer to Table 1.

Arithmetic Sharing. Arithmetic (also called Additive) sharing is performed on integer numbers in a ring $\mathbb{Z}_{2^\ell}$. Values are shared locally by subtracting random numbers as one-time-pads from the initial values, and afterwards one of the shares is sent to the other party. The reconstruction of the shares for the outputs is also straightforward: the parties exchange the shares and compute the sums of the single shares in the corresponding ring which yields the corresponding cleartext values. The main advantage of Arithmetic sharing over other sharings is that it allows local computation of addition mod 2^ℓ and cheap computation of multiplication mod 2^ℓ using Multiplication Triples (MTs) [Bea96] that can efficiently be precomputed using OT extension [DSZ15]. The drawback of this sharing is that it does not allow to trivially perform other more complicated operations. For example, secure comparisons are very expensive in Arithmetic sharing.

Boolean Sharing. In ABY, Boolean sharing stands for the GMW protocol [GMW87]. This sharing is represented as a Boolean circuit where shares represent wires in the circuit. Similarly to Arithmetic sharing, the values are shared by performing the XOR operation with a random value. Reconstruction can be performed by applying XOR on both shares (the parties, again, exchange the shares), which will eliminate the random value and yield the cleartext value. The Boolean circuit is constructed by using XOR and AND gates (any computable function can be converted to a Boolean circuit using XORs and ANDs only). There is, however, a large difference in efficiency of evaluating these gates. Whereas XOR gates can be evaluated locally (due to the associative property of XOR), AND gates require communication during the evaluation. For secure evaluation of AND gates in ABY, Beaver's MTs [Bea96] that are precomputed using OT extension [ALSZ13] are utilized. The communication requirements of GMW can be further reduced at the cost of slightly higher computation [DKS+17]. Moreover, each AND-layer in the circuit adds an additional communication round to the protocol. Consequently, we are interested in shallow circuits for Boolean sharing.

Yao Sharing. Yao's Garbled Circuits (GCs) [Yao86] are denoted as Yao sharing in ABY. Yao sharing includes all state-of-the-art enhancements, such as point-and-permute [MNPS04], free-XOR [KS08], fixed-key AES garbling [BHKR13], and half-gates [ZRE15]. Yao sharing, similar to Boolean sharing, can be used to securely evaluate a Boolean circuit. It is split into two phases: an input-independent setup phase and an input dependent online phase. In the setup phase, the party called garbler garbles the circuit and sends it to the other party called evaluator. The parties then proceed to the online phase, where only the evaluator's inputs have to be obliviously transferred via precomputed OTs [Bea96] and the evaluator can compute the garbled result locally. For reconstructing the results on the evaluator's side, the garbler sends the output keys for the corresponding shares to the evaluator, and for the garbler's side, the evaluator sends the output keys to the garbler. Yao sharing has a constant number of rounds, i.e., it does not depend on the circuit depth, and therefore generally is better suited for high-latency networks than Boolean sharing. On the other hand, Yao sharing requires more computation and communication than Boolean sharing.

SMPC Protocol Conversion. It is clear from the description of the aforementioned sharings that the choice of particular sharing is not trivial even for relatively simple tasks. To solve this problem, ABY allows to mix the protocols by implementing efficient algorithms for converting between the three different sharing types. Although conversions imply some costs, they may result in better overall performance as shown in [DSZ15]. The partitioning can even be done automatically [BDK+18].

OT-Based Multiplication. We extend the OT-based multiplication algorithm of [AHLR18] for multiplying an additively secret-shared value $\langle v\rangle^A$ in Arithmetic sharing by a secret-shared bit $\langle b\rangle^B$ in Boolean sharing. Observe that we want to compute $\langle b\rangle^B \cdot \langle v\rangle^A = (\langle b\rangle_0^B \oplus \langle b\rangle_1^B)(\langle v\rangle_0^A + \langle v\rangle_1^A)$, which we reformulate as

$$\langle b\rangle_0^B\langle v\rangle_0^A + \langle b\rangle_0^B \cdot (-1)^{\langle b\rangle_1^B} \cdot \langle v\rangle_1^A + \langle b\rangle_1^B \cdot (-1)^{\langle b\rangle_0^B} \cdot \langle v\rangle_0^A + \langle b\rangle_1^B\langle v\rangle_1^A.$$

Whereas $\langle b\rangle_0^B\langle v\rangle_0^A$ and $\langle b\rangle_1^B\langle v\rangle_1^A$ can be computed locally by the respective parties, for the computation of $\langle b\rangle_i^B \cdot (-1)^{\langle b\rangle_{i-1}^B} \cdot \langle v\rangle_{i-1}^A$: $i \in \{0,1\}$ interaction is required between P_0 and P_1. Note though that the right part, i.e., $(-1)^{\langle b\rangle_i^B} \cdot \langle v\rangle_i^A$, is computed locally. For the remaining two multiplications, we utilize two C-OTs [ALSZ13], where P_i inputs $\langle b\rangle_i^B$ and P_{1-i} inputs $(r, (-1)^{\langle b\rangle_{i-1}^B} \cdot \langle v\rangle_{i-1}^A + r)$, where r is a random value, and vice versa. P_{1-i} then sets its share to $-r$. As a result, the parties compute a valid Arithmetic share $\langle b\cdot v\rangle^A$ of value $b \cdot v$. Compared to the state-of-the-art multiplication protocol by Patra et al. [PSSY20], the online communication in our protocol is only marginally higher (their 2ℓ vs. our $2\ell + 2$), whereas our communication in the setup phase is by a factor of $(2.5 + 2.5\ell/\kappa)\times$ lower for generic (non-amortized) multiplication. With the optimization technique of the online communication rounds of C-OT from [BDST20], our protocol requires the same round complexity of one round as the protocol in [PSSY20].

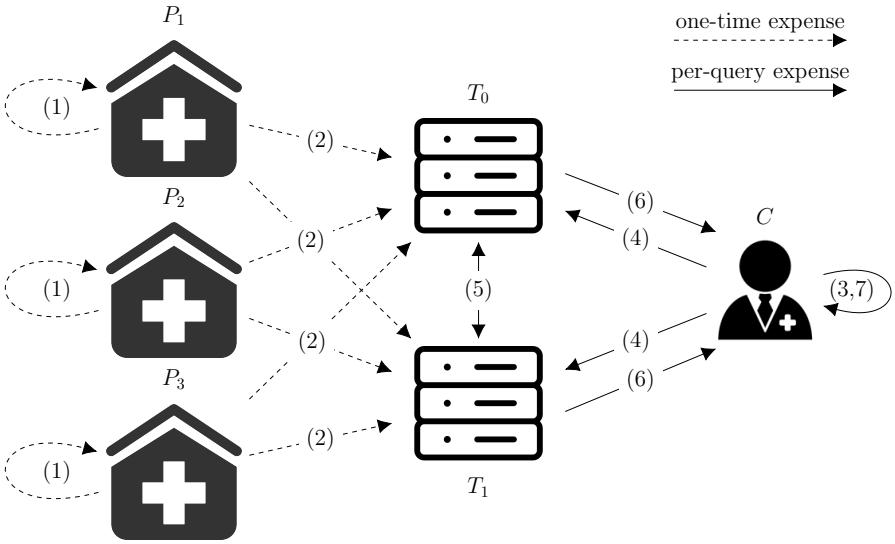

Figure 2: Privacy-preserving Similar Sequence Query system model with three medical institutions I_1, I_2, and I_3 that contribute their secret-shared genomic data to two Semi-Trusted Third Parties T_0 and T_1, and a client C who queries the secret-shared database. The communication between all parties is protected with a secure channel, e.g., TLS. See Section 4.1 for more details.

Note that the original publication [ST18, ST19] erroneously depicted a multiplication of two *Boolean* shares, which does not work for the multiplication of a Boolean with an arithmetic share. The description is fixed in this version of the paper.

Single Instruction Multiple Data (SIMD) gates. Wrapping of secret-shared data in container classes is very memory-consuming. Moreover, this adds additional overhead for managing and initialization of memory to the protocol. As a solution to this problem, SIMD gates have emerged [SZ13] and are also implemented in ABY. These gates are constructed as gates for evaluating arrays of secret-shared values rather than single values. This approach significantly reduces the RAM requirements and the online run-times of the protocols.

k-Nearest Neighbors. For finding the k most similar sequences in the Similar Sequence Query (SSQ) protocol, we utilize ABY's functionality for computing k-Nearest Neighbors (k-NN) [JLL+19], which improves over the work of Songhori et al. [SHSK15] in terms of the circuit size. This k-NN implementation is by about a factor of 5× more efficient than the one used in [AHLR18], e.g., for a database of size 500 [AHLR18] required 505 825 AND gates for computing the k-NNs, whereas we require only 92 500 AND gates.

4 OUR PRIVACY-PRESERVING SSQ PROTOCOL

Here, we describe the system model of our privacy-preserving protocol for Similar Sequence Queries (SSQs), the protocol itself, and we analyze its security.

4.1 System Model

The main idea of our protocol is to secret-share the database aggregated from data of multiple data providers between two non-colluding Semi-Trusted Third Parties (STTPs). We depict our system model in Figure 2. The communication between all parties is performed over a secure channel (e.g., TLS). Note that our protocol alternatively can be run directly between a server and a client with approximately the same efficiency (the outsourcing scenario is beneficial for data aggregation, but has the same efficiency in the querying phase). In our protocol, we have the following parties:

- Data providers (e.g., medical institutions) $P_1, \ldots, P_\psi$ that securely contribute their genome sequences to the outsourced database in a secret-shared form.
- A client C who privately queries the database with a genome sequence for finding the most similar sequences in the outsourced database.
- Two non-colluding Semi-Trusted Third Parties (STTPs) T_0 and T_1 who obliviously compute the Similar Sequence Query (SSQ) protocol on the client's query and outsourced database.

We choose two STTPs because it is the most practical and affordable model in a real-world setting, since each STTP has to be operated and maintained by different teams, and the servers must have completely different software stacks, which in total implies high costs. For running the two STTPs, one must choose two distinct organizations that have a high motivation to not collude, e.g., if they significantly loose in value/reputation if caught cheating. We can think of the following organizations: (i) health ministry, (ii) research institutes, or (iii) cloud service providers. This

outsourcing model has been widely used in the literature, e.g., in [CDC+17, ADS+17, TWSH18].

Our protocol consists of the following steps (see Figure 2):

A **Initialization**
(1) Each data provider P_i locally secret-shares its genomic data and Look-Up Table.
(2) I_i sends the shares to the Semi-Trusted Third Parties T_0 and T_1, respectively.

B **Querying**
(3) Client C locally secret-shares its query Q.
(4) C sends its secret-shared query to T_0 and T_1.
(5) T_0 and T_1 obliviously compute the SSQ protocol on the secret-shared database and the client's secret-shared query using STPC.
(6) T_0 and T_1 send the resulting output shares containing secret-shared indices of the most similar sequences in the database to C.
(7) C locally reconstructs the result from the output shares.

In contrast to querying the database (querying phase), aggregating the database from different sources (initialization phase) is a one-time expense. Data providers contribute their data only once, and any changes in the outsourced database are required for updates only. Since data providers can send their secret-shared sequences in any order and from different preprocessing sets (though using the same global parameters), the database can be updated without any further preprocessing steps on the STTPs' and data providers' side. Additional data providers in the protocol do not add any significant overhead because of the following: (i) the initialization phase is a one-time expense, (ii) the initialization phase is computed in parallel for all data providers, (iii) the STTPs do not apply any further preprocessing steps but only store the received shares, which has a very small overhead.

Client's Communication and Computation. Our model significantly reduces the amount of communication and computation performed by the client compared to the direct application of SMPC. More detailed, the client sends only 2× the amount of information compared to the non-private cleartext protocol. Moreover, the client does not require cryptographic operations in the protocol but only very efficient XOR and addition mod 2^ℓ operations. This makes our protocol even applicable for weak clients using mobile devices.

4.2 Privacy-Preserving Approximated Edit Distance

Our protocol for securely computing the Edit Distance (ED) between a genome sequence stored in the outsourced database and a client's query utilizes the idea of Asharov et al. [AHLR18] for improving the efficiency of computation by approximating ED using Look-Up Tables (LUTs) (see Section 3.2).

We extend the two-party protocol of [AHLR18] to the outsourcing scenario and carefully optimize the implementation using a mix of different sharings and minimize costly operations, such as conversions between sharings and the operations that require interaction/heavy computations. Our detailed algorithm is given in Algorithm 1 and its data representation is given in Figure 3. The Similar Sequence Query (SSQ) algorithm is given in Algorithm 2.

$\langle distance \rangle^A \leftarrow$ **ED**(*seq, query*)

```
1 :  row_dist ← ∅
2 :  for i = 1 to seq.length do
3 :    ⟨row_sum⟩^A ← ⟨0⟩^A
4 :    for j = 1 to seq.LUT.width do
5 :      //For multiple ED computations, eq values are computed only once
6 :      //Next 2 lines are equal to e_{iω+j} == q_i ? d_{iω+j} : 0
7 :      ⟨eq⟩^B ← ⟨seq[i].LUT[j].value⟩^B == ⟨query[i]⟩^B
8 :      ⟨dist⟩^A ← OTM(⟨eq⟩^B, ⟨seq[i].LUT[j].dist⟩^A)
9 :      ⟨row_sum⟩^A ← ⟨row_sum⟩^A + ⟨dist⟩^A
10 :   row_dist.insert(⟨row_sum⟩^A)
11 :  for i = 2 to seq.length do
12 :    ⟨row_dist[1]⟩^A ← ⟨row_dist[1]⟩^A + ⟨row_dist[j]⟩^A
13 :  return ⟨row_dist[1]⟩^A
```

Algorithm 1: Privacy-preserving Edit Distance (ED) algorithm between a sequence *seq* containing a Look-Up Table with genomic variants and the corresponding distances, and a client's query containing genomic variants. OTM denotes Oblivious Transfer-Based Multiplication.

$ids \leftarrow$ **SSQ**($sequences, query, k$)

```
1 :  dists ← ∅
2 :  ids ← ⟨1⟩^Y, ..., ⟨sequences.size⟩^Y
3 :  for i = 1 to sequences.size do
4 :    ⟨dist⟩^A ← ED(sequences[i], query))
5 :    dists.append(A2Y(⟨dist⟩^A))
6 :  ids ← k-NN(dists, ids, k)
7 :  return ids
```

Algorithm 2: Privacy-preserving Similar Sequence Query (SSQ) algorithm between a query and sequences of genomic data in the outsourced database. *k*-NN denotes the *k*-Nearest Neighbors algorithm.

More detailed, we improve the protocol of [AHLR18] by using more lightweight GMW [GMW87] instead of GCs [Yao86] for comparisons, Correlated Oblivious Transfers (C-OTs) [ALSZ13] instead of general OTs, and a more efficient k-NN algorithm in Yao sharing. Although we use a more efficient C-OTs, we require two C-OTs instead of one OT, which is due to the fact that the Semi-Trusted Third Parties (STTPs) do not possess the LUTs in cleartext. However, the cost for OTs remain approximately the same as in the protocol of [AHLR18] for large databases (less than 1 % overhead for a database with 10 000 sequences). In contrast to [CHW18], we compute most of the functionalities using generic protocols which enables arbitrary extensions of our protocol. A possible and cheap extension of our protocol would be a thresholding protocol that reveals only those sequence indices that have distances smaller than some threshold T. This protocol has the advantage that it dispenses with the need of finding the k most similar sequences, which improves the complexity from $O(kN)$ to $O(N)$.

We optimize the SSQ algorithm by mixing different SMPC protocols. First, the blocks of the query and LUT are secret-shared in Boolean sharing. Boolean shares are then used to compare the block values of the query with the block values of the LUT, namely, the block value i of the query with each of the ω block values of the LUT in the row i. Afterwards in Arithmetic sharing, shared distances or zeros are chosen depending on the comparison results between the query and LUT. For this, we use two C-OTs for multiplying the comparison result $r \in \{0, 1\}$ with the distances in Arithmetic sharing. Since all distances are valid for the sequence (each block yields either a valid block distance or zero) and only need to be summed up for resulting in the total distance between the query and sequence, we perform — free in Arithmetic sharing — addition operations for all distances in the sequence.

4.3 Privacy-Preserving Similar Sequence Queries

In this work, we consider a system model where the client wants to find k genome sequences that are most similar to its query among the sequences stored in the outsourced database. Here, we proceed as follows: the distances to single sequences are first calculated using the Similar Sequence Query (SSQ) algorithm, and afterwards, the distances are used along with the corresponding IDs for finding the k closest distances using the k-NN algorithm (see Algorithm 2).

Choice of the Algorithm for Finding k Most Similar Sequences. Generally, we can think of two possible methods for efficiently finding the k most similar sequences: the k-Nearest Neighbors (k-NN) algorithm and sorting networks. The first has a small AND-size for small k, but a large AND-depth of $O(n)$, because the algorithm is difficult to parallelize, whereas the second have a logarithmic AND-depth and AND-size of $O(n \log^2 n)$, which is independent of k. Since the resulting circuit size of a sorting network is by an order of magnitude larger for small k, it is practically less efficient even in Boolean sharing than k-NN in Yao sharing, so we use the efficient algorithm for finding indices of the closest distances (k-NN with precomputed distances) in Yao sharing that is already implemented in ABY [JLL+19]. In EPISODE, we utilize the highly efficient k-NN implementation in ABY with $Nk(2\ell + \lceil \log_2 N \rceil)$ AND gates, where N is the number of sequences, k is the number of most similar sequences, and ℓ is the bit-length of the distances. Since the distances are shared in Arithmetic sharing, we first convert them to Yao sharing.

Communication. Our SSQ algorithm consists of three parts: comparison, OTs, and k-NN. The first part requires $6t\psi\omega\kappa b'$ bits to be transferred (see Table 1 on p. 3 for the explanation of all parameters). For our C-OT-based protocol, $2t\omega(N \cdot \mathbf{NPoT}(\lceil \log_2(tb') \rceil + \kappa))$ bits of communication are required. $\mathbf{NPoT}(x)$ (Next Power of Two) denotes a function that takes a rational number as input and outputs the smallest number that is a power of two and is equal or greater than x. For the last part (the k-NN algorithm), we require $2Nk\kappa(2\lceil \log_2(tb') \rceil + \lceil \log_2 N \rceil)$ bits of communication. The total communication is approximately $2t\omega(3\psi\kappa b' + N \cdot \mathbf{NPoT}(\lceil \log_2(tb') \rceil)) + 2Nk\kappa(2\lceil \log_2(tb') \rceil + \lceil \log_2 N \rceil)$ bits.

4.4 Security Analysis

Our protocol is based on the outsourcing protocol described in [KR11], which gives a generic construction and a security proof for turning an N-party SMPC protocol into a secure outsourcing scheme where data is oursourced to N non-colluding Semi-Trusted Third Parties (STTPs).

We instantiate SMPC with the $N = 2$-party ABY framework [DSZ15]. Our protocols implement the algorithm for SSQs of [AHLR18]. Our protocols are even secure against malicious data providers and clients (who all send a single message in the protocol), and secure against semi-honest STTPs. Since data providers do not receive any outputs, their malicious input cannot affect the privacy of the protocol. Moreover, any changes to the client's single input message correspond to a different input to the ideal functionality, which yields security against malicious clients.

Theorem 4.1. *Assume that the protocols implemented in ABY [DSZ15] are secure against semi-honest adversaries and the two STTPs are semi-honest and non-colluding. Then our protocols securely implement the algorithm of Asharov et al. [AHLR18].*

Proof (sketch). The proof follows immediately from the proof in [KR11] and the fact that the protocols run between the two STTPs are secure against semi-honest adversaries and operate on secret-shared data. More detailed, consider shares $\langle x \rangle_i^t$ and $\langle x \rangle_{1-i}^t$ of an input value x shared by Party P_i in sharing type $t \in \{A, B, Y\}$ corresponding to Arithmetic, Boolean, and Yao sharing, respectively. Party P_{1-i} gets $\langle x \rangle_{1-i}^t$ from P_i. Party P_{1-i} cannot derive any information from $\langle x \rangle_{1-i}^t$ without knowing $\langle x \rangle_i^t$. Similarly, the security of the STPC protocols and STPC protocol conversions that ABY is based on guarantees that no information can be derived from the intermediate shares (i.e., secret-shared result of any operation on shares). The C-OT-based multiplication (which is a straightforward extension of [AHLR18]) is performed on secret-shared values and thus does not reveal any information about the cleartext values. The STTPs do not learn any new information from the secret-shared outputs. By aggregating the joint database, the data providers have no outputs and thus cannot infer any information from the protocol. In view of the above arguments: (i) no semi-honest STTP can obtain any new information on the genome data from the shares, (ii) no actively corrupted server can obtain any information on the genome data contained in the client's query or other server's database, and (iii) no actively corrupted client can obtain new information on the genome data contained in the database of any of the servers.

4.5 Data Aggregation from Multiple Data Providers

We see three possible approaches of aggregating data in the outsourced database:

(1) The most intuitive approach is to attach a Look-Up Table (LUT) to each genome sequence, which was used in [CHW18]. This approach dispenses with the need of LUT management for the Similar Sequence Query (SSQ) protocol and when updates occur. Since each genome sequence has its own LUT and is thus independent of other sequences, the data provider only has to upload the secret-shared new sequence and the corresponding secret-shared LUT to the

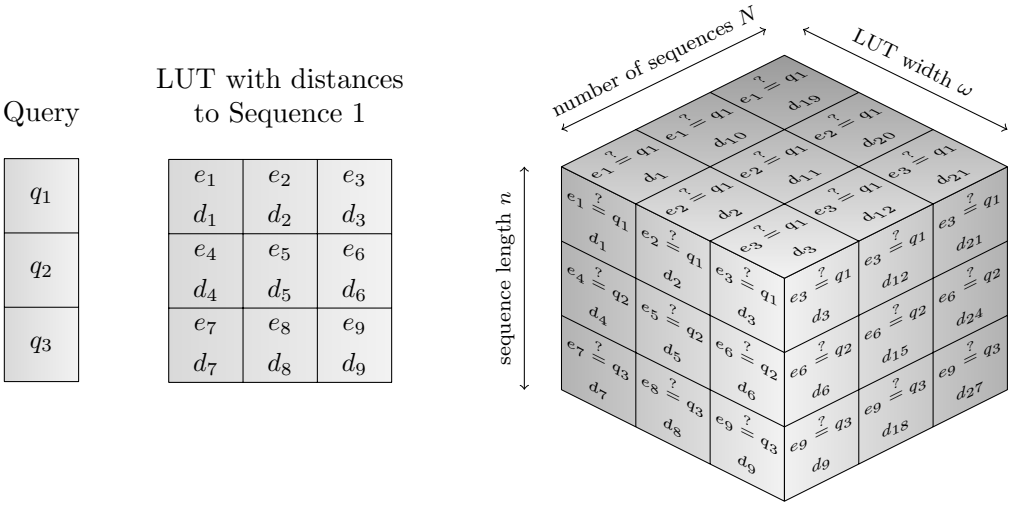

Figure 3: Data representation of the Similar Sequence Query algorithm. For computing the Edit Distance to three sequences, the values of the blocks in the client's query $\{q_1, q_2, q_3\}$ are compared with the precomputed values in the secret-shared Look-Up Tables (LUTs) $\{e_1, \ldots, e_9\}$. Based on the comparison, we process the precomputed distances $\{d_1, \ldots, d_9\}$, $\{d_{10}, \ldots, d_{18}\}$, $\{d_{19}, \ldots, d_{27}\}$ to the Sequences 1, 2, and 3, respectively. For example, for Sequence 1 in the first block e_1 is compared with q_1; if they are equal, the precomputed distance d_1 is returned, and 0 otherwise. The computation of the LUT is parallelized using Single Instruction Multiple Data gates.

Table 2: Run-time and communication comparison of our algorithm with that of Asharov et al. [AHLR18] with sequence length n=3 470, number of nearest sequences k=5, block length b=4, padded block length b'=16, and number of data providers ψ=1 for different parameters N (number of sequences) and ω (Look-Up Table width). The preprocessing stage is not included in the total run-time. A plot is given in Figure 4 in Appendix A.

N	ω	Run-time (localhost / LAN) in s			Communication in MB		
		[AHLR18]	Ours	Improvement	[AHLR18]	Ours	Improvement
1 000	25	6.03 / -	1.07 / 1.89	5.6× / -	180	130	1.3×
2 000	30	14.11 / -	2.20 / 4.07	6.4× / -	340	268	1.2×
4 000	35	31.60 / -	4.83 / 9.02	6.5× / -	660	571	1.1×

Table 3: Run-time comparison of our Edit Distance algorithm with that of Cheng et al. [CHW18] with sequence length n, Look-Up Table width ω=20, and block size b=2.

n	Run-time (LAN)		
	[CHW18]	Ours	Improvement
10	1.2 s	2.1 ms	571×
20	2.2 s	3.0 ms	733×
30	3.4 s	3.1 ms	1 096×
40	4.7 s	3.1 ms	1 516×
50	6.0 s	3.2 ms	1 875×

Semi-Trusted Third Parties (STTPs). This approach is in particular effective if there is a very large number of participating data providers in the protocol. The best-case scenario for this setting is when ψ=N, i.e., each institution uploads a single genome.

(2) A more realistic approach is to keep one LUT for each database. This approach has the advantage that the number of data providers is commonly not very large, e.g., ten big institutions is a realistic scenario. Since the STTPs know how many and which sequences came from which data provider, this approach does not violate privacy of the protocol by using predefined LUTs for particular genome sequences. Using this approach, only ψ comparisons with LUTs have to be performed, which is a small overhead if the number of sequences N is large. Due to the performance advantages over other options, we choose this approach in our SSQ protocol.

(3) The least realistic approach is to aggregate LUTs of multiple data providers. For this, the institutions count the frequencies of *all possible* alleles for a block in their database (the most commonly used block size in [AHLR18] is 16). For example, a frequency table for a 16-allele block would yield

4^{16}=4.3 billion values. These values first have to be aggregated from multiple databases. Afterwards, the w (width of the LUT) maxima have to be found from the dataset, e.g., using the k-Nearest Neighbors (k-NN) algorithm. This approach is performed for each block (there are 15 million blocks in a whole-genome sequence). Therefore, this approach is impractical and we do not discuss it further.

5 EVALUATION

We implemented all our protocols using the ABY framework [DSZ15]. We run our two Semi-Trusted Third Parties (STTPs) each on a standard PC equipped with an Intel Core i7-4770K 3.5 GHz processor and 32 GB RAM. They are connected via a 1 Gbit/s network with an average latency of 0.1 ms. All protocols are instantiated with a symmetric security parameter of 128 bit. We intentionally exclude the evaluation part for the preprocessing (extensively discussed in [AHLR18]) and aggregation of the databases (extensively discussed in [TWSH18]). Furthermore, the initialization phase (aggregation of the databases) is a negligible *one-time* expense and securing the communication channels using TLS does not add any significant overhead. Correctness and accuracy of the algorithms of Asharov et al. on real genomic data was shown in [AHLR18, Sect. 5]. Therefore, we use artificial data in our benchmarks because the performance of SMPC depends only on the size of the data, but not on concrete values.

5.1 Comparison with Cheng et al. [CHW18]

The most recent related work that covers privacy-preserving Edit Distance (ED) computations in the outsourcing scenario is [CHW18]. Unfortunately, the authors give the exact run-times only for the ED computation between two sequences in Table 3 of this paper. We compare our ED run-times in the system setting of [CHW18], i.e., the outsourcing scenario, with the run-times of their most efficient protocol. We achieve a significant run-time improvement over [CHW18] of 500× to 1 800× (see Table 3).

We can increase this even further when many sequences are computed in parallel. For this, we (optimistically for [CHW18]) approximate the benchmarking results of [CHW18, Figure 4 (a)] in Table 4 and in Figure 4 in Appendix A. As a result of the comparison, our algorithm outperforms that of Cheng et al. [CHW18] by more than factor 24 000× in run-time and by more than factor 16× in communication.

5.2 Comparison with Asharov et al. [AHLR18]

Here, we compare our privacy-preserving algorithm for ED and Similar Sequence Query (SSQ) with [AHLR18].

In Table 2 and in Figure 4 in Appendix A, we compare our algorithms in the benchmark setting of [AHLR18, Table 3], where both parties, client and server, are run on one machine (our protocol can trivially be applied for direct STPC between the client and server). In addition, we benchmark our algorithm in the LAN setting.

The authors of [AHLR18] do not detail their hardware setting, whereas we benchmark our algorithm on commodity hardware. Our algorithm outperforms that of Asharov et al. [AHLR18] by factor 5-6× in run-time which is due to more light-weight building blocks. Furthermore, our algorithm is still by factor 3× faster even on a LAN, compared to the localhost benchmarks of [AHLR18]. The communication of our algorithm is slightly lower. This is due to the more efficient C-OTs instead of general OTs (see Section 4.2) and a more efficient algorithm for finding k most similar sequences (see Section 3.3.2).

5.3 Batching the Execution for Large-Scale Benchmarks

For some of our large-scale benchmarks, we split the execution into multiple steps (i.e., we evaluate subcircuits instead of the entire circuit) because our STTPs ran out of RAM. We split our algorithms in a black-box way, i.e., without modifying the primitive protocols such as distance computation and k-Nearest Neighbors (k-NN). For the distance computation, this only increases the number of communication rounds because the computation is independent for each SNP, whereas for the k-NN algorithm we have to perform additional computation because the result depends on all input elements, which, however, turns out to be very cheap in terms of computation and does not add any significant overhead to the overall protocol. For example, for one million input genome sequences in total in the k-NN protocol with k=10 and 100 000 batch size, we have to perform 10 iterations with 100 000 input size and one iteration of k-NN on the joint output of size $10 \cdot k = 100$. Since k-NN has linear complexity in the number of inputs, the total overhead in the circuit size is $\sim 100/1\,000\,000 = 0.000\,1\,\%$, which is negligible. This also does not violate privacy, because the data and intermediate results all remain in secret-shared form and, hence, leak no information.

5.4 Large-Scale Benchmarks

For our large-scale benchmarks on thousands to millions of genomes, we define global parameters of the block size b=5, padded block size b'=16, number of data providers ψ=10, and the width of the LUT ω=30 (for better accuracy). The results of the benchmarks are given in Table 5 and in Figure 5 in Appendix A. As can be seen in the table, practical large-scale privacy-preserving SSQs on sequences of medium lengths are possible. For the sequence lengths n=1 K–10 K and any number of sequences, and n=100 with the number of sequences N=100 K, the run-times are always in the order of seconds. For all other parameters, the run-times are in the order of minutes (even for databases with N=1 M sequences). A few minutes is a reasonable delay in practice for SSQ which shows real-world applicability of our protocol to large-scale SSQ.

5.5 Whole-Genome Benchmarks

For our whole-genome benchmarks, we set the genome sequence length to n=75 M (the same as in [WHZ+15]) and the LUT width to $\omega \in \{10, 20\}$. As shown in [CHW18], a LUT width reduction slightly reduces accuracy, but significantly reduces the communication and computation of the protocol. The results of our benchmarks are given in Table 6 and in Figure 6 in Appendix A.

As can be seen in the table, running our protocol on a few hundred whole-genome sequences is practical. For example, a protocol run on up to N=1 000 sequences takes just a few hours. However, if we extrapolate the results to the dataset of [WHZ+15] with N=1 M sequences, we would require months to execute the protocol. Thus,

Table 4: Run-time and communication comparison of our Similar Sequence Query algorithm with that of Cheng et al. [CHW18] with sequence length n=500, Look-Up Table width ω=20, block size b=5, number of blocks t=20, number of most similar sequences k=10, and number of sequences N. A plot is given in Figure 4 in Appendix A

N	Run-time (LAN)			Communication		
	[CHW18]	Ours	Improvement	[CHW18]	Ours	Improvement
100	25 min	62 ms	24 193×	50 MB	3 MB	16×
200	50 min	96 ms	31 250×	100 MB	4 MB	25×
300	80 min	132 ms	36 363×	150 MB	5 MB	30×
400	105 min	171 ms	36 842×	200 MB	6 MB	33×
500	135 min	207 ms	39 130×	250 MB	7 MB	35×

Table 5: Large-scale benchmarks of our Similar Sequence Query algorithm for N sequences of length n, LUT width ω=30, number of data providers ψ=10, number of most similar sequences k=10, block size b=5, and padded block size b'=16. A plot is given in Figure 5 in Appendix A.

N	n	Run-time	Communication
1 K	100	0.5 s	21.1 MB
10 K	100	3.6 s	138.9 MB
100 K	100	24.2 s	1.4 GB
1 M	100	4.0 min	14.9 GB
1 K	1 K	1.2 s	129.5 MB
10 K	1 K	5.9 s	457.8 MB
100 K	1 K	1.1 min	3.9 GB
1 M	1 K	8.3 min	38.8 GB
1 K	10 K	8.5 s	1.2 GB
10 K	10 K	22.4 s	3.5 GB
100 K	10 K	4.1 min	26.6 GB
1 M	10 K	39.6 min	257.2 GB

Table 6: Run-times and communication for Similar Sequence Query on whole-genome genome sequences based on the computation of sub-sequences of smaller lengths with the following parameters: number of sequences N, Look-Up Table widths ω, sequence length n=75 M, block size b=5, padded block size b'=16, number of most similar sequences k=10, number of data providers ψ=10, number of blocks t=15 M, and bit-length of the distances β=$\lceil \log_2(tb') \rceil$=28. A plot is given in Figure 6 in Appendix A.

N	ω	Run-time	Communication
10	10	2.9 h	2.3 TB
100	10	3.2 h	2.4 TB
1 000	10	6.8 h	3.5 TB
10 000	10	1.2 d	14.3 TB
10	20	5.7 h	4.6 TB
100	20	6.3 h	4.8 TB
1 000	20	12.5 h	7.0 TB
10 000	20	2.4 d	28.6 TB

we propose either to use our protocol for whole-genome runs with relatively small databases (a few hundred sequences) or to use high-performance hardware.

6 CONCLUSION

In this work, we designed, implemented, and evaluated EPISODE, a scalable protocol for distributed privacy-preserving Similar Sequence Queries (SSQs), which outperforms the state of the art by orders of magnitude. Our protocol for SSQ is based on the approximation of Edit Distance (ED) computation of [AHLR18]. SSQ is performed on two Semi-Trusted Third Parties (STTPs) that obliviously compute indices of the k most similar sequences to the client's query. Our protocol is not only scalable, but it also substantially reduces the amount of communication and computation of the client. We implement our protocol using a mix of generic SMPC protocols and Correlated Oblivious Transfer (C-OT), which (i) improves the efficiency of our SSQ protocol by computing its parts using techniques that are most efficient for the particular tasks, which gives a greater than 20 000× speed-up compared to the most recent work of Cheng et al. [CHW18], and (ii) extend the protocol of Asharov et al. [AHLR18] for outsourcing while reducing its communication and computation overhead.

Acknowledgements. This work has been co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING, and by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP.

REFERENCES

[AAAM17] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure approximation of edit distance on genomic data. In *BMC Medical Genomics*, 2017.

[ADS+17] Gilad Asharov, Daniel Demmler, Michael Schapira, Thomas Schneider, Gil Segev, Scott Shenker, and Michael Zohner. Privacy-preserving interdomain routing at Internet scale. In *Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[AHLR18] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. In *Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[AKD03] Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2003.

[AL05] Mikhail J. Atallah and Jiangtao Li. Secure outsourcing of sequence comparisons. In *International Journal of Information Security*, 2005.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM SIGSAG Conference on Computer and Communications Security (CCS)*, 2013.

[Ayd16] Erman Ayday. Cryptographic solutions for genomic privacy. In *Financial Cryptography and Data Security (FC)*, 2016.

[BDK+18] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *CCS*, 2018.

[BDST20] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. MOTION - a framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, 2020. https://ia.cr/2020/1137.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *ACM Symposium on Theory of Computing (STOC)*, 1996.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Philip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[BKLS18] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Implementation and evaluation of an algorithm for cryptographically private principal component analysis on genomic data. *Computational Biology and Bioinformatics*, 2018.

[BMA+18] Charlotte Bonte, Eleftheria Makri, Amin Ardeshirdavani, Jaak Simm, Yves Moreau, and Frederik Vercauteren. Towards practical privacy-preserving genome-wide association study. *BMC Bioinformatics*, 2018.

[CDC+17] Marco Chiesa, Daniel Demmler, Marco Canini, Michael Schapira, and Thomas Schneider. SIXPACK: Securing internet exchange points against curious onlookers. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2017.

[CHW18] Ke Cheng, Yantian Hou, and Liangmin Wang. Secure similar sequence query on outsourced genomic data. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[Cor18] Zoë Corbyn. How taking a home genetics test could help catch a murderer, 2018.

[CWB18] Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 2018.

[DHSS17] Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler. Privacy-preserving whole-genome variant queries. In *Cryptology and Network Security (CANS)*, 2017.

[DKS+17] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[Eur17] European Bioinformatics Institute. Genome reference consortium human build 38, Ensembl release 91. http://dec2017.archive.ensembl.org/Homo_sapiens/Info/Annotation, 2017.

[FPI16] Roger Allan Ford and W. Nicholson Price II. Privacy and accountability in black-box medicine. *Michigan Telecommunications and Technology Law Review*, 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[HAHT15] Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. On non-cooperative genomic privacy. In *Financial Cryptography and Data Security (FC)*, 2015.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference (CRYPTO)*, 2003.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2008.

[JLL+19] Kimmo Järvinen, Helena Leppäkoski, Elena Simona Lohan, Philipp Richter, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. PILOT: Practical privacy-preserving Indoor Localization using OuTsourcing. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[Jon10] Thomas Jones. The rise of DNA analysis in crime solving, 2010.

[KR11] Seny Kamara and Marina Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*, 2011.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2008.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 1966.

[MHM17] Md Safiur Rahman Mahdi, Mohammad Zahidul Hasan, and Noman Mohammed. Secure sequence similarity search on encrypted genomic data. In *IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies*, 2017.

[MMDC19] Alexandros Mittos, Bradley Malin, and Emiliano De Cristofaro. Systematizing genome privacy research: A privacy-enhancing technologies perspective. 2019.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, 2004.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Annual International Cryptology Conference (CRYPTO)*, 2012.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Symposium on Discrete Algorithms (SODA)*, 2001.

[PF16] Alice B. Popejoy and Stephanie M. Fullerton. Genomics is failing on diversity. *Nature News*, 2016.

[PSSY20] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*, 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, 2015.

[SHSK15] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[SLH+17] João Sá Sousa, Cédric Lefebvre, Zhicong Huang, Jean Louis Raisaro, Carlos Aguilar-Melchor, Marc-Olivier Killijian, and Jean-Pierre Hubaux. Efficient and secure outsourcing of genomic data storage. *BMC Medical Genomics*, 2017.

[ST18] Thomas Schneider and Oleksandr Tkachenko. Episode: Efficient privacy-preserving similar sequence queries on outsourced genomic databases. In *Workshop on Privacy in the Electronic Society (WPES)*, 2018.

[ST19] Thomas Schneider and Oleksandr Tkachenko. EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs. In *ASIACCS*. ACM, 2019.

[SZ13] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC)*, 2013.

[TWSH18] Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[Wet17] Kris A. Wetterstrand. DNA sequencing costs: Data from the NHGRI Genome Sequencing Program (GSP), 2017. http://www.genome.gov/sequencingcostsdata.

[WHZ+15] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *ACM SIGSAG Conference on Computer and Communications Security (CCS)*, 2015.

[WSLH17] Bing Wang, Wei Song, Wenjing Lou, and Y. Thomas Hou. Privacy-preserving pattern matching over encrypted genetic data in cloud computing. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS)*, 1986.

[ZH17] Ruiyu Zhu and Yan Huang. Efficient privacy-preserving general edit distance and beyond. Cryptology ePrint Archive, Report 2017/683, 2017. https://ia.cr/2017/683.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2015.

A VISUALIZED BENCHMARKS

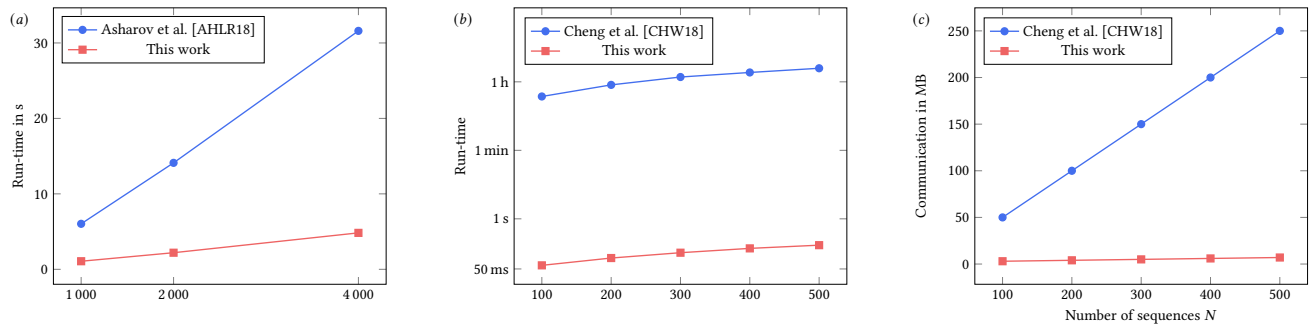

Figure 4: (a) A run-time comparison of our Similar Sequence Query algorithm with that of Asharov et al. [AHLR18] with sequence length n=3 470, number of nearest sequences k=5, block length b=4, padded block length b'=16, and number of data providers ψ=1 performed locally for different numbers of sequences N and LUT widths ω. (b,c) A run-time and communication comparison of our SSQ algorithm with that of Cheng et al. [CHW18] with n=500, b=5, k=10, ω=20, and number of blocks t=20. Numbers are given in Tables 2 (a) and 4 (b,c).

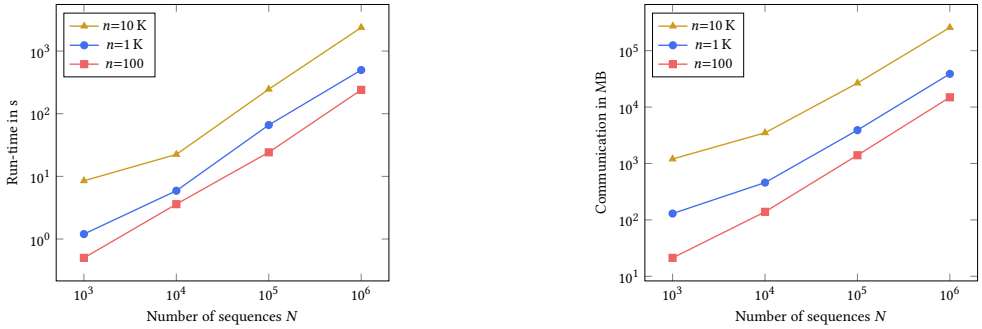

Figure 5: Large-scale benchmarks of our Similar Sequence Query algorithm for N sequences of length n, LUT width w=30, number of data providers ψ=10, number of most similar sequence queries k=10, block size b=5, and padded block size b'=16. Numbers are given in Table 5.

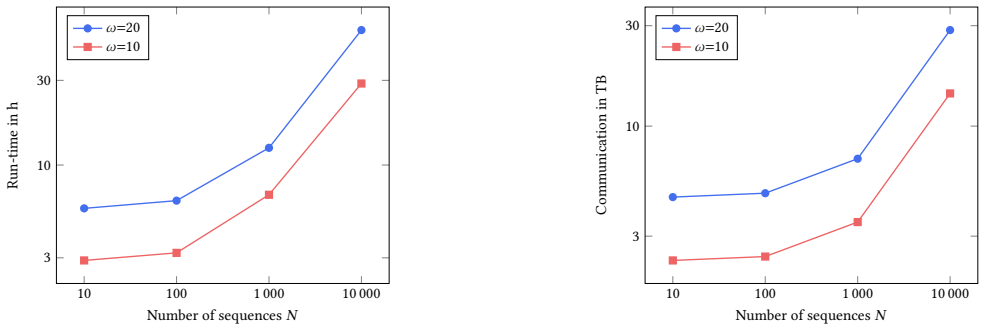

Figure 6: Run-times and communication for Similar Sequence Query on whole-genome genome sequences based on the computation of sub-sequences of smaller lengths with the following parameters: number of sequences N, Look-Up Table widths ω, sequence length n=75 M, block size b=5, padded block size b'=16, number of most similar sequences k=10, number of data providers ψ=10, number of blocks t=15 M, and bit-length of the distances $\beta=\lceil\log_2(tb')\rceil$=28. Numbers are given in Table 6.