TECHNISCHE
UNIVERSITÄT
DARMSTADT

Towards Practical Privacy-Preserving Protocols

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)

von

Daniel Demmler, M.Sc.
geboren in Neuhaus am Rennweg

Referenten: Prof. Dr.-Ing. Thomas Schneider
Prof. Dr. Amir Herzberg

Tag der Einreichung: 11.10.2018
Tag der Prüfung: 22.11.2018

D 17
Darmstadt, 2018

Erklärung

Hiermit versichere ich, Daniel Demmler, M.Sc., die vorliegende Dissertation 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, 11.10.2018

Daniel Demmler, M.Sc.

Wissenschaftlicher Werdegang

seit Oktober 2013 Promotion in der Informatik, Technische Universtität Darmstadt.

Juli 2011 – Juli 2013 Studium der Informationssystemtechnik, Technische Universität Darmstadt, Abschluss als Master of Science.

Oktober 2007 – Juni 2011 Studium der Informationssystemtechnik, Technische Universität Darmstadt, Abschluss als Bachelor of Science.

Abstract

Protecting users' privacy in digital systems becomes more complex and challenging over time, as the amount of stored and exchanged data grows steadily and systems become increasingly involved and connected. Two techniques that try to approach this issue are Secure Multi-Party Computation (MPC) and Private Information Retrieval (PIR), which aim to enable practical computation while simultaneously keeping sensitive data private. In this thesis we present results showing how real-world applications can be executed in a privacy-preserving way. This is not only desired by users of such applications, but since 2018 also based on a strong legal foundation with the General Data Protection Regulation (GDPR) in the European Union, that forces companies to protect the privacy of user data by design.

This thesis' contributions are split into three parts and can be summarized as follows:

MPC Tools Generic MPC requires in-depth background knowledge about a complex research field. To approach this, we provide tools that are efficient and usable at the same time, and serve as a foundation for follow-up work as they allow cryptographers, researchers and developers to implement, test and deploy MPC applications. We provide an implementation framework that abstracts from the underlying protocols, optimized building blocks generated from hardware synthesis tools, and allow the direct processing of Hardware Definition Languages (HDLs). Finally, we present an automated compiler for efficient hybrid protocols from ANSI C.

The results presented in this part are published in:

[BDK+18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *25. ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018, pp. 847–861. CORE Rank A*.

[DDK+15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni. **"Automated Synthesis of Optimized Circuits for Secure Computation"**. In: *22. ACM Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, pp. 1504–1517. CORE Rank A*.

[DSZ15] D. Demmler, T. Schneider, M. Zohner. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. Code: https://encrypto.de/code/ABY. Internet Society, 2015. CORE Rank A*.

MPC Applications MPC was for a long time deemed too expensive to be used in practice. We show several use cases of real-world applications that can operate in a privacy-preserving, yet practical way when engineered properly and built on top of suitable MPC protocols. Use cases presented in this thesis are from the domain of route computation using BGP on the Internet or at Internet Exchange Points (IXPs). In both cases our protocols protect sensitive business information that is used to determine routing decisions. Another use case focuses on genomics, which is particularly critical as the human genome is connected to everyone

during their entire lifespan and cannot be altered. Our system enables federated genomic databases, where several institutions can privately outsource their genome data and where research institutes can query this data in a privacy-preserving manner.

The results presented in this part are published in:

[ADS+17] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, M. Zohner. **"Privacy-Preserving Interdomain Routing at Internet Scale"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017). Full version: https://ia.cr/2017/393, pp. 143–163. CORE Rank B.

[CDC+16] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"Towards Securing Internet eXchange Points Against Curious onlooKers (Short Paper)"**. In: *1. ACM, IRTF & ISOC Applied Networking Research Workshop (ANRW'16)*. ACM, 2016, pp. 32–34.

[CDC+17] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"SIXPACK: Securing Internet eXchange Points Against Curious onlooKers"**. In: *13. International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*. ACM, 2017, pp. 120–133. CORE Rank A.

[DHSS17] D. Demmler, K. Hamacher, T. Schneider, S. Stammler. **"Privacy-Preserving Whole-Genome Variant Queries"**. In: *16. International Conference on Cryptology And Network Security (CANS'17)*. Vol. 11261. LNCS. Springer, 2017, pp. 71–92. CORE Rank B.

PIR and Applications Privately retrieving data from a database is a crucial requirement for user privacy and metadata protection, and is enabled amongst others by a technique called Private Information Retrieval (PIR). We present improvements and a generalization of a well-known multi-server PIR scheme of Chor et al. [CGKS95], and an implementation and evaluation thereof. We also design and implement an efficient anonymous messaging system built on top of PIR. Furthermore we provide a scalable solution for private contact discovery that utilizes ideas from efficient two-server PIR built from Distributed Point Functions (DPFs) in combination with Private Set Intersection (PSI).

The results presented in this part are published in:

[DHS14] D. Demmler, A. Herzberg, T. Schneider. **"RAID-PIR: Practical Multi-Server PIR"**. In: *6. ACM Cloud Computing Security Workshop (CCSW'14)*. Code: https://encrypto.de/code/RAID-PIR. ACM, 2014, pp. 45–56.

[DHS17] D. Demmler, M. Holz, T. Schneider. **"OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks"**. In: *15. International Conference on Applied Cryptography and Network Security (ACNS'17)*. Vol. 10355. LNCS. Code: https://encrypto.de/code/onionPIR. Springer, 2017, pp. 599–619. CORE Rank B.

[DRRT18] D. Demmler, P. Rindal, M. Rosulek, N. Trieu. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). Code: https://github.com/osu-crypto/libPSI. CORE Rank B.

Zusammenfassung

Es wird zunehmend schwieriger die Privatsphäre von Nutzerdaten in digitalen Systemen zu schützen, da die Menge an gespeicherten und verarbeiteten Daten stetig wächst und Systeme immer komplexer und vernetzter werden. Zwei Techniken, die dieses Problem angehen und darauf abzielen praktische Berechnungen unter gleichzeitigem Schutz der Privatsphäre zu ermöglichen, sind sichere Mehrparteienberechnung (MPC) und Private Information Retrieval (PIR). Diese Dissertation präsentiert Ergebnisse, die zeigen wie Anwendungen aus der Praxis mit Privatsphäre-Schutz versehen werden können. Dies ist nicht nur der Wunsch vieler Anwender, sondern mit der europäischen Datenschutz-Grundverordnung (DSGVO) seit 2018 auch auf einer starken rechtlichen Basis verankert.

Die wissenschaftlichen Beiträge dieser Arbeit sind in die folgenden drei Teile gegliedert:

MPC Werkzeuge Die Verwendung von MPC-Techniken benötigt fundiertes Hintergrundwissen in einem komplexen Forschungsfeld. Wir stellen dafür Werkzeuge zur Verfügung, die effizient sind und gleichzeitig einen großen Fokus auf Benutzbarkeit legen. Diese Werkzeuge diesen als Basis für viele Folge-Arbeiten und sie erleichtern es Kryptographen, Entwicklern und Forschern MPC Anwendungen zu entwickeln und zu evaluieren. Wir stellen ein Implementierungs-Framework zur Verfügung, das von Protokolldetails abstrahiert, ergänzen dieses mit Bausteinen aus der Hardware-Synthese und erlauben die direkte Verarbeitung von Hardwarebeschreibungs-Sprachen. Weiterhin stellen wir einen Compiler vor, der ANSI C Code vollautomatisiert in effiziente, hybride MPC Protokolle übersetzt.

Ergebnisse dieses Teils wurden veröffentlicht in:

[BDK+18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *25. ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018, S. 847–861. CORE Rank A*.

[DDK+15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni. **"Automated Synthesis of Optimized Circuits for Secure Computation"**. In: *22. ACM Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, S. 1504–1517. CORE Rank A*.

[DSZ15] D. Demmler, T. Schneider, M. Zohner. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. Code: https://encrypto.de/code/ABY. Internet Society, 2015. CORE Rank A*.

MPC Anwendungen MPC war lange Zeit als rein theoretisches Resultat angesehen, das aufgrund seiner Komplexität in der Praxis kaum Verwendung findet. Wir präsentieren mehrere praktische Applikationen, die die Privatsphäre der verarbeiteten Daten schützen und gleichzeitig praktikable Performanz erreichen. Eine Anwendung ist fokussiert auf die Berechnung von Routen mittels des Border Gateway Protokolls (BGP) im Internet sowie deren Verteilung bei Internet Exchange Points (IXPs). In beiden Fällen schützen unsere Protokolle

sensitive Unternehmensdaten, die für Routing-Entscheidungen benötigt werden. Ein weiterer Anwendungsfall stammt aus der Genetik und ist insofern von besonderer Relevanz, da das menschliche Genom unveränderlich ist und für die komplette Dauer eines Menschenlebens an ein Individuum gebunden ist. Unser System erlaubt es mehreren medizinischen Institutionen ihre Genomdaten sicher in eine verteilte Genomdatenbank auszulagern und diese zentrale Datenbank unter Schutz der Privatsphäre abzufragen.

Ergebnisse dieses Teils wurden veröffentlicht in:

[ADS+17] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, M. Zohner. **"Privacy-Preserving Interdomain Routing at Internet Scale"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017). Full version: https://ia.cr/2017/393, S. 143–163. CORE Rank B.

[CDC+16] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"Towards Securing Internet eXchange Points Against Curious onlooKers (Short Paper)"**. In: *1. ACM, IRTF & ISOC Applied Networking Research Workshop (ANRW'16)*. ACM, 2016, S. 32–34.

[CDC+17] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"SIXPACK: Securing Internet eXchange Points Against Curious onlooKers"**. In: *13. International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*. ACM, 2017, S. 120–133. CORE Rank A.

[DHSS17] D. Demmler, K. Hamacher, T. Schneider, S. Stammler. **"Privacy-Preserving Whole-Genome Variant Queries"**. In: *16. International Conference on Cryptology And Network Security (CANS'17)*. Bd. 11261. LNCS. Springer, 2017, S. 71–92. CORE Rank B.

PIR und Anwendungen Die private Abfrage von Daten aus einer Datenbank als Grundlage für Anonymität und den Schutz von Metadaten wird ermöglicht durch Private Information Retrieval (PIR). Wir zeigen Verbesserungen und die Generalisierung des PIR-Protokolls von Chor et al. [CGKS95] sowie eine Implementierung und Evaluation davon. Wir implementieren zudem ein effizientes anonymes Kommunikationssystem auf der Grundlage von PIR. Weiterhin stellen wir eine skalierbare Lösung für private Schnittmengenberechnung (PSI), speziell für den Kontext der privaten Kontaktsynchronisierung vor. Diese basiert auf effizienter 2-Parteien PIR in Kombination mit PSI.

Ergebnisse dieses Teils wurden veröffentlicht in:

[DHS14] D. Demmler, A. Herzberg, T. Schneider. **"RAID-PIR: Practical Multi-Server PIR"**. In: *6. ACM Cloud Computing Security Workshop (CCSW'14)*. Code: https://encrypto.de/code/RAID-PIR. ACM, 2014, S. 45–56.

[DHS17] D. Demmler, M. Holz, T. Schneider. **"OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks"**. In: *15. International Conference on Applied Cryptography and Network Security (ACNS'17)*. Bd. 10355. LNCS. Code: https://encrypto.de/code/onionPIR. Springer, 2017, S. 599–619. CORE Rank B.

[DRRT18] D. Demmler, P. Rindal, M. Rosulek, N. Trieu. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). Code: https://github.com/osu-crypto/libPSI. CORE Rank B.

Contributions

Scientific research in computer science is complex and has reached a level where single authors that publish at top venues have become the exception. Nowadays it is the norm that groups of researchers work together to achieve novel and significant results.

Similarly, the work presented here is in many cases interdisciplinary and combines comprehensive background knowledge from multiple areas. All publications that this thesis is based on are the result of extensive collaboration. Many parts result from the close collaboration of several authors combining their expertise from heterogeneous research fields, while other parts are highly complex and were only achieved through cooperation of experts within one domain. I am thankful for the opportunity to collaborate with my many great colleagues, both internationally and at TU Darmstadt — especially within the collaborative research center CROSSING, that awarded our work in [DKS+17] (cf. Chapter 4) with the CROSSING Collaboration Award 2016.

I want to thank my co-authors for the exchange of ideas and their contributions (with regard to works included in this thesis, in chronological order): Amir Herzberg, Thomas Schneider, Michael Zohner, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Shaza Zeitouni, Marco Chiesa, Marco Canini, Gilad Asharov, Michael Schapira, Gil Segev, Scott Shenker, Marco Holz, Kay Hamacher, Sebastian Stammler, Peter Rindal, Mike Rosulek, Ni Trieu, Niklas Büscher, Stefan Katzenbeisser, and David Kretzmer.

Enumerating the exact contribution and attributing parts of results to individual authors is rather difficult, as often only joint discussions and iterative processes led to the final research outcome. In this thesis I build upon the cited publications. In some cases I extended the content and modified the presentation, but many parts are adopted in verbatim form and hence might contain parts that are hard to attribute to individual authors. In almost all cases the isolated contribution of each author individually would not be very meaningful when looked at separately, and the final outcome was only achieved by putting the pieces together. Still, in the following section I aim to specify which contributions were specifically made by myself and how my co-authors, who were young researchers at the time of the respective publication, contributed to the results presented here.

Chapter 3 is based on [DSZ15], where I contributed parts of the performance evaluation and the implementation, as well as the Homomorphic Encryption (HE)-based multiplications and their evaluation. I was and I am still actively involved in maintaining and extending the code-base of ABY, which is available online on GitHub[1]. Michael Zohner contributed most core protocols and their conversions and the majority of the initial implementation.

The results in Chapter 4 bases on [DKS+17], where I was responsible for the implementation of an adapter for importing external circuits into the ABY framework, cf. Chapter 3, the implementation and extensive performance evaluation of the building block and the use cases. Many details of the circuit synthesis, the adaption of the hardware synthesis tools, and

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

many of the building blocks were contributions of my co-authors Ghada Dessouky and Shaza Zeitouni.

Chapter 5 is based on [BDK+18]. I mainly worked on the runtime estimation, the performance evaluation and implementations of interfaces between ABY, cf. Chapter 3, and the HyCC compiler. Major parts of the compiler design and protocol selection were done by Niklas Büscher. A great part of the implementation was done by David Kretzmer.

Chapter 6 is based on [CDC+17b] and [ADS+17]. Both works are very collaborative results. In [CDC+17b], I contributed significantly to the design and implementation of the system, the underlying algorithms and their performance evaluation. Marco Chiesa worked mainly on the Python implementation of the demonstrator and the operator survey. Parts of the implementation of [ADS+17] and their evaluation are my contribution, while other parts were implemented by Michael Zohner. The security considerations and ideas for failure handling were contributed by Gilad Asharov.

In Chapter 7 results from [DHSS17] are presented, where I contributed parts of the algorithm design, and significant parts of the implementation and evaluation of the protocol as well as the security considerations. Sebastian Stammler has worked on the size-efficient query format and query setting, and contributed to the implementation and evaluation.

Chapter 8 takes results from joint work with Thomas Schneider and Amir Herzberg [DHS14]. I significantly contributed to all aspects of this publication, mostly the protocol idea and design, implementation and evaluation. In the same chapter results from [DHS17] are included, where the original application idea and the implementation came from Marco Holz. I contributed to the final version of the protocol, the underlying optimizations and the system model.

The work in Chapter 9 is based on [DRRT18], which was done mostly during my 10 week visit at Oregon State University in spring 2017. I contributed to the protocol design, which was the result of many discussions with Mike Rosulek, Peter Rindal and Ni Trieu. I contributed to the implementation, while Peter was responsible for the majority of the code. Peter contributed the experiments and formulas for the Cuckoo hashing failure probability. The evaluation and the comparison with related work was done partly by me, and in part by Ni.

Acknowledgments

This thesis was certainly one of the most exciting and challenging parts of my life and only possible due to the immense support of many people. I would like to thank everyone who contributed to this process in one way or another.

First and foremost, I am incredibly grateful for the opportunity to have Thomas Schneider as my Ph.D. advisor. Without his encouragement to pursue a Ph.D. after finishing my master's thesis in his group, this Ph.D. thesis would most certainly not exist — at least not in this form. Thomas always had an open ear for questions and contributed to our joint work more than most advisors that I know of. His many ideas, the feedback, and especially the meticulous remarks on papers and their bibliographies were always very helpful. I'm immensely thankful for his continuous dedication, trust and support during the past years.

I am honored to have Amir Herzberg as external advisor and would also like to thank him for reviewing this thesis and for our successful joint work, many fruitful discussions, and exchange of ideas. I thank Matthias Hollick, Stefan Katzenbeisser, and Felix Wolf for joining my defense committee.

I want to say thank you to my colleagues and friends in the ENCRYPTO group: Michael Zohner, Ágnes Kiss, Christian Weinert, Oleksandr Tkachenko and Amos Treiber. Working with them was always productive, efficient and interesting. At the same time I enjoyed spending time outside work together, not only during lunch.

I also like to thank the people behind the curtain, who always made things possible, especially Melanie Schöyen, Heike Meißner, Stefanie Kettler, and Andrea Püchner.

There are a lot of people who work in security, cryptography, and privacy research worldwide, or in one of the many projects and groups at TU Darmstadt — too many to list them all separately. However, I'm grateful for the chance to be involved in all this and for the inspiration, encouragement, and support from so many sides.

A huge thanks to Mike Rosulek, Peter Rindal and Ni Trieu for making my visit at Oregon State University in spring 2017 so successful, exceptional, and unforgettable.

I am very grateful to have had the opportunity to work with my co-authors. I want to thank all of them for their work, and list those that I have not mentioned before: Gilad Asharov, Marco Canini, Marco Chiesa, Ghada Dessouky, Kay Hamacher, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Michael Schapira, Gil Segev, Scott Shenker, and Shaza Zeitouni.

Parts of this thesis were made possible by the hard and high-quality work of our students Lennart Braun, David Kretzmer, and Marco Holz. I thank them a lot for this.

I would also like to thank everyone involved in Bedroomdisco for spreading love for great music and offering me the opportunity to take a step away from work, allowing me to do something exciting and fulfilling in my free time.

Cheers to the iST Stammtisch that managed to survive even after everyone finished their studies. To many more years to come!

On the same note, I'd like to thank those people that spent their valuable time with me, both at work, and outside of the office — in particular: Sebastian Stammler (for genomes, crypto currencies, Hong Kong, and Mr. Robot), Daniel Steinmetzer (for the gym endeavours), Rebecca Burk (for always welcoming me back in Germany), Niklas Büscher (for the coffee breaks and the compiler), Felix Günther (for being too good to keep up, the travels, and the drink coupons), Johannes Gräbner (for the hikes in the forest), Patrick Lieser (for our hate-love, the beers, and our immense CS:GO success), and Jan Römer (for the music, the books, the movies, the travels, and all the talks).

I'm forever indebted to my parents and my family, who always supported me unconditionally, in every way possible, and encouraged me to keep going.

I also want to thank Maxine. Thank you for being there, for understanding, and for keeping me sane, especially in the last months.

Contents

1 Introduction

Privacy is the ability to express oneself selectively and to actively decide which potentially private information one discloses to others or to the public. This concept is fundamental to a functioning democratic society and a core requirement for personal autonomy.

Article 12 of the internationally almost universally accepted United Nations' Universal Declaration of Human Rights [UN48] specifies:

> "No one shall be subjected to arbitrary interference with his privacy [...]. Everyone has the right to the protection of the law against such interference or attacks."

Similarly, the European Union adopted the GDPR [EU16] in April 2016, and made it enforceable in May 2018. The GDPR obligates businesses to handle sensitive user data with "data protection by design and by default" and aims to give individuals control over their private data. GDPR violations can result in significant fines.

Along the same lines, the Indian supreme court has analogously ruled in August 2017 that privacy is a fundamental human right for its more than 1.3 Billion people.[1]

All these laws and regulations show that many societies agree, that privacy is an important concept and that sensitive data needs protection. Yet, in practice many processes require users to give up control over their private data. This problem has become even more severe with the rise of digital services that collect user data and their ubiquitous interconnection. Internet services know our shopping preferences from media to pharmaceuticals, search engines answer questions from all domains and cloud services store our contacts, calendars, pictures and backups. Messaging services almost always know our social graph (who we talk to and when) and many times even the actual content of exchanged messages (what we talk about). More than 80% of the German population are using smartphones,[2] which work heavily with the aforementioned user data and extend all these with real-time location information. All this data becomes even more valuable when multiple data sets are combined. This allows for very detailed profiling of users, is valuable for companies and happens frequently.[3,4]

[1] https://www.eff.org/de/deeplinks/2017/08/indias-supreme-court-upholds-right-privacy-fundamental-right-and-its-about-time

[2] https://www.bitkom.org/Presse/Anhaenge-an-PIs/2018/Bitkom-Pressekonferenz-Smartphone-Markt-22-02-2018-Praesentation-final.pdf

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

[4] https://techcrunch.com/2018/06/13/salesforce-deepens-data-sharing-partnership-with-google/

Another very recent report claimed that attackers allegedly used modified hardware to exfiltrate sensitive corporate data.[5] Although the involved parties declined that these incidents happened and since then doubts about the credibility of the report arose,[6] there is certainly a possiblity for these types of attacks to happen. In any case such attacks can be prevented, if computations on the most sensitive data is moved to a protected domain, that can be instantiated with suitable privacy-preserving techniques.

In light of companies tracking user data and insiders or attackers stealing corporate data, the goal of this thesis is to give end users and businesses control over their data and for this it contributes techniques for the protection of user data and corporate information.

A core concept in information security is confidentiality, i.e., the ability to make sensitive information only available to parties who are eligible to access it and to hide it from everyone else. Today, there are many practical cryptographic solutions that enable private and secure transmission and storage of data, but ultimately the *processing* of data is still challenging while maintaining confidentiality.

Homomorphic Encryption (HE) is a technique and an actively progressing research field that provides encryption schemes that allow certain operations on encrypted data, without revealing information about the contained plaintext. While huge improvements have been made that reduced ciphertext sizes and sped up operations, generic HE is still somewhat limited and applying it to generic real-life computations is far from straight forward.

Another approach that aims to solve the same problem of generic computation on private data is Secure Multi-Party Computation (MPC). The first concepts have been introduced in the 1980s [Yao86; GMW87] and were initially merely theoretic constructs. Recently these ideas have been picked up and improved up to a level where certain use cases can be considered practical. The first practical breakthrough happened 2004 with Fairplay [MNPS04] and many works followed that pushed MPC closer to practice.

An orthogonal approach that tries to solve a more specific problem of privately obtaining data from a database is Private Information Retrieval (PIR). Introduced in 1995 [CGKS95], it started a line of research that is still active today.

A core problem with the aforementioned privacy-preserving techniques is often performance, which we approach with the results in this thesis.

To sum up, this thesis aims at providing an answer to the following question:

> Can privacy-preserving techniques like MPC and PIR be applied to real-world applications and use-cases in order to protect the privacy of the data they process, while at the same time achieving efficiency that makes them usable in practice?

[5] https://www.bloomberg.com/news/features/2018-10-04/the-big-hack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies

[6] https://arstechnica.com/tech-policy/2018/10/bloomberg-stands-by-chinese-chip-story-as-apple-amazon-ratchet-up-denials/

1.1 Thesis Outline

To answer the above research question, we structure this thesis as follows:

In Chapter 2 we introduce concepts from preliminary work and explain the assumptions and notation that we use in this thesis.

Part I describes tools that we developed with the goal of providing efficient and usable MPC implementations. These tools are a foundation for our own work, as well as for other developers and researchers, who can freely use them.

Chapter 3 introduces the ABY framework [DSZ15], that was the first step of having a unified implementation of fundamental MPC protocols based mostly on symmetric cryptography and efficient conversions between them. ABY abstracts from the underlying protocol details and provides a unified interface to manually implement MPC applications using Yao's garbled circuits, the GMW protocol or arithmetic sharing, or a chosen mixture of them, called hybrid protocols.

The ABY framework is then extended with efficient buildings blocks and the ability to process circuits that were generated from hardware synthesis tools [DDK+15] in Chapter 4. This allows to implement MPC applications in a Hardware Definition Language (HDL) or to use new building blocks, such as floating-point operations. We demonstrate their efficiency by showing performance results for privacy-preserving proximity testing.

Chapter 5 is based on [BDK+18] and presents the HyCC compiler, that is the first to allow automated compilation of ANSI C code into efficient hybrid MPC protocols. For HyCC we combine ideas from CBMC-GC [HFKV12; BFH+17] with the ABY framework and show that our automatically compiled results are very performant and even able to outperform certain hand-crafted protocols for a machine learning use case.

In Part II, we present efficient applications of MPC protocols. While MPC was initially seen as a purely theoretical result, we provide examples of useful applications that can be evaluated in a privacy-preserving way, at practical performance and real-world scale.

Chapter 6 is based on [CDC+17b] and [ADS+17] and provides approaches for privacy-preserving routing using the Border Gateway Protocol (BGP). Specifically, we present two types of results: We implement two graph algorithms in MPC that model the behavior of BGP and thereby enable route computations that preserve the privacy of the underlying bussiness information that is used to make routing decisions. Furthermore, we provide a solution for efficient private route dispatch at Internet Exchange Points (IXPs), central authorities that connect multiple parties with each other. Our performance results show that our implementations allow private Internet-scale route computation and private real-time IXP route dispatch, respectively.

Chapter 7 presents the results from [DHSS17], where we designed a system for privacy-preserving querying of a federated database of genomes at large scale. These types of queries

are frequently run in medical research and our system's performance allows to do so efficiently, while protecting the privacy of both the databases and the queries.

Finally, in Part III, we present improvements of existing Private Information Retrieval (PIR) protocols and show two applications that make use of PIR in order to achieve anonymity and privacy.

Chapter 8 summarizes results from [DHS14] and [DHS17], where we generalize and optimize an existing PIR scheme in several ways. We propose an anonymous messaging system that utilizes PIR to privately retrieve users' public keys and relies on the anonymity network Tor to achieve private communication.

In Chapter 9, which is based on [DRRT18] results from efficient 2-server PIR built from Distributed Point Functions (DPFs) are combinded with Private Set Intersection (PSI). From these techniques we build a solution that can be used for private contact discovery and show that it performs well in practice and scales even for very large input sizes.

We summarize this thesis and provide an answer to many aspects of the research question in Chapter 10, where we also look into points open to be answered in future work.

2 Preliminaries

In this chapter we introduce basic concepts and notations that are used in this thesis.

2.1 Notation and Security Parameters

Throughout this thesis we use the following notation and their default parameters. If there are deviations from the default values, these are specified accordingly.

Table 2.1: Notation: Symbols and default values used.

Parameter	Symbol	Default Value
symmetric security parameter [bits]	κ	128 bits
asymmetric security parameter [bits]	φ	3 072 bits
statistical security parameter [bits]	σ	40 bits
element length [bits]	ℓ	

The default security parameters that we use in our implementations are chosen such that they achieve a security level that is expected to withstand attacks until year 2030 and possibly beyond, according to the recommendations given for cryptographic key lengths on keylength.com and from NIST [NIS12]. $\kappa \in \{80, 112, 128\}$ denotes the symmetric security parameter and $\varphi \in \{1\,024, 2\,048, 3\,072\}$ denotes the public-key security parameter, for legacy (until 2010), medium (2011-2030), and long-term security (after 2030), respectively. We set the statistical security parameter σ to 40, which means that statistical processes fail with a probability of at most $2^{-\sigma} = 2^{-40}$.

We denote public-key encryption with the public key of party P_i as $c = \mathsf{Enc}_i(m)$ and the corresponding decryption operation as $m = \mathsf{Dec}_i(c)$ with $m = \mathsf{Dec}_i(\mathsf{Enc}_i(m))$.

In MPC protocols, we denote the two parties that run the secure computation protocol as P_0 and P_1.

We write $x \oplus y$ for bitwise XOR and $x \wedge y$ for bitwise AND.

2.2 Adversary Models

Privacy-preserving protocols offer security against adversaries, meaning that the protocol aims to guarantee that adversaries can only learn the output that is intended for them and nothing else, as long as they act within a certain adversarial model. The following adversary models exist in the literature and are more formally defined in [HL10].

Semi-honest adversaries (also called *honest-but-curious* or *passive* adversaries) are adversaries that try to gain access to secret information from the protocol execution and the messages that they receive, while following the protocol specification. Semi-honest adversaries are relatively weak adversaries, but are necessary as a baseline for verifying practicality and an important step towards achieving stronger security guarantees. They are also used in scenarios where somewhat trusted parties interact with each other, or to protect against attacks from insiders that try to exfiltrate sensitive plaintext data.

Malicious or *active* adversaries can arbitrarily and actively deviate from the protocol execution in order to access private information. They can modify, re-order or omit protocol messages and are the strongest type of adversary.

Covert adversaries have all abilities of a malicious adversaries, but are guaranteed to be caught with a given probability that the protocol ensures, e.g., 50%. This probability must be high enough to discourage adversaries from attempting to cheat in the protocol in practice.

In this thesis most protocols are designed and implemented to tolerate semi-honest adversaries. For some protocols we also describe extensions for security against stronger adversaries, while maintaining practicality.

2.3 Oblivious Transfer

A core building block that serves as a foundation of the techniques used in this thesis is Oblivious Transfer (OT) [Rab81; EGL85]. In a 1-out-of-2 OT, a sender inputs two ℓ-bit messages (m_0, m_1) and a receiver inputs a choice bit $c \in \{0, 1\}$ in order to obliviously obtain the message m_c as output. OT guarantees that the receiver learns no information about m_{1-c}, while the sender learns nothing about c.

It was shown in [IR89] that OT protocols require costly public-key cryptography and cannot be built from symmetric primitives alone. However, a technique called OT extension [Bea96; IKNP03; ALSZ13; NPS99] allows to extend a few public-key-based OTs, for which we use [NP01] in our experiments, using only symmetric cryptographic primitives and a constant number of rounds. To further increase efficiency, special OT variants such as correlated OT (C-OT) [ALSZ13] and random OT (R-OT) [NNOB12; ALSZ13] were introduced. In C-OT, the sender inputs a correlation function $f_\Delta(\cdot)$ and obtains a random m_0 as output from the OT protocol while the other message is correlated as $m_1 = f_\Delta(m_0)$. In R-OT, the sender has no inputs and obtains two random messages (m_0, m_1). The random m_0 in C-OT and (m_0, m_1)

are output by a correlation robust one-way function H [IKNP03], which can be instantiated using a hash function.

2.4 Secure Multi-Party Computation (MPC)

Secure Multi-Party Computation (MPC), sometimes also referred to as secure function evaluation, multi-party computation, or simply secure computation, is an active field of research that was established in the 1980s [Yao86; GMW87]. It was followed by surprising feasibility results [BGW88; CCD88; RB89] that positioned MPC as a central and extremely powerful tool in cryptography. These works show that multiple parties can carry out a joint computation of *any* efficiently computable function on their respective inputs, without revealing any information about the inputs, except for what is logically inferred from the output.

More concretely, n parties $P_0, \ldots, P_{n-1}$ that hold private inputs $x_0, \ldots, x_{n-1}$ wish to compute some arbitrary function $f(x_0, \ldots, x_{n-1}) = (y_0, \ldots, y_{n-1})$, where the output of P_i is y_i. MPC enables the parties to compute the function f using an interactive protocol, where each party P_i learns exactly its designated output y_i, and nothing else.

In an ideal world MPC protocols can be viewed as an ideal functionality that is run by a trusted third party that collects all private inputs, computes the function on them, and sends the respective output back to the parties. In the real world the protocol is run only between the parties $P_0, \ldots, P_{n-1}$.

There are three major MPC paradigms, which we summarize in the next sections: Yao's garbled circuits protocol [Yao86] (Sect. 2.4.3), the secret-sharing-based protocols of Goldreich, Micali, and Wigderson (GMW) [GMW87] (Sect. 2.4.4) as well as arithmetic sharing [DSZ15; Gil99] (Sect. 2.4.5). We will also provide explanations for the offline-online paradigm (Sect. 2.4.1), and the setting in which we deploy these protocols (Sect. 2.4.2).

The aforementioned protocols protect the privacy of the processed data by sharing it between two parties either by using secret sharing (GMW and arithmetic sharing) or garbling and evaluating a Boolean circuit (Yao's garbled circuits). Thus, we also refer to the data processed in the respective protocols as *shares*.

2.4.1 The Offline-Online Model for MPC

MPC protocols can often be divided into several phases. A common approach that we also follow in this thesis is the separation of a *setup phase* and an *online phase*. The setup phase (also called *offline phase* or *precomputation phase*), happens before the private inputs to the protocol are known and requires only knowledge of (an upper bound of) the size of the inputs of the function to be computed. In the setup phase, helper data is created and later used to speed up the online phase. The online phase that is run as soon as the private inputs are known, is optimized in order to achieve high performance. This separation of phases allows for precomputation of expensive operations and modular protocol design.

To improve the performance of the online phase of OT we use OT precomputations [Bea95].

The opposite approach of the offline-online model is *pipelining* [HEKM11], which breaks up the execution of the MPC protocol into smaller parts, and intertwines the two phases closely to reduce the memory footprint of the MPC protocol execution.

2.4.2 Two-Party and Outsourcing Setting

The MPC protocols that we present in this thesis in Part I and Part II are two-party protocols run between the two MPC parties, which we denote as P_0 and P_1.

Naturally, such protocols can be used in *client-server applications*, e.g., for services on the Internet, where both parties provide private inputs to the computation and jointly compute the MPC protocol.

However, the protocols can also be used for *multi-party applications* where an arbitrary number of input parties provide their confidential inputs, and an arbitrary number of result parties receive the outputs of the secure computation (e.g., for auctions, surveys, etc.), cf. [FPRS04; KMR11]. We refer to this setting as *outsourcing scenario*. Following the notation used in [ABL+18], in this setting there are *input parties* that provide private inputs to a computation that is carried out using MPC by *computing parties*, and *result parties* that learn the plaintext computation outputs, or parts thereof.

For this, each input party secret-shares its inputs among two dedicated computation servers P_0 and P_1, that are assumed to not collude. Then, the two computation servers run the MPC protocol on the input shares during which they do not learn any information about inputs, intermediate values or the outputs of the computation. Finally, the computation servers send the output shares to the result parties who can reconstruct the plaintext outputs. Importantly, from the perspective of the two computational parties, the shares are indistinguishable from random bits. Naturally, the input and output parties can be the same parties. There are several frameworks that have been proposed specifically for an outsourcing scenario [CMTB13; CLT14; CMTB16].

Furthermore, there are several approaches that operate in a 3-party setting [BLW08; AFL+16; FLNW17; MR18]. Secret-sharing-based protocols like GMW can naturally be extended to more than two parties.

Recently, MPC protocols that support very large numbers of parties and achieve promising performance results have been proposed. [HSS17] is built on top of the BMR protocol [BMR90]. An alternative approach was presented in [WRK17].

A conceptual overview of the outsourcing setting is depicted in Fig. 2.1.

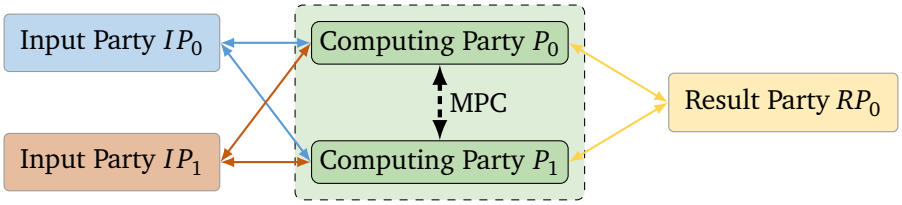


Figure 2.1: Example setting with 2 input parties that secret share their inputs with 2 computing parties P_0 and P_1. The output is received by a single result party. Thin arrows correspond to a single round of communication with small messages, while the **bold** arrow symbolizes the execution of an MPC protocol with potentially many rounds and high throughput.

2.4.3 Yao's Garbled Circuits Protocol

In Yao's garbled circuits [Yao86], two parties interactively evaluate a garbled version of a Boolean circuit, consisting of gates that have input and output wires.

One party, referred to as *garbler*, creates the garbled circuit as follows: For all wires in the circuit, including inputs and outputs, the garbler determines two random keys corresponding to the two possible bits on every wire. Using these keys, every possible gate output is encrypted with the corresponding combination of input keys and stored in a garbled table for each gate. In the evaluation step the other party, called *evaluator*, receives the garbled circuit, the encoding of the garbler input, as well as encodings of its own inputs via OT, cf. Sect. 2.3. The evaluator then iterates through the circuit gate by gate to compute the encoding of the output, which is finally decoded using a mapping from output keys to plaintext. Yao's protocol has only a constant number of communication rounds and the complexity stems from the total number of AND gates in the circuit, as XOR gates can be evaluated for free [KS08b]. Other state-of-the-art optimizations of garbled circuits that are used in today's implementations are point-and-permute [BMR90], fixed-key AES garbling [BHKR13], and half-gates [ZRE15].

The security of Yao's garbled circuits protocol was proven in [LP09].

2.4.4 The Protocol of Goldreich, Micali, and Wigderson (GMW)

In the GMW protocol [GMW87], two or more parties jointly compute a function that is encoded as Boolean circuit. The parties' private inputs to the function, all intermediate wire values, and all outputs are hidden by bit-wise XOR-based secret sharing. For this, every plaintext value v is XORed with a random value v_0 of the same length to compute $v_1 = v \oplus v_0$. The values v_0 and v_1 are called *shares* of v and are held by P_0 and P_1 respectively.

GMW allows to evaluate XOR gates locally, without interaction, using only one-time pad operations and thus essentially for free. AND gates, however, require interaction in the form of OTs [CHK+12] or Beaver's multiplication triples [Bea96; ALSZ13] that can be precomputed in the setup phase. A multiplication triple consists of correlated random bits $a_0, a_1, b_0, b_1, c_0, c_1$

that satisfy the equation $c_0 \oplus c_1 = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$. After evaluating all circuit gates in the online phase, the plaintext output can be reconstructed by computing the XOR of the resulting output shares.

The performance of GMW depends on both the total number of AND gates in the circuit, as well as the multiplicative depth of the circuit, i.e., the maximum number of data-dependent AND gates on the critical path from any input to any output. This is due to the OT that has to be performed for every AND gate, and at the same time, due to the round of communication that is performed between the parties for each layer of data-dependent AND gates.

One main advantage of the GMW protocol is that it allows to precompute *all* (symmetric) cryptographic operations in the setup phase, while the online phase consists solely of bit operations. Moreover, the GMW protocol allows to efficiently evaluate the same sub-circuit in parallel, similar to Single Instruction Multiple Data (SIMD) instructions in a CPU. Finally, the GMW protocol also allows for highly efficient instantiation of multiplexers using *vector ANDs* (cf. Sect. 3.2.3), which reduce the cost for evaluating a ℓ-bit multiplexer to the cost of evaluating a single AND gate.

The proof of security for the GMW protocol was provided in [Gol04].

The protocols that we implement in this thesis in Part II are 2-party versions of the GMW protocol with security against semi-honest adversaries.

2.4.5 Arithmetic Sharing

Arithmetic sharing, sometimes also referred to as *linear secret sharing*, works similar to the GMW protocol and uses modular addition to secret-share arithmetic values $\in \mathbb{Z}_{2^\ell}$ for a bit length ℓ. Addition can be done for free, while multiplication requires one round of interaction, analogously to XOR and AND in GMW. Multiplication is done using arithmetic multiplication triples, that can be efficiently precomputed using OTs [Gil99] or using homomorphic encryption, cf. Sect. 3.2.2.

2.4.6 MPC Protocol Implementations

There were several proposals for MPC frameworks in the recent years. In this section we provide an overview and group them into several categories, that differ by how the MPC protocols are described.

MPC from a Domain Specific Language (DSL) Domain Specific Languages (DSLs) are input languages that are designed to cover specific properties and features of a certain research domain. They can build on top of a known language, or be fully independent and designed from scratch. In all cases DSLs require developers to carefully get accustomed to specific language features. Fairplay [MNPS04], its extension to multiple parties in FairplayMP [BNP08], and the compatible PAL compiler [MLB12] compile a functionality from a domain specific input language, called Secure Function Definition Language (SFDL), into a Boolean circuit

described in the Secure Hardware Definition Language (SHDL) which is evaluated with Yao's garbled circuits protocol. Sharemind [BLW08] is a 3-party framework for arithmetic circuits evaluated using linear secret sharing-based that also offers their own DSL. The VIFF framework [DGKN09] provides a secure computation language and uses a scheduler, which executes operations when operands are available. Similarly, TASTY [HKS+10] proposed a DSL called TASTYL that allows to combine protocols that mix Yao's garbled circuits with additively homomorphic encryption. The compiler presented in [KSS12] also provides a DSL and showed scalability to circuits consisting of billions of gates that were evaluated with a variant of Yao's protocol with security against malicious adversaries. Wysteria [RHH14] is a strongly typed high-level language for the specification of secure multi-party computation protocols. More recently, ObliVM [LWN+15] introduced a DSL that is compiled into Yao's garbled circuits with support for Oblivious RAM (ORAM).

MPC Compilers from ANSI C The following secure computation tools use a subset of the ANSI C programming language as input. CBMC-GC [HFKV12] initiated this line of development and used a SAT solver to generate size-optimized Boolean circuits from a subset of ANSI C. More details on CMBC-GC can be found in Sect. 2.4.6. PCF [KSMB13] compiles into a compact intermediate representation that also supports loops. Both the initial CBMC-GC and PCF target Yao's garbled circuits protocol and hence only optimize for size. An extension for CBMC-GC that focuses on depth-optimized circuits for GMW was presented with ShallowCC [BHWK16]. PICCO [ZSB13] is a source-to-source compiler that allows parallel evaluation and uses secure computation protocols based on linear secret sharing with at least three parties.

Further results were presented, that focused on improving the compilers' scalability [KSS12], Obliv-C [ZE15], and Frigate [MGC+16]. An approach to formally verifying a tool-chain was presented with CircGen [ABB+17].

Very recently, the authors of [CGR+17] proposed an solution for hybrid compilation of MPC protocols called EzPC. However, while their main motivation is similar to ours in HyCC, cf. Chapt. 5, our results differ in several key points. In EzPC, a developer needs to *manually* split the input program into suitable modules and needs to *manually* resolve private array accesses into multiplexer-like structures, which hardly goes beyond what's already possible using the underlying ABY framework. Furthermore, EzPC does not apply circuit optimizations and does not consider depth-optimized Boolean circuits, as required for an efficient execution with the GMW protocol in low-latency networks.

MPC Libraries There is a separate line of work, where the developer composes the circuits to be evaluated securely from circuit libraries that are instantiated at runtime. This approach has been proposed in FastGC [HEKM11; HS13] and VMCrypt [Mal11] both of which are based on Yao's garbled circuits. In fact, all implementations of the GMW protocol [CHK+12; SZ13; DSZ15] are secure computation libraries. SPDZ [DPSZ12], and [LN17] are frameworks for secret sharing over arithmetic circuit-based MPC protocols, that fall in the same category.

MPC from Hardware Synthesis Tools The TinyGarble framework [SHS+15] was the first work to consider using hardware-synthesis tools to generate Yao's garbled circuits and store them as *sequential* circuits. This leads to a more compact representation and better memory locality, but identical number of cryptographic operations during garbling and evaluation. We show follow up work in Chapt. 4, that also targets the GMW protocol.

Mixed-Protocol MPC Combining multiple secure computation protocols to utilize the advantages of each of the protocols is used in several works. To the best of our knowledge, the first work that combined Yao's garbled circuits and homomorphic encryption was [BPSW07] who used this technique to evaluate branching programs with applications in remote diagnostics. The framework of [KSS13b], implemented in the TASTY compiler [HKS+10], combines additively homomorphic encryption with Yao's garbled circuits protocol and was used for applications such as face-recognition. The L1 language [SKM11] is an intermediate language for the specification of mixed-protocols that are compiled into Java programs. Sharemind [BLW08] was extended to mixed-protocols in [BLR13; BLR14]. *ABY*^{3} [MR18] is a novel framework for hybrid secure 3-party computation with a honest majority. We present our ABY framework in Chapt. 3.

There are also automated approaches to mixed-protocol MPC: In [KSS14] applications are built from primitive operations that can individually be evaluated either using HE or garbled circuits. An automated optimization based on integer programming or on a heuristic is used to determine an optimal solution. The run-time is estimated using a performance model, introduced in [SK11], that is parameterized by factors such as execution times of cryptographic primitives, bandwidth, and latency of the network. In Chapt. 5 we present our own solution to automated compilation of hybrid protocols from ANSI C input.

2.4.7 MPC Applications

At first MPC was seen as merely theoretic construct, however, a recent line of research has improved MPC primitives drastically and showed that practical implementations of MPC are possible. A very productive line of research, e.g., [MNPS04; BLW08; HEKM11; MLB12; CHK+12; HFKV12; KSMB13; CMTB13; DSZ14; LHS+14; LWN+15; BK15], has been devoted to positioning MPC as a practical tool and off-the-shelf solution for a wide variety of problems, and to minimize the complexity of the current schemes. Using these recent breakthroughs, the benefits of MPC can be utilized in some real-life applications such as [BCD+09; BTW12; BJSV15]. Despite the immense potential of MPC, it is still a great challenge to implement scalable real-world applications using MPC in practice.

Mixed-protocols have been used for several privacy-preserving applications, such as medical diagnostics [BFK+09; BFL+11], fingerprint recognition [HMEK11], iris- and finger-code authentication [BG11], computation on non-integers and Hidden Markov Models [FDH+13], and matrix factorization [NIW+13]. Privacy-preserving regression models for recommender systems are proposed in [NWI+13]. Hand-built hybrid protocols for neural networks were

presented in [LJLA17] and 3-party hybrid MPC protocols for machine learning were presented in [MR18]. A solution for MPC-based surveys for multiple participants is proposed in [BHKL18]. A privacy-preserving ridesharing system built on top of our ABY framework was proposed in [AHHK18]. Prelude [DCC18] also uses ABY to privately ensure correctness of interdomain routing using Software Defined Networking (SDN).

2.5 Private Information Retrieval (PIR)

PIR is a technique that was introduced by Chor et al. in the 1990s [CGKS95]. It refers to the privacy-preserving querying of data by a client from one or multiple data sources, such that these data sources cannot infer any information about the query or the query response. In contrast to the client's query, the available data in the database DB is considered public and does not need to be protected from the client. This allows for a trivial solution: Sending the entire database to the client, who then performs the query locally. However, this is usually impractical and expensive, especially so for large databases. PIR schemes allow clients to retrieve data without exposing their privacy, and require less communication (compared to sending the entire DB), albeit with computational overhead. PIR schemes can be viewed as a form of 1-out-of-B OT (cf. Sect. 2.3), where a receiver retrieves a single b-bit block out of B blocks. The difference is that in PIR the database DB is public, while in OT, blocks that are not queried must be hidden from the receiver. The communication in PIR needs to be strictly smaller than the size of the DB, while for OT such a restriction does not exist.

PIR protocols can be grouped into single-server schemes, that offer computational security and multi-server schemes, that can offer information-theoretic security but always require a non-collusion assumption between the PIR servers.

2.5.1 Multi-Server PIR

The first work that introduced the term PIR was presented by Chor et al. [CGKS95] and introduced information-theoretically secure PIR in a setting with multiple servers. We describe this scheme in more detail in Sect. 2.5.2. Several other multi-server PIR schemes followed: [Gol07] proposes multi-server PIR schemes with robustness properties built from cryptographic primitives like Shamir's secret sharing or HE, cf. Sect. 2.6.2. An experimental comparison of the multi-server PIR schemes of [CGKS95] and [Gol07] was given in [OG11]. A robust multi-server PIR scheme that allows multi-block queries was introduced in [HHG13]. Efficiency of robust multi-server PIR was improved in [DGH12; DG14]. Multi-server PIR with verifiability was proposed in [ZS14]. An efficient multi-server scheme based on secret sharing was presented in [Hen16].

2.5.2 The CGKS Scheme [CGKS95]

In this section we describe the original linear summation PIR scheme by Chor et al. [CGKS95]. An example query of this scheme is depicted in Fig. 2.2. A database DB is replicated on k PIR servers $\mathcal{S}_i$. The client $\mathcal{C}$ is interested in privately querying block_c at index c. The request q_i that $\mathcal{C}$ sends to server $\mathcal{S}_i$ is a randomly chosen string of B bits for $i \in \{1,\ldots,k-1\}$. The k-th request q_k corresponds to the XOR of all other requests except for one bit flipped at the index c of block_c. The result of the XOR of all requests is the elementary vector e_c with length B bits that has a 1 in position c and 0 everywhere else. The servers' responses have a length of b bits each and are the XOR of all blocks that the user requested in their query, i.e., if the bit at index j was set in the client's query q_i, the server XORs block_j into its response. When clients have received a reply from all servers they calculate the XOR of all k responses and get block_i, as all other blocks are contained an even number of times and cancel out due to the XOR. We generalize this scheme and improve the communication at the expense of a small number of symmetric cryptographic operations in RAID-PIR in Chapt. 8.

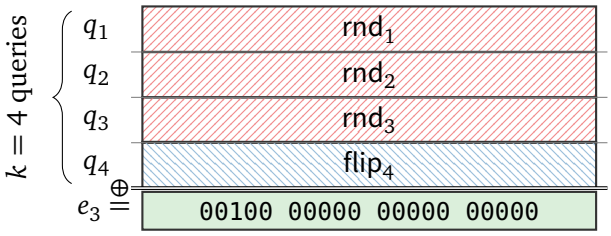


Figure 2.2: Example for querying the third block from a DB with $B = 20$ blocks using CGKS with $k = 4$ servers.

2.5.3 Single-Server PIR

Private Information Retrieval with a single computationally bounded server was first introduced in [KO97], and is often referred to as Computationally Private Information Retrieval (CPIR). Since then, several CPIR schemes have been proposed, e.g., with polylogarithmic communication [CMS99]; a survey of several CPIR schemes is given in [OS07]. In [CMO00] it was shown that CPIR implies Oblivious Transfer which gives strong evidence that CPIR cannot be constructed based on weak computational assumptions such as one-way functions. [SC07] claim that non-trivial CPIR protocols implemented on standard PC hardware are orders of magnitude less time-efficient than trivially transferring the entire database. However, a lattice-based CPIR scheme was proposed in [MG08] and experiments in [OG11] demonstrate that this scheme can be more efficient than downloading the database. By using a trusted hardware token, the computational assumptions for CPIR can be circumvented and information-theoretic security can be achieved, e.g., as shown in [WDDB06; YDDB08; DYDW10]. In [MBC13] it was shown how to exploit the massive parallelism available in cloud computing to split the server's workload on multiple machines using MapReduce. A CPIR scheme natively allowing multi-queries was given in [GKL10]. [DG14] constructs a

hybrid CPIR protocol that combines the multi-server PIR protocol of Goldberg [Gol07] with the single-server CPIR protocol of Melchor and Gaborit [MG08], for security even if *all* servers are corrupted. PIR scheme built from lattice-based cryptography was presented in [ABFK16]. A single-server scheme that allows multi-block queries and is used for anonymous messaging is presented in [AS16; ACLS18]. The combination of both approaches, called *hybrid PIR*, was presented in [DG14]. An extension of PIR, where data can directly be queried via keywords instead of locations, was proposed in [CGN98].

2.5.4 PIR Applications

There is a multitude of applications for PIR schemes, with different motivations for hiding the identity of items requested by the user. A typical reason is to prevent disclosure of personal or business interests in information from a database, e.g., patents, medical articles, company evaluations, product descriptions, or legal precedences. For example, knowledge about patent requests may allow a competitor to identify directions of a company, and knowledge about requests for medical papers by an individual may expose an illness. PIR can also be employed to improve the scalability of Tor, as proposed in PIR-Tor [MOT+11]. Cappos [Cap13] applies Chor et al.'s PIR scheme [CGKS95] to hide the specific software updates being retrieved, since knowing the requested update may allow an attacker to identify a outdated system, that might use potentially vulnerable software. PIR can also be used to privately query messages from an encrypted mailbox [SCM05; BKOS07; MOT+11] and is a building block in the private presence service DP5 [BDG15]. Moreover, building blocks of private and untraceable communication services be reused in other privacy-critical applications, such as electronic voting systems [BV14] or privacy-preserving location-based services [MCA06; HCE11; DSZ14]. A further interesting use case from [GHSG16], is to allow caching of encrypted web objects by an untrusted Content Delivery Network (CDN), preventing the CDN from learning details by identifying the requested objects.

2.6 Alternative Privacy-Preserving Techniques

There are several related techniques that aim to achieve similar goals like MPC, but are out of the main focus of this dissertation.

2.6.1 Oblivious RAM (ORAM)

Oblivious Random Access Memory (ORAM) [GO96] is more powerful than PIR as it allows not only private *retrieval* of data, but also private *write-access*. A combination of ORAM and PIR was presented recently in [MBC14]. Burst ORAM [DSS14] allows efficient online requests through precomputation. A simple ORAM scheme with small client storage was presented with Path ORAM [SDS+13]. An MPC framework that directly integrated ORAM was presented with ObliVM [LWN+15]. Recently, an efficient hierarchical ORAM scheme was proposed in [ACN+17].

2.6.2 Homomorphic Encryption (HE)

HE enables direct computation on encrypted data. The field can be grouped in additively homomorphic schemes, such as Paillier [Pai99; DGK09], that allow only additions and multiplications with public constants on ciphertexts. There are also somewhat homomorphic schemes, that additionally allow a limited number of multiplications of ciphertexts. Fully homomorphic encryption, that enables arbitrary operations on ciphertexts was initially proposed in [Gen09], but is typically not efficient enough for practical use cases. Other schemes are, e.g., [GGH+13; GKP+13]. However, such general schemes are typically too slow for practical applications [GHS12].

2.6.3 Intel SGX and Trusted Hardware

Intel Software Guard Extensions (SGX) [CD16; JDS+16] is a recent instruction set extension that allows programmers to perform computation on data stored within protected regions of memory that are not accessible by unauthorized processes. Despite its promise, SGX is currently the subject of many discussions regarding its real level of security, in contrast to MPC, which is a well-established methodology with proven security guarantees. A major concern regarding SGX programs is that timing or memory access patterns can leak information about private data. SGX does not include any mechanism for coping with such leaks [BMD+17]. While ORAM techniques can be used to mitigate these concerns [TLP+16], this comes at the price of increased complexity and non negligible obstacles to scalability.

Furthermore, there is ongoing research that focuses on applying side-channel attacks like Spectre and Meltdown to extract confidential data from SGX enclaves [BMW+18]. There also exist publicly available proof-of-concept implementations attacking SGX.[1]

Another general concern regarding SGX is Intel's role as the centralized point of trust for key distribution and attestation.

Because of the above SGX limitations, recent studies propose combining trusted execution environments like SGX with MPC to strengthen the privacy of outsourced computation [KPR+15; GMF+16]. This follows a line of research where trusted execution environments such as smart cards are used to enhance MPC protocols [FPS+11; JKSS10; DSZ14]

[1] https://github.com/lsds/spectre-attack-sgx

Part I

Tools for Efficient and Usable MPC

3 ABY: A Framework for Efficient Mixed-Protocol Secure Two-Party Computation

Results published in:

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. Code: https://encrypto.de/code/ABY. Internet Society, 2015. CORE Rank A*.

3.1 Introduction

MPC has made tremendous progress since the first theoretical feasibility results in the 1980s [Yao86; GMW87]. Ever since, several secure schemes have been introduced and repeatedly optimized, yielding a large variety of different secure computation protocols and flavors for several applications and deployment scenarios. This variety, however, has made the development of efficient secure computation protocols a challenging task for non-experts, who want to choose an efficient protocol for their specific functionality and available resources. Furthermore, since at this point it is unclear which protocol is advantageous in which situation, a developer would first need to prototype each scheme for his specific requirements before he can start implementing the chosen scheme. This task becomes even more tedious, time-consuming, and error-prone, since each secure computation protocol has its own specifig representation, like Arithmetic or Boolean circuits, in which a functionality has to be described.

The development of efficient secure computation protocols for a particular function and deployment scenario has recently been addressed by IARPA in a request for information (RFI) [IAR14]. Part of the vision that is given in this RFI is the automated generation of secure computation protocols that perform well for novel applications and that can be used by a non-expert in secure computation. As shown in Sect. 2.4.6, several tools have started to bring this vision towards reality by introducing an abstract language that is compiled into a protocol representation, thereby relieving a developer from having to specify the functionality in the protocol's (often complex) underlying representation. These languages and compilers, however, are often tailored to one particular secure computation protocol and translate programs directly into the protocol's representation. The efficiency of protocols that

are generated by these compilers is hence bounded by the possibility to efficiently represent the function in the particular representation, e.g., the multiplication of two ℓ-bit numbers has a very large Boolean circuit representation of size $\mathcal{O}(\ell^2)$, or $\mathcal{O}(\ell^{1.59})$ when using Karatsuba multiplication [KO62; HKS+10].

To overcome the dependence on an efficient function representation and to improve efficiency, several works proposed to mix secure computation protocols based on homomorphic encryption with Yao's garbled circuits protocol, e.g., [BPSW07; BFK+09; HMEK11; SK11; BG11; KSS13b; NWI+13; NIW+13; FDH+13]. The general idea behind such mixed-protocols is to evaluate operations that have an efficient representation as an Arithmetic circuit (i.e., additions and multiplications) using homomorphic encryption and operations that have an efficient representation as a Boolean circuit (e.g., comparisons) using Yao's garbled circuits. These previous works show that using a mixed-protocol approach can result in better performance than using only a single protocol. Several tools have been developed for designing mixed-protocols, e.g., [HKS+10; SKM11; BLR13; BLR14], which allow the developer to specify the functionality and the assignment of operations to secure computation protocols. The assignment can even be done automatically as shown recently in [KSS14]. However, since the conversion costs between homomorphic encryption and Yao's garbled circuits protocol are relatively expensive and the performance of homomorphic encryption scales very poorly with increasing security parameter, these mixed-protocols achieve only relatively small run-time improvements over using a single protocol.

3.1.1 Overview and Our Contributions

We present *ABY* (for Arithmetic, Boolean, and Yao sharing), a novel framework for developing highly efficient mixed-protocols that allows a flexible design process. We design ABY using several state-of-the-art techniques in secure computation and by applying existing protocols in a novel fashion. We optimize sub-routines and perform a detailed benchmark of the primitive operations. From these results we derive new insights for designing efficient secure computation protocols. We apply these insights and demonstrate the design flexibility of ABY by implementing three privacy-preserving applications: modular exponentiation, private set intersection, and biometric matching. We give an overview of our framework and describe our contributions in more detail next. ABY is intended as a base-line on the performance of privacy-preserving applications, since it combines several state-of-the-art techniques and best practices in secure computation. The source code of ABY is freely available on GitHub at `https://encrypto.de/code/ABY`.

The ABY Framework On a very high level, our framework works like a virtual machine that abstracts from the underlying secure computation protocols (similar to the Java Virtual Machine that abstracts from the underlying system architecture). Our virtual machine operates on data types of a given bit-length (similar to 16-bit `short` or 32-bit `long` data types in the C programming language). Variables are either in **C**leartext (meaning that one party knows the value of the variable, which is needed for inputs and outputs of the computation) or secret shared among the two parties (meaning that each party holds a share from which

it cannot deduce information about the value). Our framework currently supports three different types of sharings (**A**rithmetic, **B**oolean, and **Y**ao) and allows to efficiently convert between them, cf. Fig. 3.1. The sharings support different types of standard operations that are similar to the instruction set of a CPU such as addition, multiplication, comparison, or bitwise operations. Operations on shares are performed using highly efficient secure computation protocols: for operations on **A**rithmetic sharings we use protocols based on Beaver's multiplication triples [Bea91], for operations on **B**oolean sharings we use the protocol of Goldreich-Micali-Wigderson (GMW) [GMW87], and for operations on **Y**ao sharings we use Yao's garbled circuits protocol [Yao86].

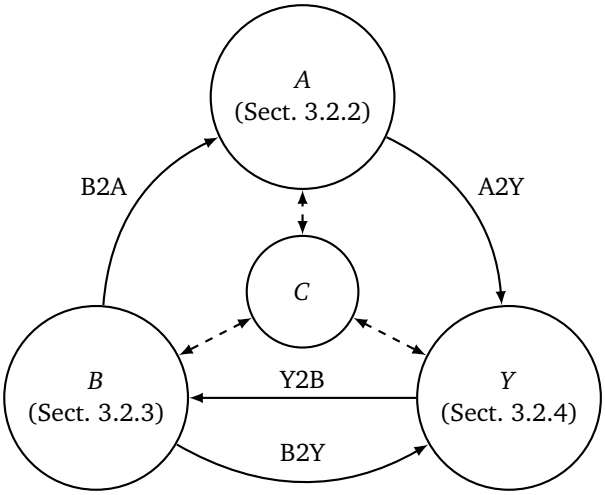


Figure 3.1: Overview of the ABY framework that allows efficient conversions between **C**leartexts and three types of sharings: **A**rithmetic, **B**oolean, and **Y**ao.

Flexible Design Process A main goal of our framework is to allow a flexible design of secure computation protocols.

1) We *abstract from the protocol-specific function representations* and instead use standard operations. This allows to mix several protocols, even with different representations, and allows the designer to express the functionality in form of standard operations as known from high-level programming languages such as C or Java. Previously, designers had to manually compose (or automatically generate) a compact representation for the specific protocol, e.g., a small Boolean circuit for Yao's protocol. As we focus on standard operations, high-level languages can be compiled into our framework (cf. Chapt. 5) and it can be used as backend in several existing secure computation tools, e.g., L1 [SKM11; SK11; KSS14], SecreC [BLR13; BLR14], or PICCO [ZSB13]. ABY has been used as backend by follow-up work, such as EzPC [CGR+17].

2) By *mixing secure computation protocols*, our framework is able to tailor the resulting protocol to the resources available in a given deployment scenario. For example, the GMW protocol allows to precompute all cryptographic operations, but the online phase requires

several rounds of interaction (which is an issue for networks with high latency), whereas Yao's protocol has a constant number of rounds, but requires symmetric cryptographic operations in the online phase.

Efficient Instantiation and Improvements Each of the secure computation techniques is implemented using most recent optimizations and best practices such as batch precomputation of expensive cryptographic operations [DKL+13; CHK+12; SZ13]. For Arithmetic sharing (Sect. 3.2.2) we generate multiplication triples via Paillier with packing [Pai99; Pul13] or DGK with full decryption [DGK08; Gei10], for Boolean sharing (Sect. 3.2.3) we use the multiplexer of [MS13] and OT extension [IKNP03; ALSZ13], and for Yao sharing (Sect. 3.2.4) we use free XOR [KS08b], fixed-key AES garbling [BHKR13], and half-gates [ZRE15] As novel contributions and advances over state-of-the-art techniques for efficient protocol design, we combine existing approaches in a novel way. For Arithmetic sharing, we show how to multiply values using symmetric key cryptography which allows faster multiplication by one to three orders of magnitude (Sect. 3.2.2). We outline how to efficiently convert from Boolean respectively Yao sharing to Arithmetic sharing, and show how to combine Boolean and Yao sharing to achieve better run-time compared to a pure Boolean or Yao instantiation in our paper [DSZ15]. Finally, we outline how to modify the fixed-key AES garbling of [BHKR13] to achieve better performance in OT extension (Sect. 3.3.1).

Feedback on Efficient Protocol Design We perform benchmarks of our framework from which we derive new best-practices for efficient secure computation protocols. We show that for multiplications in many cases OT extensions for precomputing multiplication triples is faster than homomorphic encryption (Sect. 3.3.3). With our OT-based conversion protocols, converting between different share representations is considerably cheaper than the methods used in previous works, e.g., [HKS+10; KSS14], and scales well with increasing security parameter. In fact, on a low latency network, the conversion costs between different share representations are so cheap that already for a single multiplication it pays off to convert into a more suited representation, perform the multiplication, and convert back into the source representation, as shown in Sect. 3.3.4.

Applications We show that our ABY framework and techniques can be used to implement and improve performance of several privacy-preserving applications. We present implementations of mixed protocols for modular exponentiation, PSI and biometric matching in our paper [DSZ15]. Follow-up work line MiniONN [LJLA17] relied on ABY to implement neural networks.

3.2 Sharing Types

In this section we detail the sharing types that our ABY framework uses: **A**rithmetic sharing (Sect. 3.2.2), **B**oolean sharing (Sect. 3.2.3), and **Y**ao sharing (Sect. 3.2.4). For each

sharing type we describe the semantics of the sharing, standard operations, and the state of the art in the respective sub-sections.

3.2.1 Notation

We denote a shared variable x as $\langle x \rangle^t$. The superscript $t \in \{A, B, Y\}$ indicates the type of sharing, where A denotes Arithmetic sharing, B denotes Boolean sharing, and Y denotes Yao sharing. The semantics of the different sharing types and operations are defined in Sect. 3.2. We also allow inputs and outputs to be in cleartext form denoted by C. We refer to the individual share of $\langle x \rangle^t$ that is held by party P_i as $\langle x \rangle_i^t$. In a similar fashion, we define a sharing operator $\langle x \rangle^t = \mathsf{Shr}_i^t(x)$ meaning that P_i shares its input value x with P_{1-i} and a reconstruction operator $x = \mathsf{Rec}_i^t\left(\langle x \rangle^t\right)$ meaning that P_i obtains the value of x as output. When both parties obtain the value of x, we write $\mathsf{Rec}^t\left(\langle x \rangle^t\right)$. We denote the conversion of a sharing of representation $\langle x \rangle^s$ into another representation $\langle x \rangle^d$ with $s, d \in \{A, B, Y\}$ and $s \neq d$ as $\langle x \rangle^d = s2d(\langle x \rangle^s)$, e.g., A2B converts an Arithmetic share into a Boolean share. Note that we require that no party learns any additional information about x during this conversion. When performing an operation $\odot$ on shares, we write $\langle z \rangle^d = \langle x \rangle^{s_0} \odot_d^{s_0,s_1} \langle y \rangle^{s_1}$, for $\odot_d^{s_0,s_1} : \langle x \rangle^{s_0} \times \langle y \rangle^{s_1} \mapsto \langle z \rangle^d$ and $s_0, s_1, d \in \{A, B, Y\}$. If all three variables are of the same type, i.e., $s_0 = s_1 = d$, we write $\langle z \rangle^d = \langle x \rangle^{s_0} \odot \langle y \rangle^{s_1}$.

3.2.2 Arithmetic Sharing

For Arithmetic sharing an ℓ-bit value x is shared additively in the ring $\mathbb{Z}_{2^\ell}$ (integers modulo 2^ℓ) as the sum of two values. The described protocols are based on [ABL+04; PBS12; KSS14]. First we define the sharing semantics and operations and give an overview over related work on secure computation based on Arithmetic sharing. Then we detail how to generate Arithmetic multiplication triples using homomorphic encryption. In the following, we assume all Arithmetic operations to be performed in the ring $\mathbb{Z}_{2^\ell}$, i.e., all operations are $(\mathrm{mod}\ 2^\ell)$.

Sharing Semantics

Arithmetic sharing is based on additively sharing private values between the parties. Every ℓ-bit value $\langle x \rangle^A$ of x is secret shared as $\langle x \rangle_0^A + \langle x \rangle_1^A \equiv x \pmod{2^\ell}$ with $\langle x \rangle_0^A, \langle x \rangle_1^A \in \mathbb{Z}_{2^\ell}$. To secret share arithmetic inputs, parties run $\mathsf{Shr}_i^A(x)$: P_i chooses $r \in_R \mathbb{Z}_{2^\ell}$, sets $\langle x \rangle_i^A = x - r$, and sends r to P_{1-i}, who sets $\langle x \rangle_{1-i}^A = r$. To reconstruct arithmetic outputs, parties run $\mathsf{Rec}_i^A(x)$: P_{1-i} sends its share $\langle x \rangle_{1-i}^A$ to P_i who computes $x = \langle x \rangle_0^A + \langle x \rangle_1^A$.

In an outsourcing scenario a separate input party chooses $r \in_R \mathbb{Z}_{2^\ell}$ and shares input x by computing $\langle x \rangle_1^A = x - r \pmod{2^\ell}$, and sending $r = \langle x \rangle_0^A$ to P_0 and $\langle x \rangle_1^A$ to P_1. For outsourced output reconstruction P_0 and P_1 send their shares to an output party who computes the plaintext $x = \langle x \rangle_0^A + \langle x \rangle_1^A \pmod{2^\ell}$.

Operations

Every Arithmetic circuit is a sequence of addition and multiplication gates, evaluated as follows: For addition $\langle z\rangle^A = \langle x\rangle^A + \langle y\rangle^A$ P_i locally computes $\langle z\rangle_i^A = \langle x\rangle_i^A + \langle y\rangle_i^A$. The multiplication $\langle z\rangle^A = \langle x\rangle^A \cdot \langle y\rangle^A$ is performed using a precomputed Arithmetic multiplication triple [Bea91] of the form $\langle c\rangle^A = \langle a\rangle^A \cdot \langle b\rangle^A$: P_i sets $\langle e\rangle_i^A = \langle x\rangle_i^A - \langle a\rangle_i^A$ and $\langle f\rangle_i^A = \langle y\rangle_i^A - \langle b\rangle_i^A$, both parties perform $\mathsf{Rec}^A(e)$ and $\mathsf{Rec}^A(f)$, and P_i sets $\langle z\rangle_i^A = i \cdot e \cdot f + f \cdot \langle a\rangle_i^A + e \cdot \langle b\rangle_i^A + \langle c\rangle_i^A$. We discuss protocols to precompute Arithmetic multiplication triples later in this section.

State-of-the-Art

The protocols we employ in Arithmetic sharing use additive sharing in the ring $\mathbb{Z}_{2^\ell}$. They were described in [ABL+04; PBS12; KSS14], and provide security in the semi-honest setting. The BGW protocol [BGW88] was the first protocol for secure multi-party computation of Arithmetic circuits that is secure against semi-honest parties for up to $t < n/2$ corrupt parties and secure against malicious adversaries for up to $t < n/3$ corrupt parties. The Virtual Ideal Function Framework (VIFF) [DGKN09] is a generic software framework for secure computation schemes in asynchronous networks and implemented secure computation using precomputed Arithmetic multiplication triples. The SPDZ protocol [DPSZ12; DKL+13] allows secure computation in the presence of $t = n-1$ corrupted parties in the malicious model; a run-time environment for the SPDZ protocol was presented in [KSS13a; KOS16]. Arithmetic circuits for computing various primitives have been proposed in [CH10; CS10].

Generating Arithmetic Multiplication Triples via Additively Homomorphic Encryption

Typically, Arithmetic multiplication triples of the form $\langle a\rangle^A \cdot \langle b\rangle^A = \langle c\rangle^A$ are generated in the setup phase using an additively homomorphic encryption scheme as shown in Prot. 3.1. This protocol for generating multiplication triples was mentioned as "well known folklore" in [ABL+04, Appendix A]. For homomorphic encryption we use either the cryptosystem of Paillier [Pai99; DJ01; DJN10], or the one of Damgård-Geisler-Krøigaard (DGK) [DGK08; DGK09] with full decryption using the Pohlig-Hellman algorithm [PH78] as described in [Gei10; Mak10; BG11]. In Paillier encryption, the plaintext space is $\mathbb{Z}_N$ and we use statistical blinding with parameter r; in DGK encryption we set the plaintext space to be $\mathbb{Z}_{2^{2\ell+1}}$ and use perfect blinding with parameter r. For proofs of security and correctness we refer to [PBS12] and [Pul13].

Complexity For asymmetric security parameter φ, to generate an ℓ-bit multiplication triple, P_0 and P_1 exchange 3 ciphertexts, each of length 2φ bits for Paillier (resp. φ bits for DGK), resulting in a total communication of 6φ bits (resp. 3φ bits). For Paillier encryption we also use the packing optimization described in [PBS12] that packs together multiple messages from P_1 to P_0 into a single ciphertext, which reduces the number of decryptions and reduces communication per multiplication triple to $4\varphi + 2\varphi/\lfloor \varphi/(2\ell+1+\sigma)\rfloor$ bits.

P_0	P_1, holding public key of P_0
$\langle a\rangle_0^A, \langle b\rangle_0^A \in_R \mathbb{Z}_{2^\ell}$	$\langle a\rangle_1^A, \langle b\rangle_1^A \in_R \mathbb{Z}_{2^\ell}$ $r \in_R \mathbb{Z}_{2^{2\ell+1+\sigma}}$ for Paillier (resp. $r \in_R \mathbb{Z}_{2^{2\ell+1}}$ for DGK) $\langle c\rangle_1^A = \langle a\rangle_1^A \cdot \langle b\rangle_1^A - r \pmod{2^\ell}$
	$\xrightarrow{\mathsf{Enc}_0(\langle a\rangle_0^A), \mathsf{Enc}_0(\langle b\rangle_0^A)}$
	$\xleftarrow{d = \mathsf{Enc}_0(\langle a\rangle_0^A)^{\langle b\rangle_1^A} \cdot \mathsf{Enc}_0(\langle b\rangle_0^A)^{\langle a\rangle_1^A} \cdot \mathsf{Enc}_0(r)}$
$\langle c\rangle_0^A = \langle a\rangle_0^A \cdot \langle b\rangle_0^A + \mathsf{Dec}_0(d) \pmod{2^\ell}$	

Protocol 3.1: Generating Arithmetic MTs via HE.

Generating Arithmetic Multiplication Triples via Oblivious Transfer

Instead of using homomorphic encryption, Arithmetic multiplication triples can be generated based on OT extension. The protocol was proposed in [Gil99, Sect. 4.1] and used in [BCF+14]. It allows to efficiently compute the product of two secret-shared values using OT. In our paper [DSZ15], we describe a variant of the protocol that uses slightly more efficient correlated OT extension.

3.2.3 Boolean Sharing

The Boolean sharing uses XOR-based secret sharing to share a variable. We evaluate functions represented as Boolean circuits using the protocol by Goldreich-Micali-Wigderson (GMW) [GMW87]. In the following, we first define the sharing semantics, describe how operations are performed, and give an overview over related work.

Sharing Semantics

Boolean sharing is built from XOR-based secret sharing. To simplify presentation, we assume single bit values; for ℓ-bit values each operation is performed ℓ times in parallel.

A Boolean share $\langle x\rangle^B$ of a private bit x is shared between the two parties, such that $\langle x\rangle_0^B \oplus \langle x\rangle_1^B = x$ with $\langle x\rangle_0^B, \langle x\rangle_1^B \in \mathbb{Z}_2$. To share a Boolean input value $\mathsf{Shr}_i^B(x)$ P_i chooses $r \in_R \{0,1\}$, computes $\langle x\rangle_i^B = x \oplus r$, and sends r to P_{1-i} who sets $\langle x\rangle_{1-i}^B = r$. For reconstructing a Boolean value $\mathsf{Rec}_i^B(x)$ P_{1-i} sends its share $\langle x\rangle_{1-i}^B$ to P_i who computes $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$.

In an outsourcing scenario a separate input party chooses $r \in_R \{0,1\}$, shares input x by computing $\langle x\rangle_1^B = x \oplus r$, and sending $r = \langle x\rangle_0^B$ to P_0 and $\langle x\rangle_1^B$ to P_1. For outsourced output reconstruction P_0 and P_1 send their shares to an output party who computes the plaintext $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$.

Operations

Every efficiently computable function can be expressed as a Boolean circuit consisting of XOR and AND gates, for which we detail the evaluation in the following. An XOR $\langle z \rangle^B = \langle x \rangle^B \oplus \langle y \rangle^B$ is computed locally by every party P_i as $\langle z \rangle_i^B = \langle x \rangle_i^B \oplus \langle y \rangle_i^B$. An AND $\langle z \rangle^B = \langle x \rangle^B \wedge \langle y \rangle^B$ is evaluated using a precomputed Boolean multiplication triple $\langle c \rangle^B = \langle a \rangle^B \wedge \langle b \rangle^B$ as follows [Bea91]: P_i computes $\langle e \rangle_i^B = \langle a \rangle_i^B \oplus \langle x \rangle_i^B$ and $\langle f \rangle_i^B = \langle b \rangle_i^B \oplus \langle y \rangle_i^B$, both parties perform $\mathsf{Rec}^B(e)$ and $\mathsf{Rec}^B(f)$, and P_i sets $\langle z \rangle_i^B = i \cdot e \cdot f \oplus f \cdot \langle a \rangle_i^B \oplus e \cdot \langle b \rangle_i^B \oplus \langle c \rangle_i^B$. As described in [ALSZ13], a Boolean multiplication triple can be precomputed efficiently using two R-OTs run in opposite directions. For multiplexer operations we use a protocol proposed in [MS13] that requires only R-OT$_\ell^2$, whereas evaluating a MUX circuit with ℓ AND gates requires R-OT$_1^{2\ell}$. More detailed, an ℓ-bit vector AND (or multiplexer) is evaluated using a vector multiplication triple, which consists of two random bits $a_0, a_1 \in \{0,1\}$ and four random ℓ-bit strings $b_0, b_1, c_0, c_1 \in \{0,1\}^\ell$ with $c_0[i] \oplus c_1[i] = (a_0 \oplus a_1) \wedge (b_0[i] \oplus b_1[i])$ for $1 \leq i \leq \ell$, where $[i]$ denotes the i-th bit of a string. Similar to a regular multiplication triple, a vector multiplication triple can be generated using two OTs on random inputs [PSZ14].

For further standard functionalities we use the depth-optimized circuit constructions summarized in [SZ13].

State-of-the-Art

The first implementation of the GMW protocol for multiple parties and with security in the semi-honest model was given in [CHK+12]. Optimizations of this framework for the two-party setting were proposed in [SZ13] and further improvements to efficiently precompute multiplication triples using R-OT extension were given in [ALSZ13]. These works show that the GMW protocol achieves good performance in low-latency networks. TinyOT [NNOB12; LOS14] extended the GMW protocol to the covert and malicious model.

3.2.4 Yao Sharing

In Yao's garbled circuits protocol [Yao86] for secure two-party computation, one party, called *garbler*, encrypts a Boolean function to a garbled circuit, which is evaluated by the other party, called *evaluator*. More detailed, the garbler represents the function to be computed as Boolean circuit and assigns to each wire w two wire keys (k_0^w, k_1^w) with $k_0^w, k_1^w \in \{0,1\}^\kappa$. The garbler then encrypts the output wire keys of each gate on all possible combinations of the two input wire keys using an encryption function Gb. He then sends the garbled circuit (consisting of all garbled gates), together with the corresponding input keys of the circuit to the evaluator. The evaluator iteratively decrypts each garbled gate using the gate's input wire keys to obtain the output wire key and finally reconstructs the cleartext circuit output.

In the following, we assume that P_0 acts as garbler and P_1 acts as evaluator and detail the Yao sharing assuming a garbling scheme that uses the free-XOR [KS08b], point-and-permute [MNPS04], and half-gates [ZRE15] optimizations. Using these techniques, the

garbler randomly chooses a global κ-bit string R with $R[0] = 1$. For each wire w, the wire keys are $k_0^w \in_R \{0,1\}^\kappa$ and $k_1^w = k_0^w \oplus R$. The least significant bit $k_0^w[0]$ resp. $k_1^w[0] = 1 - k_0^w[0]$ is called permutation bit. We point out that the Yao sharing can also be instantiated with other garbling schemes.

Sharing Semantics

Intuitively, P_0 holds for each wire w the two keys k_0^w and k_1^w and P_1 holds one of these keys without knowing to which of the two cleartext values it corresponds. To simplify presentation, we assume single bit values; for ℓ-bit values each operation is performed ℓ times in parallel. A garbled circuits share $\langle x \rangle^Y$ of a value x is shared as $\langle x \rangle_0^Y = k_0$ and $\langle x \rangle_1^Y = k_x = k_0 \oplus xR$. To share an input $\mathsf{Shr}_0^Y(x)$, the garbler P_0 samples $\langle x \rangle_0^Y = k_0 \in_R \{0,1\}^\kappa$ and sends $k_x = k_0 \oplus xR$ to the evaluator P_1. $\mathsf{Shr}_1^Y(x)$: both parties run C-OT$_\kappa^1$ where P_0 acts as sender, inputs the correlation function $f_R(x) = (x \oplus R)$ and obtains $(k_0, k_1 = k_0 \oplus R)$ with $k_0 \in_R \{0,1\}^\kappa$ and P_1 acts as receiver with choice bit x and obliviously obtains $\langle x \rangle_1^Y = k_x$. To obtain the plaintext output $\mathsf{Rec}_i^Y(x)$, P_{1-i} sends its permutation bit $\pi = \langle x \rangle_{1-i}^Y[0]$ to P_i who computes $x = \pi \oplus \langle x \rangle_i^Y[0]$.

Operations

Using Yao sharing, a Boolean circuit consisting of XOR and AND gates is evaluated as follows: The XOR $\langle z \rangle^Y = \langle x \rangle^Y \oplus \langle y \rangle^Y$ is evaluated using the free-XOR technique [KS08b], where P_i locally computes $\langle z \rangle_i^Y = \langle x \rangle_i^Y \oplus \langle y \rangle_i^Y$. The AND $\langle z \rangle^Y = \langle x \rangle^Y \wedge \langle y \rangle^Y$ is evaluated as follows: P_0 creates a garbled table using $\mathsf{Gb}_{\langle z \rangle_0^Y}\left(\langle x \rangle_0^Y, \langle y \rangle_0^Y\right)$, where Gb is a garbling function as defined in [BHKR13]. P_0 sends the garbled table to P_1, who decrypts it using the keys $\langle x \rangle_1^Y$ and $\langle y \rangle_1^Y$ to obtain $\langle z \rangle_1^Y$. Other standard functionalities are implemented using the size-optimized circuit constructions summarized in [KSS09].

State-of-the-Art

Beyond the optimizations mentioned above, several further improvements for Yao's garbled circuits protocol exist: garbled-row reduction [NPS99; PSSW09] and pipelining [HEKM11], where garbled tables are sent in the online phase. A popular implementation of Yao's garbled circuits protocol in the semi-honest model was presented in [HEKM11]. A formal definition for garbling schemes, as well as an efficient instantiation of Gb using fixed-key AES was given in [BHKR13]. In our implementation we use these state-of-the-art optimizations of Yao's garbled circuits protocol except pipelining (we want to minimize the complexity of the online phase and hence generate and transfer garbled circuits in the setup phase). To achieve security against covert and malicious adversaries, some implementations use the cut-and-choose technique, e.g., [KSS12; FN13; SS13; CMTB13]. The most recent optimization that we also implement in ABY is half-gates [ZRE15], that allows to reduce the communication even further at the expense of slightly more computation.

3.3 Implementation and Benchmarks

In the following section, we provide insights on the implementation of our ABY framework (Sect. 3.3.1). We then outline the deployment scenarios for our benchmarks (Sect. 3.3.2). We perform a theoretical and empirical comparison of the multiplication triple generation using Paillier, DGK, and OT (Sect. 3.3.3). We list one-time initialization costs in Sect. 3.3.3. Finally, we benchmark protocol conversions and primitive operations (Sect. 3.3.4).

3.3.1 Design and Implementation

The main design-goal of our ABY framework is to achieve an *efficient online phase*, which is why we batch-precompute all cryptographic operations in parallel in the setup phase (the only remaining cryptographic operations in the online phase is symmetric crypto for evaluating garbled circuits). If precomputation is not possible, the setup and online phase could be interleaved to decrease the total computation time. Our framework has a *modular design* that can easily be extended to additional secure computation schemes, computing architectures, and new operations, while also allowing special-purpose optimizations on all levels of the implementation. Our framework allows to focus on *applications* by abstracting from internal representations of sharings and protocol details.

We build on the C++ GMW and Yao's garbled circuits implementation of [CHK+12] with the optimized two-party GMW routines of [SZ13], the fixed-key AES garbling routine of [BHKR13], half-gates [ZRE15], and the OT extension implementation of [ALSZ13]. The generation of Arithmetic multiplication triples using Paillier and DGK is written in C using the GNU Multiple Precision Arithmetic Library (GMP) and was inspired by `libpaillier`[1]. We include several algorithmic optimizations for Paillier's cryptosystem as proposed in [DJN10] and use packing [Pai99; Pul13] to combine several multiplication triples into one Paillier ciphertext. Our implementation optimizations for both Paillier and DGK include encryption using fixed-base exponentiation and the Chinese remainder theorem (CRT), as well as decryption using CRT. In the multiplication triple protocol we use double-base exponentiations.

For Boolean and Yao sharings, we implement addition (ADD), multiplication (MUL), comparison (CMP), equality test (EQ), and multiplexers (MUX) using optimized circuit constructions described in [KSS09; SZ13; MS13]. We benchmark the Boolean sharing on depth-optimized circuits and the Yao sharing on size-optimized circuits. For arithmetic sharing, we only implement addition and multiplication. Protocols for bitwise operations on arithmetic sharings can be realized using bit-decomposition. More efficient protocols for EQ and CMP on Arithmetic shares were proposed in [CH10], but they either require $O(\ell)$ multiplications of ciphertexts in an order-q subgroup (i.e., for symmetric security parameter κ, q is a prime of bit length 2κ) and constant rounds or $O(\ell)$ multiplications of cipher-texts with elements in a small field (e.g., $\mathbb{Z}_{2^8}$) and $O(\log_2 \ell)$ rounds. In contrast, the EQ and CMP we use need only $O(\ell)$

[1] http://acsc.cs.utexas.edu/libpaillier/

symmetric cryptographic operations and constant rounds: we transform the Arithmetic share into a Yao share and perform the operations with Yao.

ABY supports inputs to be provided by the parties P_0 and P_1 or to be supplied from external input parties in an outsourcing scenario. Similarly, outputs can be reconstructed by P_0, P_1 and also by separate output parties that receive outputs from outsourced MPC.

3.3.2 Deployment Scenarios

For the performance evaluation of ABY, we use two deployment scenarios: a *local* setting (low-latency, high-bandwidth) and an intercontinental *cloud* setting (high-latency). These two scenarios cover two extremes in the design space as latency heavily affects the performance of Boolean and Arithmetic sharings.

Local setting In the local setting, we run the benchmarks on two Desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, that are connected via Gigabit-LAN. The average run-time variance in the local setting was 15%. For algorithms which use a pipelined computation process (e.g., the multiplication triple generation algorithms), we send packets of size 50 kB.

Cloud setting In the cloud setting, we run the benchmarks on two Amazon EC2 `c3.large` instances with a 64-bit Intel Xeon dualcore CPU with 2.8 GHz and 3.75 GB RAM. One virtual machine is located at the US east coast and the other one in Japan. The average bandwidth in this scenario was 70 MBit/s, while the latency was 170 ms. In our measurements we rarely encountered outliers with more than twice of the average run-time, probably caused by the network, which we omit from the results. The resulting average run-time variance in the cloud setting was 25%. For algorithms which use a pipelined computation process, we operate on larger blocks compared to the local setting and send packets of size 32 MB to achieve a lower number of communication rounds.

We run all benchmarks using two threads in the setup phase (except for Yao's garbled circuits, which we run with one thread as its possibility to parallelize depends on the circuit structure) and one thread in the online phase. All machines use the AES new-instruction set (AES-NI) for maximum efficiency of symmetric cryptographic operations. All experiments are the average of 10 executions unless stated otherwise.

3.3.3 Efficient Multiplication Triple Generation

We benchmark the generation of Arithmetic multiplication triples (used for multiplication in Arithmetic sharing, cf. Sect. 3.2.2) for legacy-, medium-, and long-term security parameters (cf. Sect. 2.1) and for typical data type sizes used in programming languages ($\ell \in \{8, 16, 32, 64\}$ bits) using two threads. We measure the generation of 100 000 multiplication triples excluding the time for the base-OTs and generation of public and private keys, which we depict separately in Sect. 3.3.3, since they only need to be computed once and amortize fairly quickly. The

communication costs and average run-times for generating one multiplication triple are depicted in Tab. 3.1.

Table 3.1: Overall amortized complexities for generating one ℓ-bit multiplication triple using Homomorphic Encryption or Oblivious Transfer Extension using two threads. Smallest values marked in bold.

	Communication [Bytes]				**Time** [μs]							
					Local				Cloud			
Bit-length ℓ	8	16	32	64	8	16	32	64	8	16	32	64
Paillier-based												
legacy	528	531	541	555	245	246	278	328	842	867	990	1 139
medium	1 039	1 043	1 051	1 067	1 430	1 475	1 572	1 748	4 485	4 654	5 198	5 669
long	1 551	1 555	1 563	1 579	4 309	4 374	4 565	4 957	12 990	13 080	13 805	14 614
DGK-based												
legacy	384	384	**384**	**384**	94	104	151	322	449	464	572	1 134
medium	768	768	**768**	**768**	259	313	465	1 020	971	1 128	1 651	3 107
long	1 152	1 152	**1 152**	**1 152**	534	629	929	2 005	1 894	2 118	3 049	6 319
Oblivious Transfer Extension-based												
legacy	**169**	**354**	772	1 800	**3**	**4**	**8**	**20**	**39**	**62**	**86**	**170**
medium	**233**	**482**	1 028	2 312	**3**	**6**	**10**	**24**	**44**	**77**	**107**	**219**
long	**265**	**546**	1 156	2 568	**3**	**6**	**11**	**27**	**46**	**82**	**110**	**224**

The OT-based protocol is in all tested cases faster than the Paillier-based and the DGK-based protocols: in the local setting by a factor of 15 – 1 400 for Paillier and by a factor of 15 – 180 for DGK and in the cloud setting by a factor of 6 – 280 for Paillier and by a factor of 6 – 40 for DGK. DGK is more efficient than Paillier for all parameters due to the shorter exponents for encryption and smaller ciphertext size. The run-time of DGK depends heavily on the bit-length ℓ of the multiplication triples, such that for very large values of ℓ Paillier might be preferable. In terms of communication, the DGK-based protocol is better than the OT-based protocol for longer bit-lengths ($\ell = 32$ and $\ell = 64$), at most by factor 4, while for short bit-lengths it is the opposite.

Overall, our experiments demonstrate that using OT to precompute multiplication triples is substantially faster than using homomorphic encryption and scales much better to higher security levels. Moreover, for homomorphic encryption our method of batching together all homomorphic encryption operations in the setup phase allows to make full use of optimizations such as packing. In contrast, when using homomorphic encryption for additions/multiplications during the online phase of the protocol, as it was used in previous works (cf. Sect. 3.1), such optimizations can only be done when the same homomorphic operations are computed in parallel, which depends on the application. This clearly demonstrates that using OT and multiplication triples is much more efficient than using homomorphic encryption.

Initialization Costs

In Tab. 3.2 we give the initialization costs for the Paillier-based, DGK-based, and OT-based multiplication triple generation. For Paillier and DGK, these costs include key generation, key

exchange and precomputations for fixed-base exponentiations. The key generation (given in parentheses) has to be done only once by the server, as keys can be re-used for multiple clients. The key exchange and fixed-base precomputation have to be performed per-client. The depicted values are for $\ell = 64$-bit multiplication triples. Smaller multiplication triple sizes will result in slightly faster key generation for DGK. For OT, the initialization costs include the Naor-Pinkas base-OTs [NP01], which have to be performed once between each client and server. Note that the base-OTs are also required for Boolean and Yao sharing, but only need to be computed once.

Table 3.2: Initialization costs for homomorphic encryption and OT-based protocols for different security parameters and $\ell = 64$-bit multiplication triples.

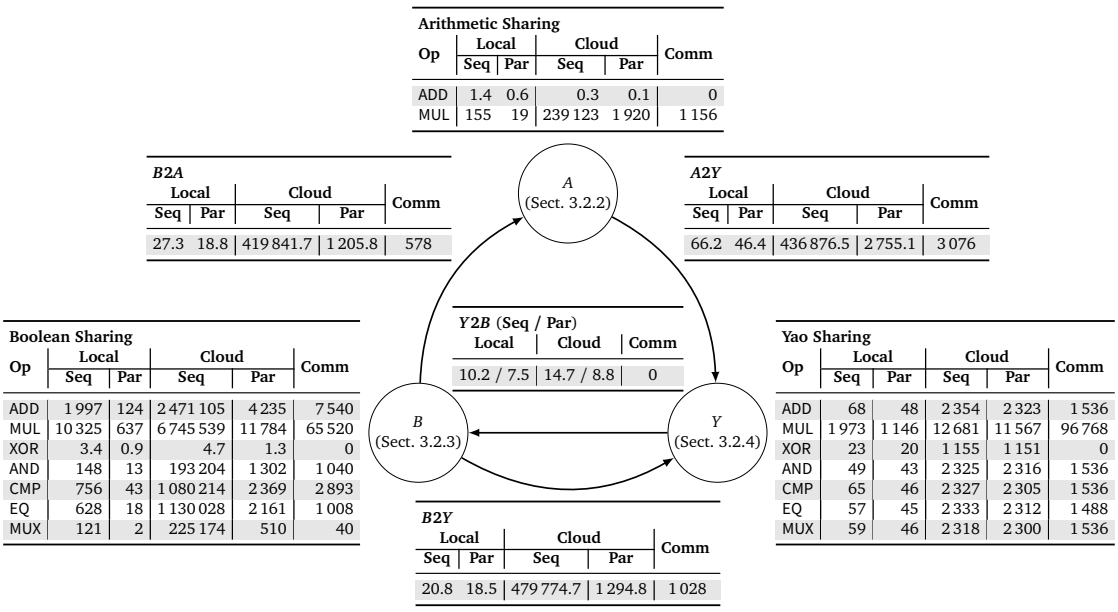
Security Level	Paillier-based	DGK-based	OT-based
Communication [Bytes]			
legacy	384	392	10 496
medium	768	776	29 184
long-term	1 152	1 160	49 920
Local Runtime [ms] (one-time key generation)			
legacy	34 (22)	42 (232)	12
medium	114 (192)	12 (10 868)	62
long-term	581 (788)	22 (104 432)	164
Cloud Runtime [ms] (one-time key generation)			
legacy	346 (32)	287 (284)	412
medium	357 (296)	217 (16 066)	657
long-term	754 (1 258)	288 (130 173)	989

3.3.4 Benchmarking of Primitive Operations

We benchmark the costs for evaluating 1 000 primitive operations of each sharing and all transformations in our framework by measuring the run-time in the local and cloud scenario and depict the asymptotic communication for $\ell = 32$-bit operands. Here we use long-term security parameters (cf. Sect. 2.1). For the online phase, we build two versions of the circuit. In the first version (Seq), we run the 1 000 operations sequentially to measure the latency of operations; in the second version (Par) we run 1 000 operations in parallel to measure the throughput of operations. The benchmark results are given in Fig. 3.2.

The first and most crucial observation we make from the results in the local setting is that the conversion costs between the sharings are so small that they even allow a full round of conversion for a single operation. For instance, for multiplication, where the best representation is Arithmetic sharing, converting from Yao shares to Arithmetic shares, multiplying, and converting back to Yao shares is more efficient than performing multiplication in Yao sharing (76 μs vs. 1 003 μs setup time, 183 μs vs. 970 μs sequential online time, and 259 μs vs. 1 973 μs total sequential run-time). The most prominent operations for which a conversion can pay off are multiplication (MUL), comparison (CMP), and multiplexer (MUX), for which we depict for each sharing the size (a measure for the number of crypto operations

needed in the setup phase and also for Yao in the online phase) and number of communication rounds in Tab. 3.3. The lowest size in Tab. 3.3 (marked in bold) matches with the lowest setup and parallel online time in the local setting. Comparison is best done in Yao sharing, because the Boolean sharing requires a logarithmic number of rounds. Multiplexer operations can be evaluated very efficiently with Boolean sharing, especially when multiple multiplexer operations are performed in parallel, since their size and number of rounds are constant. Note that the setup time for multiplication is higher compared to the evaluation of multiplication protocols in Sect. 3.3.3 since we amortize over less multiplication triples.



Arithmetic Sharing

Op	Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
ADD	1.4	0.6	0.3	0.1	0
MUL	155	19	239 123	1 920	1 156

B2A

Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
27.3	18.8	419 841.7	1 205.8	578

A2Y

Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
66.2	46.4	436 876.5	2 755.1	3 076

***Y2B* (Seq / Par)**

Local	Cloud	Comm
10.2 / 7.5	14.7 / 8.8	0

Boolean Sharing

Op	Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
ADD	1 997	124	2 471 105	4 235	7 540
MUL	10 325	637	6 745 539	11 784	65 520
XOR	3.4	0.9	4.7	1.3	0
AND	148	13	193 204	1 302	1 040
CMP	756	43	1 080 214	2 369	2 893
EQ	628	18	1 130 028	2 161	1 008
MUX	121	2	225 174	510	40

Yao Sharing

Op	Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
ADD	68	48	2 354	2 323	1 536
MUL	1 973	1 146	12 681	11 567	96 768
XOR	23	20	1 155	1 151	0
AND	49	43	2 325	2 316	1 536
CMP	65	46	2 327	2 305	1 536
EQ	57	45	2 333	2 312	1 488
MUX	59	46	2 318	2 300	1 536

B2Y

Local Seq	Local Par	Cloud Seq	Cloud Par	Comm
20.8	18.5	479 774.7	1 294.8	1 028

Figure 3.2: Total time (setup + online in μs) and communication (in Bytes) for one atomic operation on $\ell = 32$-bit values in a local and cloud scenario, averaged over 1 000 sequential / parallel operations using long-term security parameters.

Latency (Seq): The best performing sharing for sequential functionalities depends on the network latency. While in the local setting a conversion from Yao to Arithmetic sharing for performing multiplication is more efficient than computing the multiplication in Yao, multiplication using Yao's protocol becomes more efficient in the cloud setting. This is due to the impact of the high latency on the communication rounds, which have to be performed in Arithmetic and Boolean sharing. In contrast, Yao sharing has a constant number of interactions and is thus better suited for higher latency networks.

Throughput (Par): Instantiating operations in parallel greatly improves the online run-time in the Arithmetic and Boolean sharing, mainly because the number of rounds is the same as doing a single operation. While Yao's protocol also improves with parallel circuit instantiations, these benefits are smaller and mostly due to tto memory locality. Hence, Arithmetic and Boolean sharing benefit more more from parallel circuit evaluation, than garbled circuits.

Table 3.3: Asymptotic complexities of selected operations in each sharing on ℓ-bit values; smallest numbers in bold. Not implemented operations marked with — (cf. Sect. 3.3.1).

Sharing	MUL		CMP		MUX	
	size	rounds	size	rounds	size	rounds
Arithmetic	$\boldsymbol{\ell}$	1	—	—	—	—
Boolean	$2\ell^2$	ℓ	3ℓ	$\log_2 \ell$	**1**	1
Yao	$2\ell^2$	**0**	$\boldsymbol{\ell}$	**0**	ℓ	**0**

3.3.5 GMW vs. Yao

There are two different approaches for secure two-party computation on Boolean Circuits: the GMW protocol [GMW87] or Yao's garbled circuits [Yao86]. In this section we justify why GMW is beneficial for many implementations.

The properties of each protocol make it advantageous for use in different scenarios. A core difference appears when looking at network latency and the multiplicative depth of the evaluated circuit. GMW requires communication rounds that with the depth, while Yao's garbled circuits protocol requires a constant number of rounds. In some cases evaluation using GMW can still be beneficial, even if the evaluated circuits have a high depth. In an outsourcing scenario, this can be mitigated by using a low-latency network between the computational parties P_0 and P_1. Concretely, GMW has the following advantages over Yao's garbled circuits:

Precomputation GMW allows precomputation of all symmetric cryptographic operations and communication independently of the circuit and its inputs. Additionally, this setup phase can be parallelized and easily computed by multiple machines.

Multi-Party GMW allows for easy extension to multiple computing parties, which is good for settings where we might want to distribute trust to more parties.

Balanced Workload Unlike Yao's protocol, GMW balances the workload equally between all computational parties.

Memory Consumption The memory consumption for circuit evaluation is much lower for GMW, since GMW only needs to process single bits while Yao's garbled circuits needs to process symmetric keys of length $\kappa = 128$ bits.

SIMD Evaluation GMW allows more efficient parallel evaluation of gates by processing multiple bits per register, which is especially important for large circuits.

Vector ANDs GMW supports vector ANDs, that reduce the number of required OTs and allows the construction of efficient MUX gates.

4 Automated Synthesis of Optimized Circuits for MPC

Results published in:

[DDK+15] D. DEMMLER, G. DESSOUKY, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER, S. ZEITOUNI. **"Automated Synthesis of Optimized Circuits for Secure Computation"**. In: *22. ACM Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, pp. 1504–1517. CORE Rank A*.

4.1 Introduction

While designing efficient and correct MPC circuits for smaller building blocks and simple applications can be performed manually by experts, this task becomes highly complex and time consuming for large applications such as floating-point arithmetic and signal processing, and is thus error-prone. Faulty circuits could potentially break the security of the underlying applications, e.g., by leaking additional information about the parties' private inputs. Hence, an *automated* way of generating *correct* large-scale circuits, which can be used by regular developers is highly desirable.

A large number of compilers for secure computation such as [MNPS04; BNP08; HKS+10; HEKM11; Mal11; MLB12; KSS12; HFKV12; SZ13; KSMB13; ZSB13] implemented circuit building blocks manually. Although tested to some extent, showing the correctness of these compilers and their generated circuits is still an open problem.

Recently, TinyGarble [SHS+15] took a different approach by using established hardware logic synthesis tools and customizing them to be adapted to automatically generate Boolean circuits for the evaluation with Yao's garbled circuits protocol. The advantage of this approach lies in the fact that these tools are being used by industry for designing digital circuits, and hence are tested thoroughly, which is justified by the high production costs of Application Specific Integrated Circuits (ASICs). However, these tools are designed primarily to synthesize circuits on hardware target platforms such as ASICs, configurable platforms such as Field Programmable Gate Arrays (FPGAs) or Programmable Array Logics (PALs). Using hardware logic synthesis tools for special purposes such as generating circuits for secure computation, requires customizations and workarounds. Exploiting these tools promises accelerated and automated circuit generation, significant speedup, and ease in designing and generating

circuits for much more complicated functions, while also maintaining the size (and depth) efficiency of hand-optimized smaller circuit building blocks.

In this work we continue along the lines of using logic synthesis tools for secure computation and automatically synthesize an extensive set of basic and complex operations, including IEEE 754 compliant floating-point arithmetic. In contrast to TinyGarble, which generated only size-optimized circuits for Yao's garbled circuits protocol, we focus also on synthesizing depth-optimized circuits for the GMW protocol [GMW87]. Although the round complexity of the GMW protocol depends on the circuit depth, it has some advantages compared with Yao's constant-round protocol: 1) it allows to precompute *all* symmetric cryptographic operations in a setup phase and thus offers a very efficient online phase, 2) its setup phase is independent of the function being computed, 3) it balances the workload equally between all parties, 4) it allows for better parallel evaluation of the same circuit (SIMD operations) [SZ13; DSZ15] 5) it can be extended to multiple parties, and 6) the TinyOT protocol [NNOB12] which provides security against stronger active adversaries, has an online phase which is very similar to that of GMW, and its round complexity also depends on the circuit depth. A similar goal was also followed in ShallowCC [BHWK16], which appeared slightly later than our corresponding paper [DDK+15].

We combine industrial-grade logic synthesis tools with our ABY framework, presented in Chapt. 3, which implements state-of-the-art optimizations of the two-party protocols by GMW and Yao. On the one hand, our approach allows to use existing and tested libraries for complex functions such as IEEE 754 compliant floating-point operations that are already available in these tools without the need for manual re-implementation. On the other hand, this allows to use high-level input languages such as Verilog where we map high-level operations to our optimized implementations of basic functions.

4.1.1 Outline and Our Contributions

After summarizing preliminaries in Sect. 4.2, we present our following contributions:

Architecture and Logic Synthesis (Sect. 4.3) We provide a fully-automated end-to-end toolchain, that allows the developer to describe the function to be computed securely in a high-level Hardware Definition Language (HDL), such as Verilog, without the requirement to learn a new domain-specific language for MPC. Our work is the first to consider automated hardware synthesis of low-depth combinational circuits optimized for use in the GMW protocol [GMW87], as well as size-optimized circuits for Yao's protocol [Yao86]. For this, we manipulate and engineer state-of-the-art open-source and commercial hardware synthesis tools with synthesis constraints and customized libraries to generate circuits for either protocol according to the developer's choice.

Optimized Circuit Building Blocks (Sect. 4.4) We develop a library of depth- and size-optimized circuits, including arithmetic operations (e.g., addition, subtraction, multiplication, division), comparison, counter, and multiplexer, which can be used to construct more complex functionalities such as various distances, e.g., Manhattan, Euclidean, or Hamming distance. Some of the implemented building blocks show improvements in depth compared with hand-optimized circuits of [SZ13] by up to 14%, while others show at least equivalent results. Assembling sub-blocks from our customized library can be used to construct more complicated functionalities, which would otherwise be challenging to build and optimize by hand. We exploit the capabilities of our synthesis tools to bind high-level operators (e.g., the `'+'` operator) and functions to optimized circuits in our library to allow the developer to describe circuits in Verilog using high-level operators. We also utilize built-in Intellectual Property (IP) libraries in commercial hardware synthesis tools, which have been tested extensively to generate Boolean circuits for more complex functionalities such as floating-point arithmetic.

Benchmarks and Evaluation (Sect. 4.5) We use ABY to securely evaluate the Boolean circuits generated by our hardware synthesis toolchain. Moreover, we extend the list of available operations in ABY by multiple floating-point operations. In contrast to previous works that built dedicated and complex protocols for secure floating-point operations, we use highly tested industrial-grade floating point libraries. We compare the performance of our constructions with related work. For floating-point operations we achieve between 0.5 to 21.4 times faster runtime than [ABZS13] and 0.1 to 3 267 times faster runtime than [KW14]. We emphasize that we achieve these improvements even in a stronger setting, where all but one party can be corrupted and hence our protocols also work in a two-party setting, whereas the protocols of [ABZS13; KW14] require a majority of the participants to be honest and thus need $n \geq 3$ parties. We also present timings for integer division that outperform related work of [ABZS13] (3-party) by a factor of 0.6 to 3.7 and related work of [KSS13b] (2-party) by a factor of 32.4 to 274. Additionally, we present benchmarks for matrix multiplication, but these are less efficient than previous approaches [BNTW12; ZSB13; DSZ15].

Application: Private Proximity Testing (Sect. 4.6) A real-world application of floating-point calculations is privacy-preserving proximity testing on Earth [ŠG14]. We implement the formulas from [ŠG14] with our floating-point building blocks and achieve faster runtime, and higher precision compared to their protocols. This demonstrates that our automatically generated building blocks can outperform hand-built solutions.

4.2 Preliminaries

In this section we provide preliminaries and background related to hardware synthesis (Sect. 4.2.1), and the IEEE 754 floating-point standard (Sect. 4.2.2). Information about the underlying MPC protocols can be found in Sect. 2.4.

4.2.1 Hardware Synthesis

Hardware or logic synthesis is the process of translating an abstract form of circuit description into its functionally equivalent gate-level logic implementation using a suite of different optimizations and mapping algorithms that have been a theme of research over years. A logic synthesis tool is a software which takes as input a function description and transforms and maps this description into an output suitable for the target hardware platform and manufacturing technology.

Tools Common target hardware platforms for synthesized logic include FPGAs, PALs, and ASICs. ASIC synthesis tools, as opposed to FPGA synthesis tools, are used in this work due to the increased flexibility and available options, and because FPGA synthesis tools map circuits into Look-up Tables (LUTs) and flip-flop gates in accordance with FPGA architectures, but not Boolean gates. We used two main ASIC synthesis tools interchangeably: The Synopsys Design Compiler (DC) [Syn10], which is one of the most popular commercial logic synthesis tools, and the open-source academic Yosys-ABC toolchain [Wol; Ber]. In the following, we briefly describe the synthesis flow of Synopsys DC.

Synthesis Flow A HDL description of the desired circuit is provided to Synopsys DC. Operations in this description get mapped to the most appropriate circuit components selected by Synopsys DC from two types of libraries: the generic technology (GTECH) library of basic logic gates and flip-flops called cells, and *synthetic libraries* consisting of optimized circuit descriptions for more complex operations. Designware [Syn15] is a built-in *synthetic library* provided by Synopsys, consisting of tested IP constructions of standard and complex cells frequently used, such as arithmetic or signal processing operations. This first mapping step is independent of the actual circuit manufacturing technology and results in a generic structural representation of the circuit. This gets mapped next to low-level gates selected from a target *technology library* to obtain a technology-specific representation: a list of Boolean and technology-specific gates (e.g., multiplexers), called netlist.

Synopsys DC performs all of the above mapping and synthesis processes under synthesis and optimization constraints, which are directives provided by the developer to optimize the delay, area and other performance metrics of a synthesized circuit.

Input to these hardware synthesis tools can be a pure combinational circuit, which maps only to Boolean gates, or a sequential circuit that requires a clock signal and memory elements to store the current state. The output of a sequential circuit is a function of both the circuit inputs and the current state. In this work, we constrain circuit description to combinational circuits.

High-Level Synthesis Logic synthesis tools accept the input function description most commonly in a HDL format (Verilog or VHDL), whereas more recent logic synthesis tools additionally support high-level synthesis (HLS). This allows them to accept higher-level circuit descriptions in C/C++ or similar high-level programming alternatives. The HLS tools then transform the functional high-level input code into an equivalent hardware circuit description,

which in turn can be synthesized by classic logic synthesis. Although this higher abstraction is more developer-friendly and usable, performance of resulting circuits is often inferior to HDL descriptions, unless heavy design constraints are provided to guide the mapping and optimization process.

4.2.2 The IEEE 754 Floating-Point Standard

Floating-point (FP) numbers allow to represent approximations of real numbers with a trade-off between precision and range. The IEEE 754 floating-point standard [IEE08] defines arithmetic formats for finite numbers including signed zeros and subnormal numbers, infinities, and special "Not a Number" values (`NaN`) and rounding rules to be satisfied when rounding numbers during floating-point operations, e.g., rounding to nearest even. Additionally, the standard defines exception handling such as division by zero, overflow, underflow, infinity, invalid and inexact.

The IEEE 754 Standard 32-bit single precision floating-point format consists of 23 bits for significand, 1 bit for sign and 8 bits for exponent distributed from MSB to LSB as follows: sign `[31]`, exponent `[30:23]`, and significand `[22:0]`. The 64-bit double precision format consists of 52 bits for significand, one bit for sign, and 11 bits for exponent.

4.3 Our ToolChain

We describe our toolchain here by presenting our architecture followed by a detailed description of each component.

4.3.1 Architecture

An overview of our architecture is shown in Fig. 4.1. We provided the hardware synthesis tools with optimization and synthesis constraints along with a set of customized technology and synthesis libraries (cf. Sect. 4.3.2), to map the input circuit description in Verilog (or any other HDL) into a functionally-equivalent Boolean circuit netlist in Verilog. The output netlist, is constrained to consist of `AND`, `XOR`, `INV` and `MUX` gates.

The Verilog netlist is parsed and scheduled, and input into ABY (cf. Chapt. 3), which we extended to process this netlist and generate the Boolean circuit described in it.

In the following we describe in further detail the main components of our toolchain architecture: logic synthesis (Sect. 4.3.2), customizing synthesis (Sect. 4.3.3), function and operator mapping (Sect. 4.3.4), developer usage(Sect. 4.3.5), challenges of logic synthesis for MPC (Sect. 4.3.6), scheduling (Sect. 4.3.7), and extending the ABY framework (Sect. 4.3.8).

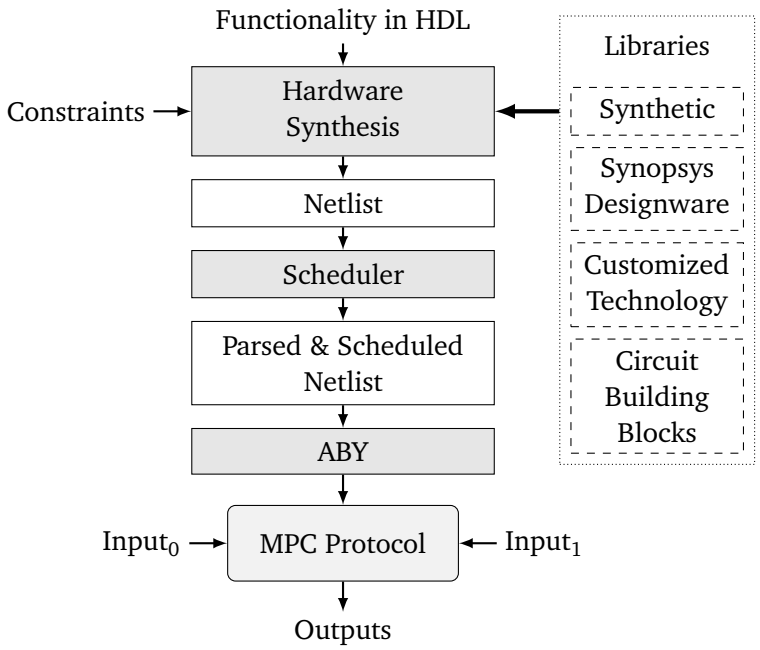


Figure 4.1: Architecture Overview

4.3.2 Hardware and Logic Synthesis

The GMW protocol and Yao's protocol require that the function to be computed is represented as a Boolean circuit. As described in detail in Sect. 2.4.7, previous work used different approaches like DSLs or custom compilation to create garbled circuits. This may be considered as "reinventing the wheel" since Boolean mapping and optimization is the core of hardware synthesis tools, and has been researched for long. It has been argued, however, that such "hardware compilers" target primarily hardware platforms and therefore involve technology constraints and metrics which are not directly related to the purpose of generating Boolean circuits for secure computation. Writing circuits in HDL, such as Verilog or VHDL, is not entirely high-level, and involves hardware description paradigms which may not be similar to high-level programming paradigms. Furthermore, they rely on the use of sequential logic rather than pure combinational logic.

Exploiting Logic Synthesis The TinyGarble framework [SHS+15] exploited these very same points, and employed hardware synthesis tools in order to generate compact sequential Boolean circuits for secure evaluation by Yao's garbled circuits protocol [Yao86]. This approach does usually not scale well for the GMW protocol, as sequential circuits have a large depth. The work in this chapter extends this approach further by using the hardware synthesis tools to generate combinational circuits of more complex functionalities for evaluation by both Yao's protocol and the GMW protocol [GMW87], while excluding all design and technology optimization metrics. The synthesis and generation of the Boolean netlist by

the synthesis tools (cf. Sect. 4.2.1) can be optimized according to the synthesis constraints and optimization options provided. Hardware synthesis tools conventionally target circuit synthesis on hardware platforms, but can be adapted and exploited for secure computation purposes to generate Boolean netlists which are AND-minimized (depth-optimized primarily for GMW or size-optimized for Yao's garbled circuits).

4.3.3 Customizing Synthesis

In the following, we focus on how we customized the synthesis flow of Synopsys DC to generate our Boolean netlists.

Synthesis Flow The synthesis and optimization constraints that can be provided to Synopsys DC allow us to manipulate it to serve our purposes in this work, and generate depth-optimized circuit netlists for evaluation with GMW. Moreover, we developed a synthetic library of optimized basic cells and depth/size-optimized circuit building blocks that can be assembled by developers to build more complex circuits, and a customized technology library to constrain circuit mapping to XOR and AND gates only. The different libraries and our engineered customizations to achieve this are described next.

Synthetic Libraries The first step of the synthesis flow is to convert arithmetic and conditional operations (`if-else`, `switch-case`) to their functionally-equivalent logical representations. By default, they are mapped to cells (either simple gates or more complex circuits such as adders and comparators) extracted from the GTECH library and the built-in Synopsys DC DesignWare library [Syn15] (cf. Sect. 4.2.1).

Our Optimized Circuit Building Blocks Library Besides the standard built-in libraries, we developed our own DesignWare circuits in a customized *synthetic library*. It consists of depth-optimized circuit descriptions (arithmetic, comparators, 2-to-1 multiplexer, etc.) customized for GMW, as well as size-optimized counterparts for Yao's garbled circuits. Synopsys DC can then be instructed to prefer automated mapping to our customized circuit descriptions (cf. Sect. 4.4) rather than built-in circuits (cf. Sect. 4.3.5 for developer usage).

Technology Library The intermediate representation of the circuit obtained in the step before is then mapped into low-level gates extracted from a technology library. A technology library is a library that specifies the gates and cells that can be manufactured by the semiconductor vendor onto the target platform. The library consists of the functional description (such as the Boolean function they represent) of each cell, as well as their performance and technology attributes, such as timing parameters (e.g., intrinsic rise and fall times, capacitance values) and area parameters.

Technology libraries targeting ASICs contain a range of cells ranging from simple 2-input gates to more complex gates such as multiplexers and flip-flops. A single cell can also have different implementations which have varying technology attributes. Ultimately, the goal of the synthesis tool is to map the generic circuit description into a generated netlist of cells

from this target technology such that user-provided constraints and optimization goals are satisfied.

Our Customized Technology Library In order to meet our requirements of the Boolean circuit netlists required in this work, we constrain Boolean mapping to non-free AND and free XOR gates. We developed a customized technology library which has no manufacturing or technology rules defined, similar to the approach in TinyGarble, and we manipulated the cost functions of the gates by setting the area and delay parameters of XOR gates to 0, and setting very high non-zero values for OR gates to ensure their exclusion in mapping. Their very high area and delay costs force Synopsys DC to re-map all instances of OR gates to AND and INV gates according to their equivalent Boolean representation ($A \vee B = \neg(\neg A \wedge \neg B)$), and to optimize the Boolean mapping in order to meet the specified area/delay constraints. We set the area and delay costs of an inverter (INV) gate to zero, as they can be replaced with XOR gates with one input fixed to constant one. For AND gates, the area and delay costs are set to reasonably high values, but not too high so that they are not excluded from synthesis. We set MUX gates to area cost equivalent to that of a single AND gate (since the 2-to-1 multiplexer construction in [KS08b] is composed of a single AND gate and 2 XOR gates), and set its delay cost equivalent to 0.25 times more than that of an AND gate due to the extra XOR operations. We concluded that these settings give the most desirable mapping results after experimenting with Synopsys DC mapping behavior in different scenarios.

Synthesis Constraints We provide constraints that make delay optimization of the circuit a primary objective followed by area optimization as a secondary objective when generating depth-optimized circuits for GMW. We set the preference attribute to XOR gates, and disable circuit flattening to avoid remapping of XOR gates to other gates. Synthesis tools are not primarily designed to minimize Boolean logic by maximizing XOR gates and reducing the multiplicative complexity of circuits within multi-level logic minimization. This is because XOR gates are only considered as "free" gates in secure computation applications, whereas in the domain of traditional hardware CMOS design, NAND gates are the universal logic gates from which all other gates can be constructed. Hence, the tools need to be heavily manipulated to achieve our objectives.

Construction of More Complex Circuits The customized circuit descriptions we developed can be used to build higher-level and more complex applications. We assembled complex constructions such as Private Set Intersection (PSI) primitives (bitwise-AND, pairwise comparison, and Sort-Compare-Shuffle networks as described in [HEK12]) using our customized building blocks, and they have demonstrated equivalent AND gate count and depth as their hand-optimized counterparts in [HEK12]. In general, all sorts of more complex functionalities and primitives can be constructed by assembling these circuit building blocks along with built-in Designware IP implementations. Consequently, these more complex circuits can then be appended to our library to be re-used in building further more complex circuits, in a modular and hierarchical way.

HDLs also allow a developer to describe circuits recursively which can be synthesized, which is often the most efficient paradigm for describing depth-optimized circuit constructions such as the depth-optimized "greater than" operation [GSV07], the Waksman permutation network [Wak68], or the Boyar-Peralta counter [BP06] for computing the Hamming weight.

4.3.4 High-level Function and Operator Mapping

An alternative to describing the circuits for HLS in high-level C/C++ is to allow developers to input their circuit descriptions in high-level Verilog, by calling operators and functions, which we map to "instantiate" circuit modules such as depth-optimized adders or comparators from our customized *synthetic library*. This allows high-level circuit descriptions without incurring the drawbacks of using HLS tools, such as inferior hardware implementation (cf. Sect. 4.2.1).

Mapping operators We prepared a library description which links our customized circuits into the Synopsys DC. This provides a description of each circuit module, its different implementations, and the operator bound to each module. These operators can be newly created, or already built-in, such as ('+', '-', '$*$', etc.), but bound to our customized circuits. For instance, when synthesizing the statement `Z = X + Y`, Synopsys DC is automated to map the '+' to our customized Ladner-Fischer adder, rather than a built-in adder implementation.

Mapping Functions We mapped functions to instantiate circuit modules by creating a global Verilog package file which declares these functions and which circuit modules they instantiate when being called. This package file is then included in the high-level Verilog description code which calls on these functions.

Explicit Instantiation Other more complex circuits can only be explicitly called from our customized building blocks library, as well as from the Designware IP library which offers a wide range of IP implementations, all of which have verified and guaranteed correctness, such as the floating-point operations we present and benchmark in Sect. 4.5.3. A list of available Designware IP implementations can be found in [Syn15].

High-level Circuit Description Example In Fig. 4.2, we show how the depth-optimized constructions of the Manhattan, Euclidean and Hamming distances [SZ13] are described using high-level Verilog. The Manhattan distance between two points is the distance in a 2-dimensional space between these two points based only on horizontal and vertical paths. The Euclidean distance between two points computes the length of the line segment connecting them. The Hamming distance between two strings computes the number of positions at which the strings are different.

In the Euclidean distance description, in lines 19 and 20 the '-' operator is mapped automatically to our Ladner-Fischer subtractor. The function `sqr` called in lines 23 and 24, is automatically mapped to instantiate our Ladner-Fischer squarer. We declared and bound this function correctly in the package file `'func_global.v'` which is included in line 6. `case`

Figure 4.2: High-level description of the Hamming, Euclidean and Manhattan distances.

statements (as are `if...else` statements) in lines 26-34 are also mapped to our depth-optimized multiplexer. In line 38, a carry-save network is explicitly instantiated from our library described in Sect. 4.4.2, since some circuit blocks are not mapped to functions and operators and have to be explicitly instantiated due to their structure and design. In the Manhattan distance description, the absolute differences are computed by calling the `'abs_diff'` function in line 12 which is also mapped to instantiate the corresponding circuit. The same high-level abstraction can be seen in the Hamming distance description. Once these distance circuits are constructed, they can be appended to our blocks library to be easily re-used in more complex functionalities.

4.3.5 Developer Usage

By default, Synopsys DC selects the most appropriate circuit description which best satisfies the constraints provided by the developer. Alternatively, the developer can also explicitly select a specific circuit description to map an operation to.

In order for developers to use our synthetic libraries instead of Designware to map to our customized circuits, they have to decide for which metric to optimize: depth or size. Accordingly, developers add the libraries' paths and a single command in the synthesis script to direct Synopsys DC to optimize for either depth (for GMW) or size (for Yao), and to prefer mapping to which set of circuit descriptions.

Optimization constraints are generally specified by the developer once for the entire top-level circuit description in the synthesis script, while some sub-circuits require specific optimization constraints. We already specified the optimization constraints for our customized circuit building blocks.

4.3.6 Challenges of Logic Synthesis for MPC

Conventionally synthesis tools are best at synthesizing sequential hardware circuits with a clock input and flip-flops. This also means that the actual circuit netlists synthesized are much more compact than combinational Boolean circuits. However, for the purpose of this work, we require combinational netlist. This implies synthesis of circuits which can reach up to 100 million gates and beyond, which is time- and resource-consuming for hardware synthesis tools. This issue can be mitigated by generating sub-blocks in a hierarchical fashion, and appending them into one top-level circuit.

4.3.7 Scheduling

The output netlist generated from the hardware synthesis tools has to be parsed in an intermediate step before being provided to ABY. A parser and scheduler topologically sorts and schedules the netlist gates [KA99], since the Verilog netlist output from some synthesis tools is not topologically sorted, i.e., a wire can be listed as input to one gate before assigning output to it from another. The scheduler generates a Boolean netlist in a format which is similar to Fairplay's Secure Hardware Definition Language (SHDL) [MNPS04]. All gates and wires are renamed to integer wire IDs for easier processing by ABY, and complex statements are rewritten as one or several available gates. These steps ensure that the final netlist contains only AND, XOR, INV and MUX gates.

4.3.8 Extending ABY

The ABY framework, cf. Chapt. 3, is an extensive tool that enables a developer to manually implement secure two-party computation protocols by offering several low-level as well as intermediate circuit building blocks that can be freely combined. We extended the ABY framework with an interface where externally constructed blocks made of low-level gates can be input in a simple text format, similar to SHDL [MNPS04] and the circuit format from [ST], that we could parse as well, with some modifications.

This interface is used to input the parsed and scheduled netlists from our hardware synthesis. ABY creates a Boolean circuit with low depth from that input netlist, i.e., it schedules AND gates on the earliest possible layer and automatically processes all AND gates in one layer in parallel. A developer has two options: 1) our hardware synthesized netlist can be used as a full protocol instance from private inputs to output or 2) the netlist's functionality can be used as a building block and combined with other synthesized or hand-built sub-circuits within ABY in order to create the whole secure computation protocol. The output of ABY is a

fully functional secure computation protocol that is split into setup phase and online phase, that can be evaluated on two parties' private inputs.

4.4 Building Blocks Library

We implemented multible building blocks in Verilog as pure combinational circuits and synthesized their Boolean netlists using both Synopsys DC and Yosys-ABC, to show that the framework is independent of the used synthesis tool. All implemented circuits have configurable parameters such as the bit-width ℓ of the inputs and/or the number of inputs n. We summarize and compare our synthesis results with their hand-optimized counterparts in [HKS+10; HEK12; SZ13]. The two main comparison metrics are size **S** which is the circuit size in terms of non-free AND gates, and depth **D** which is the number of AND gates along the critical path of the circuit. XOR gates are considered to be free, as the GMW protocol and Yao's protocol with free XORs [KS08b] allow to securely evaluate XOR gates locally without any communication. Next we show the results for functionalities that have improved depth or size compared with their hand-optimized counterparts in Sect. 4.4.1, and then in Sect. 4.4.2 we describe further functionalities and blocks that we have implemented, which show equivalent results as their hand-optimized counterparts. Finally, in Sect. 4.4.3, we describe the floating-point operations and integer division that we benchmark in Sect. 4.5.

4.4.1 Improved Functionalities

In this section, we present the implemented functionalities that achieved better results in terms of size or depth compared with [HKS+10; SZ13]. Results are given in Tab. 4.1.

Ladner-Fischer LF Adder/Subtractor The LF adder/ subtractor has a logarithmic depth [LF80; SZ13]. Our results show improvement for both depth (up to 10%) and size (up to 14%) in the subtraction circuit, while maintaining the same size and depth for addition of power-of-two numbers. Both circuits can also handle numbers that are not powers-of-two and achieve better size (up to 20%) as the hardware synthesis tool automatically removes gates whose outputs are neither used later as inputs to other gates nor assigned directly to the output of the circuit.

Karatsuba Multiplier KMUL We implemented a recursive Karatsuba multiplier [KO62] using a ripple-carry multiplier for inputs with bit-width $\ell < 20$, while for $\ell \geq 20$ inputs are processed recursively. We compare our results with numbers given in [HKS+10], which generated size-optimized Boolean circuits for garbled circuits, but did not consider circuit depth. Here we achieve up to 3% improvement in size.

Manhattan Distance $\mathbf{DST_M}$ Manhattan distance is implemented as a depth-optimized circuit using Ladner-Fischer addition ADD_{LF} and subtraction SUB_{LF} or using ripple-carry addition ADD_{RC} and subtraction SUB_{RC} for a size-optimized circuit [CHK+12; SZ13]. Our results demonstrate improvements in terms of size (up to 16%) and depth (up to 13.6%).

4.4.2 Further Functionalities

We list further functionalities that we implemented next. Their circuit sizes and depths are equivalent to the hand-optimized circuits in [HEK12; SZ13]: ripple-carry adder and subtractor [BPP00; KSS09], $n \times \ell$-bit carry-save and ripple-carry network adders [Sav97; SZ13], multipliers and squarers [Sav97; KSS09; SZ13], depth-optimized multiplexer [KS08b], comparators (equal and greater than) [SZ13], full-adder [SZ13] and Boyar-Peralta counters [BP06; SZ13], and the Sort-Compare-Shuffle circuit for private set intersection (PSI) [HEK12] and its building blocks (bitonic sorter, duplicate-finding circuit, and Waksman permutation network [Wak68]).

Matrix Multiplication We implemented size and depth-optimized matrix multiplication circuits that compute one entry in the resulting matrix by computing dot products.

This circuit is evaluated such that it computes the entries of the resulting matrix in parallel. Thereby, we can exploit the capability of the ABY to evaluate circuits in parallel, which reduces the memory footprint of the implementation.

Size-optimized matrix multiplication uses the Karatsuba multiplier and a ripple-carry network adder, whereas depth-optimized matrix multiplication uses the carry-save network multiplier and a carry-save network adder. Both implementations are configurable, i.e., we can set the bit-width ℓ and number of elements per row or column n.

The depths and sizes of these circuits for matrix multiplication are given in Tab. 4.3 and their performance is evaluated in Sect. 4.5.2.

4.4.3 Floating-Point Operations and Integer Division

We generated floating-point operations using the DesignWare library [Syn15], which is a set of building block IPs used to implement, among other operations, floating-point computation circuits for high-end ASICs. The library offers a suite of arithmetic and trigonometric operations, format conversions (integer to floating-point and vice versa) and comparison functions. The provided functionalities are parametrized allowing the developer to select the precision based on either IEEE single or double precision or set a custom-precision format. We can also enable the `ieee_compliance` parameter when we need to guarantee IEEE compatible floating-point numbers ("Not a Number" `NaN` and denormalized numbers). Some functionalities provide an `arch` parameter which can be set for either depth-optimized or size-optimized circuits.

Some of the floating-point functions provide a 3-bit optional input `round`, to determine how the significand should be rounded, e.g., `000` rounds to the nearest even significand which is

the IEEE default. They also have an 8-bit optional output flag `status`, in which bits indicate different exceptions of the performed operation allowing error detection. We can choose to truncate or use these `status` bits as desired.

We generated circuits for floating-point addition, subtraction, squaring, multiplication, division, square root, sine, cosine, comparison, exponentiation to base e, exponentiation to base 2, natural logarithm (ln), and logarithm to base 2 for single precision, double precision and a custom 42-bit precision format for comparison with [ABZS13]. The 42-bit format consists of 32 bits for significand, one bit for sign and 9 bits for exponent distributed from MSB to LSB as follows: sign `[41]`, exponent `[40:32]` and significand `[31:0]`. We extended ABY with these floating-point operations and benchmarked them. We give runtimes, depths and sizes for various floating-point operations in Sect. 4.5.3.

We also generated circuits for integer division for different bit-widths $\ell \in \{8, 16, 32, 64\}$ using the built-in DesignWare library [Syn15]. Another possibility for generating division circuits is to use the division operator `'/'` which will be implicitly mapped to the built-in division module in that library. As we optimize for depth our circuits have size $\mathcal{O}(\ell^2 \log \ell) \approx 24\,576$ gates for $\ell = 64$ but low depth 512. In contrast, optimizing for size would yield better size $\mathcal{O}(\ell^2) \approx 3\ell^2 = 12\,288$ gates (for ADD/SUB, CMP, and MUX), but worse depth $\mathcal{O}(\ell^2) = 4\,096$. We give circuit sizes and depths for integer division in Tab. 4.2 and benchmarks in Sect. 4.5.1.

In addition to the previously listed functions, we also generated two fixed-point operations, sine and cosine for 16-bit or 32-bit inputs. Depth, size, and runtime are shown in Tab. 4.6.

4.5 Benchmarks and Evaluation

We extended ABY to process the parsed and scheduled netlist generated by our hardware synthesis tool and evaluate it with ABY's optimized implementations of the GMW protocol and Yao's garbled circuits (cf. Sect. 4.3.8). In contrast to TinyGarble [SHS+15], which mainly focused on a memory-efficient representation of the circuits and gave only a single example for the time to securely evaluate the circuit, we measure the total execution times for several operations and applications: integer division (Sect. 4.5.1), matrix multiplication (Sect. 4.5.2) and an extensive set of floating-point operations (Sect. 4.5.3). For Yao's protocol we use today's most efficient garbling schemes implemented in ABY: free XOR [KS08b], fixed-key AES garbling with the AES-NI instruction set [BHKR13] and half-gates [ZRE15]. For better comparability of the runtimes we use depth-optimized circuits for both, GMW and Yao.

Compilation and synthesis times even for our largest circuits ($\mathsf{FP}_{\mathsf{EXP2}}$, $\mathsf{FP}_{\mathsf{DIV}}$) using Synopsys DC are under 1 hour on a standard PC, but this is only a one-time expense, after which the generated netlist can be re-used without incurring compilation costs again.

We provide runtimes for the *setup phase*, which can be precomputed independently of the private inputs of the participants and the *online phase*, which takes place after the setup-phase

Table 4.1: Synthesis results of improved functionalities compared to hand-optimized circuits for inputs of bit-width ℓ: Ladner-Fischer $\mathbf{ADD}_{LF}/\mathbf{SUB}_{LF}$, Karatsuba multiplication **KMUL**, and Manhattan Distance $\mathbf{DST_M}$.

Circuit	Size S			Depth D		
	Hand-optimized	Ours	Improvement	Hand-optimized	Ours	Improvement
	Depth-Optimized					
ADD_{LF} ($\ell = 20$)	151	121	**20%**	11	11	0%
ADD_{LF} ($\ell = 30$)	226	214	5%	11	11	0%
ADD_{LF} ($\ell = 40$)	361	301	16.6%	13	13	0%
SUB_{LF} ($\ell = 16$)	113	97	**14%**	10	9	10%
SUB_{LF} ($\ell = 32$)	273	241	11%	12	11	8%
SUB_{LF} ($\ell = 64$)	641	577	10%	14	13	7%
DST_{M} ($\ell = 16$)	353	296	**16%**	22	19	**13.6%**
DST_{M} ($\ell = 32$)	825	741	10%	26	23	11.5%
DST_{M} ($\ell = 64$)	1 889	1 778	5.8%	30	27	10%
	Size-Optimized					
KMUL ($\ell = 32$)	1 729	1 697	1.8%	–	63	–
KMUL ($\ell = 64$)	5 683	5 520	2.9%	–	127	–
KMUL ($\ell = 128$)	17 972	17 430	**3%**	–	255	–
DST_{M} ($\ell = 16$)	65	65	0%	34	32	**5.8%**
DST_{M} ($\ell = 32$)	129	129	0%	66	64	3%
DST_{M} ($\ell = 64$)	257	257	0%	130	128	1.5%

is done and the inputs to the circuit are supplied by both parties. All runtimes are median values of 10 protocol runs. We measured runtimes on two desktop computers with an Intel Core i7 CPU (3.5 GHz) and 16 GB RAM connected via Gigabit-LAN. In all our experiments we set the symmetric security parameter to $\kappa = 128$ bits.

4.5.1 Benchmarks for Integer Division

A complex operation that is not trivially implementable by hand is integer division, as described in Sect. 4.4.3. In Tab. 4.2 we list the runtime, split in setup phase and online phase and list the circuit parameters for multiple input sizes. We compare our runtime with the runtime prediction of 32-bit integer long division of [KSS13b] which we speed up by a factor of 32, and even more for Single Instruction Multiple Data (SIMD) evaluation. We also compare with the runtime of 3-party 64-bit integer division of [ABZS13], which outperforms our single evaluation with GMW by a factor of 1.8. However, for parallel SIMD evaluation we improve upon their runtime by up to factor 3.7. When comparing to the 3-party 32-bit integer division of [BNTW12], we achieve a speedup of factor 6.5 for single execution with GMW, while our GMW evaluation requires more than 5 times the total runtime of [BNTW12] for 10 000 parallel executions.

Table 4.2: Runtimes (setup + online phase) in ms per single integer division. '–' indicates that no numbers were given. Protocols marked with * are in the 3-party setting; all other protocols are in the 2-party setting. Entries marked with × could not be run on our machines.

Integer Division	Parallel Batch Size			AND Gates	
	1	100	10 000	Size	Depth
8-bit GMW	0.3 + 42.4	0.2 + 0.52	0.2 + 0.004	367	32
8-bit Yao	1.1 + 0.7	0.2 + 0.04	0.2 + 0.035	367	32
16-bit GMW	7.8 + 47.7	0.8 + 0.79	0.6 + 0.01	1 542	93
16-bit Yao	2.0 + 1.1	0.7 + 0.14	0.7 + 0.14	1 542	93
32-bit [KSS13b]	2 000	–	–	–	–
32-bit [BNTW12]*	400	4	0.5	–	–
32-bit GMW	3.5 + 58.2	3.5 + 3.66	2.7 + 0.04	7 079	207
32-bit Yao	5.2 + 2.1	3.3 + 0.63	×	7 079	207
64-bit [ABZS13]*	60	41	40	–	–
64-bit GMW	16.9 + 90.3	12.0 + 7.50	10.8 + 0.15	28 364	512
64-bit Yao	27.5 + 5.6	13.1 + 2.49	×	28 364	512

4.5.2 Benchmarks for Matrix Multiplication

Matrix multiplication of integer values is an important use case in many applications. Here we exploit ABY's ability to evaluate circuits in parallel in a SIMD fashion and instantiate dot product computation blocks, each of which calculates a single entry in the result matrix. In Tab. 4.3 we give the runtimes for dot product computations of 16 values of 16 bit each or 32 values of 32 bit each with two different multiplication circuits as described in Sect. 4.4.2. We compare with the 3-party secret-sharing-based implementations of [BNTW12; ZSB13] as well as the 2-party arithmetic sharing implementation of ABY. For this comparison we use the values reported in the respective papers and interpolate them to our parameters.

Solutions based on secret sharing or artihmetic sharing outperform our Boolean Circuits significantly, due to their much faster methods for multiplication.

4.5.3 Benchmarks for Floating-Point and Fixed-Point Operations

There is a multitude of use cases for floating-point and fixed-point operations in academia and industry, ranging from signal processing to data mining, but due to the complexity of the format it has only recently been considered as application for secure computation [FK11]. Until today there are only few actual implementations of floating-point arithmetic in secure computation, all of which use custom-built protocols [ABZS13; KW14]. Instead, we use multiple standard floating-point building blocks offered by Synopsys DC and synthesize them automatically (cf. Sect. 4.4.3). Tab. 4.4 and Tab. 4.5 depicts the runtime in ms per single floating-point operation, when run once or multiple times in parallel using a SIMD approach.

Table 4.3: Runtimes (setup + online phase) in ms per single dot product computation, as described in Sect. 4.4.2. Protocols marked with * are in the 3-party setting; all other protocols are in the 2-party setting. Entries marked with × could not be run on our machines. Data from referenced works are interpolated from values given in the respective paper.

Dot Product	Parallel Batch Size			AND Gates	
	1	100	10 000	Size	Depth
size-optimized RC 16×16-bit GMW	3.1 + 45.9	3.9 + 0.62	3.2 + 0.04	8 427	36
size-optimized RC 16×16-bit Yao	7.4 + 3.0	4.3 + 1.01	×	8 427	36
32×32-bit Multiplication [BNTW12]*	25.9	0.261	0.058	–	–
32×32-bit Multiplication [ZSB13]*	0.289	0.185	0.184	–	–
32×32-bit Arithmetic Multiplication [DSZ15]	5.44 + 0.196	5.44 + 0.061	5.44 + 0.060	–	–
size-optimized RC 32×32-bit GMW	55.7 + 68.6	21.0 + 1.12	21.5 + 0.30	56 314	69
size-optimized RC 32×32-bit Yao	76.7 + 18.5	28.5 + 6.74	×	56 314	69
depth-optimized CSN 16×16-bit GMW	4.3 + 69.7	5.5 + 0.62	4.7 + 0.06	12 317	30
depth-optimized CSN 16×16-bit Yao	10.1 + 3.6	6.1 + 1.37	×	12 317	30
depth-optimized CSN 32×32-bit GMW	42.1 + 87.7	33.1 + 1.61	33.6 + 0.47	89 112	38
depth-optimized CSN 32×32-bit Yao	77.2 + 30.4	44.1 + 10.1	×	89 112	30

Results for fixed-point operations benchmarks are depicted in Tab. 4.6. We compare our results for Yao and GMW with hand-optimized floating-point protocols of [ABZS13], who used a 3-party secret sharing approach with security against semi-honest adversaries and desktop computers connected on a Gigabit-LAN for their measurements. The largest runtime improvements can be achieved when evaluating our generated circuits in parallel. We improve the runtime by up to a factor of 21 for parallel evaluation and show similar or somewhat improved runtimes for the lower parallelism levels reported. We can improve upon many results of [KW14] which is in the 3-party setting, except for highly parallel multiplication, even though the 3-party setting is typically much more efficient than the 2-party setting. We show that our automatically generated circuits are able to outperform hand-crafted circuits in many cases, especially for high degrees of parallelism. We give an application for floating-point arithmetic in Sect. 4.6.

4.5.4 Benchmark Evaluation

In general, when comparing the implementations of Yao and GMW in ABY, we show that Yao outperforms GMW in most cases but scales much worse, up to a point where the largest circuits cannot be evaluated in parallel, due to the high memory consumption of Yao's protocol. GMW remains beneficial for highly parallel protocol evaluation, as the more critical online time scales almost linearly with the level of parallelism. The setup times of Yao and GMW are similar for all parameters.

Our improved performance stems from both, the optimized circuits generated by the state-of-the-art hardware synthesis tools which we manipulate to optimize the circuits for either

Table 4.4: Runtimes (setup + online phase) in ms per single floating-point operation for multiple precisions. '–' indicates that no numbers were given. Protocols marked with * are in the 3-party setting; ours are in the 2-party setting. Entries marked with × could not be run on our machines. (Continued in Tab. 4.5)

FP Operation		Parallel Batch Size					AND Gates	
		1	10	100	1 000	10 000	Size	Depth
FP_{CMP}	32-bit GMW	0.4 + 39.6	0.1 + 4.1	0.1 + 0.45	0.1 + 0.06	0.1 + 0.003	218	12
	32-bit Yao	1.1 + 0.7	0.3 + 0.1	0.5 + 0.03	0.1 + 0.03	0.1 + 0.033	218	12
	42-bit [ABZS13]*	–	5.4	3.2	2.3	2.2	–	–
	42-bit GMW	0.4 + 39.6	0.2 + 4.3	0.2 + 0.44	0.2 + 0.05	0.1 + 0.003	290	13
	42-bit Yao	1.0 + 0.7	0.3 + 0.1	0.2 + 0.04	0.2 + 0.04	0.2 + 0.043	290	13
	64-bit GMW	0.4 + 40.6	0.3 + 4.3	0.2 + 0.49	0.2 + 0.05	0.2 + 0.004	427	15
	64-bit Yao	1.1 + 0.7	0.3 + 0.1	0.2 + 0.06	0.2 + 0.06	0.2 + 0.065	427	15
FP_{SQR}	32-bit GMW	0.9 + 42.8	0.5 + 4.7	0.6 + 0.67	0.6 + 0.03	0.6 + 0.01	1 550	28
	42-bit GMW	1.2 + 245.7	3.4 + 4.9	1.5 + 0.53	1.0 + 0.02	1.0 + 0.02	2 681	30
	64-bit GMW	2.8 + 246.5	3.6 + 5.2	3.2 + 0.72	2.5 + 0.07	2.5 + 0.04	6 596	36
FP_{ADD}	32-bit [KW14]*	1 370	137.0	14.5	1.9	1.6	–	–
	32-bit GMW	3.0 + 46.1	1.1 + 5.3	1.0 + 0.66	0.7 + 0.06	0.7 + 0.01	1 820	59
	32-bit Yao	2.0 + 1.1	1.0 + 0.2	0.9 + 0.17	0.9 + 0.17	0.9 + 0.18	1 820	59
	42-bit [ABZS13]*	–	19.0	11.0	9.3	9.1	–	–
	42-bit GMW	5.3 + 46.3	1.5 + 5.8	1.3 + 1.07	1.0 + 0.07	0.9 + 0.02	2 490	69
	42-bit Yao	2.6 + 1.3	1.3 + 0.3	1.2 + 0.24	1.2 + 0.23	1.2 + 0.24	2 490	69
	64-bit [KW14]*	1 471	147.1	16.7	4.8	4.1	–	–
	64-bit GMW	2.1 + 46.9	2.2 + 6.3	2.3 + 0.73	1.6 + 0.03	1.6 + 0.03	4 303	72
	64-bit Yao	3.6 + 1.6	2.2 + 0.5	2.0 + 0.40	2.0 + 0.40	2.0 + 0.40	4 303	72
FP_{MULT}	32-bit [KW14]*	434.8	43.5	4.4	0.6	0.2	–	–
	32-bit GMW	1.8 + 42.9	1.6 + 5.6	1.4 + 0.67	1.1 + 0.05	1.1 + 0.02	3 016	47
	32-bit Yao	8.1 + 1.1	1.6 + 0.3	1.4 + 0.27	1.4 + 0.27	1.4 + 0.29	3 016	47
	42-bit [ABZS13]*	–	4.2	3.4	3.2	3.1	–	–
	42-bit GMW	2.0 + 47.3	2.4 + 6.3	2.6 + 0.82	1.9 + 0.08	1.8 + 0.03	4 757	72
	42-bit Yao	4.1 + 1.7	2.5 + 0.5	2.2 + 0.43	2.2 + 0.43	2.2 + 0.43	4 757	72
	64-bit [KW14]*	476.2	47.6	5.1	0.9	0.3	–	–
	64-bit GMW	15.5 + 170.1	5.6 + 8.7	5.0 + 0.95	4.1 + 0.08	4.2 + 0.05	11 068	111
	64-bit Yao	13.3 + 2.7	5.4 + 1.1	5.2 + 1.00	5.1 + 0.99	×	11 068	111
FP_{SQRT}	32-bit [KW14]*	11 111	1 177	142.9	41.7	31.3	–	–
	32-bit GMW	1.3 + 57.7	1.2 + 6.6	1.2 + 1.22	0.9 + 0.12	0.8 + 0.01	2 455	197
	32-bit Yao	2.6 + 0.8	1.5 + 0.3	1.2 + 0.23	1.2 + 0.22	1.2 + 0.23	2 455	197
	42-bit GMW	2.6 + 66.4	2.2 + 8.8	2.4 + 1.69	1.6 + 0.15	1.6 + 0.03	4 810	300
	42-bit Yao	3.9 + 1.2	2.4 + 0.5	2.3 + 0.43	2.2 + 0.42	2.2 + 0.44	4 810	300
	64-bit [KW14]*	12 500	1 316	217.4	103.1	96.2	–	–
	64-bit GMW	10.5 + 87.4	6.4 + 14.9	5.1 + 6.23	4.3 + 0.23	4.3 + 0.06	12 706	557
	64-bit Yao	9.4 + 2.6	6.2 + 1.3	6.3 + 1.14	5.9 + 1.12	×	12 706	557
FP_{DIV}	32-bit [KW14]*	6 250	625.0	71.4	16.9	12.7	–	–
	32-bit GMW	2.3 + 64.3	3.1 + 9.3	2.6 + 1.78	2.0 + 0.16	2.0 + 0.03	5 395	296
	32-bit Yao	4.2 + 1.9	2.7 + 0.6	2.5 + 0.49	2.5 + 0.49	2.5 + 0.49	5 395	296
	42-bit [ABZS13]*	–	15.0	12.0	12.0	12.0	–	–
	42-bit GMW	9.9 + 79.8	5.4 + 13.0	4.6 + 2.48	3.7 + 0.23	3.7 + 0.05	9 937	462
	42-bit Yao	7.0 + 2.7	4.9 + 1.0	4.7 + 0.90	4.6 + 0.89	×	9 937	462
	64-bit [KW14]*	6 667	666.7	83.3	43.5	19.2	–	–
	64-bit GMW	16.6 + 123.4	12.5 + 25.4	8.4 + 4.92	8.6 + 0.38	8.7 + 0.12	22 741	994
	64-bit Yao	15.2 + 5.0	11.1 + 2.4	10.6 + 2.06	10.6 + 2.09	×	22 741	994

Table 4.5: Runtimes (setup + online phase) in ms per single floating-point operation for multiple precisions. '–' indicates that no numbers were given. Protocols marked with * are in the 3-party setting; ours are in the 2-party setting. Entries marked with × could not be run on our machines. (Continuation of Tab. 4.4)

FP Operation		Parallel Batch Size					AND Gates	
		1	10	100	1 000	10 000	Size	Depth
FP_{EXP2}	32-bit GMW	5.5 + 144.2	5.2 + 14.7	4.7 + 0.85	3.7 + 0.09	3.8 + 0.05	9 740	100
	32-bit Yao	6.5 + 1.8	4.7 + 0.9	4.5 + 0.84	4.5 + 0.83	×	9 740	100
	42-bit [ABZS13]*	–	88.0	80.0	75.0	75.0	–	–
	42-bit GMW	14.5 + 179.1	12.6 + 23.7	10.2 + 1.14	9.4 + 0.17	9.3 + 0.12	24 357	156
	42-bit Yao	15.8 + 4.4	11.9 + 2.4	11.3 + 2.13	11.2 + 2.14	×	24 357	156
	64-bit GMW	16.7 + 455.1	12.2 + 88.9	9.2 + 17.33	8.1 + 0.51	8.2 + 0.12	21 431	1214
	64-bit Yao	14.3 + 4.2	10.6 + 2.2	10.0 + 1.91	9.9 + 1.89	×	21 431	1214
FP_{LOG2}	32-bit GMW	4.1 + 67.0	5.7 + 8.0	5.0 + 1.48	4.1 + 0.10	4.0 + 0.05	10 568	157
	32-bit Yao	7.0 + 2.1	5.1 + 1.0	4.9 + 0.91	4.9 + 0.90	×	10 568	157
	42-bit [ABZS13]*	–	159.0	103.0	97.0	96.0	–	–
	42-bit GMW	16.0 + 67.4	12.5 + 20.5	9.8 + 2.80	8.5 + 0.19	8.9 + 0.11	23 041	266
	42-bit Yao	15.9 + 4.1	11.1 + 2.3	10.7 + 2.01	10.6 + 1.99	×	23 041	266
	64-bit GMW	19.7 + 95.8	11.0 + 32.1	8.5 + 6.34	7.6 + 0.45	7.6 + 0.10	19 789	649
	64-bit Yao	13.3 + 3.9	9.7 + 2.0	9.2 + 1.76	9.2 + 1.75	×	19 789	649
FP_{SIN}	32-bit GMW	7.3 + 178.9	2.5 + 6.6	2.9 + 1.00	2.1 + 0.05	2.0 + 0.03	5 215	93
	42-bit GMW	17.0 + 57.0	7.3 + 17.2	6.0 + 1.27	5.2 + 0.12	5.1 + 0.07	13 378	172
FP_{COS}	32-bit GMW	4.3 + 50.7	2.4 + 6.6	2.9 + 0.80	2.0 + 0.06	2.0 + 0.03	5 227	96
	42-bit GMW	22.6 + 58.6	7.3 + 28.6	5.5 + 1.93	5.0 + 0.12	5.1 + 0.07	13 343	173

Table 4.6: Runtimes (setup + online phase) in ms per single fixed-point operation.

Fixpoint Operation		Parallel Batch Size			AND Gates	
		1	100	10 000	Size	Depth
$FIXP_{SIN}$	16-bit	2.0 + 41.9	0.9 + 0.62	0.6 + 0.008	1 489	45
	32-bit	5.1 + 55.2	4.8 + 0.89	3.9 + 0.052	10 171	119
$FIXP_{COS}$	16-bit	2.0 + 122.6	1.0 + 0.58	0.6 + 0.009	1 487	45
	32-bit	5.0 + 153.4	4.9 + 0.88	3.9 + 0.056	10 172	119

depth or size, and from the efficient implementation of GMW and Yao's garbled circuits with most recent optimizations in ABY. Since both protocols are based on Boolean circuits, we improve the performance of operations that require many bit operations. Operations that involve many integer multiplications are better suited for solutions based on arithmetic or secret sharing.

4.6 Application: Privacy-Preserving Proximity Testing on Earth

As application for secure computation on floating-point operations, we consider privacy-preserving proximity testing on Earth [ŠG14]. Here, the goal is to compute if two coordinates C_0 and C_1 input by party P_0 and P_1 respectively are within a given distance ϵ: $D(C_0, C_1) < \epsilon$.

This is a useful but rather privacy-critical use case that has many applications, such as finding nearby friends, points of interest or targeted advertising, and is widely used with the recent spread of end-user GPS receivers and geolocation via IP addresses. The authors of [ŠG14] present and compare three different distance metrics: UTM, ECEF, and HS described below. In their paper, the authors design secure protocols based on additively HE or Yao's garbled circuits that require to quantize all values to integers, which means a loss of precision. Instead, our framework allows to compute the distance formulas directly on floating-point numbers with multiple precision options available and thus can offer a higher precision.

Universal Transverse Mercator (UTM) This distance metric maps Earth over a set of planes and provides accurate results if P_0 and P_1 are located relatively close to each other, within the same UTM zone.

In this metric coordinates are expressed as 2-dimensional points: $C_0 = (x_0, y_0)$ and $C_1 = (x_1, y_1)$.

$D_{\text{UTM}}(C_0, C_1) < \epsilon \Leftrightarrow (\underline{x_0} - x_1)^2 + (\underline{y_0} - y_1)^2 < \epsilon^2$, where underlined variables are inputs of party P_0 and the other terms are inputs of party P_1. For computing this formula we need 2 $\mathsf{FP}_{\mathsf{SQR}}$, 3 $\mathsf{FP}_{\mathsf{ADD}}$, and 1 $\mathsf{FP}_{\mathsf{CMP}}$ operations.

Earth-Centered, Earth-Fixed (ECEF) This distance metric uses the Earth-Centered, Earth-Fixed (ECEF, also known as Earth Centered Rotational, or ECR) coordinate system which provides very accurate results when the parties are far apart.

The coordinates are expressed as 3-dimensional points where $(0, 0, 0)$ is the center of the Earth: $C_0 = (x_0, y_0, z_0)$ and $C_1 = (x_1, y_1, z_1)$.

$D_{\text{ECEF}}(C_0, C_1) < \epsilon \Leftrightarrow$
$(\underline{x_0} - x_1)^2 + (\underline{y_0} - y_1)^2 + (\underline{z_0} - z_1)^2 < 4R^2 a_\epsilon$,
with $a_\epsilon = \dfrac{(\tan\frac{\epsilon}{2R})^2}{1 + (\tan\frac{\epsilon}{2R})^2}$. Underlined variables are inputs of party P_0 and the other terms are inputs of party P_1. Computing this formula takes 3 $\mathsf{FP}_{\mathsf{SQR}}$, 5 $\mathsf{FP}_{\mathsf{ADD}}$, and 1 $\mathsf{FP}_{\mathsf{CMP}}$ operations.

Haversine (HS) This distance metric is based on the haversine (HS) formula which is a trigonometric formula used to compute distances on a sphere and is very accurate regardless of the position of P_0 and P_1.

The coordinates are expressed as spherical coordinates with latitude (lat) and longitude (lon): $C_0 = (\text{lat}_0, \text{lon}_0)$ and $C_1 = (\text{lat}_1, \text{lon}_1)$.

$D_{\text{HS}}(C_0, C_1) < \epsilon \Leftrightarrow$
$\underline{\alpha^2} \cdot \beta^2 - \underline{2\alpha\gamma} \cdot \beta\delta + \underline{\gamma^2} \cdot \delta^2 + \underline{\zeta\theta^2} \cdot \eta\lambda^2 - \underline{2\zeta\theta\mu} \cdot \eta\lambda\nu + \underline{\zeta\mu^2} \cdot \eta\nu^2 < a_\epsilon$, with a_ϵ as defined above and

$$\alpha = \cos(\text{lat}_0/2) \quad \beta = \sin(\text{lat}_1/2)$$
$$\gamma = \sin(\text{lat}_0/2) \quad \delta = \cos(\text{lat}_1/2)$$
$$\zeta = \cos(\text{lat}_0) \quad \eta = \cos(\text{lat}_1)$$
$$\theta = \sin(\text{lon}_0/2) \quad \lambda = \cos(\text{lon}_1/2)$$
$$\mu = \cos(\text{lon}_0/2) \quad \nu = \sin(\text{lon}_1/2).$$

Underlined terms are inputs of party P_0 while all other terms are inputs of party P_1. Computing this formula requires 6 $\mathsf{FP}_{\mathsf{MULT}}$, 5 $\mathsf{FP}_{\mathsf{ADD}}$, and 1 $\mathsf{FP}_{\mathsf{CMP}}$ operations.

Table 4.7: Runtimes (setup + online phase) in ms and circuit complexity per single proximity test for multiple precisions. '–' indicates that no numbers were given. All protocols are in the 2-party setting. Entries marked with × could not be run on our machines.

Distance Metric		Parallel Batch Size			AND Gates	
		1	100	10 000	Size	Depth
UTM	HE [ŠG14]	700 … 1 100	–	–	–	–
	GC [ŠG14]	401.0 + 102.0	–	–	–	–
	32-bit GMW	4.4 + 59.8	4.0 + 1.49	3.3 + 0.05	8 815	146
	32-bit Yao	18.0 + 2.4	4.2 + 0.87	×	8 815	146
	64-bit GMW	19.9 + 67.2	10.6 + 2.65	10.2 + 0.14	26 588	195
	64-bit Yao	18.1 + 5.7	12.5 + 2.54	×	26 588	195
ECEF	HE [ŠG14]	1 000 … 1 300	–	–	–	–
	GC [ŠG14]	404.0 + 105.0	–	–	–	–
	32-bit GMW	5.7 + 60.1	5.8 + 1.56	5.3 + 0.07	14 042	205
	32-bit Yao	12.8 + 3.3	6.6 + 1.32	×	14 042	205
	64-bit GMW	13.9 + 78.1	15.8 + 2.91	16.0 + 0.20	41 850	267
	64-bit Yao	27.4 + 8.8	19.9 + 3.88	×	41 850	267
HS	HE [ŠG14]	1 700	–	–	–	–
	GC [ŠG14]	409.0 + 124.0	–	–	–	–
	32-bit GMW	13.6 + 67.5	11.6 + 2.11	10.5 + 0.14	27 525	224
	32-bit Yao	17.9 + 5.6	12.8 + 2.48	×	27 525	224
	64-bit GMW	49.5 + 283.6	33.3 + 3.40	33.4 + 0.41	88 530	342
	64-bit Yao	67.8 + 18.0	41.4 + 8.03	×	88 530	342

Comparison We implemented the three proximity testing algorithms from [ŠG14] using our floating-point building blocks. In Tab. 4.7 we compare the runtime of the original implementation of [ŠG14] that uses HE and Yao's garbled circuits with our implementation based on GMW and Yao for single and parallel evaluation. We are able to achieve better runtimes for single executions of the protocol (by factor 6.2 for HS and more than factor 14 for UTM and ECEF), and more than two orders of magnitude speedup for highly parallel execution. Thereby, we show that our approach allows to substantially improve upon the runtime of hand-crafted protocols while at the same time it benefits from the heavily tested and verified circuit building blocks from industry-grade hardware synthesis libraries.

5 Automated Compilation of Hybrid Protocols for Practical Secure Computation

Results published in:

[BDK+18] N. BÜSCHER, D. DEMMLER, S. KATZENBEISSER, D. KRETZMER, T. SCHNEIDER. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *25. ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018, pp. 847–861. CORE Rank A*.

5.1 Introduction

MPC is an active field of research and the large number of protocols and optimizations has led to a significant performance improvement of MPC, yet also has the drawback that MPC becomes harder and more complex to access for people outside the field. Identifying a (near) optimal choice of MPC protocols for a given application requires experience with many MPC protocols, their optimizations, programming models, and the conversions to securely switch between protocols in hybrid computations. Furthermore, for realizing an actual application not only expert knowledge in MPC, but also a background in hardware design is needed to implement the application in an efficient Boolean and/or arithmetic circuit representation, which are the most common function representations in MPC. Consequently, creating efficient applications by hand is a tedious and error-prone task and therefore multiple compilers have been proposed, which share similarities with high-level synthesis from the area of hardware design, e.g., as shown in Chapt. 4.

Previous MPC compilers often only targeted a single protocol, e.g., Yao's garbled circuits [HFKV12; MNPS04; SHS+15], the GMW protocol [BHWK16], or linear-secret-sharing-based MPC [BLW08], or the compilers required the developer to use specific annotations to mark which protocol is used for each statement, e.g., [HKS+10; DSZ15]. While our work in Chapt. 4 can generate Boolean circuits for either Yao's garbled circuits or the GMW protocol, this only applies to the full circuit that is compiled and has to be specified by the developer manually. The only other compiler that addresses the compilation of a program using two MPC protocols (Yao's gabled circuits and arithmetic sharing) is *EzPC* [CGR+17]. However, EzPC only provides semi-automation for a domain specific language (DSL), as the input code must be manually decomposed, array accesses have to be manually resolved into multiplexer

structures, and the compiled circuits are left unoptimized. Moreover, EzPC is limited to two MPC protocols, which are selected statically and independently of the execution environment, by following a strict set of rules for each expression in the program. Similar HyCC, presented in this chapter, EzPC also uses our ABY framework from Chapt. 3 to evaluate generated circuits.

Compilation for Hybrid MPC In this chapter, we propose HyCC, a novel hybrid circuit compiler, capable of compiling and optimizing applications written in ANSI C into an efficient combination of MPC protocols. Other than previous work, we present a fully automated approach that decomposes the source code, translates the decomposed code into Boolean and arithmetic circuits, optimizes these circuits, and finally selects suitable MPC protocols for a given deployment scenario, optimizing for given criteria, such as latency (minimal total runtime), throughput (minimal per-operation runtime), or communication.

Fig. 5.1 illustrates the two main components of our approach. The first component is the (one-time) compilation of the application source code in ANSI C into a decomposed program description in the form of multiple circuits. We refer to the different parts of a decomposed program, i.e., the compact logical building blocks a larger application consists of, as *modules*. Each module is compiled into multiple circuit representations. HyCC compiles arithmetic circuits (A), depth-optimized circuits for GMW (B), and size-optimized circuits for Yao's garbled circuits (Y). The second component in HyCC is protocol selection, in which the most suitable combination of MPC protocols is selected for a decomposed program depending on the computational environment. This protocol selection can be included in an MPC framework and can also be done after compilation.

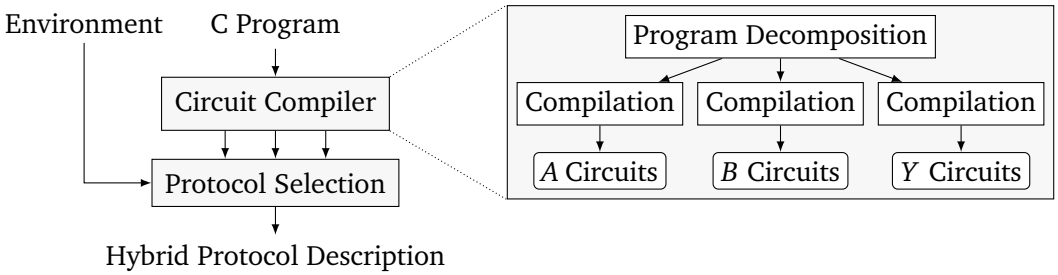


Figure 5.1: High-level overview of the HyCC compilation architecture. The circuit compiler decomposes an input program and compiles each part into multiple circuit representations. The protocol selection recombines the different parts.

Optimizing Circuit Compiler MPC is still significantly slower and more expensive than generic plaintext computation regarding both computation and communication. Thus, a tool-chain is required that optimizes the compilations of a program description into an *efficient* MPC protocol and its corresponding circuits. Even though the optimization of an input program has limits, i.e., an inefficient algorithm description cannot automatically be translated into a fast program, a programmer expects the compiler to not only correctly translate every

statement of a high-level description, but also to optimize the given representation, e.g., by removing unnecessary operations and using efficient instructions of a selected target architecture. This is of special interest for MPC compilers, where code optimizations that are too expensive to be applied by traditional compilers become affordable when considering the trade-off between compile time and evaluation costs of the program in MPC. For example, in Yao's protocol a 32×32 bit signed integer multiplication requires the evaluation of $\approx 1\,000$ non-linear Boolean gates (using the best known circuit), which results in $\approx 5\,000$ symmetric encryptions during the protocol execution. Consequently, the removal of any unnecessary operation in MPC has more impact than in traditional compilation, where only a single CPU cycle is lost during program execution. Furthermore, optimizations performed on the source code level, e.g., constant propagation, are computationally cheaper than minimization techniques applied on the gate level after the compilation to circuits.

These observations are incorporated in our compiler architecture: Before decomposing the input source code into different parts, a rigorous static analysis is performed to realize constant propagation, detect parallelism, and determine the decomposition granularity. Logic optimization techniques are then gradually applied on the circuit level. To achieve scalable optimizing compilation, we guide the logic optimization based on the results of static analysis of the source code. For example, loop bodies with a large number of iterations will be optimized with more effort than a piece of code that is used only once. Thus, in contrast to classic logic optimization or arithmetic expression rewriting, we rely on the structural information available in the high-level code.

Im summary, HyCC is capable of compiling optimized Boolean circuits and arithmetic circuits suiting the requirements of most constant- and multi-round MPC protocols. Our tool-chain is highly flexible and independent of the underlying MPC protocols, as only the respective cost models for primitive operations, e.g., addition or Boolean AND, have to be adapted to reflect future protocol developments in MPC.

Protocol Selection Protocol selection refers to mapping each part of a decomposed program to a specific MPC protocol. The circuits created by HyCC for each module and the mapping of modules into MPC protocols is sufficient to evaluate an application in a hybrid MPC framework. Overall, this is an optimization problem, where the best mapping is identified in regard to the cost model that considers the cost of evaluating each circuit in the respective MPC protocol as well as the conversion costs between them. The concept of protocol selection has previously been studied independently from compilation in [KSS14; PKUM16]. Kerschbaum et al. [KSS14] investigated protocol selection for a combination of Yao's garbled circuits and additively homomorphic encryption (HE). They conjectured that the optimization problem is NP-hard and proposed two heuristic approaches based on integer linear programming and a greedy optimization algorithm. Pattku et al. [PKUM16] used similar heuristics to optimize the protocol selection for minimal cloud computing costs.

We follow an approach that is different in several aspects. First, we show that the compilation of an efficient hybrid MPC protocol is not only a protocol selection problem, but also a scheduling problem. Second, in contrast to previous work, we make use of the structural

information in the source code before its translation into circuits. By grouping expressions that perform similar operations, e.g., loops, it becomes possible to perform an exhaustive search over the problem state for many practically relevant applications. If applications cannot be optimized to the full extent, we combine exhaustive search with heuristics. Finally, by separating compilation and protocol selection, an optimized selection can be chosen tailored to a specific deployment scenario. For this purpose, we implement a probing technique, which evaluates the computational power and network capabilities, for precise cost estimation during protocol selection.

5.1.1 Outline and our Contributions

We present the first complete tool-chain that automatically creates partitioned circuits and optimizes their selection for hybrid MPC protocols from standard ANSI C code, which makes hybrid MPC accessible to non-domain experts. We contribute techniques and heuristics for efficient decomposition of the code, scalable compilation, and protocol selection. We explain our architecture in Sect. 5.2.

We separate compilation from optimizing partitioning. Using a probing technique for MPC protocol implementations, we optimize the protocol selection at runtime for the actual deployment scenario. Protocol selection and partitioning is discussed in Sect. 5.3.

In Sect. 5.4 we provide an evaluation and comparison of HyCC with related work. We report speed-ups for our automatically compiled hybrid protocols of more than one order of magnitude over stand-alone protocol compilers, and factor three over previous handmade protocols for an exemplary machine learning application [LJLA17].

We conclude and provide an outlook into future work in Sect. 5.5.

5.2 The HyCC MPC Compiler

Here we describe our hybrid compiler HyCC[1]. After introducing the challenges, we provide details on all steps of the compilation chain.

5.2.1 Hybrid Compilation and its Challenges

In this section we describe our approach to efficient hybrid compilation.

In the first step, HyCC decomposes the input source code into distinct parts, which we call *modules*. The decomposition can happen on the source code level or on an intermediate representation of the code, like Single Static Assignment (SSA) form. Modules are the finest level of granularity used later in protocol selection and code within a module is evaluated with a single MPC protocol. In this work, we consider size-optimized Boolean circuits required for

[1] An open source implementation will be made available at https://gitlab.com/securityengineering/HyCC.

Yao's garbled circuits (Y), size- and depth-optimized Boolean circuits required for the GMW protocol (B), and arithmetic circuits (A), which we describe in more detail in Sect. 2.4.

Each module is compiled into all possible circuit representations for the different MPC protocols and then optimized. Finally, the hybrid protocol is put together during protocol selection and scheduling (cf. Sect. 5.3).

Multiple challenges (besides the complexity of compiling efficient Boolean or arithmetic circuits itself) arise when following this approach. All of them relate to a trade-off between compilation resources, i.e., time and storage, and the circuit properties (size and depth) that result from compilation. We describe identified challenges and propose solutions, which motivate our actual compilation architecture.

Determining the right size of code modules that are big enough to enable circuit level optimization and small enough to allow to benefit from protocol conversions is challenging. Furthermore determining the right scope of optimizations between local and global scale is nontrivial. The same issue occurs for loops, when trying to determine if and how far to unroll them. Even though compilation is a one-time task, its efficiency is relevant in practice and might be prohibitive for very large circuits.

We tackle these challenges and trade-offs with heuristics based on static analysis of the source code and use *Source-guided optimization* [BKJK16] to optimize circuits under configurable time constraints by distributing an optimization budget. Static source code analysis is sufficient, as MPC applications are evaluated without data-side channels, such that all possible program paths are taken during execution and thus can be studied at compile time. We provide more details on these challenge and our solutions in our paper [BDK+18].

5.2.2 Architecture

Our compilation architecture works in a resource-constrained environment and gets a source code, and compilation time limit as inputs. The compiler outputs a program description consisting of multiple modules, compiled to multiple circuit representations, and a direct acyclic dependency graph that describes the dependencies between the modules, which can be used to evaluate the program in a hybrid MPC framework.

HyCC's architecture consists of the following compilation *phases*, which themselves can consist of multiple compilation *passes*:

1. *Automated Parallelization:* Automated identification and annotation of code blocks, and in particular loops, that can be evaluated in parallel using external tools.
2. *Preprocessing, Lexing and Parsing:* Construction of an Abstract Syntax Tree (AST) from the automatically annotated input code using CBMC-GC [HFKV12].

3. *Source Code Optimization and Loop Unrolling:* The source code is partially evaluated using constant propagation, which requires a costly symbolic execution. This happens in multiple passes and runs until a configurable time limit is reached. Given enough time, this process converges to a fully optimized compilation result, while intermediate results do only optimize within functions or loops.
4. *Code Decomposition:* The input program is decomposed into multiple modules, that can be efficiently evaluated in a single protocol, and the dependencies between them are determined. Functions or procedures are naturally regarded as separate modules, and optionally split up further. The same happens for loops, where loops that are not unrolled require a pointer analysis and array accesses that depend on private variables are extracted and handled separately. Other candidates are groups of arithmetic operations or control flow and bit operations, which should end up in different modules.
5. *Circuit Compilation:* Each previously identified module is compiled into all possible circuit representations: a size-optimized Boolean circuit using CBMC-GC, a depth-optimized Boolean circuit with Shallow-CC [BHWK16], or an arithmetic circuit where each operation maps to an arithmetic gate in ABY.
6. *Inter-Procedural Circuit Optimization:* All Boolean and arithmetic circuits are optimized across multiple modules by removing unused gates and propagating constants between modules and across circuit types.
7. *Circuit Export:* The decomposed circuit with I/O interfaces is written to a file and can be used in the protocol selection.

Steps 2, 3, and 5 are also part of CBMC-GC's original tool-chain [HFKV12], while the others have been added for the compilation of hybrid protocols in HyCC. All steps are described in much more detail in our paper [BDK+18].

5.3 Protocol Selection and Scheduling

Determining an *efficient* combination of protocols (cf. Sect. 2.4) for given optimization goals is a challenging task. It depends on the use case and its complexity, the available hardware, and the network connection between the parties. In this section, we describe how to determine an optimized scheduling and mapping of the modules that were created in the compilation in an automated way.

5.3.1 Problem Definition

Our goal is to minimize the evaluation cost of a hybrid MPC application by determining an efficient protocol assignment and evaluation order of the compiled modules for a given program description and a user-specified cost model. The user can guide the optimization to optimize for the protocol's total, or online runtime, cloud computing costs as in [PKUM16],

amount of data transferred, or even energy consumption in the context of mobile devices, or a constrained combination thereof.

For determining an efficient combination of hybrid modules not only the cost of evaluating each module can be considered, but also the cost of the conversions if protocols change between modules.

Different from previous work [KSS14; PKUM16], we observe that it is crucial for the optimization to not only find an efficient module to protocol mapping but also a module scheduling, i.e., the evaluation order of parallel modules to save conversions and benefit from caching effects in large homogeneous computations.

We provide more details and an example for this, as well as a formalization of our approach in our paper [BDK+18].

5.3.2 Protocol Selection in HyCC

Scheduling and protocol selection are closely related problems, where the latter alone is conjectured to be NP-hard [KSS14; PKUM16]. Therefore, in HyCC we begin with determining an evaluation schedule from a heuristic for a given program decomposition. This schedule is then used as a starting point in a second step that optimizes the protocol selection. In HyCC we use the input source code and the automatically generated parallelism annotations to guide the protocol scheduling and with that explicitly schedule modules to be evaluated in parallel. This especially benefits secret-sharing-based protocols that require multiple rounds of interaction as well as circuit optimization that is more effective on modules that are grouped together.

Scheduling Apart from parallelization, modules are scheduled as soon as possible (ASAP). Combining both strategies in a single algorithm, parallel modules are temporarily merged in a single module which is then scheduled ASAP. Finally, the merged modules are restored and placed in the same instruction of the evaluation schedule.

Protocol Selection Protocol selection is assumed to be NP-hard in the general case [KSS14; PKUM16], however, given a relatively coarse-grained decomposition, as with HyCC, an optimal protocol selection can be computed with reasonable computational effort for many practical applications, as shown in Sect. 5.4.1. This is because the complexity of the protocol selection is mostly determined by the *width* of the program's Directed Acyclic Graph (DAG) rather than by its size.

To identify the optimal protocol selection for a given DAG of modules, we enumerate all possible protocol combinations using dynamic programming.

If the module DAG exceeds the computationally manageable width, the optimization algorithm will determine the optimal protocol selection for sub-graphs, with solvable width. For the

remaining sub-graphs, or a combination of multiple sub-graphs, heuristics, such as hill-climbing [KSS14] can be used to search for an optimized selection in the combination of different optimally solved sub-graphs.

Further details on this process and especially their algorithmic implementation as well as a formalization are described in our paper [BDK+18].

5.3.3 Cost Model and Probing

The most relevant cost factors of MPC are protocol runtime, bandwidth requirement and the number of communication rounds between the MPC parties. An accurate cost model is required for the optimizing protocol selection. The total communication complexity can precisely be determined by summing the communication costs of all individual building blocks of a protocol, whereas the runtime prediction is more complex. For large circuits with millions of non-linear gates a simple approach can give a rough estimate, where circuit depth d is multiplied by the communication latency T_{lat} and added to the number of non-linear gates G_{nl} divided by the maximum throughput of non-linear gates per second TP_{nl} to get a runtime estimate $T_{\text{estim}} = d \cdot T_{lat} + G_{nl}/TP_{nl}$. However, this is inaccurate for smaller circuits that do not fully saturate the network connection. In HyCC, we follow a more complex approach, where the input of the runtime prediction is the computation and communication costs of the individual protocol building blocks, i.e., input and output sharing, AND and XOR gates, arithmetic addition and multiplication gates, share conversions, as well as the available computation and communication resources. We automatically measure runtime, required communication and circuit depth of each individual building block for different input sizes and all available sharing types. We also evaluate them with different degrees of parallelism, to assess the efficiency gain of parallelization and also to determine the limits of the available resources. To optimize for the best possible performance, this probing happens on the systems where the final hybrid protocol will be deployed. By doing this, we can estimate the runtime and bandwidth requirement of compiled hybrid MPC protocols without actually running them by linear inter- and extrapolation of the previously measured smaller building blocks.

Fig. 5.2 shows a comparison of empirically measured runtimes (solid lines) and estimated runtimes (dashed lines) for three use cases: Minimum-Euclidean-Distance (described in Sect. 5.4.2) and AES evaluated with Yao's garbled circuits and the GMW protocol, respectively. We benchmark the building blocks for different number of inputs, evaluated in parallel. Extrapolating from runtime that was measured on small building blocks to a full-sized circuit and the influence of the network connection between the MPC parties leads to imprecision in the runtime prediction. In our measurements we found that the prediction was always within ±50% of the actual achieved runtime. For better runtime prediction, a larger number of measurements and more data points of the underlying building blocks are required to limit the influence of noise on a busy network or on shared hardware. As our results show, the runtime estimate that is interpolated from measuring the underlying building blocks captures the relative runtime between the protocols well and allows for identifying the most efficient sharing in the protocol selection step for a given deployment scenario.

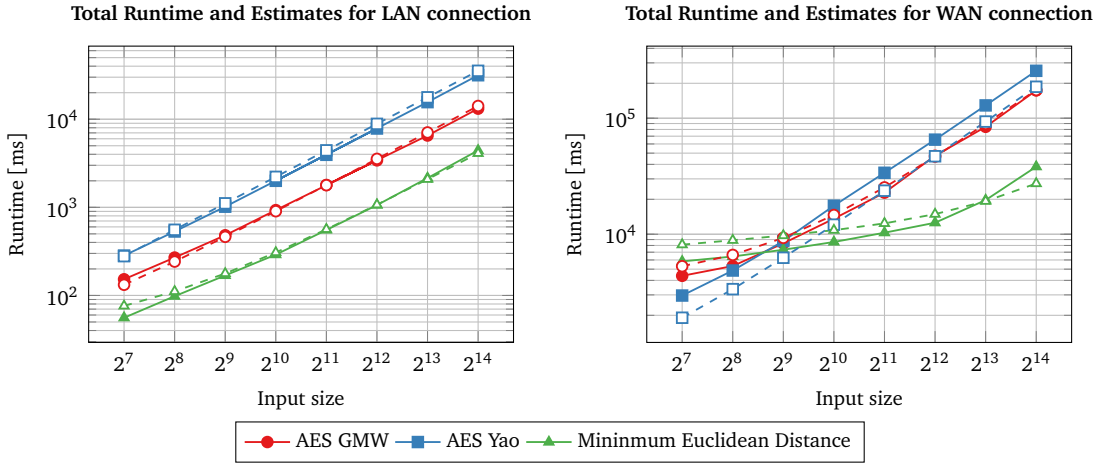


Figure 5.2: Comparison of measured runtimes (solid lines) and corresponding estimates (dashed lines) using a log-log plot.

A different approach was followed in [SK11], that performed a run-time forecast based on the number of cryptographic operations, the number of communication rounds, and the total communicated bits. We argue that our forecast mechanism is more accureate, since, especially symmetric cryptographic operations are becoming a less dominant factor, e.g., AES-NI enables a standard desktop PC to evaluate AES very efficiently.

5.4 Benchmarks

In this section, we present an experimental evaluation of HyCC. We study the efficiency of protocol selection, the circuits created by HyCC, and their performance in hybrid MPC protocols for various use cases in two different deployment scenarios. The goal of this evaluation is to illustrate that the circuits that were automatically created by HyCC from ANSI C code achieve comparable complexity as hand-crafted hybrid circuits and significantly more efficient than previous single-protocol compilers. As such, we are able to show that HyCC is simplifying the ease-of-use of hybrid MPC, and is thus a powerful tool to prototype a solution for a privacy problem, which allows to identify whether generic MPC protocols achieve sufficient efficiency or whether dedicated protocols need to be developed. We remark that the goal of this work is not to outperform dedicated secure computation protocols, which are optimized to achieve maximum efficiency for a specific use case. We start with an evaluation of the runtime of the protocol selection algorithm presented in Sect. 5.3.2.

5.4.1 Protocol Selection

To illustrate that exhaustive search is a sufficient solution for the protocol selection problem in most practical cases, we measure the runtime of the protocol selection algorithm in Fig. 5.3.

Shown are the runtimes averaged over $k = 10$ executions of a straight forward (unoptimized) implementation running on a commodity laptop for randomly generated graphs with $n = 2 \cdot w$ modules and increasing graph width w. We observe the expected exponential growth in runtime when increasing w. Albeit being a limiting factor of our approach, to the best of our knowledge all applications in privacy research studied so far have a very small branching factor in their functionalities, which leads to very small width w. For example, all use cases in this work have a width of at most $w = 3$, which is solved in less than 0.01 seconds and we remark that even larger graphs with a width of $w = 10$ are solved in seconds.

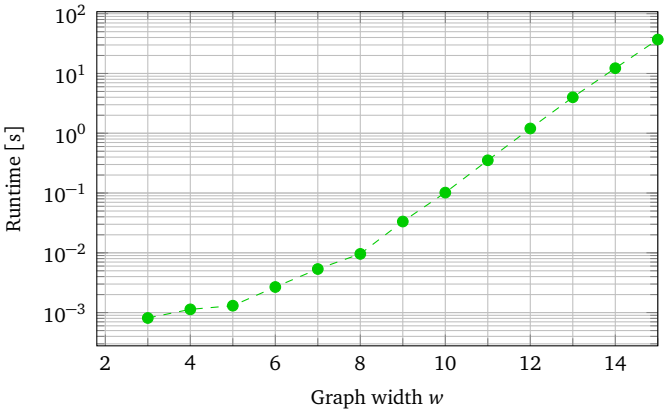


Figure 5.3: Runtime of the protocol selection algorithm for different graph widths w.

5.4.2 Use Cases

Next, we evaluate the circuits and protocol selections generated by HyCC for different use cases in ABY, cf. Chapt. 3. Note that the circuits generated by HyCC are generic and could be evaluated by any suitable MPC framework, e.g., ABY or *ABY*^{3} [MR18]. For the evaluation, we use applications that illustrate the versatility of HyCC or that have previously been used to benchmark MPC protocols and compilers.

Experimental setup All applications are implemented based on textbook algorithms and compiled with HyCC using a total optimization time of $T = 10$ minutes. The generated circuits are evaluated on two identical machines with an Intel Core i7-4790 CPU and 32 GiB RAM, connected via a 1 Gbps local network, denoted as LAN. To simulate an Internet connection between the MPC parties, denoted as WAN, we use the Linux tool `tc` to set a latency of 50 ms (100 ms RTT) and limit the throughput to 100 Mbps. We set the symmetric security parameter to κ =128 bit. Running times are median numbers from 10 measurements. "—" denotes that no values were given or benchmarked.

For all applications the number of non-linear (multiplicative) gates, communication rounds, transferred bytes, and the protocol runtime of the setup phase and of the online phase are measured. For comparison purposes we provide these numbers not only for the best protocol

Table 5.1: Modules and their circuit complexity when compiling the biometric matching example with HyCC.

Module	Non-Linear Gates			Non-Linear	# bits	
	A	*B*	*Y*	Depth	Inputs	Outputs
mpc_main	0	0	0	0	8 256	32
match (×128)	2	1 785	1 536	20	128	32
loop1 (×127)	—	120	64	10	64	32
total	128	$2 \cdot 10^5$	$2 \cdot 10^5$	909	8 256	32

selection, but also for different instantiations of the same functionality, e.g., all modules evaluated in a Boolean circuit-based protocol, or a hybrid of a Boolean circuit and arithmetic sharing. As before, we use *A* for arithmetic sharing, *B* for Boolean sharing using the GMW protocol, and *Y* for Yao's garbled circuits. We omitted *A*-only measurements for use cases that include bit-operations (e.g., minimum, comparison), since these are extremely costly in *A* sharing and therefore not implemented in ABY [DSZ15].

Biometric Matching (Minimum Euclidean Distance)

The minimum Euclidean distance is the minimum of the distances from a single coordinate to a list of coordinates. It is used in biometric matching between a sample and a database, and is a well-known benchmark for MPC, e.g., [BHWK16; DSZ15; HKS+10]. For illustration purposes a code example for the biometric matching functionality is shown in Listing 5.1 for a database of size $n = 128$ and dimension $d = 2$. The identified modules and their circuit sizes when compiling this code with HyCC are given in Tab. 5.1.

For the experimental evaluation we use databases consisting of $n \in \{1\,000; 4\,096; 16\,384\}$ samples with dimension $d = 4$, where each coordinate has bit length $\ell = 32$ bits. The performance results are given in Tab. 5.2. We compare a hand-built hybrid ABY circuit [DSZ15] to a circuit that is compiled with HyCC. The results show that the circuits that we automatically compiled from a standard ANSI C description achieve the same complexity as the hand-built and manually optimized circuits in ABY. Here, a combination of arithmetic sharing and Yao's protocol (*Y+A*) achieves the best runtime in all settings. The runtimes in both cases show a slight variation that is due to variance of the network connection. We remark that the setup phase of the ABY circuit is more efficient, because ABY allows Single Instruction Multiple Data (SIMD) preprocessing, which is currently not implemented in HyCC.

To show the efficiency gain of hybrid protocols over standalone protocols, we give experiments using *B* or *Y* sharing only. These protocols are significantly less efficient and for larger input sizes even exceed the memory resources of our benchmark hardware.

```
#define N 128
#define D 2

#include <inttypes.h>
typedef int32_t DT;

DT match(DT db1, DT db2, DT s1, DT s2) {
  DT dist1 = db1 - s1;
  DT dist2 = db2 - s2;
  return dist1 * dist1 + dist2 * dist2;
}

void mpc_main() {
  DT INPUT_A_db[N][D];
  DT INPUT_B_sample[D];
  DT matches[N];

  DT min = match(INPUT_A_db[0][0], \
     INPUT_A_db[0][1], INPUT_B_sample[0], \
     INPUT_B_sample[1]);

  for(int i = 1; i < N; i++) {
    DT dist = match(INPUT_A_db[i][0], \
     INPUT_A_db[i][1], INPUT_B_sample[0], \
     INPUT_B_sample[1]);

    if(dist < min) {
      min = dist;
    }
  }
  DT OUTPUT_res = min;
}
```

Listing 5.1: Biometric matching code example.

Machine Learning

Machine learning (ML) has many applications and is a very active field of research. Protecting the privacy of training data or ML inputs is also an active research area.

Supervised machine learning – Neural networks Deep (Convolutional) Neural Networks (CNNs) are one of the most powerful ML techniques. Therefore, many dedicated protocols for private data classification using CNNs have been proposed recently [GDL+16; LJLA17; RWT+18]. We implemented CryptoNets [GDL+16] and the very recent MiniONN CNN [LJLA17], which both have been proposed to detect characters from the MNIST handwriting data set[2]. Previously these use cases needed to be carefully built by hand, while

[2] http://cis.jhu.edu/~sachin/digit/digit.html

Table 5.2: Minimum Euclidean distance benchmarks comparing a hand-built circuit (ABY,cd. Chapt. 3) with a compilation from HyCC (best values marked in bold).

Circuit	Sharing	non-linear Gates	Comm. Rounds	Setup Phase LAN [ms]	Setup Phase WAN [ms]	Setup Phase Comm. [MiB]	Online Phase LAN [ms]	Online Phase WAN [ms]	Online Phase Comm. [KiB]
min. Euclid ABY [DSZ15] ($n = 1\,000$)	Y+A	98 936	6	167	2 878	8	**55**	**557**	**1 567**
min. Euclid HyCC ($n = 1\,000$)	Y+A	98 936	10	175	1 920	8	70	584	1 582
min. Euclid ABY [DSZ15] ($n = 1\,000$)	B+A	155 879	78	151	2 206	9	73	3 971	1 620
min. Euclid HyCC ($n = 1\,000$)	B+A	155 879	80	190	3 622	10	131	4 249	1 643
min. Euclid HyCC ($n = 1\,000$)	Y	3 166 936	3	1 498	10 239	99	1 177	1 789	4 016
min. Euclid HyCC ($n = 1\,000$)	B	3 497 879	93	550	8 228	107	2 932	7 974	1 725
min. Euclid ABY [DSZ15] ($n = 4\,096$)	Y+A	405 440	6	420	7 336	34	**211**	**1 234**	**6 416**
min. Euclid HyCC ($n = 4\,096$)	Y+A	405 440	10	536	5 162	34	330	1 406	6 480
min. Euclid ABY [DSZ15] ($n = 4\,096$)	B+A	638 855	92	417	8 016	37	303	5 606	6 629
min. Euclid HyCC ($n = 4\,096$)	B+A	635 020	94	555	4 337	41	689	5 802	6 722
min. Euclid HyCC ($n = 16\,384$)	Y+A	1 621 952	10	2 239	13 522	112	**1 419**	**4 041**	**25 920**
min. Euclid HyCC ($n = 16\,384$)	B+A	2 540 935	108	2 286	15 179	164	3 155	11 024	26 883

we achieve even better performance when conveniently compiling easily understandable C source code to a hybrid MPC protocol.

Tab. 5.3 shows the machine learning performance results. For Cryptonets, HyCC automatically determined *A* as the best sharing in the LAN setting. When changing the activation function (from the square function to $f(x) = max(0, x)$, known as RELU function), or when changing the number representation (fixed-point instead of integer), a hybrid *Y+A* protocol becomes the fastest option.

For the MiniONN CNN, HyCC proposes to use *Y+A*, where *Y* is mainly used to compute the RELU activation function, which results in a hybrid protocol that requires only a third of the online runtime, total runtime, and total communication compared to the original MiniONN protocol [LJLA17]. When expressing the entire MiniONN functionality solely as a Boolean circuit, more than 250 million non-linear gates are used. Using Yao's protocol in the LAN setting, sending the corresponding garbled circuit would take more than one minute, assuming perfect bandwidth utilization. Thus, in comparison to all existing Boolean circuit compilers for MPC, i.e., single protocol compilers, HyCC achieves a runtime that is more than one order of magnitude faster.

Unsupervised machine learning – *k*-means Clustering is another data mining task that is frequently used to identify centroids in unstructured data. One of the most well known clustering algorithms is *k*-means, and multiple works proposed dedicated privacy-preserving *k*-means protocols, e.g., [JW05; VC03]. We evaluate a textbook algorithm that detects $c = 4$ clusters in 2-dimensional data sets of size $n = 500$ using $i = 8$ iterations and show our results in Tab. 5.3. Also in this use case, a hybrid *Y+A* protocol achieves the best runtime.

Table 5.3: Machine learning benchmarks comparing with MiniONN [LJLA17], CryptoNets [GDL+16] with different activation functions, and the k-means algorithm (best values marked in bold).

Circuit	Sharing	non-linear Gates	Comm. Rounds	Setup Phase LAN [ms]	Setup Phase WAN [ms]	Setup Phase Comm. [MiB]	Online Phase LAN [ms]	Online Phase WAN [ms]	Online Phase Comm. [KiB]
MiniONN MNIST [LJLA17]	—	—	—	3 580	—	21	5 740	—	651 877
MiniONN MNIST HyCC	B+A	2 275 880	90	1 750	14 469	165	2 689	9 443	35 864
MiniONN MNIST HyCC	Y+A	1 838 120	34	1 825	14 041	150	**1 621**	**5 882**	**35 094**
CryptoNets Square [GDL+16]	—	—	—	0	—	0	297 500	—	381 133
CryptoNets Square HyCC	A	107 570	7	683	10 348	131	**134**	1 359	**2 018**
CryptoNets RELU HyCC	Y+A	195 455	19	784	11 238	134	163	1 297	3 330
CryptoNets RELU HyCC	B+A	195 455	33	735	11 298	134	187	1 917	3 360
CryptoNets Fix-Point HyCC	B+A	195 455	33	765	11 416	134	187	1 910	3 694
CryptoNets Fix-Point HyCC	Y+A	195 455	19	780	11 264	134	162	**1 296**	3 330
k-means HyCC ($n = 500$)	B+A	7 894 592	6 578	3 453	21 887	293	5 917	337 083	**30 473**
k-means HyCC ($n = 500$)	Y+A	4 991 816	125	4 414	21 007	206	**3 748**	**10 503**	38 915

Gaussian Elimination

Solving linear equations is required in many applications with Gaussian elimination being the most well known solving algorithm. We implement a textbook Gauss solver with partial pivoting for $n \in \{10, 16\}$ equations using a fixed-point number representation and present our results in Tab. 5.4. Fixed-point numbers can easily be implemented in software, and thus also using HyCC, which is illustrated in our paper [BDK+18]. In all scenarios, HyCC identifies *Y+A* as the most efficient protocol, where *Y* is mainly used to compute row permutations and divisions. Note that due to the large circuit depth, we did not measure runtime for Boolean circuits evaluated with the GMW protocol in the WAN setting.

Table 5.4: Gaussian elimination benchmarks for a 10×10 and a 16×16 matrix (best values marked in bold).

Circuit	Sharing	non-linear Gates	Comm. Rounds	Setup Phase LAN [ms]	Setup Phase WAN [ms]	Setup Phase Comm. [MiB]	Online Phase LAN [ms]	Online Phase WAN [ms]	Online Phase Comm. [KiB]
Gauss 10×10 HyCC	B+A	555 611	41 305	340	—	29	5 843	—	2 989
Gauss 10×10 HyCC	B	1 158 995	41 829	268	—	23	6 020	—	1 412
Gauss 10×10 HyCC	Y+A	494 215	147	348	2 849	17	**256**	4 235	1 997
Gauss 10×10 HyCC	Y	1 030 225	3	561	3 850	31	429	**631**	**101**
Gauss 16×16 HyCC	B+A	2 516 310	67 920	1 245	—	57	11 182	—	10 031
Gauss 16×16 HyCC	Y+A	2 294 615	243	1 515	8 842	79	**1 258**	8 126	7 740
Gauss 16×16 HyCC	Y	4 393 173	3	2 445	13 749	134	1 957	**2 190**	**257**

Database Analytics

Performing data analytics on sensitive data has numerous applications and therefore many privacy-preserving protocols and use cases have been studied, e.g., [BJSV15; DHC04]. Using generic MPC techniques is of interest for database analytics, as it allows to perform arbitrary analytics, e.g., hypothesis testing, or allows to add data perturbation techniques, e.g., differential privacy, before releasing the result with minimal effort. We study exemplary use cases, where each party provides a database (array) of size n_A and n_B that has two columns each, which are concatenated (merged) leading to a database of size $n = n_A + n_B$ or joined (inner join on one attribute) yielding a database of maximum size $n = n_A \cdot n_B$, and then the mean and the variance of one column of the combined database are computed. Our performance evaluation is shown in Tab. 5.5. We observe that in both use cases, a combination of *Y+A* achieves minimal runtime in the LAN setting, with the division (and join) being performed in Y. In the WAN setting, Y achieves optimal runtime and minimal online communication.

Table 5.5: Database operation benchmarks for merging or joining two databases with basic statistical analysis (best values marked in bold).

Circuit	Sharing	non-linear Gates	Comm. Rounds	Setup Phase LAN [ms]	Setup Phase WAN [ms]	Setup Phase Comm. [MiB]	Online Phase LAN [ms]	Online Phase WAN [ms]	Online Phase Comm. [KiB]
DB Merge 500 + 500 HyCC	B	1 441 732	1 237	593	3 776	44	1 310	63 430	733
DB Merge 500 + 500 HyCC	B+A	5 395	1 187	29	927	2	144	59 319	56
DB Merge 500 + 500 HyCC	Y	849 711	3	858	3 619	26	679	886	752
DB Merge 500 + 500 HyCC	Y+A	4 990	17	22	815	1	**4**	**606**	**30**
DB Join 50 × 50 HyCC	B	4 429 046	765	1 645	13 312	135	4 219	43 090	2 179
DB Join 50 × 50 HyCC	B+A	529 526	708	451	5 201	26	564	36 652	6 827
DB Join 50 × 50 HyCC	Y	2 550 076	3	1 725	8 317	78	1 272	**1 451**	**100**
DB Join 50 × 50 HyCC	Y+A	443 900	32	472	3 433	23	**435**	2 395	6 705
DB Join 25 × 200 HyCC	B	8 981 870	767	3 521	26 766	274	9 846	48 937	4 403
DB Join 25 × 200 HyCC	B+A	1 163 575	708	832	7 085	54	1 202	38 155	13 295
DB Join 25 × 200 HyCC	Y	5 158 825	3	3 212	15 960	158	2 660	**2 861**	**250**
DB Join 25 × 200 HyCC	Y+A	937 049	32	927	5 837	47	**942**	3 603	12 861

Summary of Experiments Summarizing the results obtained in all use cases, we observe that hybrid protocols consisting of *Y+A*, achieve efficient runtime in a LAN, whereas Y is often the fastest protocol for a WAN network. We observe that the GMW protocol (B) has barely been identified to achieve optimal runtime for any of the benchmark applications. This is because we performed all benchmarks in the function-dependent preprocessing model, which is the default setting in ABY, and which allows to garble the circuit in the setup phase. When using a function-independent cost model for preprocessing, HyCC identifies *B+A* as the fastest protocol combination in the LAN setting for many applications.

5.5 Conclusions and Future Work

In our evaluation we observed that hybrid protocols can significantly outperform standalone protocols. HyCC is capable of automatically synthesizing the required hybrid protocols from a high-level description in ANSI C and selecting them for a given deployment scenario. As such, HyCC is even capable of outperforming certain hand-optimized protocols. Moreover, as the manual creation of circuits and their selection are tedious and error-prone tasks, we conclude that HyCC makes hybrid MPC more practical and also accessible to developers without expert-knowledge in MPC.

In future work, we will extend HyCC with floating point operations and integrate more MPC protocols with different cost models. A natural candidate for extension is homomorphic encryption, similar to TASTY [HKS+10]. Another possibility would be integrating trusted execution environments such as Intel's SGX.

Part II

MPC Applications in the Outsourcing Scenario

6 Privacy-Preserving Internet Routing

Results published in:

[ADS+17] G. ASHAROV, D. DEMMLER, M. SCHAPIRA, T. SCHNEIDER, G. SEGEV, S. SHENKER, M. ZOHNER. **"Privacy-Preserving Interdomain Routing at Internet Scale"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017). Full version: https://ia.cr/2017/393, pp. 143–163. CORE Rank B.

[CDC+16] M. CHIESA, D. DEMMLER, M. CANINI, M. SCHAPIRA, T. SCHNEIDER. **"Towards Securing Internet eXchange Points Against Curious onlooKers (Short Paper)"**. In: *1. ACM, IRTF & ISOC Applied Networking Research Workshop (ANRW'16)*. ACM, 2016, pp. 32–34.

[CDC+17] M. CHIESA, D. DEMMLER, M. CANINI, M. SCHAPIRA, T. SCHNEIDER. **"SIXPACK: Securing Internet eXchange Points Against Curious onlooKers"**. In: *13. International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*. ACM, 2017, pp. 120–133. CORE Rank A.

6.1 Introduction

Routing is the process of finding a path in a network. For that, path selection is based on certain criteria that are used as inputs to a routing algorithm, which uses these inputs to determine routes and tries to achieve certain optimization goals. The optimization can target goals of technical nature, e.g., shortest latency or highest throughput, but also monetary, where the goal is to keep the routing cost as low as possible. An optimization strategy can even have political background in order to avoid routes through certain nodes or entire network segments. Naturally, inputs that influence these routing decisions are considered sensitive business information and operators want to keep them confidential.

Routing on the Internet is done using the Border Gateway Protocol (BGP) and happens in 2 processes: *Route computation*, cf. Sect. 6.1.1, is run in a decentralized process between so-called Autonomous Systems (ASes) and establishes paths to nodes in the network. *Route dispatch*, cf. Sect. 6.1.2, is the process of selecting, ranking and distributing the computed routes that takes place at central authorities, called Internet Exchange Points (IXPs).

In this chapter, we describe shortcomings with the current routing system and propose solutions that focus on achieving equivalent functionality with reasonable performance, while preserving the privacy of sensitive business information.

6.1.1 Interdomain Route Computation

Interdomain routing is the task of computing routes between the administrative domains, called "Autonomous Systems" (ASes), which make up the Internet. While there is a variety of *intra*domain routing designs to compute routes *within* an organizational network (e.g., RIP, OSPF, IS-IS), there is only one *inter*domain routing algorithm: the Border Gateway Protocol (BGP). BGP stitches together the many (over 55 000) ASes that the Internet is composed of and can thus be regarded as the glue that holds together today's Internet. BGP was specifically designed to meet the particular demands of routing between Internet domains, allowing each AS the freedom to privately and freely implement arbitrary *routing policies*, i.e., the expressiveness to both (i) select a route from the routes learned from its neighboring ASes according to its own local business and operational considerations and (ii) decide whether to advertise or not advertise this route to each of its neighboring ASes. Importantly, ASes' routing policies can leak sensitive information about their business relationships with other ASes and are therefore often kept private.

BGP achieves the dual goals of policy freedom and policy privacy through an iterative, distributed route computation. At each stage of the computation, a domain (AS) chooses which routes to use (among the routes being advertised to it by its neighbors), and then chooses to which neighboring ASes the resulting routes should be advertised. This process is repeated until convergence, thus allowing each domain to make its own policy-induced choices, without needing to explicitly reveal these choices to other domains. However, as pointed out in [MK06; GZ11] (and the references within), while BGP computation does not force domains to explicitly reveal policies, much information about routing policies can be inferred by passively observing routing choices.

While BGP has served the Internet admirably, it has many well-known drawbacks ranging from slow convergence to inability to deal with planned outages. Thus, we should explore alternative methods for interdomain routing. As suggested in [GSP+12], the use of secure multi-party computation (MPC) offers an intriguing possibility: executing the route computation centrally (among a few mutually distrustful parties) while using MPC to retain policy privacy. However, while the MPC technology provides ASes with provable privacy guarantees, scaling this approach to current Internet infrastructure sizes with over 55 000 ASes is a significant challenge. Specifically, the computation in [GSP+12] already required 0.13 s for a toy example of only 19 ASes and would hence require several minutes for today's Internet, even when assuming a low number of neighbors per AS. This chapter is devoted to the cryptographic paradigms, protocols, optimizations, and concrete tools necessary to compute interdomain routes at Internet scale in a privacy-preserving and efficient way.

Privately Centralizing BGP via MPC

The vision of (logically) centralizing interdomain routing can be regarded as the interdomain-level analogue of the SDN approach to routing within an organization (i.e., *intra*domain routing), which is revolutionizing computer networking, by allowing configuration changes in a software controller, However, reaping these benefits in the context of interdomain routing involves overcoming two grand challenges: (1) preserving *privacy* of the business-sensitive routing policies of the many independent organizations that take part in the computation, and (2) computation at very large *scale*, outputting a routing configuration spanning tens of thousands of organizations.

To overcome these challenges we combine knowledge of the two research areas of networking and secure computation. We build on top of our ABY framework (cf. Chapt. 3) and outsource the route computation to two computational parties P_0 and P_1, who are managed by two different operators, which we assume to not collude. To protect the privacy of their business relations, the ASes secret share their routing preferences with these two computational parties, such that no party gets any information about the ASes' routing preferences. The computational parties run our secure interdomain routing computation protocols to determine the routes for each AS in the network. The computation results are sent back to each AS, which can then reconstruct the plain text output of the algorithm. More specifically, the CPs only send a small message to the ASes which contains their next hop for a specific destination, while the communication- and round-intensive MPC protocol is run solely between the CPs. An example setting with 4 ASes is depicted in Fig. 6.2. In this setting all ASes are input and output parties at the same time (cf. Sect. 2.4.2).

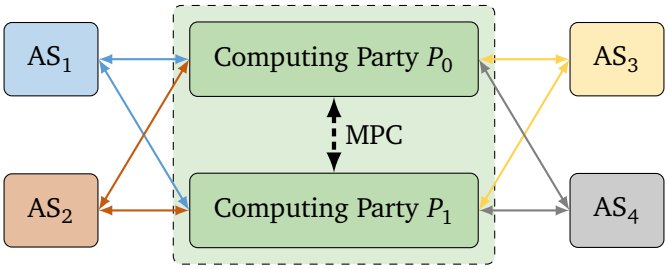


Figure 6.1: Example setting with 4 ASes that secret share their inputs with 2 computational parties P_0 and P_1. Thin arrows correspond to 1 round of communication with small messages, while the **bold** arrow symbolizes the execution of a secure computation protocol with many rounds and high bandwidth.

Motivation - Why use MPC for Interdomain Routing?

In the following section, we list some of the benefits that privately centralizing BGP via MPC can offer.

Better Convergence And Resilience To Disruption As opposed to BGP's inherently decentralized and distributed computation model, which involves communication between tens of thousands of ASes, in our scheme interdomain routes are computed by only two computational parties P_0 and P_1. BGP can take seconds to minutes to converge [LABJ00; ZMZ04; OZPZ09]. In the interim period, BGP's path exploration can have adverse implications for performance. Indeed, a huge fraction of VoIP (e.g., Skype's) performance issues are the result of bad BGP convergence behavior [KKK07]. Worse yet, BGP's long path exploration can even lead to intermittent connectivity losses. By centralizing computation and thus avoiding the long distributed (and asynchronous) path-exploration process, convergence time is reduced significantly. We demonstrate that our approach, despite harnessing MPC machinery, is much faster to compute global routing configurations than today's decentralized convergence process and, consequently, faster to recover from network failures and to adapt to changes in ASes' routing policies.

Less Congestion BGP permits ASes great expressiveness in specifying local routing policies at the potential cost of persistent global routing instability. However, as shown in [GR01], under natural economic assumptions (the so called "Gao-Rexford Conditions") BGP's convergence to a stable routing configuration is guaranteed. Unfortunately, even under these conditions, convergence might take exponential time (in the number of ASes) due to the exchange of an exponential number of messages between ASes [FSR11]. In contrast, our scheme guarantees fast convergence to the desired routing outcomes as our communication overhead is polynomial (linear in the size of the network).

Enhanced Privacy Many ASes regard their routing policies as private and do not reveal them voluntarily, as routing policies are strongly correlated to business relationships with neighboring ASes. BGP seemingly offers policy freedom and policy privacy, as each AS is free to choose which routes to use and which routes to advertise to others, without having to explicitly reveal its routing policies. However, BGP's privacy guarantees are limited, and are even fictitious. Monitoring selected BGP routes, in particular when done from multiple vantage points, can reveal much information about ASes' routing policies, e.g., their local preferences over BGP routes (see, e.g., [MK06; GZ11] and references within). In addition, AS business relationships can be reconstructed from publicly available datasets [CAI16]. We refer the reader to [GZ11] for an illustration of how monitoring the BGP convergence process can yield much information about ASes' routing policies.

Using MPC for interdomain routing can remedy this situation by providing provable privacy guarantees that cannot be achieved with today's routing on the Internet. Our scheme guarantees that no information about routing policies and inter-AS business relationships, other than that implied by the routing outcome, is leaked. In fact, each node (AS) learns *only its "next hop" node in the final routing outcome* with respect to a destination, and not even the full route. We also hide the entire convergence process, which potentially leaks information. As a side note, even with multiple vantage points, inferring routing policies is no easy task. While our scheme would not completely remove this kind of leakage, it would definitely decrease it compared to routing using BGP, where the ASes broadcast their full

routing table. Furthermore, it is unclear whether information about the policy preferences can be gained using only the next hop as information.

Enhanced Security BGP's computation model, which is distributed across all ASes, enables ASes to launch devastating attacks against the protocol, which can result in Internet outages [BFMR10]. By outsourcing BGP computation to a few parties, attacks on BGP that manipulate its decentralized computation, e.g., propagating bogus AS-level routes to neighboring nodes, are eliminated. We point out that centralizing interdomain routing is also compatible with the ongoing efforts to deploying Resource Public Key Infrastructure (RPKI) — a centralized certification infrastructure for issuing cryptographic public keys to ASes and for mapping IP addresses to owner ASes, thus preventing ASes from successfully announcing IP prefixes that do not belong to them ("prefix hijacking"). Verifying routing information through RPKI can be executed efficiently in a centralized manner in our approach.

Freedom to Adapt and Innovate The computational parties can easily update to new and more advanced protocols, thus offering more complex functionality, such as new security solutions, multiple paths per destination prefix, multicast routing, fast failover in response to network failures, etc.

'What if' Analysis Due to our low runtimes, we can precompute paths for cases of failure, i.e., simulate the removal of nodes and therefore significantly reduce the recovery times for these cases. This is possible since the network topology is known publicly and therefore topology changes can be simulated.

6.1.2 Route Distpatch at IXPs

With the rise of Internet Exchange Points (IXPs) as the emerging physical convergence points for Internet traffic, new privacy concerns arise. IXPs offer centralized Route Server (RS) services for ranking, selecting, and dispatching BGP routes to their (potentially many hundreds of) member networks [RSF+14]. However, to benefit from these centralized services, IXP members need to divulge *private* information, such as peering relationships and route-export policies to the IXP or, even worse, to other IXP members. Such information can reflect sensitive commercial and operational information, and is consequently often regarded as private [GSP+12; ZZG+16]. Indeed, our interaction with IXP administrators, and our survey of network operators (Sect. 6.2.4), reveal that such privacy concerns are widespread and that some networks even refrain from subscribing to RS services for precisely this reason. Beyond privacy, our survey reveals three additional IXP members' concerns for RS usage: limited routing policy expressiveness, reliability, and insufficient value. This situation hinders RS adoption and makes it hard to provide novel valuable routing services to IXP members, as these can rely on the exposure of (even more) sensitive data. Indeed, on one hand, advanced performance-oriented routing services are fundamental to improve performance of video and latency-critical Internet applications [TG04; JD08; CB09; PMH09; XYC09; CML10; CLM11; WLL+14; KRG+16; PTH16]. In this regard, today's largest content providers (e.g., Google and

Twitch) resort to active measurement techniques for inferring route performance [Vah17], a difficult task in practice [CML10]. On the other hand, our survey reveals that a large majority of network operators (60%) is concerned about sharing network performance information such as IXP port utilization with external entities. Therefore, the goal of supporting advanced Internet routing features while protecting sensitive information is the subject of several recent studies [MK04b; GSP+12; HR13; KSH+15; LPB+16; ZZG+16; KHH+17; BCP+17].

How should we design RSes?

To increase trust in IXPs and motivate further adoption of RS services, we argue that RSes should meet the following basic requirements:

1. *Easy management*, i.e., relieve IXP members from the burden of configuring numerous BGP peering sessions.
2. *Policy expressiveness*, i.e., provide IXP members with highly-expressive route selection at least equivalent to having multiple bilateral BGP sessions [GSG14].
3. *Performance-driven routing*, i.e., dispatching tools that leverage the IXP's superior visibility into data-plane network conditions.
4. *Efficiency*, i.e., today's RSes are required to compute and dispatch routes in the order of hundreds per second, with full routing-table transfers performed in the order of minutes [AMS12; DEC16], in order to be able to quickly react to new routing information or network failures.
5. *Privacy preservation*, i.e., neither an IXP member nor the IXP itself should be able to learn information about routing policies of the other members (except for information that can be deduced from its own RS-assigned routes).
6. *Reliability*, i.e., guaranteed connectivity upon failures in the IXP infrastructure.

SIXPACK: A privacy-preserving advanced RS

To accomplish the above, we advocate implementing an RS via MPC, cf. Sect. 2.4. With an MPC-based realization of an RS, the desired routing outcome can be computed without the IXP or IXP members gaining visibility into the "inputs" to this computation, i.e., members' private routing policies.

However, realizing our vision of a privacy-preserving RS is highly nontrivial. General-purpose MPC machinery is excessively heavy in terms of computation and communication overheads and is thus infeasible to employ for this purpose [GSP+12]. Consequently, attaining feasible runtimes and communication costs requires devising a suitable highly-optimized, privacy-preserving scheme specifically tailored to the RS context. We present SIXPACK, the first IXP

route server service that satisfies *all* the aforementioned requirements.[1] It efficiently ranks, selects, and dispatches BGP routes based on the members' expressive routing policies and any IXP-provided performance information *without* leaking any confidential business peering information.

A conceptual overview of SIXPACK is given in Fig. 6.2. The IXP route server service is jointly performed by two independent and non-colluding computational parties, P_0 and P_1, run an MPC protocol. We also refer to P_0 and P_1 as *Route Servers* (RSes). Each IXP member encrypts its BGP routes, announces them to the RSes, and creates two "shares" of its (private) business peering policy that are sent to the two RSes. Each of the RSes, in turn, sends to each IXP member, upon completion of the MPC, a share of its output, that the member can use to recover its selected routes. SIXPACK provably guarantees that as long as the RSes P_0 and P_1 do not collude with each other, neither the RSes nor other IXP members will learn any information regarding an IXP member's business and routing policies. We envision P_0 and P_1 as being run by the IXP and a neutral and well-regarded international organization (e.g., NANOG or RIPE), which is already trusted to support and operate fundamental Internet services (though not necessarily trusted for privacy).[2] In addition, one of the two RSes should be executed on a machine that is located outside the IXP but in very close proximity (i.e., within the same colocation data center) so as to be in a separate security domain while keeping latency of inter-RS communication at a minimum (i.e., < 1 ms).

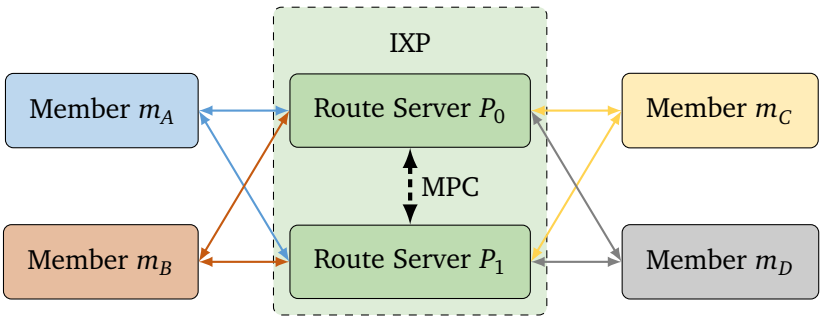


Figure 6.2: Conceptual overview of SIXPACK. The four IXP members communicate with the two route server entities P_0 and P_1, which run MPC to perform ranking, selection, and dispatch of BGP routes.

We observe (Sect. 6.5) that a naïve application of MPC to RS computation, in which the entire RS computation is carried out via MPC machinery, is infeasible in practice as a result of unrealistic computation and communication overheads in MPC. In fact, network operators rely on a highly-expressive inter-domain routing protocol to export, rank, and filter routes based on their own routing policies. In the Border Gateway Protocol (BGP), the standard de-facto

[1] While the focus in this work is not reliability, even today, at large IXPs, members are contractually requested to set multiple peering sessions with distinct RSes for redundancy requirements. Also, recently proposed Internet drafts [BHS+17] further mitigate the impact of failures in the IXP network.

[2] In addition, our survey of network operators [CDC+17a] reveals RIRs enjoy the trust of a large fraction (88%) of respondents.

inter-domain routing protocol, such operations often entail evaluating regular expressions based on the traversed networks, a computationally prohibitive operation in MPC [Kel15, §5]. To preserve the same routing expressiveness of BGP, we thus resort to the following approach: SIXPACK is carefully designed to keep complex computation outside the MPC, to the largest extent possible without compromising privacy. Specifically, in SIXPACK, we carefully decompose the RS functionality into two simple, yet crucial, MPC-based building blocks for *efficient* route-dispatch, called EXPORT-ALL and SELECT-BEST, while performing the most complex computation over unencrypted data outside of the MPC without leaking any private information. Namely, all members that wish to announce a route through the IXP will first locally compute the set of members to whom each route should be exported using any arbitrarily complex "export" routing policy. This computation is performed outside of the MPC. Using EXPORT-ALL, all available BGP routes that are exportable to an IXP member, i.e., the routes that other IXP members are willing to advertise to that member, are dispatched to the member in a fully privacy-preserving manner. Then, the member locally ranks its available routes outside of the MPC according to its possibly complex local preferences over routes and feeds the resulting ranking as input into SELECT-BEST. SELECT-BEST leverages this information and information from the IXP (e.g., port utilization) to select the best route for that member without leaking any information.[3] Performing the ranking of routes outside of the MPC greatly enhances the performance of the system.

Thus, SIXPACK both goes well beyond the services offered by today's RSes (by delegating route-selection from the RS to the member and incorporating performance-related information into the route selection process) and provides strong privacy guarantees to IXP members.

We discuss SIXPACK's optimized design, underlying assumptions, threat model, deployment challenges, IXP visibility of data-plane traffic, etc., in detail in the following sections.

6.1.3 Outline and Contributions

In the following, we summarize our contributions for privacy-preserving route computation Sect. 6.1.3 and privacy-preserving route computation in SIXPACK Sect. 6.1.3. Afterwards, we present shared preliminaries of both approaches presented in this chapter in Sect. 6.2 and an overview of related work in Sect. 6.3.

Our Contributions for Privacy-Preserving Route Computation

Interdomain Routing Algorithms for MPC (Sect. 6.4) Interdomain routing is a long-standing research topic for which several efficient algorithms exist. When transferred to the MPC domain, however, the complexity of the algorithms changes drastically, since data-dependent optimizations of the algorithms are not possible in MPC. In this chapter, we select two interdomain routing algorithms, which provide different capabilities for setting routing policies: one approach is based on neighbor *relations* and the other approach is based

[3] Thanks to our modular design, a member may skip the SELECT-BEST phase at the cost of revealing to the IXP that it disregards using IXP information.

on neighbor *preferences*. The neighbor relations-based routing algorithm is due to [GSG11] and uses business relations between ASes to perform routing decisions (Sect. 6.4.1). The neighbor preference-based routing algorithm was used in the MPC protocol of [GSP+12] and allows ASes to rank neighbors based on their preferences and give export policies which specify whether a route to a neighbor i should be disclosed to a neighbor j (Sect. 6.4.2). We implement these algorithms in a centralized setting, which allows for consistency checks of the ASes' inputs and thereby prevents malicious ASes from providing inconsistent input information (cf. Sect. 6.10.1).

Construction and Optimization of Boolean Circuits for BGP (Sect. 6.4.4) We convert the neighbor relation BGP algorithm of [GSG11] and the neighbor preference BGP algorithm of [GSP+12] into a distributed secure computation protocol between two parties to provide privacy. We explain challenges facing the implementations of these functionalities as a Boolean circuit, optimized for both low multiplicative depth (the number of AND gates on the critical path of the circuit) and low multiplicative size (the total number of AND gates) for evaluation with the GMW protocol [GMW87] implemented in ABY (cf.Chapt. 3). We provide details on several building blocks, how we optimize them, and the techniques we use to achieve high performance so as to be able to process real-world data.

Deployment and Future Directions (Sect. 6.7) Our aim is to demonstrate the practical feasibility of using MPC for interdomain routing. However, we view our work only as a first stepping stone that should serve as a basis for further research. We first explain our assumptions about the network in Sect. 6.7.2. We list promising directions for further enhancing robustness and measures against misconfigurations in Sect. 6.10.1 and discuss security against stronger adversaries in Sect. 6.10.2. Of course, deploying MPC for interdomain routing is a challenging undertaking that involves cooperation of tens of thousands of independent entities, alongside significant operational challenges. We argue, however, that our approach can also yield notable benefits when applied at a smaller scale. In addition, we show that even when applying our approach to high-density networks in fairly compact areas, namely, the German interdomain network, runtimes decrease to 0.20 s precomputation time and 0.17 s online time (cf. Fig. 6.12). We discuss the possibility of deployment and related issues of our approach in Sect. 6.7.3.

Benchmarks and Evaluation (Sect. 6.9.1) We benchmark our implementations on a recent empirically derived BGP dataset with more than 50 000 ASes with maximal degree 5 936 and almost 240 000 connections between them. We propose to exclude stub nodes from the computation as further optimization and evaluate complex Boolean circuits with several million AND gates. The neighbor relation algorithm requires about 6 s of topology-independent precomputation time and an online time of about 3 s on two mid-range cloud instances, that are comparable to off-the-shelf desktop computers. The neighbor preference algorithm takes about 13 s of topology-independent precomputation time and 10 s online time. We argue that while the online runtimes alone are not sufficient to provide adequate response to network failures, it allows the precomputation of routes for many failure scenarios, which would enable almost instantaneous failure recovery.

Our Contributions for Privacy-Preserving Route Dispatch with SIXPACK

The contributions we provide with SIXPACK are the following:

Preliminaries (Sect. 6.2.4) We analyze operators' concerns about peering with RSes at IXPs through a survey with 119 responses and a measurement of RS usage at one of the largest IXP worldwide and summarize our findings.

SIXPACK System Design(Sect. 6.5) We design and implement the first IXP route server capable of keeping the peering policies and routing preferences private, while allowing the IXP members to express arbitrary BGP routing policies *including* policies that incorporate confidential performance-related information available at the IXP.

Security (Sect. 6.6) We discuss the security and privacy guarantees of our schemes that inherit their properties from the underlying protocols and building blocks.

Deployment (Sect. 6.7.6) We believe that SIXPACK can be incrementally deployed in practice with little effort and in coexistence with traditional RS services.

Implementation (Sect. 6.8.1) We describe our implementation for applying MPC to the important and timely context of route dispatch at IXPs and our efficient and privacy-preserving MPC building blocks for RSes.

Performance Evaluation (Sect. 6.9.2) We provide a performance evaluation of our prototype. Through experiments with a BGP trace from one of the largest IXPs in the world, our results show that SIXPACK scales to hundreds of IXP members and achieves BGP processing times below 90 ms at the 99-th percentile. Via microbenchmarks we assess the online costs of MPC-based RSes to be well within *real-time* processing requirements of large IXPs.

6.2 Preliminaries

We provide preliminaries on our setting and assumptions (Sect. 6.2.1), modeling the BGP protocol (Sect. 6.2.2), and detail our protocols' input data in (Sect. 6.2.3). We provide an survey regarding privacy and IXP usage in Sect. 6.2.4 and discuss the IXP threat model and our assumptions inSect. 6.2.5.

6.2.1 Setting and Assumptions

We introduce the core concept of MPC in Sect. 2.4 and the definition of the adversary model in Sect. 2.2.

In our setting (as well as in [GSP+12]), we assume that the computing parties are semi-honest. Our basic variant of the protocols assumes that the ASes are semi-honest as well (and may collude with some of the computing parties). In the more involved variant of our protocols (unlike [GSP+12]), we tolerate even malicious behavior of the ASes (cf. Sect. 6.10.1).

In this work, we use MPC in an *outsourcing scenario*, cf. Sect. 2.4.2, where many ASes secret-share their private inputs to two computational parties, who run the secure computation on these inputs. The outputs are sent back to the ASes, who reconstruct the plaintext output.

6.2.2 Modeling BGP

We now give an overview on important aspects of modeling BGP, as discussed in [GR01; GSW02; GSG12] (and references therein). Throughout this chapter the terms AS, domain, vertex or node are used interchangeably. When peering at an IXP, ASes are called members.

The AS-Level Graph

The AS-level topology of the Internet is modeled as a network graph $G = (V, E)$ where vertices represent ASes and edges represent connections between them. Each edge is annotated with one of two business relationships: *customer-provider*, or *peering*. A *customer-provider* edge is directed from customer to provider; the customer pays its provider for transmitting traffic to/from the customer. A *peering* edge represents two ASes that agree to transit traffic between their customers at no cost. We assume that these relationships are symmetric, i.e., if AS a is a peer of AS b, then b is also a peer of a and if AS c is a customer of AS d, then d is a provider of c. ASes with customers are Internet Service Providers (ISPs). We call an AS with no customers a "*stub* AS".

Routing Policies

ASes' routing polices reflect their local business and performance considerations. Consequently, routing policies are considered sensitive information as revelation of an AS's routing policy can potentially leak information about its business relationships with others to its competitors (or other relevant information). We use the standard model of routing policies from [GR01; GSW02]. Each AS a computes routes to a given destination AS dest based on a *ranking* of simple (loop-free) routes between itself and the destination, and an *export policy*, which specifies, for any such route, the set of neighbors to which that route should be announced. We next present a specific model of routing policies that is often used to simulate BGP routing (see, e.g., [GSHR10; GSG11; GSG12]).

Ranking AS a selects a route to dest from the set of paths it learns from its neighbors ASes according to the following ranking of routes:

- **Local preference.** Prefer outgoing routes where the "next hop" (first) AS is a customer over outgoing routes where the next hop is a peer over routes where the next hop is a provider. This captures the intuition that an AS is incentivized to select revenue-generating routes through customers over free routes through peers over costly routes through providers. Optionally, an AS can have preferences within each group of neighbors, i.e., it can prefer a certain provider over another one.
- **Shortest paths.** Break ties between multiple routes with the highest local preference (if exist) in favor of shorter routes (in terms of number of ASes on them). Intuitively, this implies that an AS breaks ties between routes that are equally good from a business perspective, in favor of routes that offer better performance.
- **Arbitrary tie breaking.** Break any ties between multiple remaining routes arbitrarily.

Export policies The following simple export policy captures the idea that an AS is willing to transit traffic between two other ASes if and only if one of these ASes is a paying customer: AS b announces a path via AS c to AS a iff at least one of a and c is a customer of b.

BGP Convergence

BGP computes routes to each destination independently and so, henceforth, we consider route computation with respect to a single destination AS dest. In BGP, each AS repeatedly uses its ranking function to select a single route from the set of routes it learns from its neighbors, and then announces this route to the set of neighbors dictated by its export policy. This goes on until BGP computation *converges* to a stable routing outcome where no AS wishes to change its route. Observe that an AS can only select a single route offered to it by a neighbor. The set of selected routes upon convergence must form a tree rooted in the destination dest, referred to as the *routing tree* to AS dest. Under the routing policies specified above, BGP is guaranteed to converge to a unique stable routing tree [GR01] given the arbitrary tie-breaking strategy.

6.2.3 BGP Input Data

It is our goal to simulate the algorithms under realistic conditions and show their practicality on real-world data. To this end, we use the network topology and AS business relationships provided in the CAIDA dataset from November 2016 [CAI16]. This dataset is empirically generated and provides us with both a realistic network topology, which we can use as public input, as well as inferred business relationships between domains, which we use to simulate the private inputs of the ASes. We evaluate our protocols on datasets from the past 10 years for full topologies and recent subgraphs thereof to show how our implementations scale and provide detailed results in Sect. 6.9.1.

A possible way of deploying our solutions for MPC-based route computation could be with the help of a Regional Internet Registry (RIR) or on smaller, regional scale. Starting from the original CAIDA topology, we created subgraphs using the GeoLite database[4] for each of the 5 RIRs and Germany as an example of a regional topology (cf. Sect. 6.7.3).

The BGP Network

Here, we provide insights into the development and growth of the BGP network since 1998. With the historic data collected by CAIDA [CAI16], we can assess how the BGP network develops over time. In Fig. 6.3, we depict the count of ASes with a given number of neighbors. We show that many ASes are only sparsely connected and only very few nodes have a large number of neighbors. In Fig. 6.4, we show the number if ASes and connections between them for both the full CAIDA dataset, as well as the set with the stub nodes removed. Note that both axes are in a logarithmic scale. 91% of the nodes have at most 8 neighbors in the full topology, while in the no-stub topology the average degree is higher and the number of nodes with degree 8 and less is 69%.

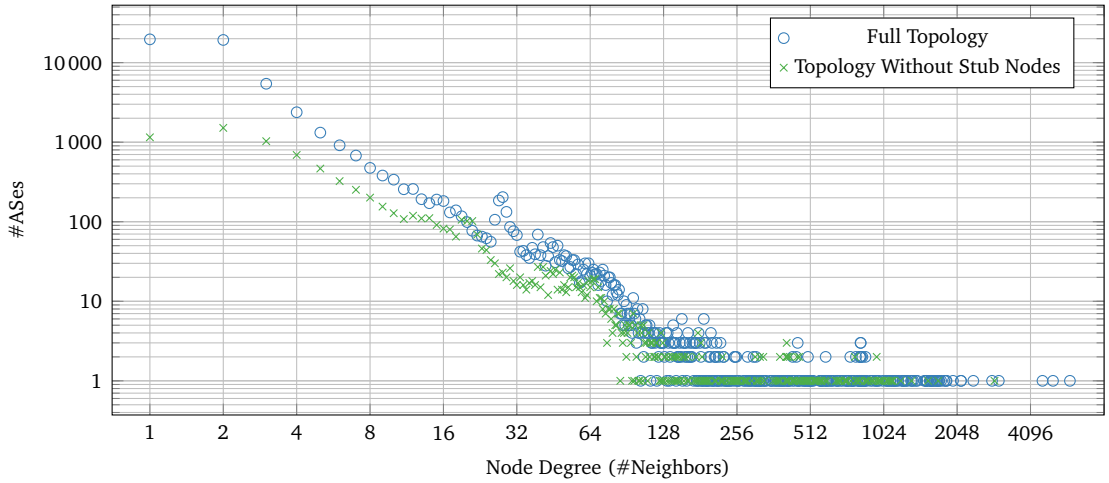


Figure 6.3: Distribution of node degree for the full 20161101 CAIDA dataset compared to the set without stub nodes.

6.2.4 On IXPs and Privacy

We present below preliminaries on Internet exchange points and quantify, via measurements, the extent to which route server services are used. We also report on our interaction with IXPs and member network administrators, including a survey of network operators, indicating that privacy concerns are a significant factor hindering widespread RS usage and corroborating assumptions underlying past research regarding the privacy of routing policies.

[4] http://dev.maxmind.com/geoip/

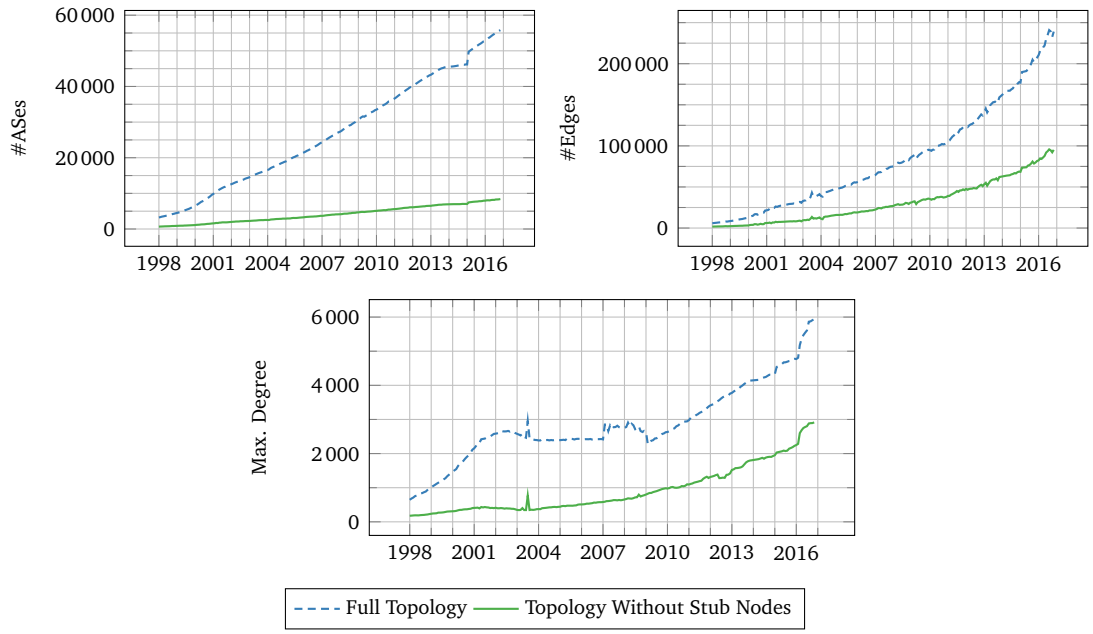


Figure 6.4: Statistics for the number of ASes, edges between them and the maximum degree for all available CAIDA ASrel datasets from January 1998 until November 2016 for the full topology compared to the topology without stub nodes.

Background on IXPs

IXPs are high-bandwidth physical networks located within a single metropolitan area. IXPs are typically geographically distributed[5,6] and hosted within colocation centers,[7] facilities operated by *third party providers* that offer high levels of physical security.

Heterogeneous economic entities, called *members*, use IXPs to exchange Internet traffic among each other [ACF+12]. To do so, each member connects its own network to one or more physical ports at the IXP network. After physical connectivity is established, each member announces the set of IP prefix destinations for which it is willing to receive traffic and starts receiving route announcements from the other members of the IXP.

The routes used to reach prefixes are spread and selected via the de facto standard inter-domain routing protocol of the Internet, i.e., Border Gateway Protocol (BGP). To this end, a full-mesh of BGP sessions among each pair of IXP members may be established. At medium to large IXPs, which can have over 800 members and carry over 5 Tbps, such full-meshes can be partially replaced by a *Route Server* (RS) service to ease the exchange of BGP announcements among members [RSF+14]. The RS establishes a BGP session with each IXP member, and

[5] https://ams-ix.net/technical/ams-ix-infrastructure
[6] https://www.de-cix.net/en/access/the-apollon-platform/setup-frankfurt
[7] http://www.interxion.com/

collects and distributes their BGP route-announcements. Note that data-plane traffic does not traverse the RS, which is only involved in control-plane traffic.

Each IXP member has the freedom to specify, for each destination IP prefix, an *export policy*, i.e., the set of other IXP members that are allowed to receive its route announcements. The RS selects, for each member, a route (per IP prefix) that is "exportable" to that member (according to members' export policies), and dispatches it to that member.

Today's IXPs do not allow their members to influence the RS's route selection with the members' *import policies*. These policies comprise of the traditional BGP *local preferences* [Cis16] and regular expressions that the RS uses to rank and filter available routes [GSG14]. For instance, an operator may be interested in routing its traffic through a certain IXP member unless the announced route traverses a specific network.

How widespread is RS usage?

Despite the fact that such an RS service eases the management of BGP sessions, facilitates peering, and lowers hardware requirements on connected BGP routers, there is anecdotal evidence of its limited usage. To corroborate this, we performed an analysis of RS usage at one of the largest IXPs worldwide. We have been reported that similar values hold for at least another of the largest IXPs worldwide. We examined both data traffic and BGP control plane messages so as to quantify the fraction of traffic that is routed along the routes dispatched by the RS. Fig. 6.5 presents a CDF graph showing the fraction of IXP members that have less than a certain fraction of "public traffic", i.e., traffic routed along the RS-computed routes. As shown in the leftmost part of the figure, 40% of members do not route their traffic via RS-prescribed routes. About two-thirds of the remaining members route less than half of their traffic according to the RS and only 20% of members route over 50% of their traffic along RS-computed routes. In terms of absolute amount of traffic, we discovered that less than 17% of the overall traffic is routed via RS-prescribed routes. While Ager et al. [ACF+12] observed that most of the networks peer with the RS, we showed that members prefer to route the vast majority of their traffic based on the information exchanged through bilateral BGP sessions.

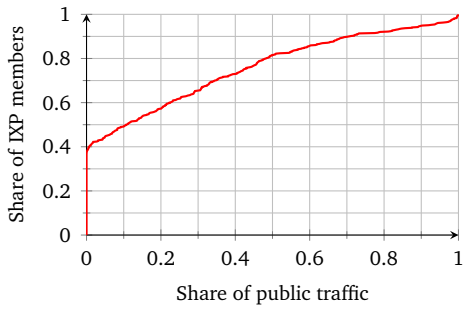


Figure 6.5: CDF of RS usage at a large IXP.

Are privacy concerns hindering wider RS usage and innovation?

One of the main barriers facing the transition from a full-mesh of BGP peering sessions to a star topology (via an RS) is that the export policy of each member (and, to support route-selection at the RS, potentially also the import policy of each member) must be revealed to the IXP. This information is considered confidential, primarily due to commercial reasons. Indeed, our interaction with IXP administrators and network operators reveals such privacy concerns and, moreover, that some networks do not connect to RSes for precisely this reason. In particular, we circulated a survey among the network operator community [CDC+17a] with the aim of exploring their perceptions about privacy at IXPs. We collected 119 responses belonging to a broad range of different networks: Tier 1 ISPs (8%), Tier 2/3 ISPs (57%) CDNs (6%), content providers (12%), and others (17%), with almost all networks connecting to an IXP and 80% of them using RS services (at least for that fraction of traffic not belonging to an established bilateral peering). According to our survey, the most critical concerns regarding RSes among respondents were: no control over best route selection, i.e., lack of import policy configuration tools (53%), reliability (40%), lack of route visibility (37%), privacy (19%), and legal restrictions (7%). Since the 1st and 3rd item require members to disclose their policies to the RS, we further investigated this privacy aspect: (i) 40-45% of respondents consider their local preferences over BGP routes to be private, both with respect to the IXP and with respect to other members, (ii) 60% of respondents expressed concerns about sharing their IXP port utilization with other members, and (iii) 1 of every 4 respondents that do not use an RS service marked concerns about disclosing export policies to the IXP as a reason. With comments ranging from "*Nothing should be considered private*" to "*Everything listed is supposed to be private/proprietary information*", our survey revealed the heterogeneous requirements of Internet domains. Despite the existence of many networks with open peering policies, our survey reveals that local-preferences over routes and port utilizations are still considered as a private information not to be divulged.

We point out that beyond privacy concerns, revealing sensitive information also entails the risk of triggering attacks on the weaker parts of the network [MK04a], e.g., easier bandwidth-exhaustion attacks if port utilization is revealed [WR04].

Overall, SIXPACK preserves the benefits of centralized route computation while tackling 3 out of 4 concerns from operators with RSes (i.e., route visibility, best route control, privacy), enhancing RS functionality with performance information, and retaining easy management.

6.2.5 IXP-Threat Model and Assumptions

To provide strong privacy guarantees, SIXPACK relies on three assumptions (A), which we list and justify next.

Assumption 1: Two Non-colluding RSes We assume that the RS service consists of two distinct route servers: one is operated by the IXP and one by an independent, non-colluding entity. The latter RS is executed on a machine that is outside of the IXP domain but connects

to the IXP network at the co-location center where the IXP is hosted. Since this instance lives in a separate security domain, this solution minimizes the possibility of an RS instance being compromised by the IXP, while keeping latency at a minimum.[8]

We believe that neutral international organizations (e.g., RIPE), which are already trusted to *support* and *operate* fundamental Internet services such as DNS and IP allocation, should be assigned the task of running an instance of an RS or, alternatively, supervising those that do. We argue that running an RS instance is a simpler task than operating a distributed DNS system. Our survey of network operators [CDC+17a] reveals that RIRs enjoy the trust of an overwhelming fraction (88%) of respondents. We point out that, even at today's large IXPs, members are requested to set up multiple peering sessions with distinct RSes for redundancy requirements.[9] In MPC, redundancy is required for both RS instances.

Assumption 2: Honest-but-curious RSes SIXPACK protects against so-called "honest-but-curious attackers", i.e., RSes that stick to the protocol but try to infer the members' private inputs. We argue that as today some networks refrain from peering with others via route servers because of fear of revealing private information to the RSes, this model captures an important desideratum in the IXP ecosystem. Furthermore, if cheating (e.g., deviation from the protocol by an RS) was detected, this would result in massive loss of trust in the service and, consequently, severe economic consequences for the IXP. We point out, however, that our MPC circuit constructions (Sect. 6.5.4 and Sect. 6.5.5) could be evaluated with a framework secure against malicious adversaries (e.g., [NNOB12]) if desired, though at higher computation and communication overheads.

Assumption 3: No Visibility into Data Traffic SIXPACK is designed to hide control-plane information from the RSes. We view data-plane privacy-preservation, i.e., preventing the IXP from inferring routing policies from observing data traffic traversing the IXP network, as an orthogonal problem that requires further exploration. We refer the reader to Sect. 6.7.5 for an explanation of why inferring routing policies from the data plane is highly challenging even when information about BGP routes is available. We point out, however, that SIXPACK actually does make the inference of routing policies from data traffic more challenging for the IXP. Guaranteeing data-plane-level privacy can also involve other approaches such as encrypting and decrypting IP headers at IXP ingress and egress. We leave this interesting topic for future research.

6.3 Related Work

The innovative idea of using MPC for BGP was first proposed in [GSP+12]. The aim of that paper was to illustrate benefits and challenges of this approach, and to explore generic

[8] https://ams-ix.net/technical/statistics/real-time-stats

[9] https://www.de-cix.net/en/news-events/news/faq-peer-in-frankfurt-and-new-york-with-one-port

cryptographic schemes towards its realization. In our work, we take an important step forward, providing a more concrete cryptographic approach that is tailored to interdomain routing, and thus leads to significant improvements, and show that we achieve reasonable performance at the scale of today's Internet.

Even though [GSP+12] outsources the route computation to few non-colluding parties, it does not utilize this approach to the fullest extent possible. In terms of functionality, the internal protocol of [GSP+12] that the clusters execute is very close to the BGP protocol. That is, the computational parties get the secret shares of routing preferences of all ASes, and then simulate an execution of the BGP protocol on "virtual" ASes with these shared preferences: The clusters run several iterations until convergence, where in each iteration, each virtual AS (i) selects a route from the routes learned from neighboring ASes and (ii) decides whether or not to advertise this route to each of its neighbors based on a set of export policies. These decisions, however, incur a high overhead in cryptographic computations since, in order to hide the policy that is applied, all policies have to be applied once per iteration. In order to securely evaluate the routing algorithm, [GSP+12] uses the MPC protocol of [BGW88], which provides passive security in the case of a honest majority, i.e., if $t < n/2$ of the n parties have been corrupted. Therefore it needs at least $n = 3$ parties, whereas our approach requires only $n = 2$ parties.

In this work, we follow up on the work of [GSP+12] and suggest the use of a second established interdomain routing algorithm that avoids these computation-heavy policies using a simpler routing strategy based on business relations [GSG11], given in Sect. 6.4.1. We compare the performance of this algorithm to the preference-based algorithm that was used in [GSP+12] and which we outline in Sect. 6.4.2. In addition, we use the secure computation protocol of Goldreich-Micali-Wigderson [GMW87] for secure evaluation of the algorithms, since it provides security in case of dishonest majority. Also, [BLO16] recently showed that the protocol of [GMW87] scales better to a larger number of parties than the protocol of [BGW88].

While the results in [GSP+12] studied how BGP routes across the whole Internet can be *computed* in a privacy-preserving manner, we additionally focus on *route dispatching* at IXPs, the crucial crossroads of the Internet that run computation on private, business-sensitive routing information. We argue that applying MPC to this narrower context is a promising approach to privacy-preserving interdomain route-computation and we built a prototype that handles real-world traces from a large IXP. Additionally to computing routes in a multihop network, we also solve the simpler problem of *exporting and ranking* sets of available routes, based on confidential business information, which even allows real-time processing of data.

Other related works proposed privacy-preserving graph algorithms, but did not consider the more complex BGP algorithm: [HR13] proposes STRIP, a protocol for vector-based routing that computes the shortest path based on the Bellman-Ford algorithm. In their protocol, the routers forward encrypted messages along the possible paths that accumulate the costs along the path using additively homomorphic encryption. This approach requires many messages until it converges and the routers need to implement costly public-key encryption

whereas in our solution all cryptographic operations are outsourced to the two mutually distrustful computational parties. [BS05] provides privacy-preserving graph algorithms with security against passive adversaries for all pairs shortest distance and single source shortest distance. [BSA13] provides data-oblivious graph algorithms for secure computation, such as breadth-first search, single-source single-destination shortest path, minimum spanning tree, and maximum flow, the asymptotic complexities of which are close to optimal for dense graphs. [CMTB13] introduces an outsourced secure computation scheme that is secure against active adversaries and uses it to compute Dijkstra's shortest path algorithm. [LWN+15] introduces a framework that compiles high-level descriptions into programs that combine secure computation and ORAM and gives speed-ups for Dijkstra's shortest path algorithm. However, the complexities of these algorithms that hide the topology of the graph are too high to scale to the size of today's Internet consisting of thousands of nodes.

A further routing-related study is SPIDER [ZZG+16], a distributed mechanism for verifying if a peering agreement between ASes (involving, e.g., a requirement to always export the shortest route available) is respected by the involved parties without revealing control-plane information (e.g., which routes are available). Applying SPIDER to our context could aid IXP members in verifying that the RS is indeed executing the protocol (in contrary to, e.g., selecting an un-optimal route for each member). Our focus is different: we guarantee that the RS does not learn anything about members' export and import policies. Finally, while SGX could be used to preserve the privacy of interdomain routing policies [KSH+15; MDL+17], we discussed its limitations in Sect. 2.6.3. Thus, we consider SGX as complementary to our solution. Our results establish that MPC alone is also a viable solution in the RS context.

Past studies utilized MPC to address privacy concerns in a variety of other networking-related problems [MK04b; CJV+11; RZ06; BSMD10]. Unlike these studies, our focus is on guaranteeing the privacy of peering policies. Additionally, the protocols proposed in these studies are either evaluated under questionable conditions, or exhibit runtimes in the order of seconds, whereas RSes are required to operate at faster runtimes.

We note that in dealing with the *privacy* of export policies, our work solves an orthogonal problem to that of securing BGP routing from IP-prefix hijacks and BGP path-manipulation, which is an active topic of research [BFMR10; GSG11; HRA11]. Finally, while higher visibility over routes has been proposed (e.g., BGP Add-Paths [WRCS16]), when deploying such techniques at the RS, no confidentiality about the export policies and IXP performance-related information is guaranteed.

6.4 Centralized BGP Route Computation

We consider two centralized algorithms for computing interdomain routes: an algorithm based on business relations (Sect. 6.4.1) and an algorithm that ranks neighbors based on preferences (Sect. 6.4.2). We first outline the pseudo-code for these algorithms, which can be con-

sidered as the "code of the trusted party" in terms of secure computation and then show how to reduce the complexity of the route computation by removing stub-nodes (Sect. 6.4.3).

6.4.1 Centralized Algorithm with Neighbor Relations

We present the algorithm from [GSG11] for computing the BGP routing tree for the routing policies described in Sect. 6.2.2. The algorithm gets as input the AS-topology $G = (V, E)$, where each outgoing edge $(u, v) \in E$ is associated with one of three labels: customer (v is a customer of u), peer (u and v are peers) or provider (v is a provider of u). The algorithm also receives as input the destination AS $\mathsf{dest} \in V$. The output of the algorithm is, for each AS, the next hop on the routing tree to destination dest. As shown in [GSG11], the induced routing tree generated by this algorithm agrees with the BGP outcome for the routing policies described in Sect. 6.2.2.

The algorithm computes for each AS its next hop on the routing tree using the following three-stage breadth-first search (BFS) on the AS graph:

1. **Customer routes.** A partial routing tree is constructed by performing a BFS "upwards" from the root node dest using only customer edges.
2. **Peer paths.** Next, single peering edges connect new ASes to the ASes already added to the partial routing tree from the first stage of the algorithm.
3. **Provider paths.** The computed partial routing tree is traversed with a BFS, and new ASes are iteratively added to the tree using provider edges.

We proceed with a detailed pseudo-code of the above algorithm. Implementing this algorithm involved several decisions that will mitigate the conversion to a secure protocol, and careful selection of data-structures. E.g., variables with Boolean values are preferred when possible, since these simplify the conversion to the Boolean circuit. Also, sometimes further optimizations of the algorithm (like breaking a loop according to some condition), are avoided as to not reveal information about the internal state. Note that all nodes are processed in parallel, i.e., the state is read once for all nodes and updated at the end of each iteration.

We distinguish between public and private algorithm inputs. We assume public inputs are global knowledge and do not reveal sensitive information. Private inputs describe the privacy-sensitive input of each AS.

A formal description of the algorithm is given in Algorithm 6.1. The state of the algorithm consists of three vectors, each of size $|V|$: next, fin and dist. The vector next stores nodes, where for every node $v \in V$, next$[v]$ stores the next hop in the routing tree to the node dest. The vector fin is a Boolean vector, where fin$[v]$ stores whether the route from v to dest is already determined. The vector dist is a vector of integers, where dist$[v]$ stores the number of hops in the current route between v and dest (this helps us to break ties between multiple routes with the highest local preference, if exist, in favor of shorter routes). It is easy to see that this pseudo-code is a concrete implementation of the algorithm presented in [GSG11],

Public inputs: $(V, \mathsf{dest}, d_{\mathsf{depth}})$, where $V = \{v_1, \ldots, v_n\}$ represents the set of vertices (ASes), $\mathsf{dest} \in V$ is the destination node, and d_{depth} is a bound on the depth of the customer-provider hierarchy, i.e., the longest route in the AS-graph in which each edge is from customer to provider. We use 10 as a very conservative upper bound on this depth. The topology of the AS-graph is assumed to be public knowledge. That is, for every $v \in V$ the list of its neighbors $\mathsf{Adj}[v] \subseteq V$ is public. We discuss hiding the topology in Sect. 6.7.4.

Private inputs: Every AS $v \in V$ inputs a private list type_v, where for every $u \in \mathsf{Adj}[v]$, $\mathsf{type}_v[u] \in \{\mathsf{customer}, \mathsf{peer}, \mathsf{provider}\}$.

Outputs: Upon completion, every AS $v \in V$ obtains its next hop in the routing tree $\mathsf{next}[v]$.

1: Initialize a vector next of size $|V|$, that stores the next hop in the routing tree to node dest, where for every $v \in V$ we set $\mathsf{next}[v] = \mathtt{DUMMY}$, where $\mathtt{DUMMY} \notin V$ is an unconnected node. Set $\mathsf{next}[\mathsf{dest}] = \mathsf{dest}$.
2: Initialize a Boolean vector fin of size $|V|$, that indicates if a route to dest was found, and set all the elements to false. Set $\mathsf{fin}[\mathsf{dest}] = \mathsf{true}$.
3: Initialize a distance vector dist of size $|V|$ that holds the number of hops to dest. Set all elements to ∞, except for $\mathsf{dist}[\mathsf{dest}] = 0$.

// BFS of customers:

4: **for** d_{depth} iterations:
5: **for all** $v \in V$ **do**: ▷ in parallel
6: **for all** $u \in \mathsf{Adj}[v]$ **do**: ▷ for all neighbors of v
7: **if** $\mathsf{fin}[u] = \mathsf{true}$ **and** $\mathsf{fin}[v] = \mathsf{false}$ **and** $\mathsf{type}_v[u] = \mathsf{customer}$ **then**
8: $\mathsf{next}[v] \longleftarrow u$
9: $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
10: $\mathsf{fin}[v] \longleftarrow \mathsf{true}$

// BFS of peers:

11: **for all** $v \in V$ **do**: ▷ in parallel
12: **for all** $u \in \mathsf{Adj}[v]$ **do**: ▷ for all neighbors of v
13: **if** $\mathsf{fin}[u] = \mathsf{true}$ **and** $\mathsf{fin}[v] = \mathsf{false}$ **and** $\mathsf{type}_v[u] = \mathsf{peer}$ **and** $\mathsf{dist}[v] > \mathsf{dist}[u] + 1$ **then**
14: $\mathsf{next}[v] \longleftarrow u$
15: $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
16: $\mathsf{fin}[v] \longleftarrow \mathsf{true}$

// BFS of providers:

17: **for** d_{depth} iterations:
18: **for all** $v \in V$ **do**: ▷ in parallel
19: **for all** $u \in \mathsf{Adj}[v]$ **do**: ▷ for all neighbors of v
20: **if** $\mathsf{fin}[u] = \mathsf{true}$ **and** $\mathsf{fin}[v] = \mathsf{false}$ **and**
$\mathsf{type}_v[u] = \mathsf{provider}$ **and** $\mathsf{dist}[v] > \mathsf{dist}[u] + 1$ **then**
21: $\mathsf{next}[v] \longleftarrow u$
22: $\mathsf{dist}[v] \longleftarrow \mathsf{dist}[u] + 1$
23: $\mathsf{fin}[v] \longleftarrow \mathsf{true}$
24: **return** next

Algorithm 6.1: Neighbor Relation Routing [GSG11]

Public inputs: Same as in Algorithm 6.1.

Private inputs: Every AS $v \in V$ inputs a private list of preferences pref_v, where for every $u \in \mathsf{Adj}[v]$, $\mathsf{pref}_v[u]$ corresponds to the preference for u, and a private bit-matrix pub_v of size $|\mathsf{Adj}[v]+1| \times |\mathsf{Adj}[v]|$ that specifies the export policy, i.e., if a route to a neighbor is published to other neighbors.

Outputs: Same as in Algorithm 6.1.

1: Initialize a vector next of size $|V|$, that stores the next hop in the routing tree to node dest. For every $v \in V$ set $\mathsf{next}[v] = \mathtt{DUMMY}$, where $\mathtt{DUMMY} \notin V$ is an unconnected node. Set $\mathsf{next}[\mathsf{dest}] = \mathsf{dest}$.

2: Initialize a Boolean vector fin of size $|V|$, that indicates if a route to dest was found, and set all the elements to false. Set $\mathsf{fin}[\mathsf{dest}] = \mathsf{true}$.

3: **for all** $v \in V$ **do**: Initialize $\mathsf{pub}_v[\mathtt{DUMMY}, u] = \mathsf{true}$ for all $u \in \mathsf{Adj}[v]$ and $\mathsf{pref}_v[\mathtt{DUMMY}] = 0$.

// BFS for all ASes:

4: **for** $2d_{\mathsf{depth}} + 1$ iterations **do**:

5: **for all** $v \in V$ **do**: ▷ in parallel

6: **for all** $u \in \mathsf{Adj}[v]$ **do**: ▷ for all neighbors of v

7: **if** $\mathsf{fin}[u] = \mathsf{true}$ **and** $\mathsf{pub}_u[\mathsf{next}[u], v] = \mathsf{true}$ **and** $\mathsf{pref}_v[\mathsf{next}[v]] < \mathsf{pref}_v[u]$ **then**

8: $\mathsf{next}[v] \longleftarrow u$

9: $\mathsf{fin}[v] \longleftarrow \mathsf{true}$

10: **return** next

Algorithm 6.2: Neighbor Preference Routing [GSP+12]

and thus we conclude that the routes computed by this algorithm agree with the outcome of BGP (where the preferences of the ASes are according to Sect. 6.2.2).

6.4.2 Centralized Algorithm with Neighbor Preferences

The algorithm in Sect. 6.4.1 can be extended such that it allows the ASes to specify preferences for each neighbor route and freely choose an individual export policy. In [GSP+12], such an algorithm was proposed, that behaves similar to the one in [GSG11], but is computationally more complex due to the added degree of freedom.

We can emulate the behavior of Algorithm 6.1 by grouping each neighbor relation to a certain range of preferences: we ensure that customers have a higher preference than all other nodes, and that providers have lower preference than others. The advantage of this algorithm is that within each neighbor relation we can have a preferred node, e.g., a favorite provider. In addition, this algorithm allows a node to freely specify his export policy, i.e., to choose whether he wants to disclose a certain route to a neighbor or not. The pseudo-code of the algorithm is given in Algorithm 6.2. We use a preference bit-length of $\ell = 4$ for good expressiveness in practice.

6.4.3 Removing Stub ASes

To reduce the complexity of the route-computation, our protocols are only run on the subgraph of the AS-graph that is induced by the non-stub ASes (i.e., by the ISPs). Stubs (by definition) have no customers, and so should never transit traffic between other ASes. Hence, in our scheme MPC is used to compute routes between ISPs (that form the core of the Internet). Then, stubs select an ISP through which to connect to the Internet according to their local routing policies. We point out that:

1. As stubs are roughly 85% of the ASes, this means that the MPC protocol needs only be run on a fairly small part of the AS graph. In our experimental evaluation in Sect. 6.9.1 we show that removing the stubs improves the runtime of the MPC protocol by a factor of ≈ 2.5. For the CAIDA topology from November 2016, the number of ASes is reduced from almost 56 000 to 8 407 ASes, when excluding stubs.
2. Whether an AS is a stub (and not an ISP) is not considered confidential information and so our partition of the AS graph into these two distinct groups of ASes does not leak any sensitive information.
3. Observe that according to the routing policies presented in Sect. 6.2.2, to select between ISPs after the MPC step is complete, a stub needs only to know whether or not the ISP has a route to the destination, and the length of the route. This information can be announced directly to the stub by its ISP.

6.4.4 Boolean Circuits for Route Computation

As a first step towards securely computing Algorithm 6.1 and Algorithm 6.2, we show how to construct Boolean circuits that implement these algorithms. In the following section we detail how to construct a Boolean circuit from the neighbor relation algorithm given in Algorithm 6.1, describe the optimizations that we apply to it (Sect. 6.8), and give a summary of the circuit for the neighbor preference algorithm in Algorithm 6.2. We emphasize that our circuits could be evaluated also with other MPC protocols (e.g., Yao's garbled circuits [Yao86]) or protocols that provide security against stronger adversaries (e.g., [NNOB12]) and/or use more than two computational parties (of which a fraction can be corrupted). However, these protocols have significantly higher communication and/or computation complexities.

'Naive' Implementation of Algorithm 6.1

We first outline the global structure of the circuit, depicted in Fig. 6.6, and then show how to implement sub-routines and analyze their complexities. We provide the complete circuit of our centralized BGP algorithm in Circ. 6.1.

1: Initialization of next, fin and dist identical to Algorithm 6.1.
// BFS of customers:
2: **for** d_{depth} times **do**:
3: **for all** $v \in V$ **do**:
4: **for all** $u \in \text{Adj}[v]$ **do**:
5: $\text{sel} \longleftarrow \text{fin}[u] \wedge !\text{fin}[v] \wedge \text{in}_v^{\text{cust}}[u]$
6: $\text{sum} \longleftarrow \text{ADD}(\text{dist}[u], 1)$
7: $\text{next}[v] \longleftarrow \text{MUX}(\text{next}[v], u, \text{sel})$
8: $\text{dist}[v] \longleftarrow \text{MUX}(\text{dist}[v], \text{sum}, \text{sel})$
9: $\text{fin}[v] \longleftarrow \text{MUX}(\text{fin}[v], \text{sel}, \text{sel})$
// BFS of peers:
10: **for all** $v \in V$ **do**:
11: **for all** $u \in \text{Adj}[v]$ **do**:
12: $\text{sum} \longleftarrow \text{ADD}(\text{dist}[u], 1)$
13: $\text{cmp} \longleftarrow \text{GT}(\text{dist}[v], \text{sum})$
14: $\text{sel} \longleftarrow \text{fin}[u] \wedge !\text{fin}[v] \wedge \text{cmp} \wedge \text{in}_v^{\text{peer}}[u]$
15: $\text{next}[v] \longleftarrow \text{MUX}(\text{next}[v], u, \text{sel})$
16: $\text{dist}[v] \longleftarrow \text{MUX}(\text{dist}[v], \text{sum}, \text{sel})$
17: $\text{fin}[v] \longleftarrow \text{MUX}(\text{fin}[v], \text{sel}, \text{sel})$
// BFS of providers:
18: **for** d_{depth} times **do**:
19: **for all** $v \in V$ **do**:
20: **for all** $u \in \text{Adj}[v]$ **do**:
21: $\text{sum} \longleftarrow \text{ADD}(\text{dist}[u], 1)$
22: $\text{cmp} \longleftarrow \text{GT}(\text{dist}[v], \text{sum})$
23: $\text{sel} \longleftarrow \text{fin}[u] \wedge !\text{fin}[v] \wedge \text{cmp} \wedge \text{in}_v^{\text{prov}}[u]$
24: $\text{next}[v] \longleftarrow \text{MUX}(\text{next}[v], u, \text{sel})$
25: $\text{dist}[v] \longleftarrow \text{MUX}(\text{dist}[v], \text{sum}, \text{sel})$
26: $\text{fin}[v] \longleftarrow \text{MUX}(\text{fin}[v], \text{sel}, \text{sel})$

Circuit 6.1: Neighbor Relation Routing Circuit

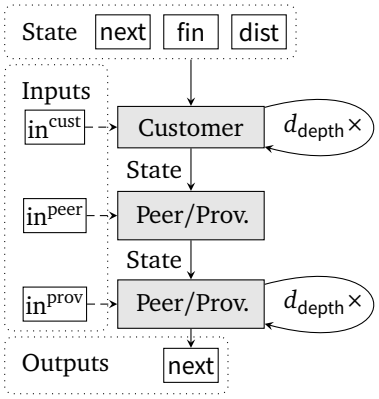


Figure 6.6: Circuit Structure Overview.

Inputs The circuit gets as input from each AS the secret shared relationship information for its neighbors. For efficiency reasons we have three separate input bit arrays for each AS, one for each type of AS relation: in^{cust}, in^{peer}, and in^{prov}. For a node with n neighbors, we have three inputs of length n bits each, where the i-th bit corresponds to the relation to the i-th neighbor. Note that the ASes only need to secret share their inputs among the computational nodes when updating their relationships since the computational nodes can re-use existing data for multiple executions. (If it should be hidden that a particular AS changed its relationships, then the secret sharing can be run again by all ASes. This might happen on a regular basis.) The routing destination dest and network topology are public inputs and thus not secret-shared.

State We operate on a secret-shared state where we store for each node: a *finish* bit fin, a δ-bit long *destination id* of the next node on the routing tree next, and a σ-bit long *hop distance* to the target node dist. Initially, fin is set to false and next is set to zero, while dist is set to the maximum value $2^{\sigma}-1$. This state is then iteratively updated using the methods for each relation. The *peer* and *provider* methods are identical, except for the different iterations and type of AS relation. We have d_{depth} iterations of the customer sub-circuit and $1+d_{\text{depth}}$ iterations of a combined peer/provider sub-circuit, where we use the peer relation as input once, and the provider relation for the remaining iterations.

Outputs After secure evaluation of the next hop on the route to dest for each AS $v \in V$, the computational parties send their share $\text{next}[v]$ to every v, who can then reconstruct the plaintext output.

Parameters According to the CAIDA dataset, we set the parameters in the protocol of Sect. 6.4 to $d_{\text{depth}} = 10$ and therefore have 10 iterations of the customer routine, a single iteration for the peer routine, and 10 iterations for the provider routine. Furthermore, we set the id-bit length $\delta = \lceil \log_2 |V| \rceil$ and the destination bit-length to $\sigma = \lceil \log_2 21 \rceil = 5$, since we can at most achieve a distance of 21 hops between any AS and the destination (1 hop per each of the 10 customer and provider iterations and one peer iteration).

Operations and Complexities The operations of the pseudo-code in Circ. 6.1 are implemented using standard circuit constructions. Apart from the standard bit-wise operations AND ($\wedge$) and NOT (!), we use the following operations:

Addition (ADD) We use the Ripple-Carry addition circuit [BPP00] with ℓ AND gates and a multiplicative depth of ℓ for addition of ℓ-bit values. While the Ladner-Fischer adder is optimized for depth, it has a much higher total number of $5/4\ell\lceil\log_2 \ell\rceil+\ell$ AND gates for ℓ-bit inputs and would thus go beyond the boundaries of our implementation.

Greater-Than (GT) We use the greater-than circuit of [KSS09] which has ℓ AND gates and a multiplicative depth of ℓ when comparing two ℓ-bit values.

`if`-Condition (MUX) To compute the `if`-condition securely, both branches must be evaluated to hide which branch has been chosen. The results are then assigned to the variables depending on the condition bit $\mathsf{sel} \in \{0,1\}$ using a multiplexer MUX. A multiplexer for ℓ-bit values requires ℓ AND gates [KS08b] and its multiplicative depth is 1. There exist optimizations for the GMW protocol that allow the evaluation of an ℓ-bit multiplexer at the cost comparable to a single AND gate [DSZ15], as explained in Sect. 6.8.

We estimate the number of AND gates and the depth of the primitive operations for the customer, peer, and provider functionality and provide the resulting total numbers for the CAIDA dataset of November 2016 in Tab. 6.4. We estimate that naively implemented circuits would have more than one hundred million AND gates, and a multiplicative depth of several hundred thousand. Furthermore, we estimate that the *total* number of gates is around 400 million. Since modern secure computation frameworks are able to evaluate 4-8 million AND gates per second [DSZ15; LWN+15], securely evaluating the circuit would require nearly 100 s, which is arguably too long. Finally, the high depth translates to a huge number of communication rounds for the GMW protocol. In the next section, we show how the circuit can be optimized substantially to overcome these limitations.

6.5 SIXPACK Privacy-Preserving Route Server

In this section, we introduce SIXPACK, a privacy-preserving RS service for IXPs. SIXPACK combines the benefits of a centralized route dispatch service with the provable guarantee of privacy preservation. Through SIXPACK, IXP members can receive the best available BGP routes according to arbitrary local route preferences and auxiliary information of the IXP (e.g., knowledge of congestion level and other performance metrics [KRG+16]).

Building on recent advances in MPC, SIXPACK employs two independent and non-colluding computational entities in the MPC outsourcing scenario (cf. Sect. 2.4.2) to correctly dispatch route announcements without gaining any visibility into the members' routing policies (i.e., export/import policies) nor leaking any private IXP performance-related information to the members.

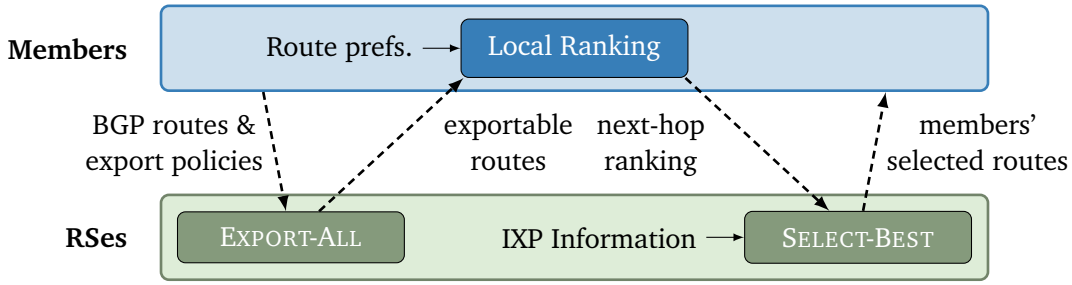


Figure 6.7: SIXPACK's 3-step route dispatching process.

To illustrate the non-trivial challenges facing SIXPACK's design, we first discuss a fairly naïve approach of applying MPC to route-dispatch at IXPs, and why this approach fails. We then describe the key design ideas behind SIXPACK, present the routing policy model, and discuss in detail the two main components of the system.

6.5.1 A Naïve Approach

As general-purpose MPC is capable of performing arbitrary computation, it would be tempting to implement SIXPACK solely within a single MPC, thus providing arbitrary privacy-preserving policy expressiveness. This task entails devising a function that takes as input the set of members' export policies, the set of arbitrarily complex members' import policies (e.g., regular expressions on the AS networks traversed by a route), and the IXP's performance-related information (e.g., port utilization), and outputs the resulting "best" BGP routes.

However, even performing a single full regular expression string matching operation in MPC using state-of-the-art implementations is overly prohibitive in practice [Kel15, §5] as this operation is shown to require runtimes in the order of *minutes*. Even by restricting the members to use a single regular expression in their import policy (a fairly restrictive assumption), evaluating the import policies in a large IXP with 500 members would take days! In contrast, SIXPACK computes and dispatches routes in the order of tens of milliseconds, improving upon the naïve approach by 5 orders of magnitude. This is made possible through a combination of several ingredients, as discussed below.

6.5.2 SIXPACK Design

To achieve practical runtimes, SIXPACK is carefully designed to keep complex computation to the largest extent possible outside the MPC, without compromising privacy (see Fig. 6.7). Specifically, the route dispatch computation is split into three operations to be performed sequentially, called EXPORT-ALL, LOCAL-RANKING, and SELECT-BEST. Both EXPORT-ALL and SELECT-BEST are MPC-based components of the system that are executed within the RS by the two non-colluding entities (depicted as a single green-colored box). The LOCAL-RANKING component, in contrast, is locally executed by each member (depicted as a single blue-colored

box). Next, we describe the SIXPACK pipeline for processing BGP route announcements. We observe that BGP withdrawal messages can be handled in a similar way. For readability, we assume members to connect with a single BGP router.

Step I: Exporting All Permissible Routes SIXPACK processes streams of BGP announcements generated by the IXP members. Through EXPORT-ALL, SIXPACK takes as input a BGP route destined to a prefix π and its associated export policy and outputs the route to the IXP members authorized to see that route. This operation is performed in MPC, and so neither the route nor the export policy is disclosed to any unintended entity. The route and its export policy are both stored in encrypted form within the RS. We discuss EXPORT-ALL in detail later in Sect. 6.5.4.

Step II: Ranking Routes Based on Local Preferences At this point, each member that received the new route executes LOCAL-RANKING to rank all its available routes towards π according to its (arbitrarily complex) local import policies. A key idea embedded into SIXPACK's design is performing this computationally-heavy operation outside of the MPC, i.e., at the member-side. Now, recall that in BGP, each IXP member only announces at most one single route towards π. Thus, a ranking of the routes destined to π corresponds to a ranking of the IXP members, where a member assigns the lowest preference to those IXP members from whom it did not receive a route to π. We call such ranking the *next-hop* ranking, and this is the output of Step II.

Step III: Incorporating IXP Information Into Route Selection When multiple routes for a certain prefix are available, members submit the next-hop ranking received to the RS service. Upon receiving a next-hop ranking from a member, the RS proceeds to run the MPC-based SELECT-BEST component for dispatching the best-selected route to that member. The best route is computed based on the next-hop ranking and, importantly, the performance-related information available to the IXP (e.g., port utilization). This operation is performed in MPC, and so neither the members' rankings nor the IXP performance-related information is revealed to any unintended entity. We discuss the SELECT-BEST in detail later in Sect. 6.5.5.

Benefits of our Design Approach Through EXPORT-ALL, an IXP member gains full visibility of the available routes and can possibly select the best one according to *any arbitrary* local import policy (e.g., next-hop preferences, shortest route, avoid specific AS networks). Then, by taking part in SELECT-BEST, the IXP member can incorporate IXP information into its route selection process. By implementing EXPORT-ALL and SELECT-BEST via carefully optimized MPC and keeping LOCAL-RANKING's potentially highly complex computation outside of the MPC framework, this pipeline preserves the member's and the IXP's privacy, and is efficiently executable. Observe that, since BGP treats each IP prefix destination independently, multiple instances of the two RSes can be easily instantiated to dispatch routes in parallel, thus enhancing the system throughput. However, multiple route announcements towards the same IP prefix have to be processed sequentially.

Peering at Multiple Sites/IXPs We have so far assumed that each organization has a single point of presence at an IXP. In practice, organizations may connect at different physical locations at the same IXP with the same goal of further reducing latencies. In SIXPACK, each connection to the same IXP from the same member is treated as an independent member. This allows operators to arbitrarily export/rank/filter BGP routes at those locations independently.

Enhancing Security with RPKI To achieve RPKI for route validation, members could reveal the IP prefix of a route and its originator so as to allow the IXP to validate that information. Alternatively, RPKI validation can be implemented within the MPC framework as it only involves a simple lookup operation on a dictionary.

Before we get into the details of the two MPC-based components of SIXPACK, we first formalize our model of export policies (and next-hop rankings) in Sect. 6.5.3. We then describe EXPORT-ALL in Sect. 6.5.4 and SELECT-BEST in Sect. 6.5.5. We discuss the schemes' security and privacy guarantees are given in Sect. 6.6.

6.5.3 Routing Policies Model

Export-Policy The EXPORT-ALL component of SIXPACK dispatches routes according to the export policies pertaining to the routes that it received from its members. Each route carries its own export policy specification, i.e., the set of members to whom that route can be exported. Since BGP computes routes independently for each destination IP prefix, w.l.o.g., we henceforth assume throughout this section that there exists only a single destination IP prefix π. Moreover, the BGP route computation only depends on the last route announced by a neighbor. Hence, our model only needs to store the set of routes *currently* available at the RS and the *currently* specified export policies. To achieve efficient MPC computation, we model BGP export policies as follows. Let $M = \{m_1, \ldots, m_{|M|}\}$ be the set of IXP members and $R = \{r_1, \ldots, r_{|R|}\}$ be the set of available routes. We define the *export policy matrix* P, with $|M|$ rows and $|R|$ columns. Entry $P_{i,j}$ in the matrix, for $1 \leq i \leq |M|$ and $1 \leq j \leq |R|$, is 1 if route r_j is exportable to member m_i, and 0 otherwise.

An example export policy matrix is shown in Fig. 6.8(a) where m_A, m_B, m_C, m_D are IXP members and r_A, r_B are routes announced by m_A and m_B, respectively. While route r_A is exported to m_C only, route r_B is exported to m_C and m_D. Observe that r_A and r_B are not exported to m_A and m_B, respectively, i.e., to the member they originate from.

In EXPORT-ALL, all permissible routes are exported. Each IXP member m_i should receive each route r_j for which $P_{i,j} = 1$. In Fig. 6.8(a), m_C receives both r_A and r_B, and m_D receives only r_B, while m_A and m_B do not receive any route. In the SELECT-BEST component, each member m_i could receive any route r_j with $P_{i,j} = 1$. The actual route that will be received depends on the ranking and the IXP's performance information.

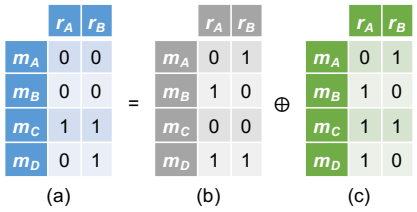


Figure 6.8: (a) Export policy matrix in plain text. (b) Random shares received by RS1. (c) Element-wise XOR of (a) and (b) received by RS2.

Next-Hop Ranking (a.k.a. Local Preferences over Routes) SELECT-BEST receives as input, from each participating IXP member, its next-hop ranking with respect to the destination IP prefix. Each next-hop corresponds to a route announced by a member, thus next-hop rankings model *local preferences* over the received routes. For each IP prefix π, we model preferences over the available routes at the RS as a matrix Ψ with size $|M| \times |M|$. Each element $\psi_{i,j}$ of that matrix represents member m_i's local preference (value) for routes announced by m_j, where routes announced by members with higher local preference are preferred over routes announced by members with lower local preference. Using SELECT-BEST, each IXP member m_i thus receives a single route r announced by m_j such that $\psi_{i,j}$ is the highest priority value in row i of Ψ, for which $P_{i,j} = 1$ holds. Ties are broken deterministically.

The matrix Ψ can easily be extended to represent preferences over routes based on a combination of members' local preferences and IXP performance-related recommendations. For instance, if each preference value ψ is encoded as an $\rho = 8$-bit integer, we can use the four most significant bits of ψ to create 16 different classes of members' local preferences over routes and use the four least significant bits to create another 16 additional classes for the IXP performance-related recommendations. In this way, the IXP information is used only to break ties among routes with the same rank. Alternatively, IXP information can be given higher priority and members' local preferences can be used to break ties.

6.5.4 The Export-All Component

Through EXPORT-ALL, the RSes export to each member all permissible (i.e., exportable) routes, while keeping each member's export policy private. We note that this problem could also be solved by using public-key cryptography if we assume that each sending member knows the public keys of all other IXP members. However, a public-key solution alone cannot incorporate IXP performance-related information without revealing it — a concern for 60% of the surveyed operators. Furthermore, MPC circumvents all key management challenges, protects against side-channel attacks, and easily integrates with the SELECT-BEST MPC component.

Observe that, for EXPORT-ALL, not only is the computation *per-prefix independent*, i.e., the computation is executed independently for each destination IP prefix, but it is *per-route independent*, in the sense that the announcement of a specific route to a member does not

depend on what other routes to the same prefix are announced to that member. Hence, w.l.o.g., we describe EXPORT-ALL with respect to a single route to a single prefix π.

Fig. 6.9 illustrates an example of the EXPORT-ALL computation. Two independent RSes, P_0 and P_1 (center of the figure), perform the redistribution of a route from m_A according to its export policy. We consider the scenario presented in Fig. 6.8(a). The computation operates over a policy that is kept private using MPC and the route is dispatched in such a way that neither the RSes nor the IXP members can distinguish whether a route is announced to any other member or not. Each member attempts to decrypt the information received from the RSes (right side of the figure). This operation succeeds iff the route is actually exported to that member, which then learns the route. We now discuss in more detail the different parts of EXPORT-ALL.

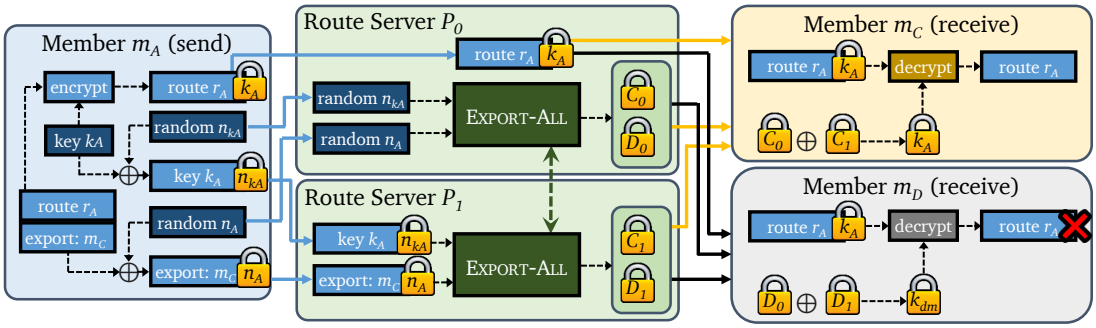


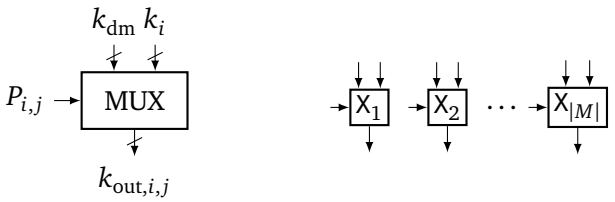
Figure 6.9: The EXPORT-ALL component. The RSes export to IXP members all permissible routes.

Encrypted Routes Each route that needs to be distributed is first encrypted. W.l.o.g., we assume that member m_A wants to send a route r_A as shown in Fig. 6.9. Then, m_A encrypts r_A using a route-specific key k_A and a symmetric encryption scheme (we use AES) and sends the encrypted route to P_0, which, in turn, redistributes it to all the members. Receiving members can decrypt r_A only if they possess the route key k_A. SIXPACK guarantees that an IXP member receives the key k_A only if the route can be exported to it, and that routes are not revealed to the IXP. We use a "dummy key" k_{dm} to notify a member when a route cannot be exported to it. Recall that a receiving member does not have visibility of the routes in plain text, so it does not know which routes are announced. Even if a member colludes with one of the two RS entities, it cannot distinguish whether a certain route is exported to any of the other IXP members.

Alternatively, we could make plaintext routes available for the IXP and move the encryption step to the RSes. This mighe be viable since information about the route itself is far less sensitive than peering information, local preferences and export policies. More specifically, the sending members could simply announce their routes (with sensitive information stripped off) to the IXP, who selects random symmetric keys and encrypts these routes before it broadcasts them to the receiving members. The remainder of the protocol remains the same. This would

reduce the complexity for sending members since they don't have to carry out symmetric encryption operations, even though they are very cheap, especially due to the wide-spread deployment of AES-NI acceleration.

Exporting Keys via MPC We now leverage MPC to dispatch the key k_A in a privacy-preserving manner. Specifically, we devise a tailored MPC circuit (shown in Circuit 6.2) that is jointly executed via MPC by the two RSes P_0 and P_1. The EXPORT-ALL circuit consists of one multiplexer X_i per member m_i, which outputs either the valid or dummy key k_{dm} depending on the export policy entry $P_{i,j}$ (cf. Sect. 6.5.3), where i (j) is the member announcing (receiving) a route. Since we process the inputs in a SIMD fashion, the routes for all receiving members are processed in parallel MUX blocks. If only an incremental update to a single route is processed, the circuit consists only of a single MUX.



(1) Circuit Building Block X_j **(2)** Circuit Structure

Circuit 6.2: The EXPORT-ALL circuit is a set of $|M|$ multiplexers, each of which outputs either the valid or dummy key depending on entry $P_{i,j}$, where i (j) is the member announcing (receiving) a route. Since we process the inputs in a SIMD fashion, the routes for all receiving members are processed in parallel. If only an incremental update to a single route is processed, the circuit consists only of a single MUX.

To generate the MPC input (left of Fig. 6.9), member m_A secret-shares r_A's export policy and the corresponding key $k_A$with P_0 and P_1. Assuming non-collusion between P_0 and P_1, they are able to decrypt neither the export policy nor the key. We show in Fig. 6.8 on page 100 an example of (a) the export policies of two routes r_A and r_B, (b) the random values chosen by m_A and m_B, respectively, and (c) the resulting inputs to P_1.

Once the two RSes receive their shares of the export policy and key k_A, P_0 and P_1 (center of Fig. 6.9) output to every member m_X two shares X_0 and X_1 of the output, respectively. XORing X_0 with X_1 (right of Fig. 6.9) produces a key that can be used to decrypt the encrypted route r_A if and only if $P_{X,A} = 1$, (i.e., route r_A can be exported to member m_A), or the dummy key k_{dm}, otherwise. In Fig. 6.9, m_C receives C_0 and C_1; when XORed, they give k_A. Similarly, m_D receives D_0 and D_1; when XORed, they give k_{dm}, leading m_D to discard the encrypted route r_A.

We formally describe the EXPORT-ALL protocol in Prot. 6.1.

The EXPORT-ALL Protocol

Sending Member	P_0	P_1	**Receiving Member** j
	Secret-share export policies		
$n \in_R \{0,1\}^{\lvert ep \rvert}$			
$n' \leftarrow n \oplus ep$			
$\xrightarrow{n}$			
$\xrightarrow{n'}$			
	Broadcast encrypted routes		
$\xrightarrow{R}$			
	$\mathsf{k} \in_R \{0,1\}^{\lvert \mathsf{k} \rvert}$		
	$R' \leftarrow \mathsf{Enc}(\mathsf{k}, R)$		
	broadcast R'		
	$\xrightarrow{R'}$		
			store R'
	MPC execution		
	run MPC on(k, n)	run MPC on$(\{0\}^{\lvert \mathsf{k} \rvert}, n')$	
	MPC execution $\longleftrightarrow$	MPC execution	
	$\mathsf{k}_0 \leftarrow \mathrm{MPC}(\mathsf{k}, n)$	$\mathsf{k}_1 \leftarrow \mathrm{MPC}(\{0\}^{\lvert \mathsf{k} \rvert}, n')$	
	Send resulting key share		
	$\xrightarrow{\mathsf{k}_0[j]}$		
			store $\mathsf{k}_0[j]$
	$\xrightarrow{\mathsf{k}_1[j]}$		
			store $\mathsf{k}_1[j]$
	Member receives MPC output		
			$\mathsf{k}' \leftarrow \mathsf{k}_0[j] \oplus \mathsf{k}_1[j]$
			if $\mathsf{k}' \neq \mathsf{k}_{dummy}$ **then**
			$R \leftarrow \mathsf{Dec}(\mathsf{k}', R')$
			endif

Protocol 6.1: The EXPORT-ALL Protocol. Inputs: R: a BGP route stripped off its export policy ep; $|ep|$: the length in bits of ep; $|\mathsf{k}|$: the bit length of key k.

6.5.5 The Select-Best Component

To leverage the superior IXP's visibility into dataplane conditions, we design SELECT-BEST, a privacy-preserving component that allows IXP members to select the best permissible route according to both *their own* local preferences and the *IXP* performance-related information.

To execute SELECT-BEST, each IXP member will first translate its ranking of available routes (e.g., prefer shortest routes) to a ranking of the corresponding IXP members that announced them, where members that are neither exporting nor announcing a route to this member are assigned the lowest preference. Analogously, the IXP translates its sensitive performance-related information, such as members' port utilization, to preference values. As explained in Sect. 6.5.3, these preferences values can be combined into a single preference value ψ per route, where we envision the members' local preference to be given precedence over the IXP's inputs. This information is provided as input to SELECT-BEST, which privately computes the best route. We note that this functionality *cannot* be realized with public-key cryptography alone and is thus an interesting and practical real-world application of MPC.

In Fig. 6.10 we provide an example of the SELECT-BEST computation, where we consider the export policy scenario of Fig. 6.8(a). Namely, two members m_A and m_B announced two routes r_A and r_B, respectively. Through EXPORT-ALL, m_C received both r_A and r_B while m_D only received r_B, with the latter deciding not to execute SELECT-BEST. Based on port utilization levels and assuming m_C equally ranks r_A and r_B, the IXP gives route r_B a higher preference.

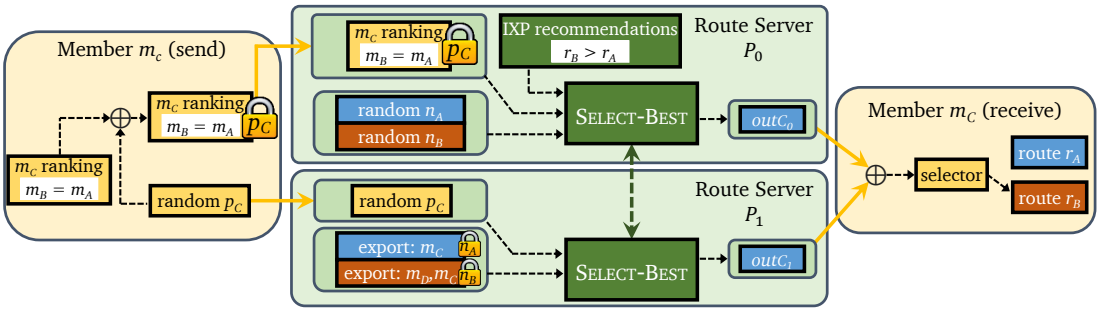
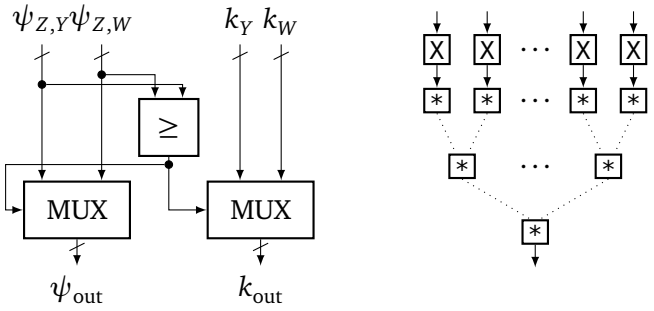


Figure 6.10: The SELECT-BEST component. The RSes export to each IXP member the best preferred permissible route.

Choosing the Best Route via MPC We now leverage MPC to select the best route of each IXP member in a privacy-preserving manner. To this end, we devise a tailored MPC circuit (shown in Circuit 6.3) that, for each member m, takes as input the next-hop ranking (i.e., preferences over members) and the IXP route recommendations, and outputs to m the identifier of the best route.

The first step (left of Circuit 2) is similar to the EXPORT-ALL circuit: based on the export policy, the preference of all non-exportable routes is set to zero. After that (right of Circuit 2), we feed the resulting priorities as well as the route keys (which are used as identifiers) into a *MaxIdx* tree circuit [KSS09, Sect. 3.3], that determines the index of the maximum in a given

list of values. With it, we determine the best route, i.e., output the route key with the highest preference. We also input the dummy key k_{dm}, which is returned if the receiving member does not have any permissible route. The comparison among two routes r_Y and r_W (announced by members m_Y and m_W, resp.) is performed by the $*$ circuit (Circuit 1), where $\psi_{Z,Y}$ and $\psi_{Z,W}$ are the m_Z's preferences over r_Y and r_W, resp., while k_Y and k_W are the keys of routes r_Y and r_W, resp. The selection bit of the MUX is chosen such that the route key with the higher preference value gets propagated to the next level of the tree. The multiplicative depth of the SELECT-BEST circuit, for a priority value of ℓ bit, is $\lceil \log_2(\#\text{Routes}) \rceil \cdot (\lceil \log_2(\ell) \rceil + 2) + 1$.



(1) Recursive Circuit Building Block $*$

(2) Circuit Tree Structure

Circuit 6.3: The SELECT-BEST circuit for exporting the single highest ranked route key to a member m_Z. Each $*$ circuit compares 2 routes r_Y and r_W announced by m_Y and m_W, resp., where $\psi_{Z,Y}$ and $\psi_{Z,W}$ are the m_Z's preferences over r_Y and r_W, resp., while k_Y and k_W are the keys of routes r_Y and R_W, resp.

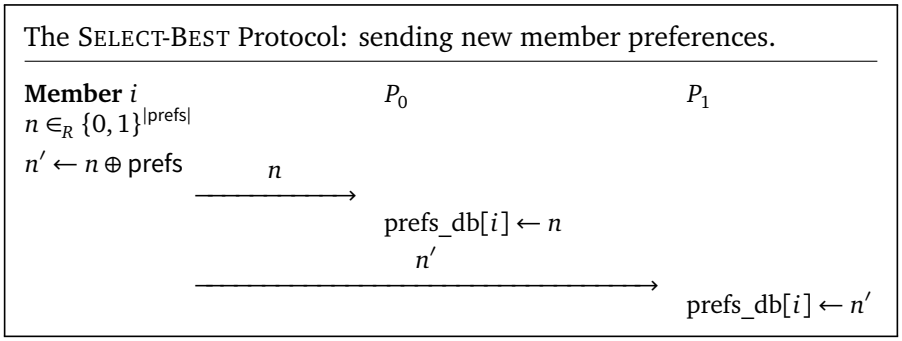
Both input and output are considered private information, and are not visible to the RS in clear. The IXP members are responsible for generating and then reconstructing the MPC's input and output through secret sharing.

The input to the MPC (i.e., the next-hop ranking) is generated similarly to the input of the EXPORT-ALL component. In the example (left side of Fig. 6.10), m_C generates a $|M| \cdot \rho$-bits random mask p_C that is XORed with its next-hop ranking (i.e., row ψ_C of the ranking matrix Ψ, Sect. 6.5.3). The XORed result is sent to P_1, while p_C is sent to P_0. Neither P_0 nor P_1 are able to decrypt the ranking as they are assumed to not collude.

At this point, the two RSes combine the member's preferences with the IXP preferences, where only P_0, which runs at the IXP supplies the preferences based on performance information; P_1 uses a vector of zeros. The two RSes execute the SELECT-BEST circuit on the given inputs (center of Fig. 6.10). Then, m_X receives two values out_{X1} and out_{X2}; when XORed, they produce an identifier of the best route. In Fig. 6.10, m_C receives r_B as the best route based on the IXP performance-based preference. Observe that if m_C had preferred r_A over r_B, it would have received an identifier to r_A.

As for the other circuit, it is worth noting that our design of the SELECT-BEST circuit can process multiple next-hop rankings at once from different members for improved efficiency. In fact, SELECT-BEST takes as input a ranking matrix Ψ (Sect. 6.5.3), which may consist of just one row as a special case.

We detail how new local preferences are shared in Prot. 6.2 and provide a formal description of the SELECT-BEST protocol in Prot. 6.3.



Protocol 6.2: Secret sharing of the member preferences prefs in the SELECT-BEST protocol.

Route Oscillations

As with any algorithm that dynamically adapts the routing paths based on network performance, there exists a risk of not converging to a stable routing configuration. For instance, when an IXP ranks routes based on congestion levels, traffic might move back and forth between IXP ports as these become more or less congested and so are assigned new priorities. We propose addressing this in SELECT-BEST by ensuring that when a member is offered a new route this is either because (1) its old route was withdrawn, or (2) a new route, ranked higher by *the member's local preference*, appears. We mention next two ways of accomplishing this: one on the member side and one on the RS side.

Importantly, in both schemes, members' local route preferences should be given higher preference than the IXP's preference. When a member receives its best route from the MPC, it could temporarily (slightly) increase the priority of that route to ensure its selection in successive iterations, unless a better route is available, and communicate its new preferences to the RSes. Alternatively, this could also be realized within the MPC itself, thus minimizing communication between members and RSes, though at the price of higher circuit complexity.

6.6 Security and Privacy

The overall security and privacy of our MPC-based route computation (Sect. 6.4) and SIX-PACK (Sect. 6.5) stems from the proven security and privacy of the GMW protocol (cf. [GMW87; Gol04] for proofs).

The SELECT-BEST Protocol.

Sending Member	P_0	P_1	**Receiving Member** j
$n \in_R \{0,1\}^{\lvert ep \rvert}$			
$n' \leftarrow n \oplus ep$			
$\xrightarrow{R,n}$			
	$i \leftarrow ind[R.dst][R.mem]$		
	if $i == null$		
	$\quad i \leftarrow len(ind[R.dst]) + 1$		
	$\quad ind[R.dst][R.mem] \leftarrow i$		
	endif		
	$ep[R.dst][i] \leftarrow n$		
	$\mathsf{k} \in_R \{0,1\}^{\lvert \mathsf{k} \rvert}$		
	$keys[R.dst][i] \leftarrow \mathsf{k}$		
	$R' \leftarrow \mathsf{Enc}(\mathsf{k}, R)$		
	broadcast $R', R.dst, i$		
		$\xrightarrow{R', R.dst, i}$	
			routes$[R.dst][i] \leftarrow R'$
		$\xrightarrow{i}$ store i	
	$\xrightarrow{R.dst, n'}$		
		$ep[R.dst][i] \leftarrow n'$	
.....	MPC execution	.....	
	MPC$(keys[R.dst], ep[R.dst],$ prefs_db$[R.dst], recc)$	MPC$(\{0\}^{\lvert \mathsf{k} \rvert \cdot m}, ep[R.dst],$ prefs_db$[R.dst], \{0\}^{rl})$	
	MPC execution $\longleftrightarrow$	MPC execution	
	$(\mathsf{k}_0, i_0) \leftarrow$ MPC$(\ldots)$	$(\mathsf{k}_1, i_1) \leftarrow$ MPC$(\ldots)$	
.....	Send resulting key and index	.....	
		$\xrightarrow{\mathsf{k}_0[j], i_0[j]}$	
			store $\mathsf{k}_0[j]$, store $i_0[j]$
		$\xrightarrow{\mathsf{k}_1[j], i_1[j]}$	
			store $\mathsf{k}_1[j]$, store $i_1[j]$
.....	Member decryption	.....	
			$\mathsf{k}' \leftarrow \mathsf{k}_0[j] \oplus \mathsf{k}_1[j]$
			$i \leftarrow i_0[j] \oplus i_1[j]$
			if $\mathsf{k}' \neq \mathsf{k}_{dummy}$ **then**
			$\quad R \leftarrow \mathsf{Dec}(\mathsf{k}', \text{routes}[R.dst][i])$
			endif

Protocol 6.3: The SELECT-BEST protocol. Inputs: R: a BGP route stripped off its export policy ep; $|ep|$: the length in bits of ep; $|\mathsf{k}|$: the bit length of key k; $recc(R)$: route recommendation at the IXP; m: the number of members.

In SIXPACK, we additionally employ symmetric encryption for routes that are transferred from sending to receiving members. In our implementation we use AES with a key size of 128 bit in counter-mode (CTR). We are not aware of any significant attacks on AES in this mode of operation and thus believe it to be a viable choice for our purpose. Alternatively any suitably block cipher can be used for encryption.

We ensure that the symmetric keys that are required to decrypt a route are only provided to those members who are explicitly allowed to receive them. This is achieved by relying on the GMW protocol that provably guarantees correctness, security and privacy. The correctness property of GMW together with the correctness of the circuits we evaluate with it ensure that only those members who are explicitly allowed by the export policy of a sending member will receive the correct key to decrypt an encrypted route. All other members receive a dummy key, that will not successfully decrypt the route. We describe the circuits that implement this behavior in Sect. 6.5.4 and Sect. 6.5.5.

The privacy property of GMW ensures that the computational parties, who carry out the operations on the routes and secret-shared export policies cannot gain access to this data unless they break the non-collusion assumption. This is guaranteed by using information-theoretic XOR-based secret sharing that masks plaintext inputs with random data and splits it between the computational parties. These parties evaluate our circuits gate by gate using the GMW protocol, while maintaining the invariant that the values on each wire in the circuit is secret-shared among the parties. The circuits' outputs are sent only to the eligible receiving members who are able to reconstruct the plaintext output values.

It is easy to see that our circuits (e.g., Circ. 6.1) correctly implement our algorithms. These are direct translations of the algorithms into lower level components, such as AND ($\wedge$) and NOT (!) gates, as well as ADD, GT and MUX. We rely on the correctness of the implementations of [BPP00] for ADD-gates, [KSS09] for GT and [DSZ15] for MUX, to conclude our final circuits, that use only AND and XOR gates. The correctness of the protocols is derived from correctness of the GMW protocol, and the correctness of our circuits.

We summarize our observations above in the following Theorem 1.

Theorem 1. ***Correctness, Security, Privacy of* SIXPACK *and privacy-preserving centralized route computation:*** *The* EXPORT-ALL *protocol (Circuit 6.2), the* SELECT-BEST *protocol (Circuit 6.3), the protocol for computing Algorithm 6.1, and the protocol for computing Algorithm 6.2 correctly, securely, and privately compute the respective functionality described in Prot. 6.1, in Prot. 6.3, in Algorithm 6.1, and in Algorithm 6.2 respectively, in the presence of a semi-honest adversary, corrupting either P_0 or P_1, but not both, in addition to at most all but one of the ASes.*

6.7 Deployment

In this section, we explain our network assumptions and propose a method for enforcing input consistency. We discuss how to handle failures and byzantine behavior, and possible deployment.

6.7.1 Disadvantages of Centralized Routing

The centralized routing schemes that are discussed in this chapter have some disadvantages over decentralized approaches. First, the ASes have to trust that the computational parties do not collude and the computational parties become a single point of failure. Second, the computation of light-paths for rapid restoration is more complex compared to a distributed model. Finally, the centralized approach is contradictory to the decentralized philosophy which has driven the rapid development of the Internet. Another issue with centralized approaches is latency. We propose to mitigate this, by applying our techniques not on global scale, but on smaller topologies, as described in Sect. 6.7.3.

6.7.2 Network Setting

The connection between the CPs is a critical point in our system and has to be low-latency and high-throughput. We argue that this is realistic, since reputed entities that run the CPs are often co-located in the same data centers, yet managed by different authorities. The communication between ASes and CPs can be an arbitrary Internet connection with no special requirements. Error correction can be applied on the usual network layers (e.g., within TCP/IP or on application level). Packet loss between ASes and CPs does not cause severe problems, as old inputs from ASes can be re-used in multiple protocol iterations. Lost outputs have the same consequences as lost BGP messages nowadays. However, packet loss is covered by the use of TCP/IP that re-sends lost packets. The ASes have to do one round of communication with the computational parties that run our protocols. Thus, we have measured the round trip time (RTT) between our computational parties and other Amazon EC2 regions and observed average RTTs between 90 ms and 311 ms, as depicted in Tab. 6.1. This time has to be added to the MPC runtime to get the time an individual AS has to wait for a computation result. The standard deviation of the RTTs and packet loss was less than 1%.

6.7.3 Route Computation Deployment

Our approach to privacy-preserving route computation is primarily intended as a broad vision for the future of interdomain routing. Of course, transitioning to MPC of interdomain routes is an extremely challenging undertaking that involves cooperation of tens of thousands of independent financial and political entities, alongside significant deployment and operational challenges. We believe, however, that our approach can also yield significant benefits (e.g., in terms of privacy, security, and ability to innovate) when applied at a country/region-level scale,

Table 6.1: Average network round trip times between Amazon EC2 regions measured from EU (Frankfurt) to the listed regions.

Location	∅ RTT
US East (Northern Virginia)	90.2 ms
US West (Northern California)	162.5 ms
South America (São Paulo)	193.4 ms
Asia Pacific (Mumbai)	112.5 ms
Asia Pacific (Tokyo)	228.5 ms
Asia Pacific (Sydney)	311.3 ms

while alleviating many of the challenges a global transition entails. Often, for performance and security reasons, traffic within a geographic/political region is expected to not leave the boundaries of that region. One example for this is that many end-users retrieve content from servers in their geographic region due to the popularity of content delivery networks (à la Akamai). Thus, a natural deployment scenario for MPC of interdomain routes is focusing on a specific region and executing MPC only for routes between the ASes in that region, while all other routes will be computed via traditional (decentralized) BGP routing.

This would also offer very natural instantiations of the MPC parties: The computation could be done by the RIR as well as a local IXP, such as DE-CIX. While being independent entities, they typically share a fast and low latency network connection, which is required for our protocols. We give the runtimes for such subgraphs for the RIRs in Fig. 6.12 and observe that such a regional execution also drastically decreases the runtime of our algorithms (e.g., 0.20 s setup time and 0.17 s online time for the German RIR RIPE-DE). We point out that such a scheme, beyond MPC's inherent privacy guarantees and the other benefits listed in Sect. 6.1.1, can provide the guarantees that all routes between ASes in the region will indeed only traverse other ASes in that region. This should be contrasted with today's insecure routing with BGP, which allows a remote AS to manipulate the routing protocol so as to attract traffic to its network. Our evaluations show that, beyond the above guarantees, it also yields better running times due to the smaller size of the "input". We believe that a region/country-level implementation of such MPC of routes is a tangible and beneficial first step en route to larger-scale deployment scenarios.

6.7.4 Hiding the Network Topology

Currently, we exploit the fact that the network topology is public for many implementation optimizations in the circuit. However, we could also keep the topology private, which comes at an overhead of $O(n^3)$, where n is the number of ASes [ACM+13]. For country-level ASes, especially when excluding stub-nodes, this overhead seems tolerable. E.g., the RIPE-DE country-level AS has 250 non-stub ASes, which would result in a circuit with 30 million AND gates for the neighbor relations algorithm of Sect. 6.4.1 and around 200 million gates for the neighbor preference algorithm of Sect. 6.4.2.

6.7.5 Leakage Through Received Routes

Naturally, this raises the question of what can be inferred from the received routes and how effective the protection of SIXPACK is.

While many techniques have been designed to infer peering relationships, and even routing policies, in the Internet, using control- and data-plane traffic information [CDE+10; DKF+07; WG03; Gao01; MKA+09; SBS08; JCC+13], such solutions not only have limited accuracy [RWM+11] (e.g., globally visible AS-paths neither reveal local preferences nor "negative" export policies, i.e., not exporting a route) but might also be detectable (e.g., BGP AS-path spoofing [JCC+13]). Finally, SIXPACK supports ranking routes based on the IXP members' port utilization, a fundamental performance metric that is challenging to infer in practice [CML10].

6.7.6 SIXPACK Deployment

SIXPACK can easily and incrementally be deployed at IXPs since it needs not replace the traditional RS service, i.e., the two can coexist. Each member can independently decide whether to share its routes in a privacy-preserving manner or not. Observe that the routes shared via the traditional RS can be forwarded to SIXPACK so that the early adopters have the same route visibility of the members peering at the traditional route server. We believe that the desire to keep peering relationships private can incentivize adoption.

Moreover, SIXPACK only requires small modifications at both the IXP and member side (i.e., route servers and BGP border routers). Members could run SIXPACK as a BGP proxy that receives updates intended for the RS from its BGP border router and performs the MPC input sharing process. Similarly, the BGP proxy will receive the output of the MPC from the two MPC-enabled route servers and translate it into BGP updates that will, in turn, be forwarded to the member's BGP border router. Members not using SIXPACK require no changes to their infrastructure.

We believe that bringing innovation to today's networks is no longer as prohibitive a task as it has been in the past. The rise of programmable networks with SDN is boosting innovation in different environment such as data centers [SOA+15] and, more recently, IXP networks [SFLR13; GVS+14; Lig15; END15; AMS16; GMB+16]. Deploying SIXPACK in a so-called Software-Defined-eXchange (SDX) could provide the opportunity for using custom interfaces and avoid the need for compatibility with BGP. Advanced route dispatch services, based for instance on port congestion metrics, should obliviously be designed to avoid route oscillations (cf. Sect. 6.5.5).

6.8 Implementation

In this section we explain details of our implementations.

Optimized Implementation of Algorithm 6.1

In order to counter the problem of the high number of AND gates, the memory complexity for storing the circuit, and the high number of communication rounds, we propose to use the following three optimizations: reduce the complexity for evaluating AND gates by using *vector ANDs*, decrease the number of gates in the circuit description by using *SIMD circuits*, and decrease the circuit depth by building certain parts of the circuits as *tournament-like evaluation*. In Tab. 6.5 we give the improvements of our optimizations and break down the resulting complexities into the sub-functionalities in Tab. 6.6. We describe these optimizations in more detail next.

Vector ANDs The naive circuit in Circ. 6.1 consists of many multiplexer gates operating on ℓ-bit values, needed to realize `if` conditions. As outlined in [DSZ15] these multiplexers can be instantiated using vector ANDs that reduce the precomputation cost from ℓ AND gates to the cost of one AND gate, cf. Sect. 3.2.3.

Overall, the multiplexers constitute to around 75% of the total number of AND gates in our circuits. By using the vector AND optimization, we can therefore reduce the number of AND gates by factor 3 for the neighbor relation algorithm in Algorithm 6.1 and by up to factor 45 for the neighbor preference algorithm in Algorithm 6.2 (as shown in column Total ANDs vs. Vector ANDs in Tab. 6.2). Note, however, that this optimization can only be applied when performing the evaluation with the GMW protocol, while an evaluation with Yao's garbled circuits has to process the total number of AND gates (cf. middle column in Tab. 6.6).

SIMD circuits In order to cope with large circuits, there are two common approaches: pipelining the circuit construction and evaluation [HEKM11] and building a Single Instruction Multiple Data (SIMD) circuit. While the approach of pipelining the circuit construction and evaluation is especially suited for processing circuits of arbitrary size, we decided to pursue a solution based on SIMD techniques. A SIMD circuit also consists of gates, but instead of operating on single bits, it operates on multiple bits in parallel. Thereby, the time for the load / process / store operations of a gate amortizes, which drastically speeds up the evaluation [SZ13]. In contrast, a pipelined construction and evaluation approach would need to perform a load / process / store operation per bit of evaluation. We will now describe how to build such a SIMD circuit that evaluates our BGP functionality.

Note that for the customer, peer, and provider functionality, we perform the same operation for each node $v \in V$ in parallel. Using SIMD circuits, we can combine the values for each node into vectors instead of single bits and thereby only build a single copy of the functionality. Thereby, we can operate on multiple values in parallel, which allows us to reduce the memory footprint of the circuit as well as to decrease the time for circuit evaluation. However, applying the SIMD programming style is not straight-forward since for each node the circuit depends on its degree, i.e., the number of its neighbor nodes n, which differs drastically between ASes. The obvious solution, that builds the circuit for the node with the highest degree $n_{\max}$ and pads the number of neighbor nodes for all other nodes to $n_{\max}$, introduces a non-tolerable overhead in terms of AND gates. We solve this problem as described next.

All nodes are divided into groups of similar degree. After each iteration the results from all groups are merged into a state, that is used as input to the next iteration. The challenge is to find the right amount and size of groups to partition the nodes. For our experiments, we use the following partitioning: $\{1, 2, \ldots, 6, 8, 12, 20, 32, 64, 128, 256, \ldots, n_{\max}\}$, where $n_{\max}$ is the highest number of neighboring nodes that any AS in the topology has. This partitioning was chosen based on the degree distribution shown in Fig. 6.3, and resulted in good overall runtimes for all datasets that we performed our benchmarks on.

Tournament Evaluation The current circuit has a high multiplicative depth, which makes it inefficient for MPC protocols which require communication rounds linear in the circuit depth, e.g., the GMW protocol. The reasons for the high depth of the circuit are the iterative structure of Algorithm 6.1 and the sequential processing of neighbors, which results in a circuit depth linear in $n_{\max}$, i.e., the highest number of neighbors of any AS in the graph (for the CAIDA dataset, $n_{\max}$=5 936). In order to reduce the depth for processing the neighbors, we adopt a tournament evaluation style by arranging operations in form of a tree and thereby achieve a logarithmic depth. We give the selection function for the customer functionality in Func. 6.4 and for the peer / provider functionality in Func. 6.5. Note that we can compute sel in Func. 6.4 as well as sum_L and sum_R in Func. 6.5 once in the beginning and pass/re-use them during the tournament evaluation to decrease the number of AND gates. Thereby, the overall number of AND gates in the circuit remains the same as for the sequential circuit.

Input: $v \in V$, $(\mathsf{next}[u_L], \mathsf{dist}[u_L], \mathsf{fin}[u_L], \mathrm{in}_v^{\mathrm{cust}}[u_L])$,
$(\mathsf{next}[u_R], \mathsf{dist}[u_R], \mathsf{fin}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_R])$ with
$u_L, u_R \in \mathsf{Adj}[v], u_L \neq u_R$

1: $\mathsf{sel} \longleftarrow \mathsf{fin}[u_L] \wedge !\mathsf{fin}[v] \wedge \mathrm{in}_v^{\mathrm{cust}}[u_L]$
2: $\mathsf{next}' \longleftarrow \mathrm{MUX}(\mathsf{next}[u_R], \mathsf{next}[u_L], \mathsf{sel})$
3: $\mathsf{dist}' \longleftarrow \mathrm{MUX}(\mathsf{dist}[u_R], \mathsf{dist}[u_L], \mathsf{sel})$
4: $\mathsf{fin}' \longleftarrow \mathrm{MUX}(\mathsf{fin}[u_R], \mathsf{fin}[u_L], \mathsf{sel})$
5: $\mathrm{in}' \longleftarrow \mathrm{MUX}(\mathrm{in}_v^{\mathrm{cust}}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_L], \mathsf{sel})$

Output: $(\mathsf{next}', \mathsf{dist}', \mathsf{fin}', \mathrm{in}')$

Circuit 6.4: Selection Function customer

Implementation of Algorithm 6.2

The structure of the neighbor preference algorithm described in Sect. 6.4.2 is very similar to that of the peer/provider part of Algorithm 6.1 described in Sect. 6.4.1. Thus, we can use the same structure, ideas and optimizations as described before to efficiently realize it as a Boolean circuit optimized for the evaluation with the GMW protocol. The main difference between the neighbor preference and the relation algorithm is the publish matrix pub, held by each AS. This matrix has dimension $|\mathsf{Adj}[v]| \times |\mathsf{Adj}[v]|$ and hence becomes very large for

Input: $v \in V, (\mathsf{next}[u_L], \mathsf{dist}[u_L], \mathsf{fin}[u_L], \mathrm{in}_v^{\mathrm{cust}}[u_L])$,
$(\mathsf{next}[u_R], \mathsf{dist}[u_R], \mathsf{fin}[u_R], \mathrm{in}_v^{\mathrm{cust}}[u_R])$ with
$u_L, u_R \in \mathsf{Adj}[v], u_L \neq u_R$

1: $\mathsf{dist}_{\max} = 2^\sigma - 1$
2: $\mathsf{sum}_L \longleftarrow \mathrm{MUX}(\mathsf{dist}_{\max}, \mathsf{dist}[u_L], \mathsf{fin}[u_L] \wedge \mathrm{in}_v^{\mathrm{peer/prov}}[u_L])$
3: $\mathsf{sum}_R \longleftarrow \mathrm{MUX}(\mathsf{dist}_{\max}, \mathsf{dist}[u_R], \mathsf{fin}[u_R] \wedge \mathrm{in}_v^{\mathrm{peer/prov}}[u_R])$
4: $\mathsf{sel} \longleftarrow \mathrm{GT}(\mathsf{sum}_R, \mathsf{sum}_L)$
5: $\mathsf{next}' \longleftarrow \mathrm{MUX}(\mathsf{next}[u_R], \mathsf{next}[u_L], \mathsf{sel})$
6: $\mathsf{dist}' \longleftarrow \mathrm{MUX}(\mathsf{dist}[u_R], \mathsf{dist}[u_L], \mathsf{sel})$
7: $\mathsf{fin} \longleftarrow \mathrm{MUX}(\mathsf{fin}[u_R], \mathsf{fin}[u_L], \mathsf{sel})$
8: $\mathrm{in} \longleftarrow \mathrm{MUX}(\mathrm{in}_v^{\mathrm{peer/prov}}[u_R], \mathrm{in}_v^{\mathrm{peer/prov}}[u_L], \mathsf{sel})$

Output: $(\mathsf{next}', \mathsf{dist}', \mathsf{fin}', \mathrm{in}')$

Circuit 6.5: Selection Function peer / provider

ASes with many neighbors. In fact, for the full CAIDA dataset from November 2016, only the AS with the most neighbors ($n_{\max} = 5\,936$) has a matrix with 35 236 096 bits. Each bit of this matrix has to be accessed once for each of the $2d_{\mathsf{depth}} + 1$ rounds in order to hide the current next hop, which costs one AND gate per bit. Overall, the total number of AND gates in the circuit for the full CAIDA dataset from November 2016 amounts to nearly 8 billion (cf. Tab. 6.2). The vector AND optimization allows us to perform a more efficient access and reduces the cost for processing this matrix in the setup phase to 130 million AND gates (cf. Tab. 6.2). However, during the online phase we have to evaluate the total number of AND gates, regardless of the vector AND optimization, which results in a communication of approximately 2 GiB of data which is an order of magnitude higher than for the relation-based algorithm of Sect. 6.4.1.

Additionally, in the neighbor preference algorithm, the computation parties need to perform lookups by secret-shared values in Step 7 (i.e., the lookups $\mathsf{pub}_u[\mathsf{next}[u], v]$ and $\mathsf{pref}_v[\mathsf{next}[v]]$). We implement these lookups by updating the values pub_u and pref_v for all nodes each time a new next hop is chosen. Note that updating the values is done using the vector AND optimization, which greatly reduces the costs.

6.8.1 SIXPACK Implementations

MPC Circuit Representation

In this section, we describe the Boolean circuits representing EXPORT-ALL Sect. 6.5.4 and SELECT-BEST Sect. 6.5.5 that we built for evaluation with the GMW protocol.

We rely only on two gates: a greater-or-equal gate ($\geq$), and a multiplexer (MUX) that outputs one of its two data inputs, depending on a selection input bit. We intentionally choose

to restrict our circuit construction to a minimalistic design in order to achieve practical runtimes. Our circuits are evaluated in a SIMD (Single Instruction Multiple Data) fashion that allows to efficiently process the inputs for each of the members in parallel. Furthermore, our circuits have a very low multiplicative depth, so they can be evaluated efficiently with the GMW protocol, whose round complexity is linear in the multiplicative depth. For the EXPORT-ALL approach the depth is constant and for the SELECT-BEST approach, it grows only logarithmically with the number of routes, which is never larger than 30 in our real-world example data (see Sect. 6.9.2). We stress that our circuits are generic Boolean circuits that could potentially be evaluated with arbitrary MPC solutions and can thus be extended to active security or more than 2 parties, given additional resources.

SIXPACK Implementation

We implemented the two MPC components shown in Fig. 6.9 and Fig. 6.10 in `C++` using ABY (Chapt. 3) in $\approx$ 700 lines of code.

With ABY's SIMD evaluation, we can run our two MPC circuits for inputs from arbitrarily many members in parallel while vector operations allow the efficient processing of long bit strings, such as the route keys. The circuits that we built are optimized to process multi-bit values efficiently, which benefits the processing of route keys and comparing preference values. The SELECT-BEST circuit is evaluated in a tournament fashion by arranging the preference comparison gates in a tree. Thereby we achieve a circuit depth that grows logarithmically in the number of members, thus resulting in optimized latency for GMW. We designed our circuits to be minimalistic and performant, while still being as expressive as needed for our use cases.

We rely on the proven security of a symmetric cipher for encrypting/decrypting routes, which we instantiate with 128-bit AES in CTR mode. We verified that AES adds negligible overhead compared to the MPC, which holds in general and especially on machines with the AES-NI instruction set.

Our route server service, which wraps the MPC components and handles the distribution and processing of all the BGP update information among the IXP members, is implemented as 1 800 lines of code in Python. The RSes run as independent processes, each executing its own instance of MPC as a daemon subprocess, and communicate via TCP sockets. To improve the efficiency of SELECT-BEST, we observe that, in practice, it is convenient to batch several next-hop ranking messages together before executing SELECT-BEST as shown in our evaluation (Sect. 6.9.2).

Although we did not optimize our implementation to the fullest extent possible, our evaluation (Sect. 6.9.2) shows that our approach already scales to the size of the largest IXPs in the world. We discussed deployment consideration in Sect. 6.7.6.

Optimizations

We believe that the running time of SIXPACK can be drastically reduced through the following two improvements:

First, the current implementation of the MPC does not allow to separate the setup and online phases. For example, the runtime of the MPC part for the SELECT-BEST approach, when processing 32 inputs and using 4 bits for representing preferences, currently takes a total time of 158.0 ms, while the online phase only amounts to 35.9 ms of that. Since for all our results the setup phase accounts for at least 77% of the total MPC processing time, we believe that the per-prefix processing time can be improved significantly as soon as our implementation becomes capable of precomputing the setup phase. We stress that this is possible in theory and the current situation is merely a limitation of our implementation.

Second, our Python code could be rewritten in a more performance-oriented programming language such as C/C++, thus gaining an additional decrease in runtimes.

6.9 Benchmarks and Evaluation

In this section we evaluate the complexity and performance of our solutions. In Sect. 6.9.1, we provide evaluation results for the BGP route computation algorithms from Sect. 6.4. The performance analysis of SIXPACK (Sect. 6.5) can be found in Sect. 6.9.2.

6.9.1 Route Computation Benchmarks

In this section, we provide benchmark results of our protocols and evaluate the practicality of our solution privacy-preserving route computation. We show that we are capable of securely evaluating the circuit for the full CAIDA dataset in a reasonable runtime and further improve it using the algorithmic optimization of excluding stubs from the computation (cf. Sect. 6.4.3).

We implement our protocols in the GMW protocol using ABY. The main reason for the GMW protocol are the optimizations from Sect. 6.8, that are only possible with GMW. Using Yao's garbled circuits, runtimes would become impractical. We provide further arguments for choosing the GMW protocol in Sect. 3.3.5.

To the best of our knowledge, the optimizations described in Sect. 6.8 are only implemented in ABY. We are not aware of automated tools capable of using the same optimizations to the same extent that we do in our hand-built circuits. We would like to point out, however, that our efficient circuits are generic Boolean circuits that could be evaluated with any secure computation framework and could thus be extended to more than 2 parties or even security against malicious adversaries (e.g., using [NNOB12] or [NST17]), with additional cost in communication and runtime.

Benchmarking Environment Our MPC benchmarks are run on two Amazon EC2 `c4.2xlarge` instances with 8 virtual CPU cores with 2.9 GHz and 15 GiB RAM located in the same region, connected via a Gigabit network connection. The symmetric security parameter in our experiments is set to 128 bits. All runtime results are median values of 10 protocol executions and their standard deviation. The communication numbers provided are the sum of sent and received data for each party.

CAIDA (Fig. 6.11) The graphical evaluation of both protocols (*neighbor relationship* Algorithm 6.1 and *neighbor preference* Algorithm 6.2) on CAIDA datasets of the past 10 years is provided in Fig. 6.11. The detailed results are given in Tab. 6.2. Our results show that both protocols spend most of the time in the setup phase which takes 15.5 s for the neighbor relation algorithm (35.1 s for the neighbor preference algorithm) for the full CAIDA November 2016 topology and reduces to 6.4 s (12.8 s) if we exclude stub nodes. This part of the computation is less critical than the online phase for two reasons: a) it is independent of the network topology and input of the ASes and thus can be precomputed at any time and b) it can be ideally parallelized by adding more machines and thus is just dependent on the available resources. The online phase is the time required from a secret shared input of the ASes until the resulting next hop on the routing tree can be provided to them. For the full network, the online runtime is 6.1 s for the neighbor relation algorithm (29.9 s for the neighbor preference algorithm) and decreases to 3.2 s (10.5 s) when ignoring stub nodes.

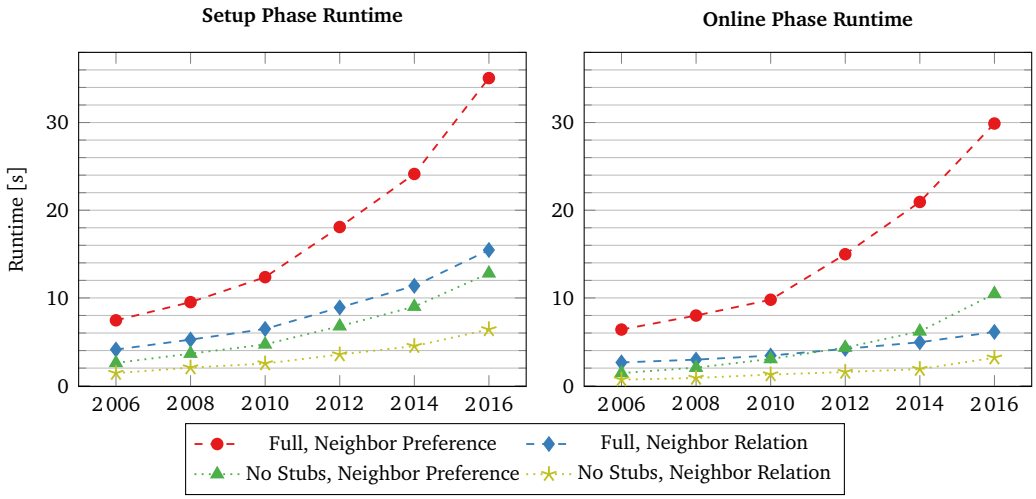


Figure 6.11: Median runtimes for setup and online phase, for the CAIDA topology of November of every year, with and without stub nodes, comparing the neighbor relation algorithm (Algorithm 6.1, Sect. 6.4.1) and the neighbor preference algorithm (Algorithm 6.2, Sect. 6.4.2).

Generally speaking, the algorithmic improvement of removing stub nodes from the network topology speeds up both protocols by a factor of 2 to 3.

The required bandwidth between the two MPC parties in the online phase for the neighbor relationship algorithm is less than 60 MiB, while the more complex neighbor preference algorithm requires ≈ 800 MiB for the newest topology without stub nodes. Our results also show that the online runtime of the preference algorithm scales worse with a growing topology size. In general, communication between the two computing parties takes approximately 1/3 of the runtime of the online phase and is also the part that induces the biggest runtime variations, even on a local network. The remaining 2/3 of the online phase runtime are spent on local computations consisting of bit operations and memory accesses.

RIRs (Fig. 6.12) We show similar measurements for subgraphs of CAIDA's November 2016 topology for 5 RIRs and a regional topology for German ASes (RIPE-DE) in Fig. 6.12 and provide detailed numbers in Tab. 6.3. When considering the smaller RIR topologies, the online time decreases below 10 s for both algorithms, even for big sub-topologies such as RIPE or ARIN; for smaller sub-topologies even further. For instance, on the full RIPE-DE sub-topology, the online runtime for the neighbor relation algorithm is 0.3 s (0.5 s for the neighbor preference algorithm) and decreases to 0.2 s (0.3 s), when leaving out stubs. In a similar fashion, the required bandwidth decreases to1.0 MiB for the neighbor relation algorithm and to3.6 MiB for the neighbor preference algorithm without stub nodes.

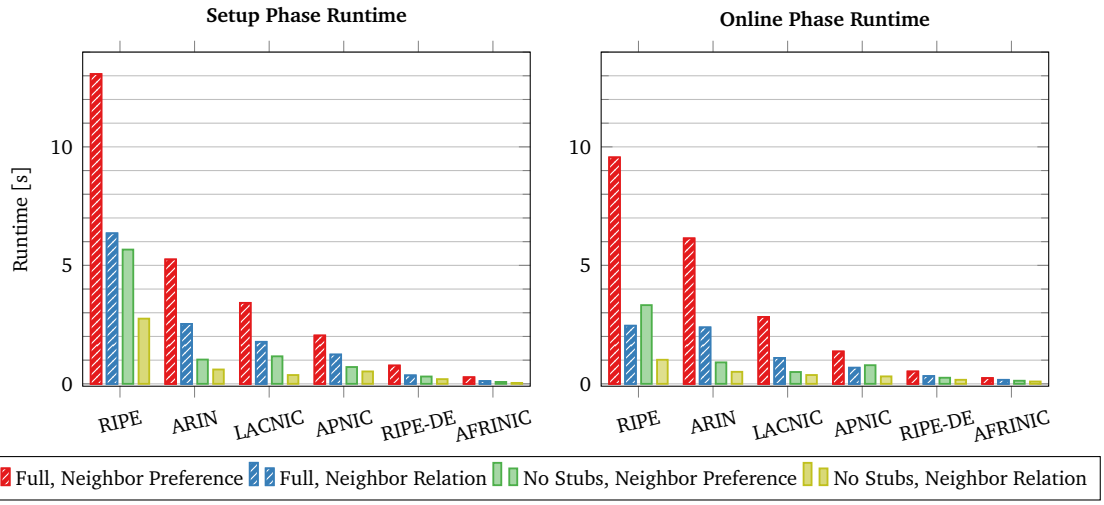


Figure 6.12: Median runtimes for setup and online phase, for subgraphs of CAIDA's November 2016 topology, with and without stubs, comparing the neighbor relation algorithm (Algorithm 6.1, Sect. 6.4.1) and the neighbor preference algorithm (Algorithm 6.2, Sect. 6.4.2).

In Tab. 6.2, we provide detailed evaluation results for both of our protocols on the CAIDA datasets of the past 10 years, where the most recent results for the topology of November 2016 are marked in bold. We list the number of ASes, the connections between them, and the maximum degree for each benchmarked topology. Furthermore, we list the circuit sizes as total number of AND gates (that one would have to evaluate *without* using the vector gate

optimization of the GMW protocol respectively when using Yao's protocol), the number of gates when using the vector gate optimization for GMW, and the depth of the circuit, i.e., the number of communication rounds between the two computational parties.

Table 6.2: Comparison of topology, circuit and MPC runtimes of both algorithms from Sect. 6.4, using CAIDA datasets from past years, comparing the full dataset with the topology without stubs. The depicted communication happens solely between the CPs. We used the November dataset from every respective year. Most recent values are marked in **bold**.

			Topology			**Circuit**			**Benchmarks**			
									Setup Phase		Online Phase	
CAIDA Dataset			ASes	Edges	max. Degree	Total ANDs [$\cdot 10^6$]	Vector ANDs [$\cdot 10^6$]	AND Depth	Runtime [s]	Comm. [MiB]	Runtime [s]	Comm. [MiB]
Full Topology	2008	Alg. 6.1 Sect. 6.4.1	30 018	82 630	2 632	123	37	1 042	5.2 (± 1%)	1 136	3.0 (± 1%)	47
	2012		42 847	138 306	3 703	217	63	1 042	8.9 (± 1%)	1 921	4.2 (± 0%)	82
	2016		55 809	239 064	5 936	380	110	1 118	**15.5** (± 0%)	**3 344**	**6.1** (± 1%)	**143**
	2008	Alg. 6.2 Sect. 6.4.2	30 018	82 630	2 632	2 074	58	1 430	9.5 (± 1%)	1 767	8.0 (± 2%)	520
	2012		42 847	138 306	3 703	4 428	99	1 430	18.1 (± 0%)	3 028	15.0 (± 2%)	1 105
	2016		55 809	239 064	5 936	6 603	184	1 535	**35.1** (± 0%)	**4 577**	**29.9** (± 0%)	**2 130**
No Stubs	2008	Alg. 6.1 Sect. 6.4.1	4 550	29 275	764	43	14	890	2.1 (± 3%)	418	0.9 (± 1%)	16
	2012		6 483	52 661	1 384	78	25	966	3.5 (± 2%)	760	1.6 (± 0%)	30
	2016		8 407	95 157	2 913	147	45	1 042	**6.4** (± 1%)	**1 374**	**3.2** (± 1%)	**55**
	2008	Alg. 6.2 Sect. 6.4.2	4 550	29 275	764	434	22	1 220	3.7 (± 2%)	687	2.0 (± 1%)	111
	2012		6 483	52 661	1 384	1 173	41	1 325	6.8 (± 1%)	1 255	4.3 (± 1%)	296
	2016		8 407	95 157	2 913	3 142	75	1 430	**12.8** (± 0%)	**2 283**	**10.5** (± 1%)	**785**

In Tab. 6.3, we list the same values for subgraphs of the CAIDA dataset from November 2016 that correspond to the RIR networks and a local topology as described in Sect. 6.7.3.

Estimating Circuit Gate Counts

In Sect. 6.4.4 we described a 'naive' circuit for the neighbor relation algorithm Algorithm 6.1. Here we detail the gate counts that we estimated.

From the results in Tab. 6.5 we observe that rewriting the circuit as SIMD circuit increases the total number of AND gates by factor 3, and by factor 1.4 when using GMW with vector ANDs. However, at the same time the *total* number of gates (AND plus XOR) that have to be held in memory is reduced by a factor of 130 from 400 Million to 3 Million. This allows us to hold the complete circuit in memory, which greatly improves the runtime.

The tournament evaluation method (cf. Sect. 6.8) allows us to construct a tree of all neighbor nodes that has a worst-case depth logarithmic in the highest number of neighbors $\log_2(n_{\max})$. Thereby, we are able to reduce the multiplicative depth of the circuit by two orders of magnitude, as depicted in Tab. 6.5. From Tab. 6.6 we can observe that most of the multiplicative depth of the optimized circuit is due to the peer and provider functionalities. This can be explained by the comparisons that need to be done between each pair of neighbor distances for the peer and provider iterations. During a customer iteration, if a node that has no route

Table 6.3: Comparison of topology, circuit and MPC runtimes of both algorithms from Sect. 6.4, using subgraphs of the CAIDA datasets from November 2016, comparing the full dataset with the sub-topology, with and without stub nodes. The depicted communication happens solely between the CPs. Most recent values are marked in **bold**.

			Topology			Circuit			Benchmarks			
									Setup Phase		Online Phase	
	Dataset		ASes	Edges	max. Degree	Total ANDs [$\cdot 10^6$]	Vector ANDs [$\cdot 10^6$]	AND Depth	Runtime [s]	Comm. [MiB]	Runtime [s]	Comm. [MiB]
Full Topology	CAIDA 2016	Alg. 6.1 Sect. 6.4.1	55 809	239 064	5 936	380	110	1 118	**15.5** (± 0%)	**3 344**	**6.1** (± 1%)	**143**
	RIPE		21 723	95 909	1 769	149	44	966	6.3 (± 1%)	1 358	2.5 (± 1%)	56
	ARIN		16 942	39 563	3 047	56	17	1 042	2.5 (± 3%)	518	2.4 (± 1%)	21
	APNIC		7 505	18 802	727	25	8.1	890	1.2 (± 4%)	249	0.7 (± 1%)	9.5
	LACNIC		5 283	28 374	1 066	39	12.5	966	1.8 (± 3%)	381	1.1 (± 1%)	15
	RIPE-DE		1 328	5 375	372	6.8	2.4	814	0.4 (±11%)	72	0.3 (± 1%)	2.6
	AFRINIC		916	1 644	199	1.8	0.7	738	0.1 (±16%)	21	0.2 (± 2%)	0.7
	CAIDA 2016	Alg. 6.2 Sect. 6.4.2	55 809	239 064	5 936	6 603	184	1 535	**35.1** (± 0%)	**4 577**	**29.9** (± 0%)	**2 130**
	RIPE		21 723	95 909	1 769	2 844	71	1 325	13.1 (± 1%)	2 185	9.6 (± 2%)	711
	ARIN		16 942	39 563	3 047	1 325	26	1 430	5.2 (± 1%)	789	6.1 (± 1%)	330
	APNIC		7 505	18 802	727	200	12.5	1 220	2.0 (± 4%)	386	1.4 (± 1%)	52
	LACNIC		5 283	28 374	1 066	693	20	1 325	3.4 (± 3%)	622	2.8 (± 2%)	174
	RIPE-DE		1 328	5 375	372	44	3.8	1 115	0.8 (± 8%)	118	0.5 (± 2%)	12
	AFRINIC		916	1 644	199	6.3	1.0	1 010	0.3 (±14%)	33	0.2 (± 3%)	1.8
No Stubs	CAIDA 2016	Alg. 6.1 Sect. 6.4.1	8 407	95 157	2 913	147	45	1 042	**6.4** (± 1%)	**1 374**	**3.2** (± 1%)	**55**
	RIPE		3 646	41 274	918	58	19	890	2.8 (± 2%)	583	1.0 (± 1%)	22
	ARIN		1 849	8 501	665	11	3.7	890	0.6 (± 7%)	112	0.5 (± 1%)	4.0
	APNIC		1 140	5 398	338	6.7	2.3	814	0.4 (±11%)	71	0.3 (± 1%)	2.5
	LACNIC		1 012	8 367	484	9.9	3.6	814	0.5 (± 9%)	109	0.4 (± 1%)	3.8
	RIPE-DE		250	2 219	167	2.5	1.0	738	0.2 (±16%)	31	0.2 (± 2%)	1.0
	AFRINIC		178	371	77	0.4	0.2	662	0.0 (±12%)	5.1	0.1 (± 3%)	0.2
	CAIDA 2016	Alg. 6.2 Sect. 6.4.2	8 407	95 157	2 913	3 142	75	1 430	**12.8** (± 0%)	**2 283**	**10.5** (± 1%)	**785**
	RIPE		3 646	41 274	918	884	32	1 220	5.7 (± 2%)	971	3.3 (± 3%)	223
	ARIN		1 849	8 501	665	86	6	1 220	1.0 (± 6%)	182	0.9 (± 1%)	23
	APNIC		1 140	5 398	338	36	3.7	1 115	0.7 (± 9%)	117	0.5 (± 2%)	9.7
	LACNIC		1 012	8 367	484	109	5.8	1 115	1.2 (± 6%)	182	0.8 (± 2%)	28
	RIPE-DE		250	2 219	167	13	1.6	1 010	0.3 (±14%)	53	0.3 (± 2%)	3.6
	AFRINIC		178	371	77	1.0	0.2	905	0.1 (±10%)	9.2	0.1 (± 4%)	0.3

to the target has a neighbor with a route to the target (i.e., its finish bit is set to 1), this route is automatically the shortest path to the target AS. If there was a closer neighbor, it would have been found in an earlier iteration of the customer circuit. If there are multiple neighbor nodes who have a route, it is not required to determine the node with the minimal distance, since all are equally far away from the destination node dest.

6.9.2 SIXPACK Evaluation

We evaluate our SIXPACK prototype to demonstrate that our approach is both feasible and practical. We first provide insights into the performance of the MPC part of SIXPACK by performing micro benchmarks across a realistic range of numbers of IXP members and inputs. We then evaluate our system by replaying a real trace of BGP announcements from one of the largest IXPs worldwide and by performing a stress test. Our results highlight the following:

Table 6.4: Estimated number of AND gates per edge and AND depth for the sub-routines of the circuit for Algorithm 6.1 for $d_{\text{depth}} = 10, \delta = 16, \sigma = 5, n_{\max} = 5\,936$ for CAIDA 2016 full and $n_{\max} = 2\,913$ for CAIDA 2016 without stubs.

	Asymptotic		CAIDA 2016 full		CAIDA 2016 no stubs	
Sub-Function	#AND gates	AND depth	#AND gates	AND depth	#AND gates	AND depth
customer	$d_{\text{depth}}(2\sigma+\delta+3)$	$d_{\text{depth}}n_{\max}(\sigma+1)$	69 328 560	356 160	27 595 530	174 780
peer	$3\sigma+\delta+3$	$n_{\max}(\sigma+3)$	8 128 176	47 488	3 235 338	23 304
provider	$d_{\text{depth}}(3\sigma+\delta+3)$	$d_{\text{depth}}n_{\max}(\sigma+3)$	81 281 760	474 880	32 353 380	233 040
Total			158 738 496	878 528	63 184 248	431 124

Table 6.5: Total number of AND gates (as required by Yao's garbled circuits), number of AND gates that need to be evaluated with the vector AND functionality of GMW, and multiplicative (AND) depth for each of our optimizations.

	CAIDA 2016 full			CAIDA 2016 no stubs		
	#AND gates	#Vector ANDs	AND depth	#AND gates	#Vector ANDs	AND depth
Original	158 738 496	—	878 528	63 184 248	—	431 124
Vector AND	158 738 496	55 940 976	878 528	63 184 248	22 266 738	431 124
SIMD	380 481 105	109 540 680	878 528	147 251 331	44 997 018	431 124
Tournament	380 481 105	109 540 680	1 118	147 251 331	44 997 018	1 042

Table 6.6: Number of AND Gates (Total and Vector) and Depth for 1 iteration of each sub-circuit.

Dataset	**Sub-Circuit**	**Total AND Gates**	**Vector ANDs**	**AND Depth**
2016 No Stubs	customer	5 399 507	530 254	16
2016 No Stubs	peer/provider	8 463 319	3 594 066	81
2016 Full Topology	customer	14 479 639	1 365 866	17
2016 Full Topology	peer/provider	21 893 495	8 779 722	87

(1) While MPC is (as expected) the costliest part performance-wise, our results show that the online phase is even at worst below 38 ms. The *maximum* setup and online runtime we measured in our evaluation were 131.9 ms and 37.2 ms, respectively for 32 inputs in the SELECT-BEST component.

(2) Our still unoptimized SIXPACK prototype achieves BGP processing times below 90 ms at the 99-th percentile, and, specifically, below 23.6 ms and 62.8 ms for EXPORT-ALL and SELECT-BEST at the 99-th percentile, respectively. Furthermore, we measured negligible bandwidth requirements. SIXPACK processes a full-routing-table of 250 000 prefixes in ≈11 minutes, comparable to today's RSes. We stress the fact that our prototype can be improved to achieve better performance by precomputing the MPC setup phase.

It is worth comparing these numbers with the convergence time of BGP on the Internet, which can be in the order of minutes [MBGR03], that is, several orders of magnitude higher than time overhead due to dispatching routes with SIXPACK.

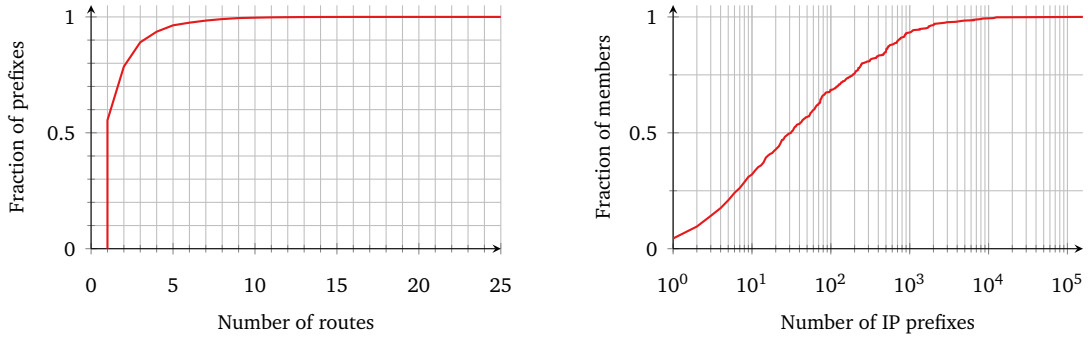
IXP Dataset

We assess our system using a trace of BGP updates from one of the largest IXPs worldwide, which interconnects more than 600 members. Our data shows that 511 IXP members connect to the IXP's route server. The trace covers a two-hour time window in August 2015. It contains 25 676 BGP update messages, consisting of 76 506 IP prefix announcements and withdrawals. The average number of BGP updates per second is 3.57, the first and third quartiles are 2 and 4, respectively, while the minimum and maximum numbers per second are 1 and 29, respectively. The average number of IP prefix announcements or withdrawals per second is 10.62, the first and third quartiles are 6 and 12, respectively, while the minimum and maximum numbers per second are 2 and 379, respectively. In addition to that trace, our data also contains a snapshot of the RS routing table at the beginning of the trace of updates. The routing table contains roughly 400 000 routes towards ≈ 240 000 IP prefixes.

In Fig. 6.13b, we use a CDF to show what fraction of the announced IP prefixes (y-axis) are reachable through no more that a certain number of routes (x-axis). We observed that for more than half of the IP prefixes there exists a single available route, with an average of 1.9, the 95-th percentile of 5 and a maximum of 25 routes per prefix. In Fig. 6.13b, we use a CDF to show what fraction of the IXP members (y-axis) announced no more than a certain number of routes (x-axis). We observed that on average each member announces 626.5 routes, the 95-th percentile is 1 581 and the maximum is approximately 150 000.

SIXPACK MPC Microbenchmarks

To benchmark the MPC circuits, we consider an IXP with 750 members, which is ≈ 1.5 times the number of members connected to the RS encountered in the IXP dataset we analyzed. From the dataset, we observe that at most 27 members export a route for the same IP prefix. Thus, we run benchmarks for a number of route keys up to 32.



(a) IP prefixes reachable via a given maximum number of routes.

(b) Members announcing at most a given maximum number of IP prefixes.

Figure 6.13: IP prefix CDFs for routes per prefix and prefixes per member.

We measure the runtime of the MPC setup and online phase. One invocation of MPC corresponds to processing a single IP prefix announcement. For all experiments, we perform 50 circuit executions and report median values of the runtimes and their standard deviation. Measurements were performed on two servers with 2.6 GHz CPUs and 128 GiB RAM, connected via a local 10 Gbps network. We evaluate with a preference value bit length of 8 bits. The reported communication is the sum of sent and received data by one party.

The number of total AND gates reported in Tab. 6.7 is the number of AND gates we would have to evaluate if we would use Yao's garbled circuits protocol for our MPC implementation. By using vector gates, that precompute multi-bit AND gates for the cost of 1-bit AND gates and that are only possible in the GMW protocol, we can save more than two orders of magnitude for the runtime of the setup phase.

Runtimes Tab. 6.7 shows the setup and online runtimes as well as the required communication and circuit complexity. Our results show that the setup phase takes between 1.7 ms and 122.4 ms, and depends on the number of inputs processed and the circuit used. In the best case, an IP prefix is announced by a single member and the EXPORT-ALL component can be used to dispatch the announcement to the legitimate members since no route comparison is needed. In the EXPORT-ALL component, each route is processed independently, even those towards the same destination IP prefix. This case corresponds to the computation with just 2 inputs (i.e., a given route and the dummy one) and requires an online computation of only 0.6 ms and an amount of transferred data of 25 KiB. However, as both setup and online runtimes grow sub-linearly with the number of routes, it is beneficial to compute on many routes in parallel, e.g., at times where several BGP announcements happen simultaneously. The runtimes of SELECT-BEST are large due to the deeper MPC circuit. Note that SELECT-BEST may not be used when an IP prefix is announced by a single route (i.e., 2 inputs). We also verified that our MPC circuits scale linearly with respect to the number of members, i.e., with 1 500 members, SELECT-BEST takes less than 70 ms to process 32 routes.

Table 6.7: MPC micro benchmarks: Circuit properties and runtimes of both building-blocks from Sect. 6.5 for 750 members, and a given number of input keys, where one input is the dummy key and the remaining inputs are route keys.

		Circuit			Benchmarks			
					Setup Phase		Online Phase	
Approach	#Inputs	Total ANDs	Vector ANDs	AND Depth	Runtime [ms]	Comm. [KiB]	Runtime [ms]	Comm. [KiB]
Export-All	2	102 000	750	1	1.7 (± 8%)	28	0.6 (±17%)	25
	4	306 000	2 250	1	3.6 (± 5%)	76	1.3 (±11%)	75
	8	714 000	5 250	1	7.2 (± 3%)	172	2.4 (± 7%)	176
	16	1 530 000	11 250	1	14.2 (± 2%)	360	4.4 (± 5%)	377
	32	3 162 000	23 250	1	24.0 (± 3%)	732	8.3 (± 4%)	778
Select-Best ($\ell = 4$ bit)	2	114 000	7 500	5	6.2 (± 5%)	248	1.9 (± 8%)	30
	4	342 000	22 500	9	14.9 (± 3%)	720	4.6 (± 4%)	89
	8	798 000	52 500	13	24.8 (± 3%)	1 652	9.4 (± 5%)	208
	16	1 710 000	112 500	17	44.4 (± 5%)	3 540	17.7 (± 2%)	445
	32	3 534 000	232 500	21	77.6 (± 8%)	7 293	34.0 (± 1%)	920
Select-Best ($\ell = 8$ bit)	2	128 250	15 750	6	10.6 (± 4%)	504	2.3 (± 5%)	35
	4	384 750	47 250	11	19.8 (± 4%)	1 492	5.3 (± 4%)	106
	8	897 750	110 250	16	34.6 (± 6%)	3 469	10.3 (± 3%)	247
	16	1 923 750	236 250	21	63.7 (± 9%)	7 409	19.3 (± 3%)	528
	32	3 975 750	488 250	26	122.4 (± 8%)	15 286	35.9 (± 1%)	1 091

Memory Consumption We also measure the memory consumption of our MPC implementation. Even when processing larger inputs of 32 routes and 2 000 members, the memory consumption of the dispatching operation, which is consumed only during a route dispatch, remains below 15 MiB. We determined that memory consumption grows sub-linearly with increasing parameters, which shows that our implementation will remain practical in the future.

In summary, these performance numbers confirm our MPC implementation's practicality.

SIXPACK Prototype Evaluation

To assess the feasibility of SIXPACK, we focus on evaluating its two main building blocks, i.e., EXPORT-ALL and SELECT-BEST, against a real-world trace of BGP updates collected from one of the largest IXPs in the world. To assess SIXPACK's scalability, we performed a stress test of EXPORT-ALL and SELECT-BEST and considered edge case scenarios such as the connection of a new member to the IXP network.

Experimental Setup We performed our experiments on three servers with 16 hyper-threaded cores at 2.6 GHz with 128 GiB of RAM and Ubuntu Linux 14.04, each two connected through 10 Gbps links. The average latency is 100 μs, similar to co-location data centers.[10] We use one server to replay the stream of BGP updates from our dataset and to handle the

[10] https://ams-ix.net/technical/statistics/real-time-stats

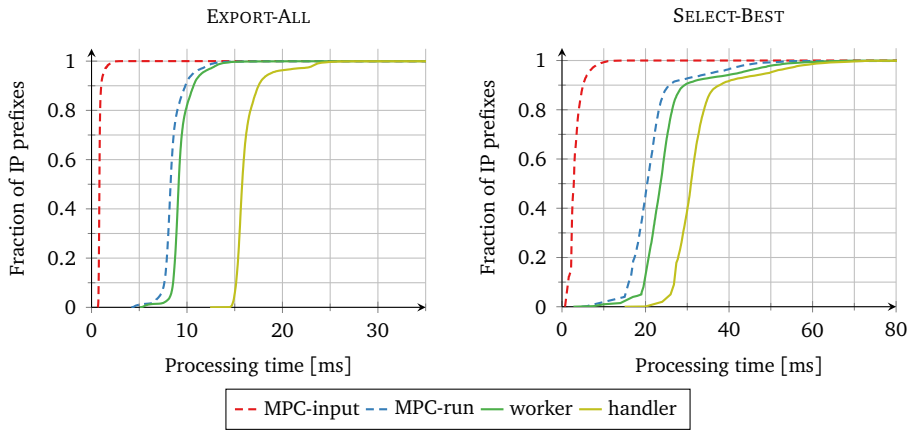


Figure 6.14: Processing time CDFs for EXPORT-ALL and SELECT-BEST components with 8 workers per RS instance.

receiving part of the SIXPACK mechanism. In each experiment, we load the full RS routing table contained in our dataset into the two RS instances.

Each RS instance runs on a server and consists of a set of *worker* processes orchestrated by a single *handler* process. The latter one (i) redistributes received BGP updates to the workers, (ii) guarantees that the other RS instance also knows which worker must be used to process a BGP update, (iii) synchronizes SIXPACK's operations, (iv) redistributes the MPC outputs to the members, and (v) guarantees that two BGP updates for the same IP prefix are not processed simultaneously by two different workers. Each worker runs an instance of the MPC party and computes its inputs.

Each BGP update can be either an announcement or a withdrawal of a set of IP prefix destinations that share the same export policy. In order to evaluate SELECT-BEST, we assign random local preferences among all the IXP members.

Performance Analysis on a Real-World BGP Trace For each BGP route announcement r, we measure the amount of time required to create the input values for the MPC processing, the time required to run the MPC computation, the amount of time the announcement is handled by a worker process, and the total time between the reception of r and the transmission of its output to the IXP members.

We plot the processing time of each IP prefix from our dataset in Fig. 6.14. We observe similar trends in both EXPORT-ALL and SELECT-BEST (i.e., Step I and Step III of SIXPACK), with the latter one being 2 times slower than the former one. In both graphs, we see a large gap between worker and handler times. This is mostly due to the unoptimized utilization of shared data structures in Python. Second, worker execution is dominated by the MPC processing, with MPC input creation requiring only a negligible amount of computational resources. We also measure the time required to encrypt and decrypt routing announcements, finding AES operations to be a negligible amount of time compared to the handler.

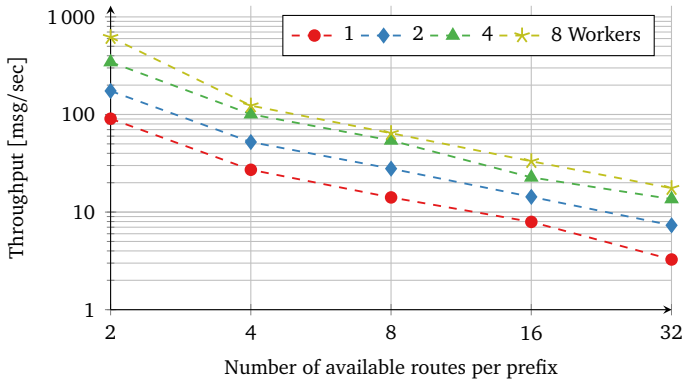


Figure 6.15: Throughput test of SIXPACK for different number of parallel workers.

To summarize, average processing times are 15.9 ms and 32.7 ms for EXPORT-ALL and SELECT-BEST, respectively, while the 99-th percentiles are 23.6 ms and 62.8 ms. These low running time reflects the fact that most of the routes are announced by few members (see Sect. 6.9.2). Currently the MPC part makes up the majority of the runtime, which could be further optimized by precomputing the setup phase ahead of time.

We also measured the amount of communication required by SIXPACK during the experiment. We found that for both EXPORT-ALL and SELECT-BEST the average bandwidth requirements from an IXP member to the RSes are negligible (i.e., 20 kbps). The requirements are higher for the communication between the two RSes. In the EXPORT-ALL component, the average bandwidth requirement is less than 2.79 Mbps while in SELECT-BEST it is no more than 10.9 Mbps. These figures show that even a 1 Gbps link between the two RSes would be more than sufficient for supporting SIXPACK.

Stress Test To assess the scalability of our system, we flooded SIXPACK with announcements and counted the number of routes dispatched per second. Our evaluation plotted in Fig. 6.15 follows two dimensions: number of routes announced for the same IP prefix (including a dummy route) and number of parallel workers. We observed that SIXPACK processes 619.28 route announcements per second in the EXPORT-ALL component and 64.33 announcements per second for IP prefixes announced through 8 routes. We observe that eight workers in parallel provide limited improvements for computations of more than a single announced route. We observed that sixteen parallel workers allow SIXPACK to process over 1 010 routes per second with EXPORT-ALL (not shown in the graph) but do not improve performance for routes announced by more than one member (i.e., in SELECT-BEST). The bandwidth requirements at full-speed are always below 1.5 Gbps.

Connecting a New Member: Comparing with Today's RSes When a member M connects to the RS, either because of a newly established connection or after recovering from a failure, two operations are performed: (i) the RS propagates to the new member all the best permissible routes towards the IP prefixes that were previously announced by the other

members and (ii) the RS recomputes and propagates to all the members the best route for each IP prefix announced by M. As for operation (i), at large IXPs, ≈250 000 best-route computations must be performed, one for each prefix known to the RS. While this may sound problematic, as observed in Fig. 6.13(b), most of the IP prefixes are announced by a single member, hence enabling us to consistently leverage the EXPORT-ALL component. Moreover, each best route is computed only for one IXP member and not for all of them. This allows us to considerably speed up the MPC execution. For instance, while executing the SELECT-BEST component for 500 members with 32 routes takes on average 158 ms, the same computation for a single member takes just 9 ms. We verified that operation (i) takes on average 92.8 s with our dataset. As for operation (ii), in Fig. 6.13(b), we observed that most of the customers do not announce more than 1 000 BGP routes. Our stress test shows that such operations would not take more than a few seconds. For large customers announcing ≈250 000 prefixes, we verified that the announcement operation takes roughly 11 minutes. In comparison, today's RSes report convergence times ranging between 3 and 10 minutes [AMS12; DEC16], even without incorporating import policies, performance-driven information, and privacy functionality.

6.9.3 GMW vs. Yao – Performance

We discusses in Sect. 3.3.5 why GMW is beneficial to Yao's garbled circuits in some cases. Here we provide performance numbers that show that an evaluation with Yao's protocol would perform much worse for our use case.

Communication Yao's garbled circuits protocol, we will have higher bandwidth requirements, due to the missing vector ANDs and the fact, that even today's most communication-efficient method for garbled circuits [ZRE15] requires $2 \cdot 128 = 256$ bit of communication per AND gate. Looking at the neighbor relation algorithm (cf. Sect. 6.4.1) and assuming an ideal Gigabit network connection the resulting runtime for communication alone will be $380 \cdot 10^6$ AND gates $\cdot$ 256 bit per AND gate / 1 Gbps ≈ 100 s for the full topology and ≈ 35 s without stubs (using values from Tab. 6.2). We emphasize that these ideal runtimes are already higher than our combined setup and online time.

Computation The fastest available implementation for Yao's garbled circuits, JustGarble [BHKR13], requires 48.4 cycles for evaluating and 101.3 cycles for garbling gates in "larger" circuits [BHKR13, Fig. 10]. Using a 3.5 GHz CPU, we need ($380 \cdot 10^6$ AND gates + $797 \cdot 10^6$ XOR gates) $\cdot$ 101.3 cycles per gate / ($3.5 \cdot 10^9$ cycles per second) ≈ 34 s for the full topology and ≈ 12 s without stubs for circuit garbling of the neighbor relation algorithm (cf. Sect. 6.4.1). The evaluation can be parallelized when using pipelining.

Overall, the runtime for Yao's garbled circuits, even with the fastest available implementation, most recent optimizations, and an ideal network, would be significantly slower than the runtimes we achieve with the GMW protocol. Evaluating the full neighbor relation algorithm using the CAIDA dataset from November 2016, we estimate at least ≈ 134 s using Yao's protocol, compared to less than 22 s in our GMW benchmarks.

We want to point out, however, that the Boolean circuit we designed for computing the routing tree of BGP is independent of the underlying secure computation scheme. Hence, for networks with high latency, one could easily use an implementation of Yao's protocol [Yao86] with pipelining [HEKM11] for evaluation instead of GMW.

6.10 Conclusion and Future Work

In this section, we provide an outlook into future work and summarize our results for privacy-preserving BGP routing.

6.10.1 Input Consistency

The centralized evaluation gives us the powerful ability to check the ASes' inputs for consistency. Since our solutions aim at protecting AS relations and local preferences, the CPs could have an overview of the announced prefixes and can detect malicious behavior such as prefix hijacking or misconfigurations. One more specific attack that can be prevented, is the following: AS a claims that AS b is its peer, while b claims that a is its provider. Clearly, one of them is lying. To verify the symmetry of AS input relations (cf. Sect. 6.2.2), we require only a single layer of AND gates that processes these inputs, adding negligible complexity to the overall algorithm. This check has to be done only once whenever an AS changes its inputs. Further, more complex sanity checks of encrypted data can be added at the expense of longer runtimes.

When inconsistencies are detected, the CPs can discard these new and inconsistent inputs and fall back to previous inputs. Involved ASes can be queried to re-send their inputs if we suspect that the inconsistency happened erroneously or due to faulty transmission. If an AS is detected as malicious or permanently faulty, the computational parties, can virtually remove this AS from the public topology and ignore it until recovery. This has the effect that no route will be sent via a faulty or malicious AS.

6.10.2 Handling Failures and Byzantine Behavior

Our approach preserves the privacy of interdomain route-computation against honest-but-curious attackers. However, the MPC itself is a single point of failure, as the routing depends on the availability and the honesty of two computational parties. We propose to add robustness by running multiple, independent 2-party MPC sessions in parallel. Alternatively, one could also use secure multi-party computation protocols based on t-out-of-n secret sharing that work even if all but t out of the n computing parties fail. Identifying more efficient schemes is an interesting direction for future research.

In Sect. 6.10.1 we showed that our approach can be adapted efficiently to the case of malicious ASes, but where the computational parties are still semi-honest. Another future direction

is to protect against malicious computational parties, while keeping runtimes practical, at least at country- or region-level scales. Another possible approach is to obtain security in a slightly weaker adversarial model, which is the covert security model [AL07]. By small adaptations of our protocol we can obtain a semi-honest version of GMW for the multi-party case (instead of two-party as we considered). Assuming honest majority, such a protocol can be transformed easily (and in a black-box way) to efficient protocols in the more robust setting of covert adversaries [AL07], at the expense of just running the protocol several times in parallel [DGN10]. In this setting, the corrupted party might not follow the protocol, and by doing so it can also sometimes break the security of the protocol. Nevertheless, the security guarantee is that any cheating attempt can be detected by the honest party with a fixed probability (say, 50%). Furthermore, by some additional (cheap) adaptations of the protocol, any cheating attempt can also be publicly verified [AO12; KM14], which enables the honest party to persuade other third parties (e.g., a "judge") about the cheating attempt. Since the computational parties in our settings are reputed authorities such as IANA/NANOG/RIPE, etc., we believe that the fear of being caught, the public humiliation or even the legal consequences are enough of a deterrent to prevent any cheating attempt.

6.10.3 MPC for Connectivity Marketplaces

We believe that future research should concentrate on extending the functionality of privacy-preserving RS services under realistic runtimes. One very interesting research direction along these lines could be designing a privacy-preserving marketplace, where networks dynamically buy and sell connectivity without revealing confidential information.

6.10.4 Conclusion

In this chapter, we presented SIXPACK, a privacy-preserving IXP RS design with provable guarantees. SIXPACK dispatches routes according to highly expressive members' routing policies and IXP performance-related information. We showed that an *efficient* realization of SIXPACK with MPC can be attained through a careful redistribution of the route dispatching responsibilities between the RS and IXP members. We devised optimized MPC circuits tailored to RS computation. We built a SIXPACK prototype and assessed its practical feasibility with a real-world trace of BGP updates collected from one of the largest world-wide IXPs.

We also showed that BGP route computation on Internet-scale is in principle possible in a privacy-preserving way, when we centralize the computation and employ the right MPC schemes to implement it. We achieve realistic performance numbers when our graph algorithms are run on a smaller scale. We show implementation details and algorithmic improvements that enabled us to achieve these results.

7 Privacy-Preserving Whole-Genome Matching

Results published in:

[DHSS17] D. DEMMLER, K. HAMACHER, T. SCHNEIDER, S. STAMMLER. **"Privacy-Preserving Whole-Genome Variant Queries"**. In: *16. International Conference on Cryptology And Network Security (CANS'17)*. Vol. 11261. LNCS. Springer, 2017, pp. 71–92. CORE Rank B.

7.1 Introduction

Genomic data holds the key to the understanding of many diseases and medical conditions. Some genomic variations in individuals in particular might be crucial in the diagnosis of a disease or a treatment regime. As a first step, a doctor or researcher might want to query the world's pool of sequenced genomes for such a variation in order to identify if it has been encountered and studied before. For this purpose, the *Beacon project* was established by the Global Alliance for Genomics & Health[1] to evaluate the eagerness of institutions around the globe to engage in a distributed variant query service. Participating institutions can be queried for a variation and confirm or deny its existence in their database.

However, this form of collaboration raises privacy concerns about, e.g., the re-identification risk [SB15]. Thus, it would be highly desirable to secure participants of such a variant querying service, as well as individuals in their genome databases. Furthermore, it can be assumed that many small institutions will not be comfortable in joining the Beacon scheme in its current form, since re-identification risk impacts small databases even more severely.

We address these (genomic) privacy demands by developing a privacy-preserving federated variant query service – eventually an extending the Beacon project – in which the count of matches is learned, while hiding which institution contributed to what extent. Optionally, our solution allows to only release match results if there are more matches than a threshold t. This can mitigate the re-identification risk when querying for rare mutations.

Here, MPC gives us the powerful ability to run arbitrary computations on sensitive data, while protecting the privacy of this data. In this work, we use two parties, called proxies, to achieve high efficiency. We rely on MPC for *secure outsourcing* of genomic data from arbitrarily many

[1] http://genomicsandhealth.org/

sources to two proxies that enable clients to run *private queries* on the data. The proxies are assumed to not collude and therefore learn nothing about the outsourced data or the client's query and its response. We focus on the use-case of privately querying a large, aggregated genome database.

7.1.1 Our Contributions

In this chapter, we provide the following contributions:

We allow private queries to multi-center genome databases. We hide the query, which elements it accesses, and what elements match the query (Sect. 7.3).

Our protocol allows to privately aggregate data from multiple data sources. Usually this is prohibited by patient privacy laws, which aim at protecting sensitive medical data. We retain privacy, while at the same time providing a larger search space that leads to more expressive query results. Due to the generic nature of our protocol we can perform additional multi-property queries that add only negligible overhead to the database lookup (Sect. 7.4).

We develop a custom format (Variant Query Format – VQF) for the lossy storage of genomic variants that also allows *similar* variants to match. The widely used Variant Call Format (VCF) can easily be compressed to VQF, providing a bridge to existing genomic variant databases (Sect. 7.3.2).

We present a prototypical implementation of our protocol in C++ using ABY (cf. Chapt. 3) and achieve practical runtimes for real-world inputs (Sect. 7.5 and Sect. 7.6).

7.2 Preliminaries

In this section we explain our deployment setting and provide an overview of related work.

7.2.1 Deployment Setting

We depict our setting in Fig. 7.1, which is set in the outsourcing scenario, cf. Sect. 2.4.2. An arbitrary number of genomic database providers $\mathsf{DB}^{(i)}$ act as input parties and privately outsource their data to two non-colluding proxies P_0 and P_1, who simply aggregate the received data as one large dataset. Privacy is achieved by using XOR-based secret sharing, as described in more detail in Sect. 7.4. $\mathsf{DB}^{(i)}$ can extend the database by simply sending Boolean shares of their entries $\langle \mathsf{DB}^{(i)} \rangle_0^B$ and $\langle \mathsf{DB}^{(i)} \rangle_1^B$ to the proxies at any time. Updates of existing entries require sending only the bitwise XOR of the updated entries to one of the proxies.

A client C who wants to query entries from the aggregated databases sends an XOR-secret-shared query to both proxies. Privacy-preserving computation is made possible by the protocol

of Goldreich, Micali and Wigderson [GMW87], which enables efficient computation on secret-shared data. Optionally, we allow results to be t-threshold released, i.e., the client receives a query response only if more than t database entries overall match the query criteria.

As a special case, our setting can also be used by a client that runs private queries on a single genomic database held by a server without involving additional parties. For this, the client runs C and P_0, and the server runs $\mathsf{DB}^{(1)}$ and P_1.

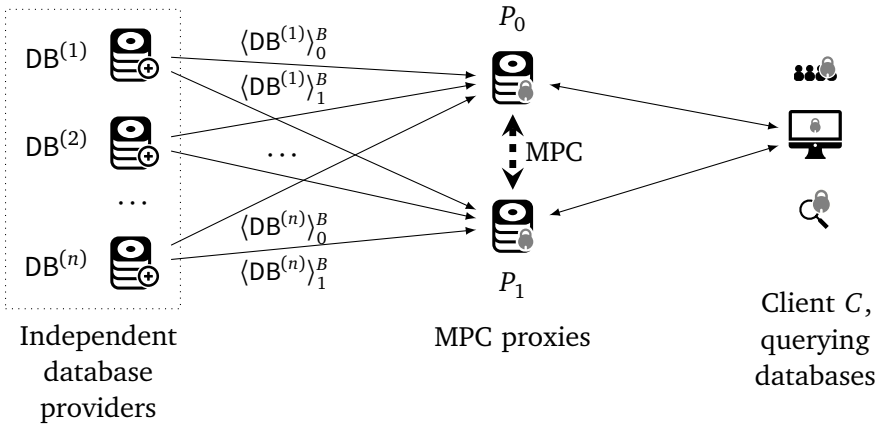


Figure 7.1: Deployment setting overview. The plus sign denotes the federation of genomic databases and the grey lock symbol marks secret-shared data. The client only interacts with the MPC proxies, which hold XOR secret-shared copies of the genetic variant data.

7.2.2 Related Work in Genomic Privacy

There exists a long line of work in genomic privacy, starting with [HHT14]. In the subsequent section we provide an overview of recent work related to our solution. We compare the key aspects of related work with our proposal in Tab. 7.1.

In [HMM17], count queries on fixed-indexed single-nucleotide polymorphisms (SNPs) databases are performed. They convert the database into an index tree structure and perform an encrypted tree traversal using Paillier's additively homomorphic encryption scheme [Pai99] and Yao's garbled circuits [Yao86] for comparison. The queried SNP indices are leaked and the index tree needs to be built by a single and trusted certified institution that must know all the data in plaintext (coming from possibly multiple data sources as in our setup).

In the field of privacy-preserving genomic testing many approaches securely calculate the edit distance (ED) with protected genetic databases. ED is a way of measuring the closeness of two strings by calculating the minimum number of operations to transform between the two. Most of the time, the Levensthein distance is meant by ED and it is defined as the minimum amount of substitutions, insertions and deletions to transmute between the strings. It is

an important measure in genomics to estimate the closeness of two genomes, and it finds application in similar patient querying (SPQ).

The authors of [AHMA16] implemented a form of secure count and ranked similar patient queries, running in seconds. However, their setup is quite different from ours since the query is known in plaintext to each data center, and the output is learned by a central server (CS). The aggregation/sorting is done on the CS and only individual contributions are hidden from the CS and querier.

In [JKS08], the authors developed a privacy-preserving ED algorithm leveraging Yao's garbled circuits. While their technique calculates the exact edit distance, it is computationally infeasible on a whole-genome scale. Their implementation takes several minutes to calculate the ED of just a few hundred-character long strings. [HEKM11] improved those results by up to a factor of 29.

In the whole-genome context it is often more sensible to only approximate the ED, taking advantage of the fact that the human genome is mostly preserved ($>$ 99% of genomic positions) and most variations are simple substitutions. Two important works leveraging these factors are [WHZ+15] and [AHLR18].

The authors of [WHZ+15] test their distributed query system GenSets over, 250 simulated distributed hospitals, each holding 4 000 genome sequences of around 75 million nucleotides each It took their system 200 minutes to search through one million cancer patients.

In [AHLR18], the authors partition the sequences into smaller blocks and then precompute the ED within the blocks Since the human genome shows high preservation, this greatly reduces the number of distinct blocks E.g., for a realistic data set of 10 000 genome snippets of length about 3 500 from a region of high divergence, after partitioning them into 15-letter blocks, for each group of blocks they observed at most 40 unique blocks, instead of the theoretic maximum of 10 000 Still, they could determine the t best matching sequences against a query sequence with high success.

In [KBS14], an attack by Goodrich [Goo09] on genome matching queries was investigated They developed a detection technique against such inference attacks employing zero-knowledge proofs to ensure querier honesty.

The authors of [DFT13] developed a novel method for *secure genomic testing with size- and position-hiding private substring matching* (SPH-PSM) In their setup, a testing facility holds a DNA substring (e.g., a marker) and a patient possesses their full genome sequence The patient sends their homomorphically encrypted genome and public key to the facility, which computes an accumulator applying their substring The still encrypted accumulator is sent back to the patient, who decrypts it to learn the binary answer — whether the substring is included in her genome In the process, the facility doesn't learn anything and the patient doesn't learn the substring or its position in their genome.

The work [ZCD16] presents an innovative and efficient approach to outsourced pattern matching employing a new outsourced discrete Fourier transform protocol called OFFT. It

solves a similar problem to the aforementioned method [DFT13] and scales logarithmically in the string to be matched.

In [SLH+17], a genomic cloud storage and query solution is presented VCF data is symmetrically encrypted and sent to a cloud storage. Using a custom method based on private information retrieval (PIR, see also Sect. 2.5), the data owner, or an authorized entity, can query the cloud storage for a specific variant (utilizing a homomorphically encrypted 0–1–array mask). This solution cannot be generalized to multiple patients as the querier would need access to all VCFs' symmetric keys. A generalization of this work [FES+17] allows computations that are similar to ours and offers strong security guarantees. However, in this case, the runtime depends on the size of the response and thus reveals meta information about the query, which we specifically intend to hide in our work. Padding could solve this issue, but would increase the runtime drastically.

The protocol [BBD+11, Sect. 4.2] uses Authorized Private Set Intersection [CT10] to allow for authorized queries of a list of specific SNPs (a SNP profile, e.g., of SNPs relevant for drug selection and dosage): The querier first submits the SNP profile to a certified institution, which sends back an authorization. The querier can then use this authorization to query for his SNP profile in a patient's genome, learning the matching SNPs. The query is hidden from the patient/database. The protocol [PC17] extends this idea and uses additively homomorphic Elliptic Curve-based El-Gamal [Gam85] to calculate a weighted sum over a set of SNPs, where the weights are authorized by a certified institution.

In conclusion, several solutions have addressed the problem of secure genome queries (see Tab. 7.1), but most of them work directly on the sequence instead of called variants and none grant the easy extensibility that our solution provides and at the same time deliver whole-genome scale protection for all included parties.

7.3 Genetic Variant Queries on Distributed Databases

We consider a federation of databases storing genomic variants in our custom Variant Query Format (VQF, see Sect. 7.3.2). They jointly offer a privacy-preserving *Beacon Service*: individual privacy is guaranteed in the sense that it is not learned which dataset from which data-center contributed to which extent to the final count. Optionally, privacy can be strengthened by enforcing a threshold criterion on the query: the actual count will only be returned if it is larger than a predefined threshold parameter t.

7.3.1 Beacon Network and Potential Extensions

The term *Beacon* stems from the Beacon Network Project, which is "a global search engine for genetic mutations". It was instantiated by the Global Alliance for Genomics & Health to "test the willingness of international sites to share genetic data". The joint service answers queries

Table 7.1: Comparison of features and limitations of related work to our solution.

	[HMM17]	[AHMA16]	[JKS08; HEKM11]	[WHZ+15; AHLR18]	[DFT13; ZCD16; BBD+11; PC17]	[SLH+17]	[FES+17]	Our work
Variants(V)/ Sequences(S)/ Both(*)	S	S	S	S	S	V	*	V
Single trusted party eliminated	✗	✗	✓	✓	✓	✓	✓	✓
Query protected	✗	✗	✓	✓	✓	✓	✓	✓
Top similar patient query	✗	✓	n.a.	✓	n.a.	✗	✗	(✓)[2]
Whole-genome scale	✗	✓	✗	✓	✓	✓	✓	✓
Easy extensibility	✗	(✓)	✗	✗	✗	✗	✗	✓
Output	count	count & similar patients	exact ED	similar patients	substr. match? (Y/N) / matching SNPs / weighted average over SNPs	variant present? (Y/N)	count & sum[3]	(thresh.) count/matches[4]

[2] not implemented

[3] optionally with differential privacy

[4] for an arbitrary (aggregation) function f

of the form "Do you have any genomes with an 'A' at position 100 735 on chromosome 3?" and participants each give a simple *Yes/No* answer.

Privacy Concerns It has been shown in [SB15] that in its original form, Beacon queries are susceptible to re-identification attacks. The authors showed that with 5 000 queries, a person can be re-identified from a Beacon holding 1 000 genomes. Our proposed framework lowers this risk twofold: Firstly, the querier doesn't learn immediately which database contributes to the count. Only in a follow-up consultation can the databases and querier reveal a match should they mutually agree to do so. And secondly, because of an optional t-threshold check in the final step, it is harder for the querier to craft individual-identifying queries. While recent changes to the project's architecture address some privacy concerns by access-control checks[5], those solutions cannot withstand a malicious man-in-the-middle or potential exploits

[5] https://www.elixir-europe.org/services/compute/aai

of access-control vulnerabilities. That is, the project doesn't offer privacy by design since queries are sent in clear to the beacons and the central web-interface, which also learns all beacons' answers during aggregation of the results.

7.3.2 Genomic Variant Representation Format

We developed a machine-friendly and simple format for storing and comparing genetic variants in the context of variant querying that also makes it possible for *similar* variations to match. We call this new format Variant Query Format (VQF). It captures the core features of the commonly used Variant Call Format (VCF) [DAA+11], while abstracting from specifics that are unlikely to match across individual genomes. The VQF incorporates domain-specific knowledge, which makes it very compact and allows us to achieve small sizes and high performance.

The VQF stores a fixed-size dictionary with $\psi = 16$ bit variations at $\phi = 32$ bit addresses, aligned to a known reference genome. A more detailed explanation of the VQF and its encoding is provided in our paper [DHSS17].

7.3.3 Queries

The queries that we support are of the form

```
SELECT f(*) FROM Variants
WHERE ((locus_1, var_1), ..., (locus_m, var_m)) IN Genome
   AND cancertype = X AND ... AND age_min ≤ age ≤ age_max ...
```

when expressed in SQL for illustration purposes. Here, m is the number of variant equalities to check, while the remaining auxiliary queries can be more versatile and include ranges, besides equalities. `f(*)` is an optional aggregation or threshold function. For most of our benchmarks, we choose $m = 5$.

There are typically two ways to analyze a person's genetic variation. The first option is to store the full genetic variation, which leads to around $N = 3$ million entries. The second, and more viable option is to only process a person's exome, consisting of $N = 100\,000$ variations, which contain the most relevant regions for research and disease testing. In our database model, a person's full genetic variation is mapped to N entries of size $\phi + \psi = 48$ bit. Besides this information on genetic variation, the databases can also hold auxiliary data like sex, age, weight and health data like blood pressure or disease indicators. For those fields, our protocol allows for range or threshold conditions in the queries, where desired.

The querier specifies the values for the highlighted parts of the query. These values are secret-shared and remain private, such that the proxies do not learn them. The *structure* of `f` cannot be chosen by the querier and is fixed a priori. However, since we use the ABY framework, cf. Chapt. 3, the function `f` can generally be an arbitrary (aggregation) function

on the bit string of matching genomes, like the identity (simple output of matches) or the count (Hamming weight).

Technically, any query prompts the two proxies to generate a secret-shared bit string representing the matching genomes. As can later be seen in Sect. 7.6.1, this task causes the bulk computation and communication cost. Next, the proxies apply function `f` to this bit string and output it to the querier (or any other predefined party).

Query Scenario For our experiments, we choose a scenario in which the querier receives the count of matches, together with a random query ID. Each database with at least one match receives the corresponding sub-bit mask of the genomes only in their own database together with the same query ID. This way, the database doesn't learn the query which matched their genomes but can contact the querier for a follow-up discussion of the matched patients.

Optionally, we can substitute the simple count by a t-threshold count, which returns the count only if it is larger than t. This would mitigate the re-identification risk as presented in [SB15]. However, since Beacon queries are often used to query for rare mutations, this extension brings its own problems. Our solution can easily realize both options and due to the generic nature of MPC it can also be adapted for additional requirements.

7.4 Our Protocol for Private Genome Variant Queries

Our protocol ensures the privacy of both, the query to a genomic database and its response. At the same time, the entire database is hidden from the two proxies. This matches closely the cloud computing paradigm where computation and data storage is outsourced to a powerful set of machines that are maintained by an external party. In Fig. 7.1 on page 132, we depict our setting where multiple databases $\mathsf{DB}^{(1)}, \ldots, \mathsf{DB}^{(n)}$ outsource their data to two proxies P_0 and P_1 that run the MPC protocol and are assumed to not collude. A client C queries the combined data from all databases for the counts of entries that a) match the query criteria and optionally b) fulfill a pre-defined t-threshold level.

We achieve privacy by using XOR-based secret sharing between the MPC parties, as used in Boolean sharing in the GMW protocol, described in Sect. 2.4.4. More precisely, for every plaintext bit p, we choose a random masking bit r. We send r to P_0 and $s = p \oplus r$ to P_1. The values r and s are called *shares* of the plaintext value p. For bit strings of length ℓ we apply this technique ℓ times in parallel. To further improve communication, we could send to P_0 a single seed for a pseudo-random generator instead of the values r.

We use the protocol of Goldreich, Micali and Wigderson (GMW) [GMW87] to *privately* evaluate a Boolean circuit that corresponds to our functionality, i.e., querying a genome database. GMW operates directly on XOR-secret-shared values. We use the GMW protocol in the offline/online computation model, i.e., an *offline phase* that is preprocessed at any time before the actual private inputs are known. The data from the offline phase is then used in the efficient *online phase* to compute the function on the private data.

7.4.1 Protocol Description

In this section, we describe the phases of our overall protocol and depict it graphically in Fig. 7.2. Optionally, before the protocol, the database providers and proxies agree on a privacy threshold t that defines the minimum amount of matching records that a query response must contain. If a query matches $\leq t$ records, the query response will be the empty set $\emptyset$, i.e., the same as if no record matched the query.

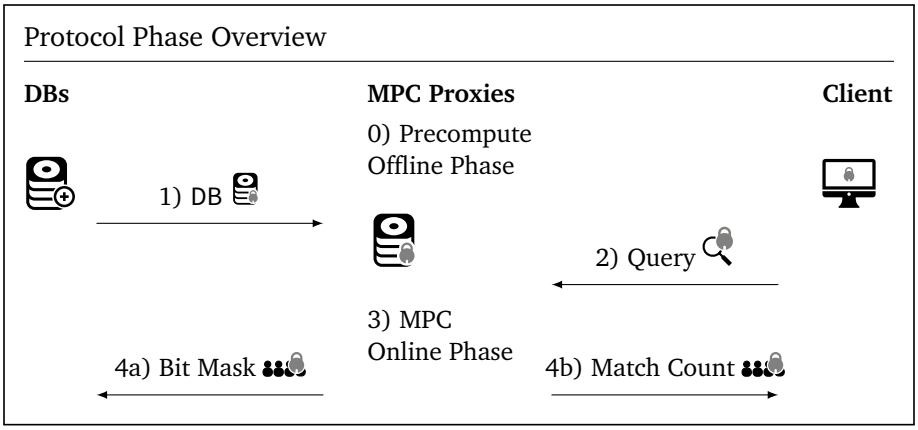


Figure 7.2: Protocol phases. Note that all communication with the MPC proxies happens secret-shared, such that the proxies never gain access to any plaintext.

Phase 0) MPC Offline Phase At any point before the MPC online phase in step 3, P_0 and P_1 precompute the MPC offline phase, which is independent of the actual inputs from other parties.

Phase 1) Database Outsourcing Each database provider DB_i is assumed to hold their data in the VQF format (see Sect. 7.3.2). The provider then generates a random mask of the size of its database and sends the random mask to proxy P_0 and the XOR of the mask and its database to P_1. The proxies concatenate the shares they receive from all database providers and keep track of the mapping of shares to DB providers. Note that this phase needs to be performed only once. The secret-shared database can be queried multiple times. Databases need only send new entries or updates to existing entries if they have changed.

Phase 2) Client Query Client C secret-shares its query between P_0 and P_1 and sends in plain text the type of auxiliary queries it wants to run. Note that we only reveal the *operation* of the auxiliary queries and not the values that are compared with the datasets. As described in Sect. 7.5.2 Private Function Evaluation (PFE) could be used to also hide the query structure at extra cost.

Phase 3) MPC Online Phase The proxies P_0 and P_1 run the MPC protocol on the databases and the query they received. Due to memory constraints, the client query is run on a single patient dataset, consisting of up to 3 million variants at a time. Multiple patients' datasets can be evaluated in separate MPC instances, which can be run sequentially or in parallel. Given enough hardware, this step can be ideally parallelized. The individual outcomes (match / no match) are stored in a bit mask, still secret-shared and thus unknown to the proxies.

In our scenario, after the query was run against all patients' datasets, the resulting bit mask is processed by a final circuit that counts the number of matches, i.e., the Hamming weight, optionally compared to a threshold.

Phase 4) Output Reconstruction Both proxies P_0 and P_1 hold the output of the computation in secret-shared form and send their output shares to C, who computes the XOR and thereby receives the plaintext output. If the optional threshold t privacy is enforced, C will only receive the count if there are more than t matches. Each database also receives the two shares of its sub-bit mask of matches, together with a random query ID for potential follow-up. Reconstruction will reveal the matching genomes, if any, or an all-zero bit string otherwise.

7.4.2 Security Considerations

We discuss the security of our scheme and the attacker model in this section.

The goal of our protocol is to ensure the privacy of the queries clients send to the service, as well as the corresponding responses they receive. At the same time, our protocol ensures the privacy of the genomic databases that outsource their data to our service. We achieve these properties by directly relying on the proven security of the GMW protocol [GMW87; Gol04], which allows to privately evaluate any computable function that is represented as a Boolean circuit. GMW uses XOR-based secret sharing as underlying primitive, which protects private values. In its original form secret sharing offers information-theoretic security since plaintexts are masked with randomness of the same length. The security of outsourcing the computation from multiple parties to two computational proxies was shown in [KMR11]. We use a PRG to expand a short seed to the length of the plaintext, thus reducing information-theoretic security to computational security of the PRG. In our implementation we rely on AES as PRG.

We make the assumption that adversaries behave semi-honestly and corrupt at most one of the two proxies at the same time. The latter corresponds to a non-collusion assumption between the two proxies. Given the semi-honest non-collusion assumptions we can proof that our protocol is secure, since the transcript of every party can be simulated given their respective inputs and outputs. We consider the following cases of semi-honest corrupt parties, malicious clients or external adversaries:

Corrupt Semi-Honest Client /Corrupt Database A corrupt client or corrupt database with input query Q and response R can be simulated by a simulator playing the role of the two proxies, running the GMW protocol. This implies that a corrupt client or database learns no additional information from the protocol execution.

Corrupt Semi-Honest Proxy For a set of databases and a single client query each separate proxy's view consists of a share of the client's query and a share of each database. In all cases XOR secret sharing is used, which makes the corresponding strings appear uniformly distributed and leaks no information about the content. Each proxy's view also contains the other proxy's inputs for the GMW protocol. These are proven not to leak information about private inputs in [Gol04].

Malicious Clients Since the proxies and the client interactively agree on the query structure beforehand, a malicious client can only influence values within the boundaries of the query, i.e., the client can only send an input bit string of the pre-defined length. Malicious clients can send bogus queries with the correct length, which will be processed by the proxies. This leads to corrupt outputs that leak no more information than valid queries. The number of queries a client can send can be controlled by using rate-limiting.

External Adversaries Data in transit is protected from external adversaries by using state-of-the-art secure channels, e.g., TLS, to ensure confidentiality, integrity and authenticity between all communicating parties.

7.5 Implementation

In this section, we describe our implementation decisions and the software design of our application, as well as its limitations. We implemented our protocols within ABY, cf. Chapt. 3. More specifically, we rely on the included Boolean sharing, which is an implementation of the GMW protocol Sect. 2.4.4, based on XOR secret sharing and thus is well-suited for our outsourcing scenario.

7.5.1 Boolean Circuit Design

The GMW protocol operates on Boolean circuits. Our protocol contains two circuit designs that we optimized for a low multiplicative depth in order to minimize the number of communication rounds between the proxies. The biggest circuit is the query circuit that checks if a user query matches the genome of a patient. It consists of an equality gate that compares every patient variant with a query variant. On the circuit level, this is done in parallel on a person's entire variants and all query variants. From this we get 1 bit per patient variant, which is fed into an OR tree that returns 1 if at any position the query matched with a patient's variant. For all query variants the results from the OR trees are fed into an AND tree that returns 1 if all query variants are in the patient's variants. These gate trees are the reason for the circuit

depth logarithmic in the number of compared variants. The circuit also checks for auxiliary patient properties, which are implemented as single equality or comparison gates.

The circuit's output is a single bit that indicates if all query variants are in the person's variant dataset and if all auxiliary queries matched. The circuit output for each patient record is stored (still secret-shared). As soon as all patient queries have been run, the previously stored result shares are fed into a smaller (threshold-)counting circuit, that optionally checks that the count is larger than the threshold t. It consists of a Hamming weight circuit that counts the number of 1-bits, and a comparison gate that controls if a multiplexer gate outputs the string of matches or an all-zero bitstring. Its output is a bit string with one bit per patient. Bits are set to one at the indices where the query matched. If it contains more than t ones, it is output, otherwise an all-zero bit string is output, in case threshold t counting is applied. For N variants, query length Q, A auxilliary queries with bit length ℓ, and entries of length $\phi + \psi$ bit, our circuit depth is $\max\left(\lceil \log_2(\phi+\psi)\rceil + \lceil \log_2 N\rceil, \lceil \log_2 \ell\rceil\right) + 1 + \lceil \log_2 A\rceil$, while there are in total $Q \cdot N \cdot (\phi+\psi) + (Q+A)\cdot(3\ell - \log_2(\ell) - 1)$ AND gates.

7.5.2 Limitations of our Approach

While we only ran queries where *all* conditions must be met, i.e., all conditions are connected with `AND` ($\wedge$) expressions, the ABY framework would easily allow for more complex or nested queries, such as $A \vee \big((B \wedge C) \vee D\big)$. The performance impact would be minimal and would only depend on the size of the formula. PFE built from Universal circuits [Val76; KS08a; KS16; LMS16; GKS17] could be used to also hide the structure of the formula at extra cost.

The translation of variants into 16 bit strings (see Sect. 7.3.2) certainly is a limitation and cannot reflect the full spectrum of possible variations. However, as elaborated before, we used this compression as a way to match similar variations while only using strict equality queries.

7.6 Benchmarks

In this section we provide benchmark results of our implementation. We performed runtime benchmarks of our MPC implementation on two identical desktop computers with 16 GiB RAM and a 3.6 GHz Intel Core i7-4790 CPU, connected via a local 1 Gbps network. For all measurements we instantiate the parameters to achieve a symmetric security level of $\kappa = 128$ bits. All results are averaged over 25 iterations. The provided communication numbers are the sum of sent and received data of one MPC proxy. In the following section we use the term offline phase to refer to step 0) from Fig. 7.2, while online phase refers to step 3). We did not measure the time for steps 1), 2), and 4), i.e., the conversion and transmission of databases and query/response, as these are simple and efficient plaintext operations and data transmissions over a TLS connection that scale linearly with the size and available bandwidth.

7.6.1 Variant Query Performance

In this section we measure the performance of running a query with a certain length against a person's dataset with a given number of variants. As default parameters we use $N = 100\,000$ variants, query length $= 5$, and $\phi + \psi = 48$ bit, which corresponds to a query on a person's exome. This data point is marked with a gray circle in each figure on page 143. In Fig. 7.3 we show the runtimes of the offline and the online phase for *varying number of patient variants* with queries of length 5. In Fig. 7.4 we fix the number of a patient's variants to $N = 100\,000$ entries and show how the runtimes scale for *varying query length*. Fig. 7.5 shows how a *varying entry bitlen* ($\phi + \psi$) influences the protocol runtimes. We provide the corresponding detailed numbers in Tables 7.2, 7.3, and 7.4.

In all cases the offline and online runtime and circuit size (number of AND gates) increase linearly with the database size, query length, and entry bitlength. The circuit depth, i.e., the number of communication rounds between the two proxies, scales only with the logarithm of the input sizes.

Regarding the auxiliary queries, we fixed five different equality and range queries, which didn't have any noticeable performance impact. They are thus omitted in the following discussions.

For querying a patient's exome variants ($N = 100\,000$), assuming a query of length 5 and our proposed entry format with key length $\phi = 32$ bit, and value length $\psi = 16$ bit, we achieve an offline runtime of 3.4 s and an online runtime of 178 ms. In this case we need to transfer 733 MiB in the offline phase and the online phase requires 28 communication rounds with a total transmission of 11 MiB. The circuit for these parameters consists of 24 million AND gates. Using the same parameters to query a patient's full genome ($N = 3\,000\,000$) requires an offline and online runtime of 99.5 s and 4.6 s, respectively.

Our performance is comparable to the single-variant query in [SLH+17], which takes 2.4 – 4.3 s online runtime for 5 million variants. However, their query is not extensible and can only answer whether a single variant is present, without the option for further privacy-preserving aggregation.

7.6.2 Count Performance

The circuit (`f` in Sect. 7.3.3) that determines the total number of matches and compares this to the privacy threshold t is very small. We are computing the Hamming weight using a Boyar-Peralta counter [BP06] with logarithmic depth. For processing the results of 100 000 patient records, the t-threshold count circuit consists of 100 040 AND gates and has a depth of 22. It requires 75 ms (55 ms) runtime and 439 KiB (2 124 KiB) communication in the online (offline) phase. Communication, runtime, and circuit size scale linearly with the input size, while circuit depth grows logarithmically. Since these numbers are negligible for the total runtime, we omit a more detailed analysis at this point.

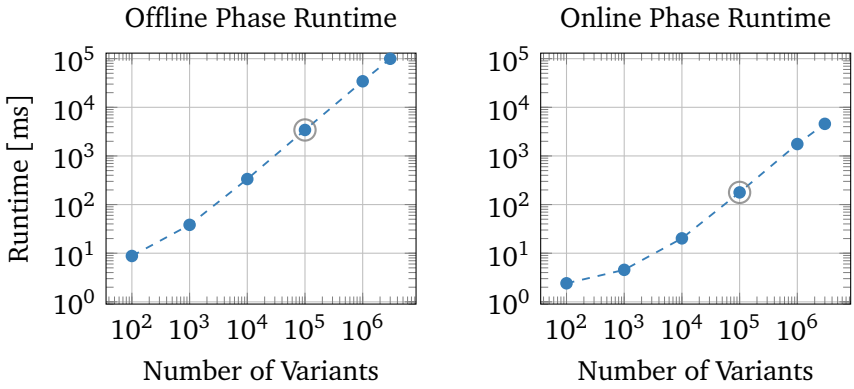


Figure 7.3: Offline and online runtime in ms for *varying number of variants per patient* and fixed key length $\phi = 32$ bit, value length $\psi = 16$ bit, and a query with 5 components.

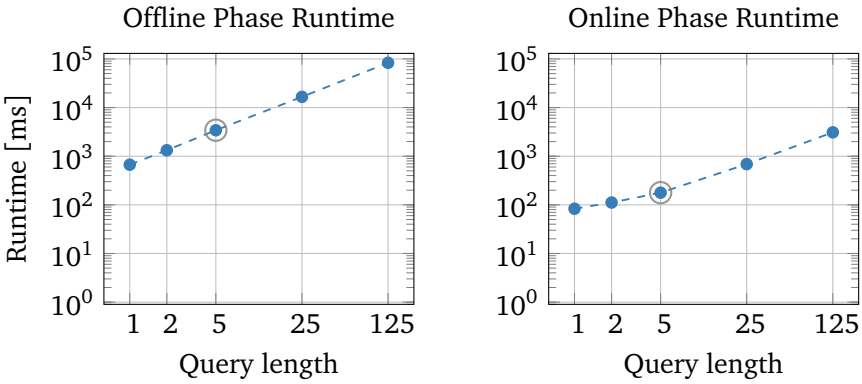


Figure 7.4: Offline and online runtime in ms for *varying query length* and fixed key length $\phi = 32$ bit, value length $\psi = 16$ bit, and variant count of $N = 100\,000$ entries.

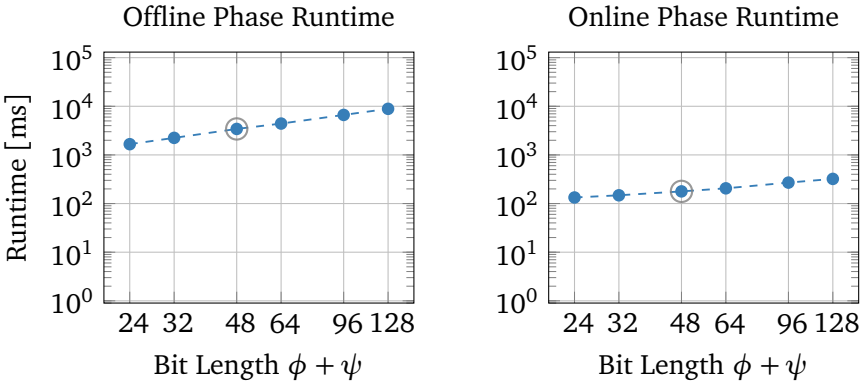


Figure 7.5: Offline and online runtime in ms for *varying total element size* $\phi + \psi$ at a fixed variant count of $N = 100\,000$ entries and query length of 5.

Table 7.2: Benchmark results and circuit properties for *varying variant count* at fixed key length $\phi = 32$ bit, value length $\psi = 16$ bit, and query length 5.

Variants N	Query Length	$\phi + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100	5	48	$2.4 \cdot 10^4$	18	9	1	2.4	0
1 000	5	48	$2.4 \cdot 10^5$	21	38	7	4.5	0
10 000	5	48	$2.4 \cdot 10^6$	25	335	73	20.2	1
100 000	5	48	$2.4 \cdot 10^7$	28	3 420	733	177.9	11
1 000 000	5	48	$2.4 \cdot 10^8$	31	34 297	7 325	1 756.2	114
3 000 000	5	48	$7.2 \cdot 10^8$	33	99 507	21 975	4 567.7	343

Table 7.3: Benchmark results and circuit properties for *varying query length* at fixed key length $\phi = 32$ bit, value length $\psi = 16$ bit, and variant count $N = 100\,000$.

Variants N	Query Length	$\phi + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100 000	1	48	$4.8 \cdot 10^6$	27	669	146	83.2	2
100 000	2	48	$9.6 \cdot 10^6$	27	1 327	293	111.5	5
100 000	5	48	$2.4 \cdot 10^7$	28	3 420	733	177.9	11
100 000	25	48	$1.2 \cdot 10^8$	29	16 629	3 662	687.5	57
100 000	125	48	$6.0 \cdot 10^8$	32	83 141	18 312	3 096.9	286

Table 7.4: Benchmark results and circuit properties for *varying total element size* $\phi + \psi$ at fixed query length of 5 and variant count $N = 100\,000$.

Variants N	Query Length	$\phi + \psi$ [bit]	#AND Gates	Circuit Depth	Offline Phase [ms]	Offline Phase [MiB]	Online Phase [ms]	Online Phase [MiB]
100 000	5	24	$1.2 \cdot 10^7$	27	1 662	366	133.2	6
100 000	5	32	$1.6 \cdot 10^7$	27	2 241	488	147.9	8
100 000	5	48	$2.4 \cdot 10^7$	28	3 420	733	177.9	11
100 000	5	64	$3.2 \cdot 10^7$	28	4 409	977	205.6	15
100 000	5	96	$4.8 \cdot 10^7$	29	6 640	1 465	269.6	23
100 000	5	128	$6.4 \cdot 10^7$	29	8 860	1 953	321.3	31

7.6.3 Conclusions from the Benchmarks

We consider our solution practical for typical private genome queries. While the computation of responses for databases with thousands of patients still do not answer instantaneously, we can run these private queries over night or increase throughput by running them in parallel on dedicated hardware and faster networks. As we can see from our performance evaluation, both runtime and communication complexity scale linearly with the input size. Our circuit constructions are optimized for use with the GMW protocol and their depth grows only logarithmically with increasing input size. Network traffic and memory requirements are well within the limits of modern hardware.

The generation of the bit string representing the matching genomes takes the bulk computation and communication cost, while the cost for evaluating the auxiliary conditions, aggregation, and threshold comparison is negligible. Therefore, possibly complex and versatile post-processing functions can be applied to the matches thanks to the use of generic MPC techniques. This ability sets our system apart from related works in this field.

7.7 Conclusion

In this chapter we have presented a new, privacy-preserving protocol to allow multi-center variant queries on genomic databases. The achieved performance renders this approach applicable in real-world scenarios with some dozens of centers. Full genome studies are supported based on the state-of-the-art VCF format. Our approach leverages the custom VQF, which can be built from existing VCF data. Variants must be called against a pre-defined reference genome, global for federated data analysis platform. An interesting and demanding research question immediately arises: how would one use our (and many previously developed) genomic privacy techniques on genomic data while facing the problem of different reference genomes? The simple answer is to regenerate the genomic data against the new reference genome via the same pipeline. But this approach might not always be feasible or possible, if the pipeline is not fully automated or recorded. Note that this applies to almost all previous work where genomic data from different patients is compared, as is the case in the Beacon project. This “transcription” to other reference genomes is beyond the scope of the present work but will be addressed in the future.

Another open problem is how to effectively mitigate the re-identification risk as presented in [SB15], while still handling queries of rare mutations in a sensible way. We described two query types which our framework supports: a regular count, which is susceptible to the aforementioned re-identification risk, even though to a lesser extent, since the querier doesn’t learn in which databases the matches occurred, and a threshold-t-count, which only outputs the count if it is larger than t. While the latter method provides more privacy, it may render the system unusable for very rare mutations.

Part III

Private Information Retrieval and Applications

8 Improving Multi-Server PIR for Anonymous Communication

Results published in:

[DHS14] D. DEMMLER, A. HERZBERG, T. SCHNEIDER. **"RAID-PIR: Practical Multi-Server PIR"**. In: *6. ACM Cloud Computing Security Workshop (CCSW'14)*. Code: https://encrypto.de/code/RAID-PIR. ACM, 2014, pp. 45–56.

[DHS17] D. DEMMLER, M. HOLZ, T. SCHNEIDER. **"OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks"**. In: *15. International Conference on Applied Cryptography and Network Security (ACNS'17)*. Vol. 10355. LNCS. Code: https://encrypto.de/code/onionPIR. Springer, 2017, pp. 599–619. CORE Rank B.

8.1 Introduction

Nowadays, the need for confidential communication and privately retrieving data from the Internet is bigger than ever. Data can be encrypted during transport, thus preventing a man-in-the-middle attacker from getting access to the data. However, a content provider needs to know what files a user requested, in order to deliver the content, and even if the contents are encrypted, large amounts of metadata are collected. Therefore, the user must trust the content provider to keep his request safe and to be protected against attacks. This trust cannot easily be established, especially since today many websites are not directly hosted by the content provider, but instead located in a CDN or on machines in the cloud. Furthermore, even if content providers are trustworthy, they can be forced, e.g., by government agencies, to reveal information about user queries.

It has been shown that there is a large demand for communication systems that protect the anonymity of their users and that do not leak metadata [Lan15; MMM16]. This is of special importance, e.g., for people in suppressive regimes, for whom trying to interact with government-critical organizations can be a tremendous risk.

A solution to these problems is *Private Information Retrieval (PIR)*, which allows to protect the privacy of the users' queries and their metadata. While PIR schemes with a single server exist, today's most efficient PIR schemes use multiple servers with the assumption that not all

of them collude. Multi-server PIR is particularly suited to distribute trust between multiple parties, as these could be run on different machines operated by different cloud providers, different administrators and/or in different regions or jurisdictions. Our schemes further improve the efficiency of known multi-server PIR protocol.

8.1.1 Outline and Contributions

After introducing our preliminaries in Sect. 8.2, we present our main contribution in Sect. 8.3: We present RAID-PIR, a family of multi-server PIR schemes that improve upon and generalize Chor et al.'s linear summation PIR protocol [CGKS95] and have several desirable properties. We use the name RAID-PIR, since our work shares ideas with RAID storage systems. Redundant Array of Inexpensive Disks (RAID), introduced in [PGK88], is a method to virtualize data storage that combines multiple disks into one logical unit. Our multi-server PIR scheme has similar properties to RAID systems: the data is distributed to multiple servers (comparable to disks in a RAID) for better performance, where each server stores only parts of the database. The data from the different servers is combined in a simple, efficient operation. The use of multi-server PIR implies that the entire system is trustworthy, even if some but not all of the servers are corrupted. RAID-PIR has a *balanced workload* amongst all participating servers, *reduces the communication complexity*, and uses only highly efficient cryptographic primitives, namely a Pseudo-Random Generator (PRG). RAID-PIR has several parameters that allow adaption to the specific deployment scenario and trust assumptions: the number of available servers and the maximum number of servers that can collude. In RAID-PIR, we *reduce the storage requirements for the servers* as each server is comparable to a disk in a striped RAID that stores only a part of the database and computes only a part of the answer to a user's query. The user can even *query multiple blocks of data in parallel* for higher efficiency. Communication and computation requirements are reduced – in particular the upstream communication from the client to the servers. A detailed analysis of the properties of our schemes is given in Sect. 8.4. We provide a publicly available open-source Python implementation on GitHub[1] that we describe in Sect. 8.5, and perform extensive performance benchmarks and evaluation of our PIR schemes in Sect. 8.6.

Our PIR schemes can be implemented with low memory for the client. This makes them well-suited for resource-limited devices, such as web browsers, smartphones, embedded systems, or smartcards. For this, we instantiate the PRG with the AES block cipher in counter mode and iteratively compute the queries. In Sect. 8.7, we discuss several deployment aspects, including preserving privacy when reading multiple blocks concurrently, identifying the necessary blocks for a given query/file, and ensuring object integrity and availability in spite of server failures. Our approach of building such features on top of RAID-PIR allows us to maintain a very simple and extremely efficient PIR design, while providing practical solutions to failures, which other designs solve by non-trivial extensions and modifications to the PIR scheme, e.g., [DGH12; OG11; Gol07].

[1] https://encrypto/code/RAID-PIR

In Sect. 8.8 we propose an anonymous communication system called OnionPIR that builds on PIR to privately distribute keys.

8.2 Preliminaries

In this section we explain our deployment scenario and introduce the used notations. More detailed background informaton on PIR is provided in Sect. 2.5.

8.2.1 Notation

The public database is denoted as DB. We denote the content provider as $\mathcal{CP}$, the k PIR servers as $\mathcal{S}_i$, with $1 \leq i \leq k$, and the client as $\mathcal{C}$. DB consists of B blocks of size b bits each. The block at position j is denoted as block_j. We partition the B blocks into k so-called *chunks*, each containing $\lceil B/k \rceil$ blocks of the database.

The protocol makes operations over vectors of size B; in particular, for $c \in [1, B]$, the (B-bits) elementary vector e_c has a single bit set to one at position c, and all other bits are set to zero. For a given B-bit vector α, the notation $\alpha[i]$ refers to the i-th chunk of α, i.e., the $\lceil B/k \rceil$ bits beginning with $(i-1) \cdot \lceil B/k \rceil$, as described in Sect. 8.3.1.

From Sect. 8.3.2 on, our PIR schemes also use a *redundancy parameter* r, with $2 \leq r \leq k$, such that the scheme is secure as long as the number of colluding servers is less than r.

Finally, in Sect. 8.3.3, we improve the performance using a pseudo-random generator PRG, with symmetric security parameter κ. For simplicity, we use $\mathrm{PRG}(s, i)$ to denote the i^{th} 'block' of $\lceil B/k \rceil$ bits output by PRG for seed $s \in \{0,1\}^\kappa$.

8.2.2 Deployment Scenario for RAID-PIR

A content provider $\mathcal{CP}$ holds a database DB and distributes it to k PIR servers $\mathcal{S}_i$ with $1 \leq i \leq k$. This is common practice for distributing data over the Internet in a large scale for the purpose of load balancing and scalability. The direct communication between clients and $\mathcal{CP}$ should be kept to a minimum. A client $\mathcal{C}$ wants to retrieve data from $\mathcal{CP}$ and gets forwarded to several servers that deliver the requested data to him. The PIR protocols we consider here, are single-round protocols where $\mathcal{C}$ sends a query q_i to server $\mathcal{S}_i$ from whom he receives the answer a_i. An example setting with $k = 3$ servers is depicted in Fig. 8.1.

The properties that a PIR scheme must satisfy are that the client $\mathcal{C}$ can correctly recover his desired data (*correctness*), but neither the content provider $\mathcal{CP}$ nor any combination of less than r servers $\mathcal{S}_i$ learns anything about the data the client is interested in from the client's queries or their corresponding responses (*security*).

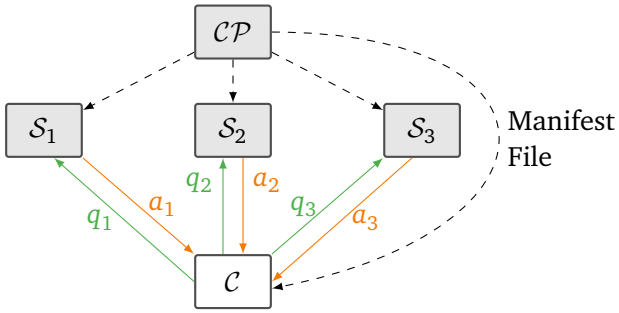


Figure 8.1: Example PIR setting with $k = 3$ PIR servers $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. q_i denotes a query to, a_i denotes a response from $\mathcal{S}_i$. Dashed lines represent communication in the PIR setup phase (see Sect. 8.2.3).

8.2.3 PIR System Phases

RAID-PIR has two phases, a setup phase run between the content provider and the servers, and a PIR phase run between the client and the servers as described next.

Setup Phase The content provider $\mathcal{CP}$ sets up the database DB of his files by partitioning all available data into B blocks of size b. For this, $\mathcal{CP}$ creates a *manifest file* that maps every file to one or multiple blocks, optionally using the method described in Sect. 8.3.6. Every block has a unique index and its content can optionally be hashed with a secure hash function in order to guarantee its integrity and thereby allowing to identify malicious servers. The manifest file is static and identical for all clients and servers. The partitioning is done in a compact way, such that files can start in the middle of a block, wasting no storage space. The files to be distributed and the manifest file are sent to all PIR servers. The PIR servers can individually apply precomputation, as explained in Sect. 8.3.5. The client $\mathcal{C}$ requests the manifest file either the content provider directly or from some delegated content povider, that could also be a PIR server.

PIR Phase With the manifest, $\mathcal{C}$ determines what data is available and for creating queries to the servers. With the manifest file, $\mathcal{C}$ identifies the desired blocks and runs the PIR protocol with the servers (cf. Sect. 8.3), which consists of sending a message to and receiving a response from each PIR server. All received responses are combined with an XOR operation in our scheme to reconstruct the plaintext block(s).

8.3 RAID-PIR

In this section, we present RAID-PIR and the optimizations that we introduce in order improve the efficiency of Chor et al.'s protocol [CGKS95]. First, we introduce the protocol changes and notation in Sect. 8.3.1. We then show how to reduce the server storage and workload

in Sect. 8.3.2. In Sect. 8.3.3, we improve the query size and in Sect. 8.3.4 we allow to request multiple blocks in a single query. We speed up the generation of the PIR responses at the server-side by using precomputations as shown in Sect. 8.3.5, while the final optimzation in Sect. 8.3.6 achieves a significant speedup when querying multiple blocks in parallel by using a database layout where blocks are uniformly distributed.

8.3.1 Protocol Overview

Our work is based on Chor et al.'s linear summation PIR for multiple servers [CGKS95], which we denote as CGKS in the following. A detailed description of CGKS can be found in Sect. 2.5.2. Following earlier work, e.g., [Cap13], we query blocks of data instead of single bits to improve efficiency. The overall goal is to allow privacy-preserving retrieval of data and outsourcing of workload from $\mathcal{CP}$ to the PIR servers $\mathcal{S}_i$. After introducing a slightly modified variant of the original CGKS scheme, we present our improved PIR schemes.

Initialization of RAID-PIR

Before clients can privately retrieve data, the content provider $\mathcal{CP}$ has to setup the database DB of his files and distribute it to the PIR servers. This is realized by partitioning all available data into B blocks of size b bits and sending them to the PIR servers. The mapping of actual files to blocks is discussed in Sect. 8.7.2.

A Variant of CGKS [CGKS95]

In the following, we describe a slightly modified variant of the original CGKS scheme that has the same properties in terms of complexity, but with the structure of our improved protocols presented in the following sections. For completeness, we provide a description of the original CGKS protocol in Sect. 2.5.2.

As in [CGKS95], client $\mathcal{C}$ is interested in privately querying block_c at index c and represents this plaintext query as the elementary vector e_c (i.e., a vector of B bits where the c-th bit is set to one and all other bits are zero). The queries received by each PIR server appear random; only the XOR of all queries equals to the plaintext query e_c. However, differently from [CGKS95], in our variant all servers and queries are 'symmetrical', as follows.

As shown in Fig. 8.2, we partition each query into k so-called *chunks*, where k is the number of servers. A chunk in a query, i.e., a column in Fig. 8.2, corresponds to $\lceil B/k \rceil$ adjacent *blocks* in the DB and thus contains $\lceil B/k \rceil$ bits. The query sent to $\mathcal{S}_i$ contains random bits $\mathsf{rnd}_i[x]$ for all the chunks $x \neq i$. Chunk i is denoted flip_i, and is computed to cancel out all other (randomly chosen) chunks $\mathsf{rnd}_j[i]$ sent to other servers $j \in \{1,\ldots,k\} \setminus i$, leaving exactly $e_c[i]$, i.e.,

$$\mathsf{flip}_i \leftarrow e_c[i] \oplus \bigoplus_{j \in \{1,\ldots,k\} \setminus i} \mathsf{rnd}_j[i],$$

where $e_c[i]$ is the i-th chunk of elementary vector e_c.

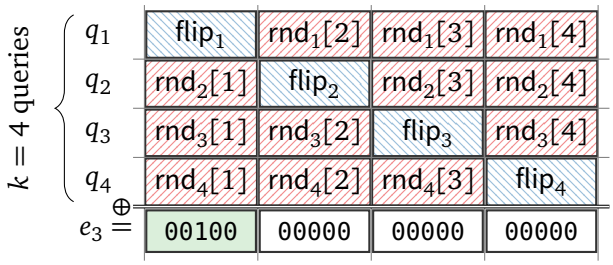


Figure 8.2: CGKS: The queries q_i sent by the client to servers $\mathcal{S}_i$ and their XOR. In this example we use $k = 4$ servers and $B = 20$ blocks. The block that the client is interested in has index 3, as e_3 the third bit set to 1.

Each request q_i that $\mathcal{C}$ sends to server $\mathcal{S}_i$ for $i \in \{1, \ldots, k\}$ contains one flip chunk and $k-1$ randomly chosen rnd chunks, and hence has a total length of B bits, as in the original CGKS scheme. The idea of distributing the flip chunks to multiple queries can be seen as analogous to RAID-5 (Rotating Parity), where parity information is distributed over all disks (see our paper [DHS14] for a background on RAID levels).

As in [CGKS95], the servers' responses have a length of b bits each, and are the XOR of all blocks that the user requested in his query, i.e., if the bit at index j was set in the client's query, the server XORs block_j into his response. When the client has received the replies from all k servers he calculates the XOR of all responses and gets block_c, as all other blocks are contained an even number of times and cancel out due to the XOR.

8.3.2 Redundancy Parameter r

As our first optimization of Chor et al.'s protocol we introduce the *redundancy parameter* r with $2 \leq r \leq k$, which sets the minimum number of servers that need to collude in order to recover the block that is queried. In our protocol, depicted in Fig. 8.3, the redundancy parameter specifies the number of chunks in each query and how often the chunks overlap throughout all queries, thus setting the storage requirements of the servers and the security of our scheme. Each of the k servers stores only $(r/k) \cdot B$ blocks of the DB, and each query now consists of r chunks, with a length of $\lceil B/k \rceil$ bits each. A small r parameter allows to reduce the percentage of the DB each server has to store and the length of each query to a fraction of r/k, but also reduces the protection against colluding servers. Hence, the redundancy parameter r allows a trade-off between storage/communication and security and can be chosen in accordance with the deployment scenario and trust assumptions. For $r = k$ we obtain exactly the variant of the original CGKS scheme described in Sect. 8.3.1; for better performance, we can use more servers (with a fixed r).

Our idea stems from the striping technique used in RAID systems, where data is distributed over multiple disks in order to improve efficiency. However, to achieve security we have to also rely on mirroring, in order to protect against colluding servers. Our partially overlapping structure of chunks is comparable to a hybrid of RAID-0 (Striping) and RAID-1 (Mirroring).

Such a hybrid RAID construction allows to trade-off protection against data loss in case of a disk failure and performance. An example of the partitioning into chunks and the use of the redundancy parameter r is depicted in Fig. 8.3.

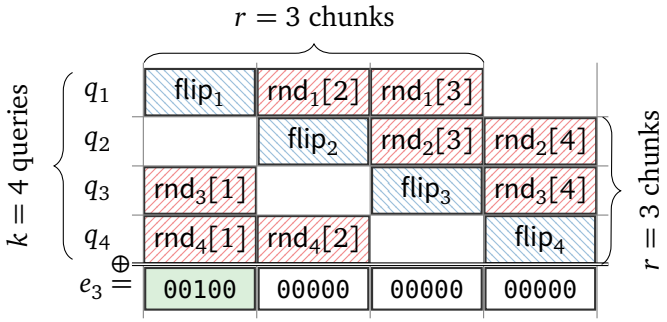


Figure 8.3: PIR with Redundancy Parameter r: Queries q_i sent by the client to server $\mathcal{S}_i$ and their XOR. In this example we use $k = 4$ servers, redundancy parameter $r = 3$, and $B = 20$ blocks. The block that the client is interested in has index 3, as e_3 the third bit set to 1.

8.3.3 SB: Single Block Queries with Seed Expansion

For the next optimization we use a pseudo random generator (PRG) to further improve the communication complexity by reducing the query size. The flip chunk in each query is chosen as before and sent as a bit string, while the remaining $r-1$ rnd chunks are generated from a PRG and expanded from a seed s of length κ bits, where κ is the symmetric security parameter (set to 128 bit in our implementation). The seed expansion is depicted in Fig. 8.4.

This technique reduces the communication complexity, as soon as the symmetric security parameter κ is smaller than $B(r-1)/k$, at the cost of the evaluation of few symmetric cryptographic operations, and is very effective for large databases with a high number of blocks B. The efficiency of the PIR scheme is improved, since typically an end user's upstream is significantly slower than his downstream. Therefore, reducing the amount of data a user has to send to the servers will reduce the overall protocol runtime. We argue that, for large number of blocks B, the additional costs of evaluating a small number of symmetric cryptographic operations, e.g., by instantiating the PRG with AES, are very low compared to the bandwidth savings, especially due to the massive increase of computational power and the availability of the AES-NI instruction set in today's CPUs. See performance evaluation results in Sect. 8.6.

The servers' replies are identical to the ones in the original protocol. The trick is to flip the bit at the beginning of the query, in the flip chunk that is not generated by a PRG. All queries start with such a flip chunk, and therefore all servers are equally likely to receive the query with the flipped bit.

We can combine this technique with the redundancy parameter to further increase efficiency. We refer to the scheme that uses chunks, the redundancy parameter r, and the seed expansion as single block scheme SB. A formal description of SB is given in Algorithm 8.1 (variant SB).

From Sect. 8.3.3 on, our schemes use a symmetric security parameter κ.

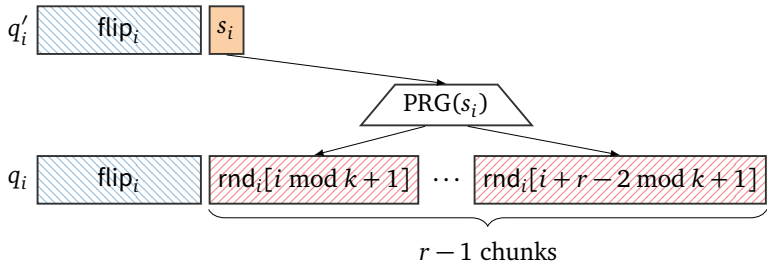


Figure 8.4: SB: Query expansion from seed s_i.

Fixed Session Seeds Instead of including one seed to every query message that is sent from the client to the servers, a more effective strategy is to choose a fixed seed per server for one session, comparable to the master secret of a TLS connection. This reduces the amount of data sent from $\mathcal{C}$ to $\mathcal{S}$ and can also improve the runtime of the PRG if AES is used, since the key schedule has to be done only once. Additionally, the server can precompute a certain amount of PRG outputs to further reduce the online runtime, however, in order to protect forward secrecy of our protocol, a seed should only be used in one session and not be fixed per client-server pair for a longer period of time.

8.3.4 MB: Multiple Block Queries

Finally, we extend the protocol to allow $\mathcal{C}$ to request multiple blocks with a single query. For this, the servers reply with one block per query chunk and calculate the XOR of each response block only within each chunk. The size of the reply from each server to $\mathcal{C}$ is increased from b bits to $r \cdot b$ bits. This approach has the limitation, that the requested blocks must be located in different locations of the DB, as they must be queried in different chunks. Every chunk can contain at most one block of data, comparable to the original scheme, where every block has to be queried separately. However, we argue that the assumptions of blocks being located in different chunks is practical, especially for requests of a large amount of data. An optimization that targets this is proposed in Sect. 8.3.6. An example of the parallel query is depicted in Fig. 8.5 (variant MB).

We refer to the scheme, that incorporates all of our optimizations and that allows to query up to k blocks with one query as multi block scheme MB. A formal description of MB is given in Algorithm 8.1.

This optimization is again inspired by RAID, since blocks can be read from multiple disks in parallel. However, similar to RAID-5 (Rotating Parity) and RAID-0 (Striping), blocks can only be queried in parallel if they are located on different disks.

Input: (SB) single block block_c or (MB) k blocks $\mathsf{block}_c[i]$ for $i \in \{1,\ldots,k\}$

1. $\mathcal{C}$ randomly picks the seeds $s_i \in_R \{0,1\}^\kappa$ for $i \in \{1,\ldots,k\}$.
2. $\mathcal{C}$ expands each seed s_i to generate the $r-1$ chunks $\mathsf{rnd}_i[j] \leftarrow \mathrm{PRG}(s_i, j)$ for $j \in \{(i \bmod k)+1,\ldots,(i+r-2 \bmod k)+1\}$.
3. For each i, $\mathcal{C}$ sets the chunk flip_i as the XOR of the $r-1$ chunks $\mathsf{rnd}_j[i]$ in column i: $\mathsf{flip}_i \leftarrow \bigoplus_j \mathsf{rnd}_j[i]$ with $j = (i-1 \mod k)+1, (i-2 \mod k)+1, \ldots$.
4. (SB) $\mathcal{C}$ identifies the flip chunk that contains block_c (the block $\mathcal{C}$ is interested in) and flips the bit at the corresponding position.
   (MB) $\mathcal{C}$ identifies all flip chunks that contain a $\mathsf{block}_c[i]$ he is interested in and flips the bit at the corresponding positions.
5. $\mathcal{C}$ sends the queries q_i' consisting of one flip chunk and one seed s_i to the servers $\mathcal{S}_i$.
6. $\mathcal{S}_i$ expands his seed s_i to generate $r-1$ random chunks $\mathsf{rnd}_i[j] = \mathrm{PRG}(s_i, j)$ for $j \in \{(i \bmod k)+1,\ldots,(i+r-2 \bmod k)+1\}$ and gets his full query q_i, as depicted in Fig. 8.4.
7. (SB) $\mathcal{S}_i$ calculates his answer $a_i \leftarrow \bigoplus_{x \in q_i} \mathsf{block}_x$ and sends answer a_i to $\mathcal{C}$.
   (MB) For each of the r chunks j, $\mathcal{S}_i$ calculates $a_i[j] \leftarrow \bigoplus_{x \in q_i[j]} \mathsf{block}_x$ and sends all $a_i[j]$ blocks to $\mathcal{C}$.
8. (SB) $\mathcal{C}$ calculates the plaintext block by XORing the k answers: $\mathsf{block}_c \leftarrow \bigoplus_{1 \le i \le k} a_i$.
   (MB) For each chunk j, $\mathcal{C}$ calculates the plaintext block by XORing the r answers in column j :
   $\mathsf{block}_c[j] \leftarrow \bigoplus_i a_i[j]$ with $i = (j-1 \mod k)+1, (j-2 \bmod k)+1, \ldots$.

Algorithm 8.1: Description of our PIR schemes for retrieving a single block (SB, cf. Sect. 8.3.3) or multiple blocks in parallel (MB, cf. Sect. 8.3.4). See Sect. 8.2.1 for notation.

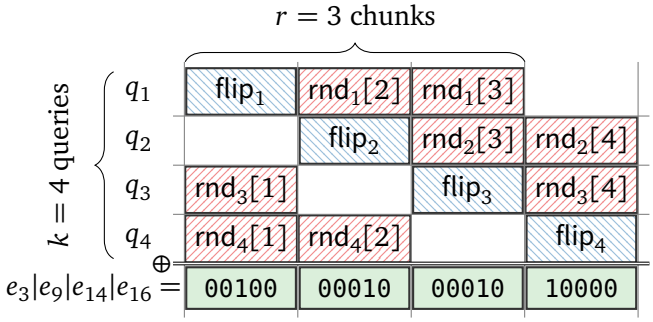


Figure 8.5: MB: The queries q_i sent by the client to server $\mathcal{S}_i$ and their XOR. In this example we use $k = 4$ servers, redundancy parameter $r = 3$, and $B = 20$ blocks. The client queries for the blocks 3, 9, 14, and 16 in parallel.

8.3.5 Precomputation using the Method of four Russians

The so called *Method of four Russians* [ADKF70], is a precomputation technique for efficient matrix multiplication. This method can be used to reduce the computation time of the PIR servers since RAID-PIR uses these multiplications to compute PIR query responses. This yields lower latency and higher throughput at the (low) cost of a preprocessing phase for the servers and increased memory requirements. The precomputation has to run only once in the setup phase (cf. Sect. 8.2.3) before the queries can benefit from it.

In the following, we assume the matrix multiplication $\mathsf{Z} = \mathsf{X} \cdot \mathsf{Y}$, done in the field with two elements $\mathbb{F}_2$ with the goal to speed up the computation by precomputing Z, or parts thereof. Applied to PIR, X would be a set of queries, Y is DB and Z are the query responses. While there are in total 2^n possible combinations of rows in Y, we can precompute them all with just n XOR operations, when using a Gray code to reorder them such that two consecutive combinations only differ by a single bit. For each of the 2^n combinations, it is now sufficient to XOR the last precomputed result with the row of Y that changed in the Gray code. The binary Gray code g of a number m can be calculated as $g = (m \oplus (m \gg 1))$. In Fig. 8.6 we depict the calculation of LUT entries for a given example matrix Y^*.

Instead of precomputing a LUT for the full matrix Y, it is divided into smaller groups of $t \in \mathbb{N}$ rows each. The precomputation is done within these n/t groups using the Gray code as described above, resulting in a computational complexity of $\mathcal{O}\left(2^t \cdot n^2/t\right)$. If $t = \log_2(n)$ the complexity simplifies to $\mathcal{O}\left(n^3/\log_2(n)\right)$, resulting in a speedup of factor t, compared to traditional matrix multiplication.

Y is divided into groups horizontally, while the matrix X has to be divided into vertical groups of t columns. To multiply the two $n \times n$ matrices, X is now traversed column-wise, processing a group of t columns at a time. For each group, the corresponding LUT can now be created for t rows of Y and later a lookup can be performed for all t-bit parts of the n rows of X in this group. This procedure is depicted in Fig. 8.7. We note that this optimization is generic

$$Y^* = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Gray Code	Changed Bit	Result
0000	0	0000
0001	4	$1011 = 0000 \oplus Y^*[4,:]$
0011	3	$0010 = 1011 \oplus Y^*[3,:]$
0010	4	$1001 = 0010 \oplus Y^*[4,:]$
0110	2	$0101 = 1001 \oplus Y^*[2,:]$
...	...	...

Figure 8.6: Example LUT precomputation for matrix Y^*. $Y^*[i,:]$ denotes the i-th row of matrix Y^*. The previous result gets XORed with the row in Y^* corresponding to the changed bit in the Gray code.

and could potentially be used in any PIR scheme in which queries can be expressed as matrix multiplication, e.g., [CGKS95; LG15; Hen16], as well as in PIR-PSI, cf. Chapt. 9.

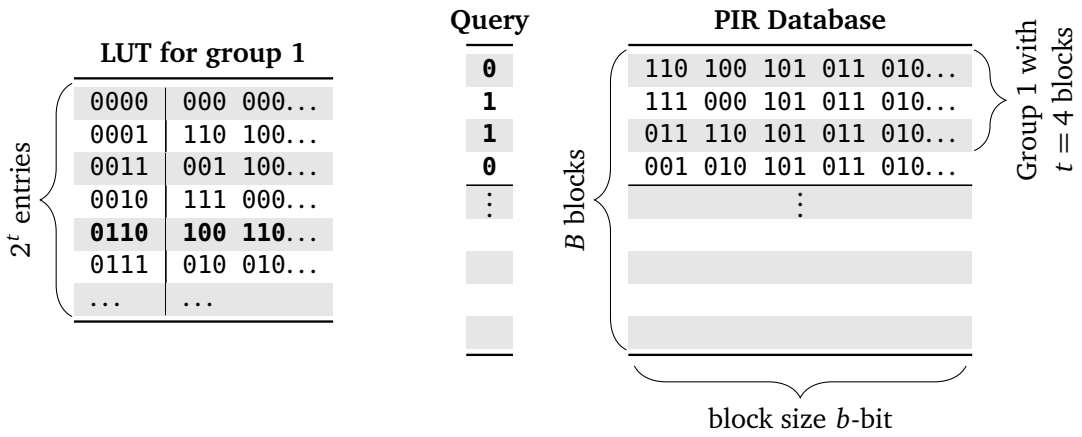


Figure 8.7: Precomputation example in the PIR database for $t = 4$. A query vector is one *row* in the matrix X. The first 4 bits of the query vector represent the index in the LUT for the first group of Y (the PIR database).

Implementation: An implementation of the Method of four Russians is provided in [ABH10]. However, since it only offers full matrix-matrix multiplications and uses different data structures than RAID-PIR, we implemented our own version that we integrated into RAID-PIR to make use of memory locality. While the traditional Method of four Russians assumes the multiplication of two full matrices, this is not the case in RAID-PIR. Often, only a single PIR request has to be answered, making X effectively a vector. However, several PIR queries could be batched to create matrix X at the cost of increased latency of the individual queries, similar to [LG15].

In our implementation we do the LUT precomputation once while loading the database and store the LUTs in main memory. For every query we achieve a theoretical speedup of t. The

downside of this approach is the increased memory requirement. Without loss of generality, we now assume that Y is a RAID-PIR database of size $B \times b$, i.e., B blocks of b bits each. While a larger t decreases the computation time for the generation of response vectors, it also excessively increases the memory needed to store the LUTs. When t blocks of the PIR database are put in a group, only B/t XOR operations have to be performed per query, resulting in a speedup of factor t. On the other hand, each of the B/t LUTs requires $2^t \cdot b$ bits of memory, where b is the database block size. The choice of $t = 4$ gives a good trade-off between theoretical speedup and memory requirements for larger databases (see Tab. 8.1) and is used in our implementation.

Table 8.1: Comparison of speedup and memory for the Method of four Russians.

t (speedup)	2	3	4	5	6	8
$\frac{1}{t} \cdot 2^t$ (memory overhead)	2.00	2.67	4.00	6.40	10.67	32.00

We implemented these optimizations in C. The precomputation only affects the calculation of the XOR responses by the PIR servers and is completely transparent to the PIR clients. This means that clients do not have to be aware of the changes and can send the same queries as before. Furthermore, different PIR server operators could decide independently whether they want to support the precomputation or not, or pick a t that suits their available resources.

In [LG15] a related approach to speedup server computation, based on the Strassen algorithm for matrix multiplication is introduced. This comes at the expense of higher latency for each individual query, as multiple queries have to be collected and processed together. Furthermore, their approach only introduces a speedup of $q/q^{0.80735} = q^{0.19265}$, where q is the number of queries that are processed together, which is less efficient than our approach (constant speedup t) when q is small (e.g., for $q < 1\,334$ and $t = 4$).

8.3.6 Uniform Distribution of Blocks in the Database

In the following we present an optimization specific to multi-block (MB) queries from Sect. 8.3.4, that enable a client to privately request more than one block in a single query. When database entries are larger than the block size b, they occupy multiple blocks. Naïvely, large entries could simply be placed adjacent to each other, usually residing within a single DB chunk. However, MB queries only allow to query at most one block per chunk. Therefore, in order to make use of the benefits of MB queries, it is desirable to spread large entries over the whole DB, such that they are equally spread over all chunks.

To improve the performance of MB for files larger than block size b, we change the layout of the database. Instead of placing files in adjacent blocks, they are now uniformly distributed over the whole database, as shown in Fig. 8.8 on the bottom. For each file, the required number of blocks n is calculated. The file is then placed with a mapping algorithm, such that between two of its blocks $B/n - 1$ blocks are used for other files, where B is the total number of blocks in DB. If a chosen block is already fully filled with another file, the next

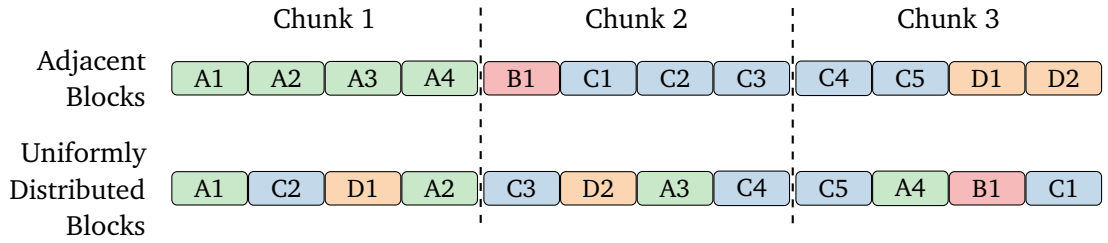


Figure 8.8: Uniform distribution of the four logical data entries {A,B,C,D} in the PIR database with 3 chunks, containing 4 blocks each. While entry A (green) was placed only in the first chunk in the original adjacent RAID-PIR database layout (top), it is now uniformly distributed over the whole database (bottom).

possible free block in DB is used instead. We also ensure a compact database layout, i.e., if the mapping algorithm encounters a partially occupied block, it is filled up to its full size.

8.3.7 Proxied Block Download

As proposed in [BCKP01], an additional download proxy (which can even be one of the PIR servers itself) can collect the responses from all servers, XOR them and send them to the client to reduce the download size by a factor of k. To ensure confidentiality, i.e., prevent the download proxy from reconstructing the plaintext response block, all responses have to be masked and blinded with an additional XOR operation that can efficiently be undone by the client. The blinding value can be generated from the PRG in a similar fashion as the rnd chunks, as described in Sect. 8.3.3. We do not implement this optimization in RAID-PIR. A related idea is introduced as designated output PIR in Sect. 9.3.1.

8.4 Analysis

In the following we analyze the complexity, correctness, and security of our PIR schemes.

8.4.1 Complexity

The complexities of the PIR schemes described in Sect. 8.3 are summarized in Tab. 8.2.

Our improved PIR protocols are well-suited for cloud-based applications where customers are charged for server computation and communication to/from the cloud. In contrast to the original Chor protocol [CGKS95], where multiple servers are used only to increase the security of the protocol, our protocols can use multiple servers for better efficiency, similar to the properties of a RAID system: for k servers, the upload communication from the client to each server is only about $1/k$-th of the original communication, and each server has to process only $(rB)/(2k)$ blocks in the database, since on average only every second block has

Table 8.2: Comparison of complexity and efficiency of the original CGKS scheme with our optimizations from Sect. 8.3. k: #servers, r: redundancy parameter ($2 \leq r \leq k$), B: #blocks, b: block size, κ: symmetric security parameter, t: 4 Russian precomputation parameter. Values in square brackets ('[]') only apply when 4 Russian precomputation is used.

	[CGKS95]	Sect. 8.3.2	SB Sect. 8.3.3	MB Sect. 8.3.4
Query size $\lvert q_i \rvert$ [bit]	B	$(r/k) \cdot B$	$(1/k) \cdot B + \kappa$	$(1/k) \cdot B + \kappa$
Answer size $\lvert a_i \rvert$ [bit]	b	b	b	$r \cdot b$
$\mathcal{S}$ Precomp. [#blocks to XOR]	$[2^t \cdot B/t]$	$[2^t \cdot B/t]$	$[2^t \cdot B/t]$	$[2^t \cdot B/t]$
$\mathcal{S}$ Comp. [#blocks to XOR]	$B/2[\cdot 1/t]$	$(r/k) \cdot B/2[\cdot 1/t]$	$(r/k) \cdot B/2[\cdot 1/t]$	$(r/k) \cdot B/2[\cdot 1/t]$
$\mathcal{S}$ Storage [#blocks]	$B[\cdot 2^t/t]$	$(r/k) \cdot B[\cdot 2^t/t]$	$(r/k) \cdot B[\cdot 2^t/t]$	$(r/k) \cdot B[\cdot 2^t/t]$
$\mathcal{C}$ Comp. [pseudorandom bits]	$(k-1) \cdot B$	$(r-1) \cdot B$	$(r-1) \cdot B$	$(r-1) \cdot B$
Result blocks per query	1	1	1	k
Max. #colluding servers	$k-1$	$r-1$	$r-1$	$r-1$
Communication efficiency	$b/(kB+kb)$	$b/(rB+kb)$	$b/(B+k\kappa+kb)$	$b/(B/k+\kappa+rb)$

to be processed due to the uniformly random queries. When using redundancy parameter $r = 2$, each server loads and processes only about $1/k$-th of the blocks in the DB.

We note that queries are sent to the servers concurrently with responses sent back from the servers; hence, the communication delay is essentially the maximum, rather than the sum, of the upload (query) and download (response) times.

8.4.2 Correctness

Our proposed PIR schemes are adaptations of the original CGKS scheme. The main difference is that we do not compute the XOR over the entire database, but over smaller chunks (column-wise). Therefore, correctness carries over from [CGKS95].

8.4.3 Security

The security of the schemes of Sect. 8.3.1, and Sect. 8.3.2 follows naturally using the same proof as of the original scheme in [CGKS95]. Intuitively, for each column i, the flip_i chunk is the XOR of $r-1$ random (rnd) values. Therefore, these flip chunks can be seen as an r-out-of-r XOR-based secret sharing of either the zero-vector or an elementary vector with the single bit set to one in which the client is interested. From the security of the secret sharing scheme follows that a collusion of up to $r-1$ servers cannot gain any information about the block the client is interested in.

The security of the SB scheme of Sect. 8.3.3 follows by standard reduction to the security of the PRG used. Namely, if the SB scheme leaks information, then we can distinguish between the output of the PRG and truly random strings of the same length, by running

the protocol with the given bits and checking if the attacker can learn information (which is proven impossible using truly random bits, see above and in [CGKS95]). However, while the original CGKS scheme achieves information-theoretic security, RAID-PIR reduces this to computational security in the symmetric security parameter of the PRG.

The security of the MB scheme of Sect. 8.3.4 also follows by a simple reduction argument, this time reducing to the SB scheme. Namely, assume an attacker can leak information from the MB scheme (but not from the SB scheme); for simplicity, assume this holds for the case shown in Fig. 8.5. Then the attacker runs four queries against the SB scheme, each time for a block from a different chunk, and then XORs the views received by the corrupt servers; this is equivalent to a single run of the MB algorithm, so we can use the attack on MB to leak information – which contradicts the security of the SB scheme. Note that for convenience, we ignored the fact that both SB and MB use pseudorandom strings (which can be easily dealt with as explained in the previous paragraph).

Note that to additionally achieve security against external man-in-the-middle attackers, the client can connect to the servers via secure channels, e.g., TLS.

8.5 Implementation

We base our implementation of RAID-PIR on the publicly available code of upPIR [Cap13] which implements the original CGKS protocol [CGKS95]. We modified and extended the existing Python code and implemented our improved PIR protocols described in Sect. 8.3: support for redundancy parameter r (Sect. 8.3.2), smaller upload by using a PRG for the queries (SB – Sect. 8.3.3), enabling parallel requests (MB – Sect. 8.3.4), 4 Russians precomputation (Sect. 8.3.5), and uniform distribution of DB entries (Sect. 8.3.6). We set the symmetric security parameter to $\kappa = 128$ and instantiate the PRG with AES128-CTR.

Implementation Optimizations Python is used for the implementation of the control flow, while the bit operations are implemented in C for efficiency. We introduced several implementation improvements in RAID-PIR, that are included in all measurements in Sect. 8.6. We pipelined the communication on the client side and decoupled the sending of queries and reception of answers. All queries are sent out immediately and answers are processed as they arrive. Furthermore, only a single seed is sent from the client to the servers during a session and a state is kept while the connection is alive. We also used SSE2 intrinsics on very wide data types in our C code. This feature is available in almost all x86 CPUs since several years and leads to a more efficient bit-wise XOR computation, which is one of the most crucial operations in RAID-PIR for both clients and PIR servers.

8.6 Benchmarks

In this section we evaluate the performance of RAID-PIR and show how the optimizations from Sect. 8.3 influence the overall performance. We compare the performance of our different variants to that of linear summation PIR [CGKS95]. In Sect. 8.6.7 we compare to the PIR system of [Gol07], using the implementation from [GDL+14].

8.6.1 Benchmark System

We benchmark our PIR schemes for different parameters and deployment scenarios. We deploy the k PIR servers as Amazon EC2 `r3.xlarge` instances with 30.5 GiB RAM, an Intel Xeon E5-2670 v2 processor and 1 Gbps Ethernet. The client is a consumer notebook with an Intel Core i7-4600U CPU with up to 3.3 GHz and 12 GiB RAM. We use two different network settings between the client and EC2: a consumer grade DSL connection (1 MBit/s upstream and 16 MBit/s downstream) and a WAN connection (350 MBit/s for both up- and downstream). The latency averaged to 30 ms for both network settings.

As DB we use a release of Ubuntu security updates, containing 964 updates adding up to a total size of 3.8 GiB. The average file size of the DB is 4 MiB, while the median file size is only 267 kiB, as many small patches are contained. We run each experiment 5 times and list the average runtimes.

8.6.2 PIR Server Startup Time

Loading the database DB into the PIR servers' RAM took ≈ 45 s for our benchmark database of 3.8 GiB and is mostly independent of the chosen block size. This time includes reading from disk, storing it into main memory and precomputing the lookup tables introduced by the Method of four Russians. The four Russians precomputation took ≈ 7 s.

For a group size of $t = 4$ and a total number of B blocks in the database, only $4 \cdot B$ XOR operations have to be performed during the precomputation phase, which is equivalent to the generation of 4 single-block responses. Note, that in this case, in contrast to the generation of PIR responses, data has to be written to main memory.

Loading the PIR database from disk into RAM is mostly bounded by the disk speed and therefore increases linearly with the database size, as depicted in Fig. 8.9. Since the number of operations needed for the four Russian precomputation is also proportional to the database size, the total startup time scales linearly.

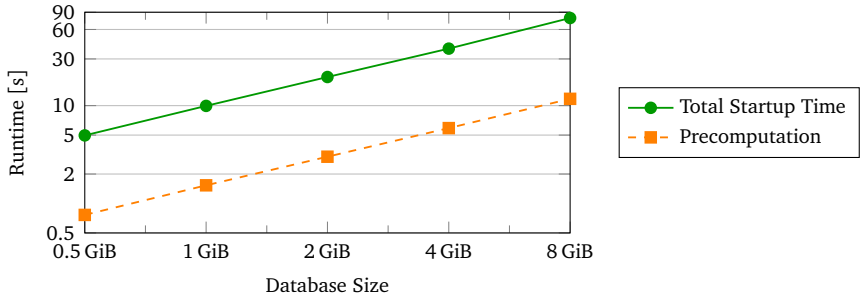


Figure 8.9: Time to load the PIR database from disk to RAM and doing the precomputation for varying database sizes. Block size $b = 256$ kiB, four Russians size $t = 4$.

8.6.3 Query Time for Varying Block Size

We show the influence of block size b on the query runtime for a given database DB in Fig. 8.10 and Fig. 8.11. We use $k = 3$ servers and a redundancy parameter of $r = 2$. The block size b is varied from 16 kiB to 4 MiB and the total runtime for the different PIR schemes is shown: CGKS (Sect. 8.3.1), $r = 2$ (Sect. 8.3.2), SB (Sect. 8.3.3), and MB (Sect. 8.3.4).

Our results indicate, that our improved PIR protocols result in larger runtime improvements when the upload bandwidth of the network connection is limited as the client sends less data (cf. Upload in Tab. 8.2). Our results also confirm that our multi-block PIR scheme MB described in Sect. 8.3.4 is beneficial when querying for multiple files distributed across multiple chunks.

The computation improvements for the servers cannot be seen clearly in these experiments as for parameters $r = 2$ and $k = 3$ the servers have in our protocols only 2/3 of the workload of the original Chor PIR scheme (cf. Server Computation in Tab. 8.2). Therefore, we also vary the number of servers k in later experiments. We compare performance of our schemes with further optimizations on and off.

Small File Query via WAN The performance results for querying 10 small files with a total size of 2.9 MiB in a WAN setup are depicted in Fig. 8.10 on the left. Single-block queries are denoted as SB, multi-block queries as MB, the Method of four Russians as 4R and the uniform distribution of the data entries as UD. In this network, the MB queries are significantly faster than MB queries even without further optimizations since the requested files are already distributed among the database. Therefore, uniformly distributing them does not improve the overall performance. Four Russians precomputation improves the runtimes by factor 2. For a block size of $b = 16$ kiB, runtime decreases from 6.8 s for the originally best performing multi-block queries to 3.4 s with all optimizations applied. Best performance is achieved for $b = 256$ kiB where 7 queries are performed to retrieve 18 blocks in 0.5 s.

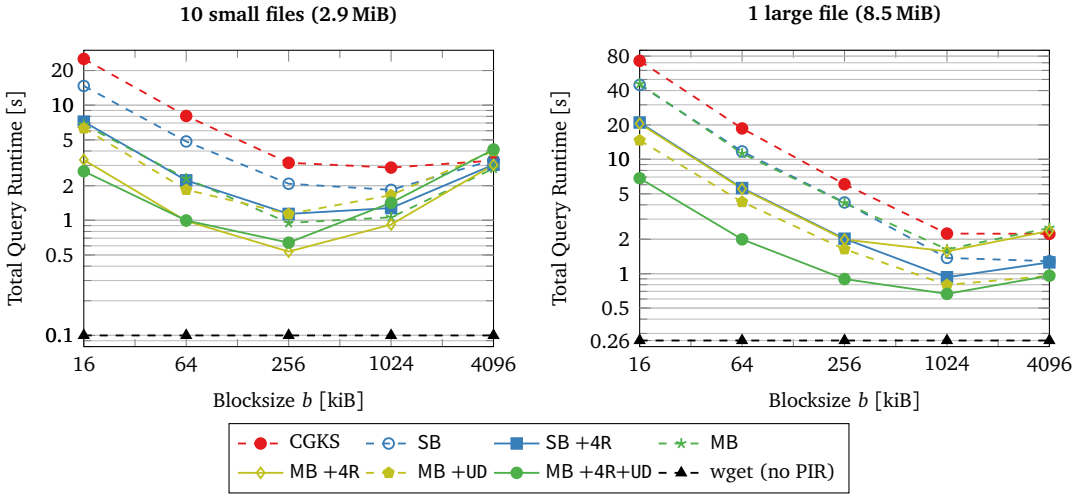


Figure 8.10: WAN Benchmarks: Runtimes for varying block size b with $k = 3$ servers, and redundancy parameter $r = 2$ for 10 small files (2.9 MiB, left) and a large file (8.5 MiB, right). CGKS: [CGKS95], SB: Single-Block scheme, MB: Multi-Block scheme, 4R: Four Russians precomputation, UD: Uniform distribution of data entries. DB size is 3.8 GiB.

Large File Query via WAN We query one large 8.5 MiB file and show the corresponding time on the right side of Fig. 8.10 Single-block and multi-block queries take similar time when querying one large file in a WAN network, that is typically not the sole bottleneck. The four Russians precomputation causes a significant speedup and effectively improves the runtime in most test cases by factor ≈ 2. When the CPU cache cannot hold precomputed LUTs and interim results for large block sizes, the Method of four Russians' speedup decreases, and disappears for blocks of size $b \geq 4$ MiB.

Uniformly distributing the data entries results in an even greater speedup and improves the overall runtime by approximately a factor of 3 in comparison to the unoptimized multi-block queries for small block sizes. In a DB layout where blocks are stored adjacent to each other, they most likely end up in the same chunk and have to be queried separately. Now it is possible to retrieve 3 blocks from 3 different chunks in one multi-block query in parallel.

When both optimizations are combined, the runtime decreases from 46 s (or 83 s for the original CGKS scheme) to 6.8 s for block size $b = 16$ kiB and reaches its minimum for $b = 1$ MiB where the runtime decreases from 1.3 s for the originally best performing single-block queries to 0.66 s for multi-block queries using both optimizations.

Large File Query via DSL In Fig. 8.11, we show the results for querying one large file (8.5 MiB, as in the first test case) over a consumer-grade DSL connection. The first observation is that the results for single- and multi-block queries with a block size of $b = 16$ kiB do not differ significantly, due to the slow upload speed of the DSL connection. For small block

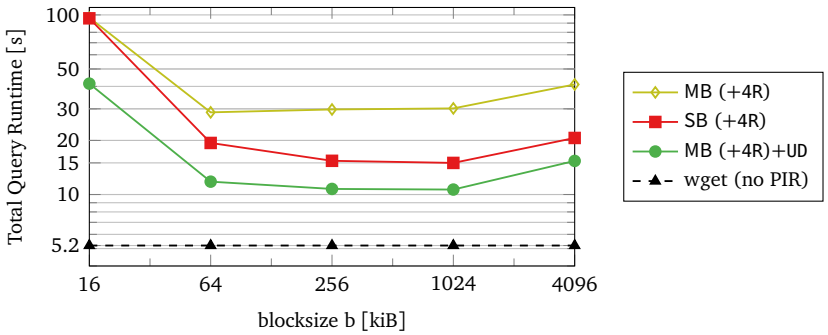


Figure 8.11: Large File, DSL: Runtimes for varying block sizes b with $k = 3$ servers, and redundancy parameter $r = 2$ for one large file (8.5 MiB). `CGKS`: original PIR scheme [CGKS95], `SB`: Single-Block scheme, `MB`: Multi-Block scheme, `4R`: Four Russians precomputation, `UD`: Uniform distribution of data entries. `DB` size is 3.8 GiB.

sizes, the required upstream bandwidth, which depends on the number of blocks, has a larger impact than the downstream bandwidth, which mostly depends on the block size. To query a `DB` with $B = 246\,360$ blocks, independent of the block size b, $B/3$ bits have to be transferred to each server (plus additional 16 Bytes for the PRG seeds) resulting in a total transmission time of $3 \cdot (B/3\,\text{bit} + 16\,\text{Byte})/(1\,\text{Mbit/s}) \approx 247\,\text{ms}$ solely for sending every query.

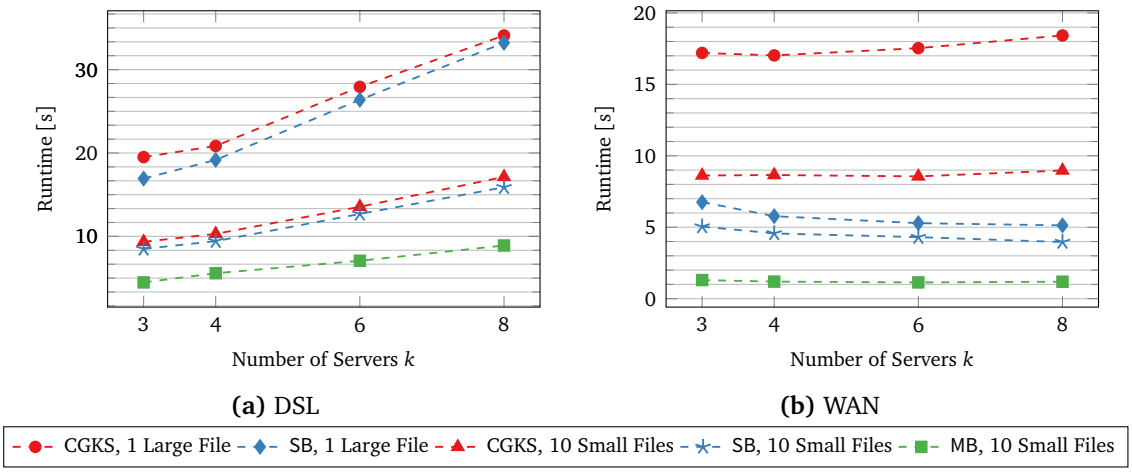
Due to this high communication overhead of the PIR queries, the server-side four Russians precomputation has almost no effect on the overall runtime. As in the first test case, `SB` queries provide better performance than `MB` queries without the uniform distribution. However, when the data entries are distributed uniformly, a large speedup for `MB` queries can be seen, since the number of requests is reduced by about factor 3, even outperforming `SB` queries.

For $r = 2$, the client needs to download approximately twice the size of the raw data. Retrieving the file without PIR, using `wget` over an unencrypted HTTP connection takes 5.0 s and is only 2.1 times faster, indicating that the runtime of 10.7 s for the PIR approach almost reached the theoretical optimum for the DSL connection. Our results show, that when the network is the bottleneck, computation has only a small influence on total query time. Our results also indicate that the previously assumed optimal block size of $b = B = \sqrt{|\mathsf{DB}|}$ does not necessarily yield good performance, especially in a DSL setting with asymmetric up- and downstream. For a `DB` size of 3.8 GiB that would be ≈ 22 kiB block size, which our results show to be far worse than larger block sizes, and the results we achieved for,e.g., $b = 1$ MiB.

8.6.4 Query Time for Varying Number of Servers

In the following, we analyze the effect of the number of servers on the total runtime of the PIR protocol, including network communication. From the results depicted in Fig. 8.12 we observe that increasing k only improves the total runtime if the network bandwidth is

high enough. For the DSL scenario the higher communication complexity, caused by the downstream from each server, also increases the overall runtime. For the WAN scenario the performance increase is clearly visible, but limited due to the increased communication and computation requirements for the client. The runtime for CGKS increases slightly, because the client has to wait for all k servers to respond and latency in the cloud typically varies.



(a) DSL **(b)** WAN

Figure 8.12: Total Runtimes [s] for varying number of servers k, redundancy parameter $r = 2$, and block size $b = 128$ kiB.

8.6.5 Server Computation Workload

Next, we measure the total time that one server needs to respond to a client's query for either the large consecutive file or the 10 small files that are distributed throughout the DB. This server computation time includes expanding the seed (for SB and MB), reading the blocks, and calculating their XOR. The results are depicted in Fig. 8.13. Our schemes benefit from an increasing number of servers k as the computation time for each server decreases, whereas the original CGKS scheme has a constant server workload that is independent of k. The runtimes for $r = 2$ and SB are almost identical (for a single consecutive file and 10 files, respectively), showing that the seed expansion with a PRG has negligible computation overhead.

8.6.6 Query Time for Varying Database Size

To demonstrate how the size of the database influences the query runtime, we perform the same queries as in Sect. 8.6.3 using multi block queries with four Russians precomputation and uniformly distributed database blocks. We varied the size of the database and depict our results in Fig. 8.14. The time that the clients needs to generate the queries to the servers is almost constant and always ≈ 50 ms. The communication varies due to queries being sent over a public Internet connection and also remains mostly constant around 200 ms. This is due to the fact, that even though the database and thus the number of blocks increases, the

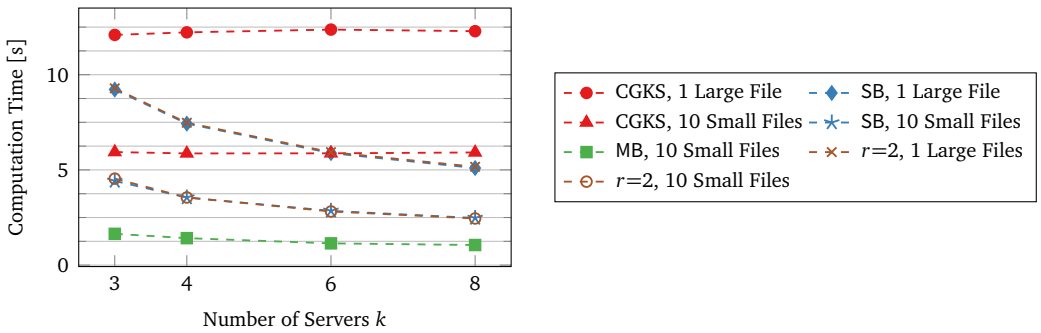


Figure 8.13: Server computation time [s] for varying number of servers k, redundancy parameter $r = 2$, and block size $b = 128$ kiB.

query size only grows from 250 Bytes to 4 kiB, which is negligible compared to the block size of $b = 256$ kiB, that the client receives. Server computation time and thus total time increase linearly with the database size.

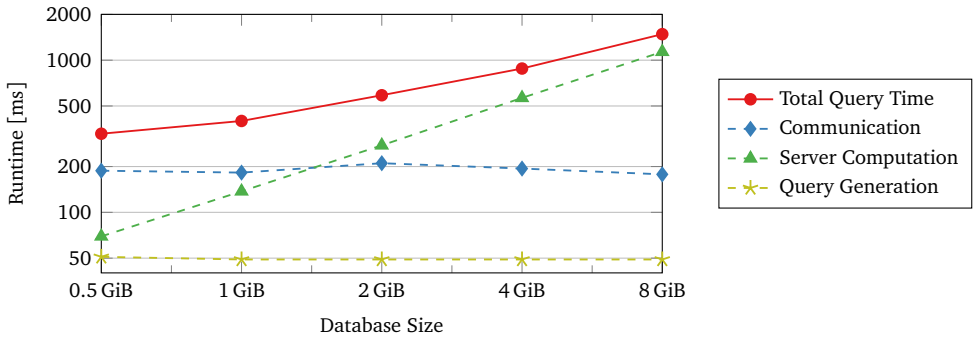
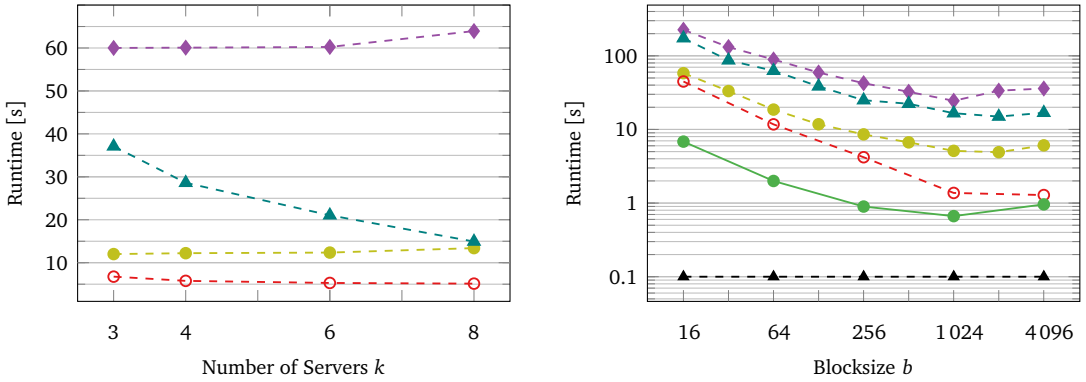


Figure 8.14: Varying DB size: Runtimes for varying database size, $k = 3$ servers, redundancy parameter $r = 2$ for querying one large file (8.5 MiB) and block size $b = 256$ kiB. All queries are multi-block (MB) queries using four Russians precomputation and uniformly distributed database blocks. Y-axis in logarithmic scale.

8.6.7 Comparison with [Gol07]

We evaluate the recent implementation of [GDL+14; Gol07] on the same machines and data that we used to benchmark our code and depict the results in Fig. 8.15. All results are for a WAN network connection, therefore the results in Fig. 8.15b and Fig. 8.10 are comparable. We also plot our RAID-PIR query time for SB and MB in Fig. 8.15b, as well as the plain download time using wget. We show performance results for their implementation of CGKS, which are comparable to our CGKS implementation. We compare it with a version of their protocol that allows to query one block per query sequentially and a version that allows to query $k-1$ blocks per query in parallel. The resulting runtimes are slower than the plain CGKS



(a) Runtimes for varying numbers of k, block size $b = 128$ kiB and the query of 10 random files.

(b) Runtimes for different Blocksizes b for $k = 3$ servers and the query of 10 random files.

Figure 8.15: Runtimes for the scheme of [Gol07] as implemented in [GDL+14] for varying number of servers k and block sizes b.

protocol for $k = 3$ servers. The implementation of [GDL+14] scales very well with increasing k, and for a large number of servers, may be competitive with ours. However, as shown in Fig. 8.15a, for a relatively low number of servers, our schemes are more efficient. More significantly, [Gol07] ensure robustness against malicious servers, which we only address by the higher-layer mechanisms described in Sect. 8.7.3.

8.7 Applying RAID-PIR

PIR can be used for different applications and scenarios, where clients wish to hide the storage locations they are reading. However, the applications may have additional requirements, beyond those provided by basic PIR schemes. Some of these may require significant extensions to existing PIR schemes, or a new scheme; for example, several works study PIR schemes which are robust to benign and/or byzantine failures [Gol07; DGH12; DG14]. Addition of such properties to RAID-PIR is subject for further research.

However, as we show below, some requirements can be achieved quite easily, by extending RAID-PIR or adding a layer on top of it. In the following, we discuss private retrieval of multi-block objects, PIR lookup mechanisms (identifying the block(s) belonging to a query/object), and finally object integrity, availability and accountability.

8.7.1 Private Multi-Block Object Retrieval

While PIR hides from a database which specific objects were queried, the amount of blocks a client requests in one session might give away information about possible corresponding objects. In this case, using a block size equal to the object size involves significant communication and computation overhead. To reduce this overhead, we would normally use a smaller block size, s.t. retrieving a large object would imply retrieval of all the blocks in the objects. However, when using such a scheme, the number of blocks retrieved may allow the attacker to identify the object (if it has a unique number of blocks) or to know that the object is within the limited set of objects with the same number of blocks. To mitigate this problem, clients may always retrieve a fixed number of blocks, including dummy blocks they are not interested in, possibly using the parallel block query (Sect. 8.3.4) to reduce the overhead. By using a smaller block size, this design is less wasteful in storage, although it has the same communication cost as the use of blocks of the same total size. Hence, these solutions fix the privacy concern, at the cost of more communication.

In many scenarios, an alternative would be to address the privacy concern by increasing the latency. Specifically, if a system has some known distribution of requests, say in rate of β blocks per second (taking into account the total number of blocks in requested objects), then a possible way to hide the identity of the requested objects would be to request objects at fixed rate of β blocks per second (or a bit higher, to avoid excessive latency), delaying requests if necessary and adding 'dummy' requests if none are present. Again, this combines well with the parallel block query (Sect. 8.3.4), allowing to retrieve k blocks with each request, improving request (upstream) bandwidth and processing overhead. Further optimizations may be possible, especially when the distribution of requests and of object-sizes is known or has certain properties.

8.7.2 PIR Lookup Mechanisms

PIR schemes allow clients to retrieve a block by specifying the block number, without exposing the block number to eavesdroppers or to the PIR servers. However, in many applications, clients do not necessarily know in advance the block number containing the information they need. Instead, they have some higher-layer identifier such as a URL or file name. How can clients learn the block number(s) corresponding to the object identifier without exposing their interest in a particular object?

One solution is used in upPIR [Cap13]: the content provider maintains a *manifest file*, mapping all object identifiers to the corresponding block numbers. Clients always retrieve the entire manifest file (from the origin or any PIR server, or even from another location). The manifest file can also allow to ensure integrity by including a collision-resistant hash of each object, and the origin signs the manifest file itself.

This use of such a manifest file is fine, as long as its size is reasonable; it may be less appropriate to ensure privacy for queries to a huge collection of many (smaller) items, such

as the Domain Name System, as proposed, e.g., in [ZHS07]. In such cases, the distribution of such a file may cause significant overhead, especially if it has to be updated periodically.

In such cases, where distributing a manifest file causes significant overhead, it may be better to use other approaches. A simple approach is to store an object with identifier (URL) i, in block $h(i)$ where h is a hash function; this has obvious limitations, in particular, no control over placement. More elaborate schemes for object lookup in PIR based on keywords or identifiers were studied, e.g., in [CGN98].

Another approach can be built from searchable encryption, where clients send a lookup into an encrypted dictionary of locations and thereby privately retrieve the indices of the blocks that they are interested in.

8.7.3 Object Integrity, Availability and Accountability

PIR schemes do not necessarily ensure integrity. Namely, a benign or malicious (byzantine) failure in one or more of the servers, may result in clients receiving corrupt data. Some PIR schemes are *robust*, i.e., designed to handle benign failures (e.g., transmission errors, or disk failures), or even byzantine failures (e.g., active attacks), e.g., [DG14; DGH12]. Our RAID-PIR constructions, however, are not robust to server failures; it is an interesting challenge to adopt them to achieve robustness (with comparable performance). However, we note that clients may easily recover from failure of few servers, by using only the operative servers in the protocol. We now show how to take advantage of this and use RAID-PIR to ensure integrity, availability and accountability.

We first explain how to ensure object integrity: one natural solution is for the origin to sign every object in the DB, and to provide the signature as part of the object. Alternatively, when a manifest file is used, the origin can only sign the manifest file, and include therein a collision-resistant hash value for each object (cf. [Cap13]). In some applications, the content 'belongs' to a specific customer of the origin; in this case, the origin may use a Message Authentication Code (MAC) for higher efficiency.

The above mechanisms only detects corrupted objects, which still allows malicious servers to deny service to the clients. However, we can efficiently deal with this threat, as follows: First, we assume that all the communication between client and each server is authenticated. This allows the client to ignore servers to which there are repeated communication failures. We therefore focus on server corruptions and assume that all messages are received correctly as sent. Second, we assume that the client uses some of the mechanisms above, e.g., signed objects, to detect corrupted objects. It remains to explain how we deal with this case, i.e., client receiving corrupted objects due to a server failure. In this case, the client proceeds with a *resolution protocol*, which identifies the corrupted server, as follows: (A) The client simply re-sends the same query to the servers; however, this time it indicates that the responses should be *signed*. If the client detects that the server failed, by not returning the same response as before (or not returning a valid signature), then the client marks this server as 'bad' and continues the protocol with the remaining servers. The signature is over both the request

received by the server and over the response. (B) Once the client receives the signatures from all the (operative) servers, then it selects one or more of the servers, and sends the corresponding signed responses from these servers to the content origin or a trusted third party. The origin can validate the responses and blacklist servers in case of failure so that all clients will stop using it.

If clients wish to preserve query privacy from the origin, then they should send less than r of the responses from the servers to the origin (each containing the corresponding request that the client sent to that server). However, surely, after few such interferences, a faulty server would certainly be detected.

8.8 OnionPIR: A System for Anonymous Communication

As an application that utilizes RAID-PIR and our previously presented improvements, we introduce a private communication system called *OnionPIR* for anonymous communication. We use RAID-PIR with our optimizations as a building block for public key distribution, and onion routing to get a system efficient enough for practical use.

8.8.1 Motivation

In order to achieve anonymous communication, two parties could attempt to register under pseudonymous identities to a classical messaging service and connect to the service by using Tor [DMS04]. However, a malicious server can link those two pseudonyms by building a social graph isomorphic to the one built from information from other sources. By comparing these two graphs, sensitive information about users can be revealed. Therefore, Tor alone is not enough to provide protection against metadata leakage. Additionally, in practice it is technically difficult for users to establish a shared secret. In order to provide *practical* anonymity, a system to establish private communication channels based on already existing contact information such as phone numbers or email addresses is needed.

8.8.2 Related Work in Anonymous Communication

Nowadays, end-to-end encryption is available and deployed at large scale. In the past, these technologies were not accessible to a large user base because they required expert knowledge or were just inconvenient, and thus only used by enthusiasts. For example, in OpenPGP [CDF+07] users have to manually build a web of trust and should be familiar with public key cryptography, while S/MIME [Ram99] requires certificate management. When the Signal Protocol was integrated into popular messaging services like WhatsApp[2] or Facebook Messenger[3] in 2016, private messaging became available to an extremely large

[2] https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf
[3] https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf

user base. However, protecting not only communication content but also its metadata is still under active research. OnionPIR currently relies on RAID-PIR for public key distribution, but could also employ different PIR schemes, e.g.,[DG14; Hen16; ABFK16] or the DPF-based PIR from Sect. 9.2.1. Similarly, we rely on Tor to provide anonymity, which can also be achieved by using alternative techniques such as mix networks [Cha81].

Next, we present systems that are related to OnionPIR. *Redphone* was one of the first applications that tackled metadata leakage using Bloom filters [Blo70] for private contact discovery. Redphone's encrypted call features were integrated into Textsecure/Signal, however, without anonymity due to scalability [Ope14]. With *DP5* [BDG15] users can anonymously exchange online presence information based on PIR. DP5 divides time in long-term and short-term epochs which are in the order of days and minutes respectively, which makes it impractical for real-time communication. Users must share symmetric keys before the protocol run, which can be hard to achieve in practice. AnNotify [PHG+17] is a service for efficient and private online notifications that combines several techniques. Recently, *Alpenhorn* [LZ16] was proposed, which is based on identity-based encryption (IBE) for key distribution, a mix network [CDJ+17; Cha81] for privacy and a so called keywheel construct for forward secrecy. Alpenhorn was integrated into the messaging system Vuvuzela [HLZZ15], which supports 10 Mio. users using three Alpenhorn servers with an average dial latency of 150 s and a client bandwidth overhead of 3.7 KiB/s. *Riposte* [CBM15] enables a large user base to privately post public messages to a message board. Its security is based on distributed point functions that are evaluated by a set of non-colluding servers. Time is divided into epochs, in which all users who post messages form an anonymity set. *Ricochet*[4] is a decentralized private messaging service based on Tor hidden services, whose addresses must be exchanged out-of-band to establish connections. Recent research showed that HSDirs are used to track Tor users [SN16], which might be problematic for Ricochet's privacy. *Riffle* [KLDF16] provides scalable low-latency and low-bandwidth communication through mix networks [Cha81] using verifiable shuffles [BG12] for sender anonymity and PIR for receiver anonymity. In Riffle, time is divided into epochs and each client sends and receives messages even if they do not communicate. *The Pynchon Gate* [SCM05] is an anonymous mail system that guarantees only receiver anonymity by using PIR.

8.8.3 System Model and Goals

There is a number of privacy-preserving communication systems available, but in many cases, these do not scale well for large numbers of users or require the out-of-band exchange of a shared secret, which is error-prone and leads to usability issues most users have problems with. The success of the popular messaging app Signal[5] and the adaption of its protocol in WhatsApp and Facebook Messenger are founded on the combination of strong security and seamless usability. Since most users do not have a background in cryptography, it is necessary

[4] https://ricochet.im

[5] downloaded by more than 5 Million users via the Google Play Store, https://play.google.com/store/apps/details?id=org.thoughtcrime.securesms

to build technologies that provide privacy and usability by design and to set reasonable defaults for its users.

In this work, we combine the strong privacy guarantees of PIR protocols with the efficiency of established onion routing protocols, such as Tor [DMS04]. The interaction with the server is divided into two phases: an *initialization phase* where the communication channels are established and keys are exchanged via PIR and a *communication phase* where the actual message exchanges take place through Tor.

In the initialization phase, PIR is used to *privately* exchange information between the two parties that want to communicate securely, such that no external entity will find out that the communication between the parties happened. This procedure could theoretically be used to exchange all kinds of data. However, when it comes to practical applicability, querying large amounts of information via PIR does not scale well for a large number of users. Hence, in the communication phase no PIR techniques will be used.

Instead, the information retrieved via PIR is used to place messages, encrypted using Authenticated Encryption with Associated Data (AEAD), in an anonymous inbox, called "dead drop". This dead drop is read and written at the OnionPIR control server in the communication phase by clients that know its long and unpredictable identifier. We employ onion routing to hide the identity of the communication partners, such that the server is unable to determine who sent and received messages. For real-time communication, the users could establish direct connections using TCP streams or web sockets [FM11] through the server.

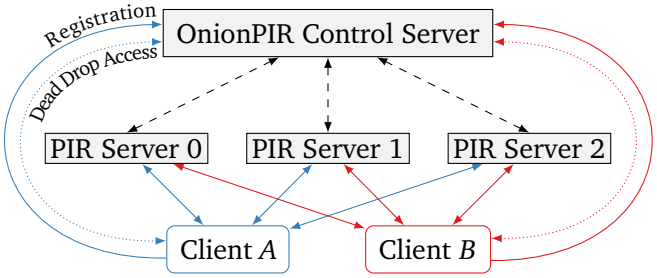


Figure 8.16: OnionPIR sytem. The OnionPIR control server handles user registrations, distributes users' public keys to the PIR servers and holds the dead drops. Clients perform PIR queries to privately receive other users' public keys from the PIR servers. Clients connect to the OnionPIR control server directly or via Tor (dotted lines).

The OnionPIR system, depicted in Fig. 8.16, serves clients who want to communicate with each other and two types of servers. All honest clients form the anonymity set among which a user is anonymous. That means that a potential adversary cannot determine which users within this set communicate with each other. The central OnionPIR control server handles user registration and serves as a content provider for the PIR servers. It also acts as a database for the "dead drops" in the communication phase. The PIR servers are used only in the initialization phase to privately distribute the public keys of clients.

8.8.4 Protocol Description

In this section we describe the OnionPIR protocol. An overview is depicted in Fig. 8.17.

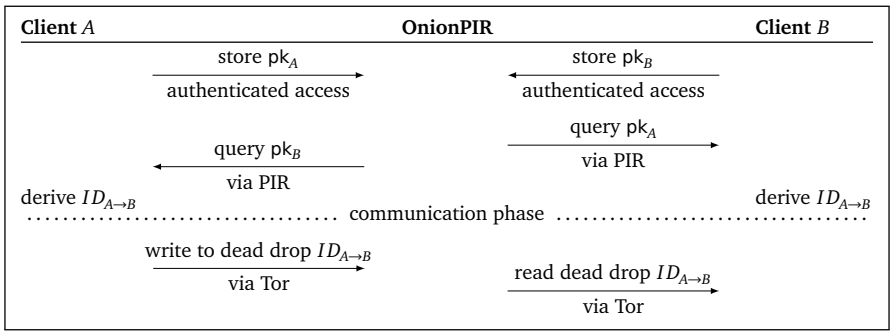


Figure 8.17: Simplified OnionPIR protocol. The registration, dead drop and PIR servers were abstracted into one server. Key renewal and the answer of client B are not shown.

When a client A registers for the service, it first runs through an account verification process and sends its public key to the OnionPIR control server which will later distribute it to the PIR servers (see Sect. 8.6.2 for precomputation performance). Another client B, that has an address book entry for A (e.g., a mobile number or e-mail address), will later *privately* query A's public key via PIR. In addition, each user periodically queries for his or her own public key to make sure the OnionPIR control server does not distribute bad keys to the PIR servers (see Sect. 8.8.5 for details). B will then use his own private key and the received public key to generate a shared secret K_{AB} between A and B by performing an Elliptic Curve Diffie-Hellman (ECDH) key agreement. Since no communication with the server is required to derive the shared secret, this type of key agreement protocols is often also called *private key agreement*. In the same way, A is also able to derive the shared secret K_{AB} using her own private key and the public key of B.

Symmetric Keys The shared secret K_{AB} and both parties' public keys are used to derive identifiers of the dead drops and keys used to exchange messages. Client A generates two different symmetric keys derived from the initial shared secret K_{AB} and the respective public keys by a keyed-hash message authentication code (HMAC): $K_{A\to B} = \mathrm{HMAC}_{K_{AB}}(pk_B)$ for sending messages to B and $K_{B\to A} = \mathrm{HMAC}_{K_{AB}}(pk_A)$ for receiving messages from B. These secrets constantly get replaced by new ones, transmitted alongside every message to provide forward secrecy.

Dead Drop IDs The identifier of the dead drop for sending from A to B is built by computing an HMAC with the key $K_{A\to B}$ of a nonce $N_{A\to B}$ that increases after a given time period. Hence, the identifier $ID_{A\to B} = \mathrm{HMAC}_{K_{A\to B}}(N_{A\to B})$ changes in a fixed interval, even if no messages were exchanged at all. This prevents the server from identifying clients that disconnect for several days and would otherwise reconnect using the same identifiers. However, a fixed

point in time at which the nonce changes (e.g., a unix timestamp rounded to the current day) is not a good choice. Assuming synchronized clocks is often error-prone.[6] If all clients would update their identifiers at a given point in time, e.g., at midnight, the server could detect a correlation between all identifiers of a user whose clock is out of sync. Thus, the nonces will be handled per contact and the secret key $K_{A\to B}$ is used to generate a point in time at which the nonce $N_{A\to B}$ will be increased. This leads to a different update time for all identifiers of dead drops.

Sending Messages When a client A wants to send a message m to B, it encrypts the message using AEAD using $K_{A\to B}$ so that only B can read it. Note, that the ciphertext of message m does not reveal any information about the sender or the receiver as explained in [Ber09, Sect. 9]. A then stores the encrypted message in the dead drop $ID_{A\to B}$ via Tor. B can now fetch and decrypt A's message from this dead drop. The shared secret will only be replaced by a new one transmitted alongside a message when B acknowledges the retrieval of the new secret in a dead drop derived from the *existing* shared secret. A will retransmit messages until B sends an acknowledgment. This procedure guarantees that messages are not lost. Note, that $ID_{A\to B}$ can still change over time because of the used nonce $N_{A\to B}$ which is known by both parties.

Querying for new public keys using PIR is done in a fixed interval, e.g., once a day, to discover new users of the system. It is mandatory not to write to the dead drop specified by the shared secret immediately after receiving a new public key. This would allow an adversary to correlate a PIR request of a given user with (multiple) new requests to the dead drop database, even if the server does not know which public keys were queried. Instead, the first access to the dead drop database for new identifiers is delayed until a random point in time between the current query and the next one derived from the shared key $K_{A\to B}$. This ensures that the server cannot correlate the dead drop access to the PIR request because it could also have been initiated by any other clients that did a PIR request in the fixed interval. Note, that this delay is only necessary when a new contact is discovered. New users joining the service will therefore also have to delay their first interaction with other users.

Initial Contact If B wants to contact A without A knowing about that (and thus not checking the associated dead drop at $ID_{B\to A}$), an anonymous signaling mechanism is required. We propose a per-user fixed and public dead drop that *all* other users write to, to establish contact. The fixed dead drop ID is $ID_A = \mathrm{HMAC}_0(pk_A)$. Messages into this dead drop are encrypted using hybrid encryption, similar to PGP, where A's public key pk_A is used to encrypt an ephemeral symmetric key, which encrypts the request message. This reveals how many contact requests A receives, which we consider as non-critical. However, this number could be obscured by sending dummy requests.

[6] The need for secure time synchronization protocols lead to a number of secure time synchronization concepts such as ANTP [DSZ16], NTS (https://tools.ietf.org/html/draft-ietf-ntp-network-time-security-14) or Roughtime (https://roughtime.googlesource.com/roughtime).

8.8.5 Analysis

Correctness Correctness of RAID-PIR is shown in Sect. 8.4.2, correctness of Tor is explained in [DMS04] and has already been well-proven in practice. Messages are acknowledged and retransmitted if identifiers change and the message has not been read yet. Therefore message delivery can be ensured. Correctness of the ECDH key exchange is shown in [Ber09]. All operations involved in the private establishment of identifiers for the dead drops are deterministic and therefore result in the same identifiers for both parties.

Security OnionPIR's security is based on the security guarantees of the underlying protocols and their assumptions. First, we assume that RAID-PIR is secure and does not leak any metadata. A security argumentation for this is given in Sect. 8.4.3. In particular, it is important that PIR servers are run by different non-colluding operators. Note, that security still holds if less than r servers collude, where r is the redundancy parameter. A good choice for the PIR server operators could be NGOs located in different legal territories. The different PIR servers must also not be in control of the same data center operator.

Next, we assume that Tor provides anonymity for the users that tunnel their connections through this anonymity network. This assumption implies that there is no global passive adversary which is able to monitor and analyze user traffic and colludes with the operator of the PIR servers or gains unauthorized access to them. Note, that it is necessary to at least de-anonymize two specific Tor connections in order to learn if two users are communicating with each other. Therefore, an attacker would have to be able to de-anonymize all users of the service to gain the full social graph of a given user. However, as shown in [DS18], tagging attacks are preventable. OnionPIR does not hide the fact that a user is using the service.

Anonymity is guaranteed among all honest users. A malicious user revealing its list of dead drop requests, would effectively remove himself from the anonymity set. Note, that no user can gain information about communication channels it is not participating in or prove that communication took place because the AEAD guarantees repudiability.

OnionPIR uses a Trust On First Use (TOFU) strategy to lessen the burden of manually exchanging keys. For detecting the distribution of faulty keys, a client queries not only its contacts' public keys, but also its own key. This query can be performed at nearly no cost when querying together with other contacts using RAID-PIR's multi-block query `MB`, cf. Sect. 8.3.4. Since the PIR servers cannot determine which keys are being requested, they are not able to manipulate the resulting response in a meaningful way. Of course, additional security can be achieved by adding out-of-band key verification, e.g., by announcing public keys on personal websites. Another option are plausibility checks for the updates of the PIR database.

Access to the dead drops is not protected against manipulation since identifiers are only known to the involved parties. An adversary interested in deleting the messages for a specific client would have to brute-force the dead drop identifier. Protection against a server deleting messages or blocking access to the system is out of scope of this work and would require a federated or decentralized system.

Protection against attackers flooding the dead drops with large amounts of data could be achieved by using blind signatures [Cha83]. A client could encrypt a number of random tokens and authenticate against the server who then blindly signs them. The tokens are then decrypted by the client and sent along with each write access. While the server can verify the signature of these tokens, it cannot identify which client generated them. Since a server only signs a small number of tokens per client, this approach rate-limits DB write requests.

Complexity and Efficiency OnionPIR aims at providing efficient anonymous communication. Many existing systems (see Sect. 8.8.2) require a high communication overhead or high computational costs. The dead drop database is therefore combined with scalable onion routing and can be implemented as simple key-value storage. The servers needed in the communication phase can be deployed with low operating costs, comparable to traditional communication services.

The initialization phase in which the PIR requests are performed is crucial for the scalability of the system. As shown in Sect. 8.6, RAID-PIR with our optimizations is a valid choice that offers reasonable performance and allows users to detect malicious actions of the servers, cf. Sect. 8.7.3. The 3.8 GiB PIR database used for the benchmarks in Sect. 8.6 is sufficient to store public keys of 127 Mio. users, using 256 bit elliptic curve public keys.

The DB size and the servers' computation grows linearly with the number of users. The PIR servers' ingress traffic for PIR queries depends on the number of blocks B in the DB and therefore also scales linearly. Thanks to the PRG used in RAID-PIR (see Sect. 8.3.3), the size of a PIR query is $\lceil B/8 \rceil$ Bytes for all servers combined (excluding PRG seeds and overhead for lower level transport protocols). The egress bandwidth is constant since the response size only depends on the block size b (and the number of chunks for multi-block queries).

8.8.6 Implementation

We implemented a desktop application for private messaging with metadata protection, called OnionPIR. Our implementation is written in Python and C, building on top of RAID-PIR, the Networking and Cryptography library (NaCl) [BLS12] for cryptographic operations, and Stem[7] as controller library for Tor. The system is divided into a client, the OnionPIR control server and PIR servers. The client can use the messenger through a GUI as depicted in Fig. 8.18. Our open-source implementation is publicly available on GitHub[8].

The clients' user interface is built using web technologies, while the OnionPIR client itself is written in Python. We designed the client and the OnionPIR protocol with usability in mind, such that the user will not see any cryptographic keys or other technical details. In order to increase reusability of the code, major functionalities, e.g., sending/receiving messages, or the registration process, were bundled in the *OnionPIR client library*.

[7] https://stem.torproject.org/
[8] https://github.com/encryptogroup/onionPIR

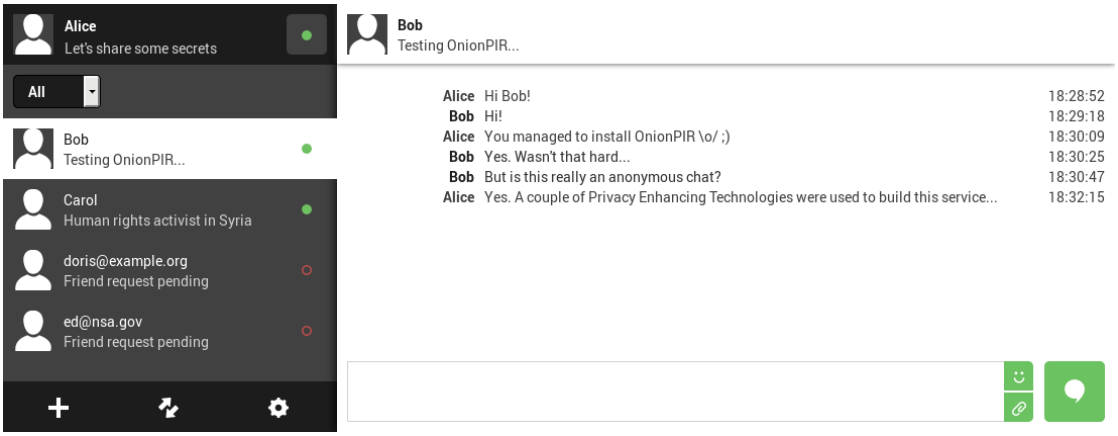


Figure 8.18: Screenshot of the OnionPIR client GUI.

The OnionPIR control server handles user registrations and acts as content provider for RAID-PIR, that provides the DB of public keys, while the PIR servers answer PIR requests. The client runs at the user's end system. The control server is operated by the service provider while the PIR servers should be operated by a number of independent, non-colluding third-parties, e.g., NGOs. Currently, email addresses are used as an identifier for contacts, since they are unique and easily verifiable. Identifiers are hashed before they are used to determine the location inside the PIR database at which the user's public key is stored. We could also allow arbitrary user names with a first-come-first-serve policy, or phone numbers with SMS verification.

Our current implementation is intended for testing and demonstration only. A minimalistic end-to-end encrypted chat protocol is implemented in the client, which offers repudiability.

8.9 Conclusion and Future Work

In this chapter, we presented and evaluated RAID-PIR, a family of simple and practical multi-server PIR schemes, which are efficient in computation, storage and communication, and especially upload from clients. Due to its simplicity and efficiency, RAID-PIR can be used by practical systems. Built on top of RAID-PIR, we presented OnionPIR, a practical anonymous communication system that combines private key distribution via PIR with private communication through an onion routing system like Tor.

RAID-PIR does currently not provide robustness against faulty servers, however, we show (in Sect. 8.7.3) how to efficiently use it to handle erroneous or malicious servers. It would be an interesting challenge, to find another RAID-PIR mode, that will provide fault-tolerance, which may also help speed up results by not waiting for response from the slowest server. Of course, the challenge is to maintain the RAID-like simplicity and efficiency. A more recent scheme that is more efficient for large databases and two servers is PIR based on DPFs, as presented in Sect. 9.2.1.

9 PIR-PSI: Scaling Private Contact Disvocery

Results published in:

[DRRT18] D. DEMMLER, P. RINDAL, M. ROSULEK, N. TRIEU. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). Code: https://github.com/osu-crypto/libPSI. CORE Rank B.

9.1 Introduction

With the widespread use of smartphones in the last decade, social networks and their connected instant messaging services are on the rise. Services like Facebook Messenger or WhatsApp connect more than a billion users worldwide.[1]

Contact discovery happens when a client initially joins a social network and intends to find out which of its existing contacts are also on this network. Even after this initial client join, it is also run periodically (e.g., daily) in order to capture a client's contacts that join the network later on. A trivial approach for contact discovery is to send the client's entire address book to the service provider, who replies with the intersection of the client's contacts and the provider's customers. This obviously leaks sensitive client data to the service provider. In fact, a German court has recently ruled that such trivial contact discovery in the WhatsApp messaging service violates that country's privacy regulations.[2] Specifically, users cannot send her contact list to the WhatsApp servers for contact discovery, without written consent of all of their contacts.

A slightly better approach (often called "naïve hashing") has the client *hash* its contact list before sending it to the server. However, this solution is insecure, since it is prone to offline brute force attacks if the input domain is small (e.g., telephone numbers). Nonetheless, naïve hashing is used internally by Facebook and was previously used for contact discovery by the Signal messaging app. The significance of a truly private and efficient contact discovery was highlighted by the creators of Signal already in 2014.[3]

[1] https://blog.whatsapp.com/10000631/Connecting-One-Billion-Users-Every-Day

[2] http://www.huffingtonpost.de/2017/06/27/whatsapp-abmahnung-anwalt-medien-gericht-nutzer_n_17302734.html

[3] https://whispersystems.org/blog/contact-discovery/

9.1.1 State of the Art and Challenges

Contact discovery is fundamentally about identifying the intersection of two sets. There is a vast amount of literature on the problem of *Private Set Intersection (PSI)*, in which parties compute the intersection of their sets without revealing anything else about the sets (except possibly their size). A complete survey is beyond the scope of this work, but we refer the reader to Pinkas et al. [PSZ18], who give a comprehensive comparison among the major protocol paradigms for PSI.

In contact discovery, the two parties have *sets of vastly different sizes*. The server may have 10s or 100s of millions of users in its input set, while a typical client has less than 1 000.[4] However, most research on PSI is optimized for the case where two parties have sets of similar size. As a result, many PSI protocols have communication and computation costs that scale with the size of the larger set. For contact discovery, it is imperative that the client's effort (especially communication cost) scales *sublinearly* with the server's set size. Concretely, when the client is a mobile device, we aim for communication of at most a few megabytes. A small handful of works [CLR17; KLS+17; RA18; PSZ18] focus on PSI for asymmetric set sizes. We compare these works to ours in Sect. 9.1.3 and Sect. 9.7.

Even after solving the problems related to the client's effort, the server's computational cost can also be prohibitive. For example, the server might have to perform expensive exponentiations for each item in its set. Unfortunately no known technique allows the server to have computational cost sublinear in the size of its (large) input set. The best we can reasonably hope for (which we achieve) is for the server's computation to consist almost entirely of fast symmetric-key operations, which have hardware support in modern processors.

If contact discovery were a one-time step only for new users of a service, then the difference between a few seconds in performance would not be a significant concern. Yet, *existing users* must also perform contact discovery to maintain an up-to-date view. Consider a service with 100 million users, each of which performs maintenance contact discovery once a week. This is only possible if the marginal cost of a contact discovery instance costs less than 6 milliseconds for the service provider (one week is roughly 600 million milliseconds)! To be truly realistic and practical, private contact discovery should be as fast as possible.

9.1.2 Overview of Results and Contributions

We propose a new approach for private contact discovery that is practical for realistic set sizes. We refer to our paradigm as *PIR-PSI*, as it combines ideas from private information retrieval (PIR) and standard 2-party PSI.

[4] A 2014 survey by Pew Research found that the average number of Facebook friends is 338 [Smi14].

Techniques (Sect. 9.3) Importantly, we split the service provider's role into two to four non-colluding servers. With 2 servers, each one holds a copy of the user database. When a third server is used, 2 of the servers can hold secret shares of the user database rather than hold it in the clear. With 4 servers, all servers can hold secret shares. By using a computational PIR scheme, a single-server solution is possible. Most of our presentation focuses on our main contribution, the simpler 2-server version, but in Sect. 9.8 we discuss the other variants in detail. Note that multiple non-colluding servers is the traditional setting for PIR, and is what allows our approach to have sublinear cost (in the large server set size) for the client while still hiding its input set.

Roughly speaking, we combine highly efficient state-of-the-art techniques from 2-server PIR and 2-party private set intersection. The servers store their sets in a Cuckoo table so that a client with n items needs to probe only $O(n)$ positions of the server's database to compute the intersection. Using the state-of-the-art PIR scheme of Boyle et al. [BGI15; BGI16], each such query requires $O(\kappa \log N)$ bits of communication, where N is the size of the server's data, and κ is the symmetric security parameter. In standard PIR, the client learns the positions of the server's data in the clear. To protect the servers' privacy, we modify the PIR scheme so that *one of the servers* learns the PIR output, but blinded by masks known to the client. This server and the client can then perform a standard 2-party PSI on masked values. For this we use the efficient PSI scheme of [KKRT16].

Within this general paradigm, we identify several protocol-level and systems-level optimizations that improve performance over a naïve implementation by several orders of magnitude. For example, the fact that the client probes *randomly distributed* positions in the server's database (a consequence of Cuckoo hashing with random hash functions) leads to an optimization that reduces cost by approximately 500×.

As a contribution of independent interest, we performed an extensive series of experiments (almost a trillion hashing instances) to develop a predictive formula for computing ideal parameters for Cuckoo hashing. This allows our protocol to use very tight hashing parameters, which can also yield similar improvements in all other Cuckoo hashing based PSI protocols. A more detailed description can be found in Section 9.3.2.

Privacy/Security For The Client (Sect. 9.4) If a corrupt server does not collude with the other server, then it learns nothing about the client's input set except its size. In the case where the two servers collude, even if they are malicious (i.e., deviating arbitrarily from the protocol) they learn no more than the client's *hashed* input set. In other words, *the failure mode for PIR-PSI is to reveal no more than the naïve-hashing approach.* Since naïve-hashing is the status quo for contact discovery in practice, PIR-PSI is a strict privacy improvement.

Furthermore, the non-collusion guarantee is forward-secure. Compromising both servers leaks nothing about contact-discovery interactions that happened previously (when at most one server was compromised).

In PIR-PSI, malicious servers can use a different input set for each client instance (e.g., pretend that Alice is in their database when performing contact discovery with Bob, but not with

Carol). That is, the servers' effective data set is not anchored to some public root of trust (e.g., a signature or hash of the "correct" data set).

Privacy/Security For The Server (Sect. 9.4) A semi-honest client (i.e., one that follows the protocol) learns no more than the intersection of its set with the servers' set, except its size. A malicious client can learn different information, but still no more than $O(n)$ bits of information about the servers' set (n is the purported set size of the client). We can characterize precisely what kinds of information a malicious client can learn.

Performance (Sect. 9.6) Let n be the size of the client's set, let N the size of the server's set ($n \ll N$), and let κ be the computational security parameter. The total communication for contact discovery is $O\big(\kappa n \log(N \log n/\kappa n)\big)$. The computational cost for the client is $O\big(n \log(N \log n/\kappa n)\big)$ AES evaluations, $O(n)$ hash evaluations, and κ exponentiations. The exponentiations can be done once-and-for all in an initialization phase and re-used for subsequent contact discovery events between the same parties.

Each server performs $O\big((N \log n)/\kappa\big)$ AES evaluations, $O(n)$ hash evaluations, and κ (initialization-time) exponentiations. While this is indeed a large number of AES calls, hardware acceleration (i.e., AES-NI instructions and SIMD vectorization) can easily allow processing of a billion items per second per thread on typical server hardware. Furthermore, the server's computational effort in PIR-PSI is highly parallelizable, and we explore the effect of parallelization on our protocol.

Other Features (Sect. 9.8) In PIR-PSI the servers store their data-set in a standard Cuckoo hashing table. Hence, the storage overhead is constant and updates take constant time.

PIR-PSI can be easily extended so that the client privately learns associated data for each item in the intersection. In the case of a secure messaging app, the server may hold a mapping of email addresses to public keys. A client may wish to obtain the public keys of any users in its own address book.

As mentioned previously, PIR-PSI can be extended to a 3-server or 4-server variant where some of the servers hold only *secret shares* of DB, with security holding if no two servers collude (cf. Sect. 9.8.2). This setting may be a better fit for practical deployments of contact discovery, since a service provider can recruit the help of other independent organizations, neither of which need to know the provider's user database. Holding the user database in secret shared form reduces the amount of data that the service provider retains about its users[5] and gives stronger defense against data exfiltration.

[5] https://www.reuters.com/article/us-usa-cyber-signal/signal-messaging-app-turns-over-minimal-data-in-first-subpoena-idUSKCN1241JM

9.1.3 Related Work and Comparison

PSI with Asymmetric Set Sizes As discussed in the previous section, many protocols for PSI are not well-suited when the two parties have input sets of very different sizes. For example, [KKRT16] is one of the fastest PSI protocols for large sets of similar size, but requires communication of at least $O\big(\sigma(N+n)\big)$ where σ is a statistical security parameter. This cost makes these approaches prohibitive for contact discovery, where N is very large.

Only a handful of works specifically consider the case of asymmetric set sizes. Chen et al. (CLR) [CLR17] use somewhat-homomorphic encryption to reduce communication to logarithmic in the large set size N. This work was recently extended in [CHLR18] to also support associated data. [PSZ18] compares several approaches. Building on top of that, Kiss et al. (KLSAP) [KLS+17] describe an approach that defers $O(N)$ communication to a pre-processing phase, in which the server sends a large Bloom filter containing $\mathsf{AES}(k,x)$ for each of its items x. To perform contact discovery, the parties use Yao's protocol to obliviously evaluate AES on each of the client's items, so the client can then probe the Bloom filter for membership. Resende and Aranha (RA) [RA18] describe a similar approach in which the server sends a large message during the preprocessing phase. In this case, the large message is a more space-efficient Cuckoo filter.

In Sect. 9.7 we provide a detailed comparison between CLR, KLSAP, RA, and PIR-PSI. The main qualitative difference is the security model. The protocols listed above are in a two-party setting involving a single server, whereas PIR-PSI involves several *non-colluding* servers. We discuss the consequences of this security model more thoroughly in Sect. 9.4. Besides this difference, the KLSAP and RA protocols require significant offline communication and persistent storage for the client.

Keyword PIR Chor, et al. [CGN98] defined a variant of PIR called *keyword PIR,* in which the client has an item x, the server has a set S, and the client learns whether $x \in S$. Our construction can be viewed as a kind of multi-query keyword PIR with symmetric privacy guarantee for the server (the client learns *only* whether $x \in S$). In [BGI16, Appendix A] a similar method is proposed that could be used to implement private contact discovery.

The keyword-PIR method would have computation cost of $O(Nn\ell)$ AES invocations, where the items in the parties' sets come from the set $\{0,1\}^\ell$. By contrast, our protocol has server computation cost $O(N \log n)$. We can imagine all parties first hashing their inputs down to a small ℓ-bit fingerprint before performing contact discovery. But with N items in the server's set, we must have $\ell \geq s + 2\log N$ to limit the probability of a collision to 2^{-s}. In practice $\ell > 60$ is typical, hence $n\ell$ is much larger than $\log n$.

Signal's Private Contact Discovery In late 2017 the Signal messaging service announced a solution for private contact discovery based on Intel SGX, which they plan to deploy.[6] The idea is for the client to send their input set directly into an SGX enclave on the server, where

[6] https://signal.org/blog/private-contact-discovery/

it is privately compared to the server's set. The enclave can use remote attestation to prove the authenticity of the server software.

The security model for this approach is incomparable to ours and others, as it relies on a trusted hardware assumption. Standard two-party PSI protocols rely on standard cryptographic hardness assumptions. Our PIR-PSI protocol relies on cryptographic assumptions as well as an assumption of non-collusion between two servers. There are several disadvantages of SGX, which we address in Sect. 2.6.3.

9.2 Preliminaries

In this section we introduce the preliminaries and the notation used in this chapter. Background information on PIR is provided in Sect. 2.5. In this work, we are specifically interested in 2-server PIR schemes.

Throughout this chapter we use the following notation, which we summarize in Tab. 9.1: The large server set has N items, while the small client set has n items. The length of each item is ℓ bits. In our implementation we use $\ell = 128$ bits. We write $[m] = \{1, \ldots, m\}$. The computational and statistical security parameters are denoted by κ, σ, respectively. In our implementation we use $\kappa = 128$ and $\sigma = 40$.

Table 9.1: Notation: Parameters and symbols used.

Parameter	Symbol
symmetric security paramter [bits]	$\kappa = 128$ bits
statistical security paramter [bits]	$\sigma = 40$ bits
element length [bits]	$\ell = 128$ bits
client set size [elements]	n
server set size [elements]	N
Cuckoo table expansion (Sect. 9.3.2)	e
Cuckoo table size [elements] (Sect. 9.3.2)	$m = e \cdot N$
DPF bins (Sect. 9.3.4)	β
DPF bin size [elements] (Sect. 9.3.4)	μ
PIR block size (Sect. 9.3.5)	b
scaling factor (Sect. 9.5.4)	c
server database	DB
server Cuckoo hash table	CT
the number of Cuckoo hash functions	k

9.2.1 Distributed Point Functions

Gilboa and Ishai [GI14] proposed the notion of a distributed point function (DPF). For our purposes, a DPF with domain size N consists of the following algorithms:

- $\mathsf{DPF.Gen}$: a randomized algorithm that takes index $i \in [N]$ as input and outputs two (short) *keys* k_1, k_2.
- $\mathsf{DPF.Expand}$: takes a short key k as input and outputs a long *expanded key* $K \in \{0,1\}^N$.[7]

The correctness property of a DPF is that, if $(k_1, k_2) \leftarrow \mathsf{DPF.Gen}(i)$ then $\mathsf{DPF.Expand}(k_1) \oplus \mathsf{DPF.Expand}(k_2)$ is a string with all zeros except for a 1 in the i-th bit.

A DPF's security property is that the marginal distribution of k_1 alone (resp. k_2 alone) leaks no information about i. More formally, the distribution of k_1 induced by $(k_1, k_2) \leftarrow \mathsf{DPF.Gen}(i)$ is computationally indistinguishable from that induced by $(k_1, k_2) \leftarrow \mathsf{DPF.Gen}(i')$, for all $i, i' \in [N]$.

PIR from DPF Distributed point functions can be used for 2-party PIR in a natural way. Suppose the servers hold a database DB of N strings. The client wishes to read item $\mathsf{DB}[i]$ without revealing i. Using a DPF with domain size N, the client can compute $(k_1, k_2) \leftarrow \mathsf{DPF.Gen}(i)$, and send one k_b to each server. Server 1 can expand k_1 as $K_1 = \mathsf{DPF.Expand}(k_1)$ and compute the inner product:

$$K_1 \cdot \mathsf{DB} \stackrel{\text{def}}{=} \bigoplus_{j=1}^{N} K_1[j] \cdot \mathsf{DB}[j].$$

Server 2 computes an analogous inner product. The client can then reconstruct as:

$$(K_1 \cdot \mathsf{DB}) \oplus (K_2 \cdot \mathsf{DB}) = (K_1 \oplus K_2) \cdot \mathsf{DB} = \mathsf{DB}[i]$$

since $K_1 \oplus K_2$ is zero everywhere except in position i.

This scheme works analogous to Chor et al.'s linear summation PIR (CGKS), described in Sect. 2.5.2. The DPF keys k_i are the corresponding queries q_i in CGKS.

BGI construction Boyle et al. [BGI15; BGI16] describe an efficient DPF construction in which the size of the (unexpanded) keys is roughly $\kappa(\log N - \log \kappa)$ bits, where κ is the symmetric security parameter.

Their construction works by considering a full binary tree with N leaves. To expand the key, the $\mathsf{DPF.Expand}$ algorithm performs a PRF evaluation for each node in this tree. The (unexpanded) keys contain a PRF block for each level of the tree.

As described, this gives unexpanded keys that contain κ bits for each level of a tree of height $.\log N$. To achieve $\kappa(\log N - \log \kappa)$ bits total, BGI suggest the following "early termination optimization": Treat the expanded key as a string of N/κ characters over the alphabet

[7] The original DPF definition also requires efficient random access to this long expanded key. Our usage of DPF does not require this feature.

PARAMETERS: Set sizes m and n; Two parties: sender $\mathcal{S}$ and receiver $\mathcal{R}$

FUNCTIONALITY:

- Wait for an input $X = \{x_1, x_2, \ldots, x_n\} \subseteq \{0,1\}^*$ from sender $\mathcal{S}$ and an input $Y = \{y_1, y_2, \ldots, y_m\} \subseteq \{0,1\}^*$ from receiver $\mathcal{R}$
- Give output $X \cap Y$ to the receiver $\mathcal{R}$.

Figure 9.1: Private Set Intersection Functionality $\mathcal{F}_{\text{psi}}^{m,n}$

$\{0,1\}^\kappa$. This leads to a tree of height $\log(N/\kappa) = \log N - \log \kappa$. An extra κ-bit value is required to deal with the longer characters at the leaves, but overall the total size of the unexpanded keys is roughly $\kappa(\log N - \log \kappa)$ bits. In practice, we use hardware-accelerated AES-NI as the underlying PRF, with $\kappa = 128$.

The major difference between this scheme and CGKS-based schemes is the size of the queries: in CGKS and RAID-PIR, the queries q_i scale linearly with the size of the database, while in the newer DPF-based PIR the queries k_i scale logarithmic in the database size. Currently PIR from DPFs can only efficiently be built for two servers and the extension to more than two servers requires much more expensive computation [BGI15].

9.2.2 Private Set Intersection (PSI)

Private Set Intersection (PSI) is an application of secure computation that allows parties, each holding a set of private items, to compute the intersection of their sets without revealing anything except for the intersection itself. We describe the ideal functionality for PSI in Fig. 9.1. Based on oblivious polynomial evaluation, the first PSI protocol was formally introduced in 2004 by [FNP04]. However, this protocol requires a quadratic number of expensive public key operations. Over the last decade, many PSI protocols [DKT10; DT10; DCW13; PSZ14; PSSZ15; KKRT16; PSZ18; OOS17; RR17; CLR17; KLS+17; RA18] were proposed with linear (or even sub-linear) communication and computation complexity, which made PSI become practical for many applications. These PSI protocols can be classified into two different settings based on the size of party's input set: (1) symmetric, where the sets size have approximately the same size; (2) asymmetric, where one of the sets is severely smaller than the other.

The most efficient PSI approaches [KKRT16; PSZ18] for symmetric sets are based on efficient Oblivious Transfer (OT) extension, cf. Sect. 2.3.

9.3 Our Construction: PIR-PSI

We make use of the previously described techniques to achieve a practical solution for privacy-preserving contact discovery, called **PIR-PSI**. We assume that the service provider's large user database is held on 2 separate servers. To perform private contact discovery, a client interacts with both servers simultaneously. The protocol's best security properties hold when these two servers do not collude. Variants of our construction for 3 and 4 servers are described in Sect. 9.8.2, in which some of the servers hold only *secret shares* of the user database. We develop the protocol step-by-step in the following sections. The full protocol is shown in Prot. 9.1 on p. 194.

9.3.1 Warmup: PIR-PEQ

At the center of our construction is a technique for combining a Private Equality Test (PEQ) – a special case of PSI when the parties have one item each) [PSZ14] with a PIR query. Suppose a client holds private input i, x and wants to learn whether $\mathsf{DB}[i] = x$, where the database DB is the private input of the servers.

First recall the linear summation PIR scheme from Sect. 2.5. This PIR scheme has *linear reconstruction* in the following sense: the client's output $\mathsf{DB}[i]$ is equal to the *XOR of the responses* from the two servers.

Suppose a PIR scheme with linear reconstruction is modified as follows: the client sends an additional mask r to server #1. Server #1 computes its PIR response v_1 and instead of sending it to the client, sends $v_1 \oplus r$ to server #2. Then server #2 computes its PIR response v_2 and can reconstruct the masked result $v_2 \oplus (v_1 \oplus r) = DB[i] \oplus r$. We refer to this modification as **designated-output PIR**, as the client designates server #2 to learn the (masked) output.

The client can now perform a standard 2-party secure computation with server #2. In particular, they perform a PEQ with input $x \oplus r$ from the client and $\mathsf{DB}[i] \oplus r$ from the server. As long as the PEQ is secure, and the two servers do not collude, the servers learn nothing about the client's input. If the two servers collude, they can learn i but not x.

This warm-up problem is not yet sufficient for computing private set intersection between a set X and DB, since the client may not know which location in DB to test against. Next we will address this by structuring the database as a Cuckoo hash table.

9.3.2 Cuckoo Hashing

Cuckoo hashing has seen extensive use in Private Set Intersection protocols [PSZ14; PSSZ15; KKRT16; PSZ18; OOS17] and in related areas such as privacy preserving genomics [CCL+17]. This hashing technique uses an array with m entries and k hash functions $h_1, \ldots, h_k : \{0,1\}^* \to [m]$. The guarantee is that an item x will be stored in a hash table at one of the locations indexed by $h_1(x), ..., h_k(x)$. Furthermore, only a single item will be assigned to each entry.

Typically, k is quite small (we use $k = 3$). When inserting x into the hash table, a random index $i \in [k]$ is selected and x is inserted at location $h_i(x)$. If an item y currently occupies that location, it is evicted and y is re-inserted using the same technique. This process is repeated until all items are inserted or some upper bound on the number of trials has been reached. In that latter case, the procedure can either abort or place the item in a special location called the stash. We choose Cuckoo hashing parameters such that this happens with sufficiently low probability (see Sect. 9.5.2 and Sect. 9.5.3), i.e., no stash is required.

In our setting the server encodes its set DB into a Cuckoo hash table CT of size $m = e \cdot N$, where $e > 1$ is an expansion factor. That way, the client (with much smaller set X) must probe only $k|X|$ positions of the Cuckoo table to compute the intersection. Using the PIR-PEQ technique just described makes the communication linear in $|X|$ but only logarithmic in $|\mathsf{DB}|$, due to the complexity of the DPF-PIR queries.

9.3.3 Hiding the Cuckoo Locations

There is a subtle issue if one applies the PIR-PEQ idea naïvely. When the client learns that $y \in (\mathsf{DB} \cap X)$, he/she will in fact learn whether y is placed in position $h_1(y)$ or $h_2(y)$ or $h_3(y)$ of the Cuckoo table. But this *leaks more than the intersection* $\mathsf{DB} \cap X$, in the sense that it cannot be simulated given just the intersection! The placement of y in the Cuckoo table CT depends indirectly on all the items in DB.[8] Note that this is not a problem for other PSI protocols, since there the party who processes their input with Cuckoo hashing receives the output from the PEQs. For contact discovery, we require these to be different parties.

To overcome this leakage, we design an efficient oblivious shuffling procedure that obscures the Cuckoo location of an item. First, let us start with a simple case with two hash functions h_1, h_2, where the client holds a single item x. This generalizes in a natural way to $k = 3$ hash functions. Full details, and the extension to multiple items are provided in our paper [DRRT18, Appendix].

The client will generate and send two PIR queries, for positions $h_1(x)$ and $h_2(x)$ in CT. The client also sends two masks r_1 and r_2 to server #1 which serve as masks for the designated-output PIR. Server #1 randomly chooses whether to swap these two masks. That is, it chooses a random permutation $\sigma : \{1,2\} \to \{1,2\}$ and masks the first PIR query with $r_{\sigma(1)}$ and the second with $r_{\sigma(2)}$. Server #2 then reconstructs the designated PIR output, obtaining $\mathsf{CT}[h_1(x)] \oplus r_{\sigma(1)}, \mathsf{CT}[h_2(x)] \oplus r_{\sigma(2)}$. The client now knows that if $x \in \mathsf{DB}$, then server #2 must hold either $x \oplus r_1$ or $x \oplus r_2$.

Now instead of performing a private equality test, the client and server #2 can perform a standard **2-party PSI** with inputs $\{x \oplus r_1, x \oplus r_2\}$ from the client and the designated PIR

[8] For instance, say the client holds set X and (somehow) knows the server has set $\mathsf{DB} = X \cup \{z\}$ for some unknown z. It happens that for many $x \in X$ and $i \in [k]$, $h_i(x)$ equals some location ρ. Then with good probability some $x \in X$ will occupy location ρ. However, after testing location ρ the client learns no $x \in X$ occupies this location. Then the client has learned some information about z (namely, that $h_i(z) = \rho$ is likely for some $i \in [k]$), even though z is not in the intersection.

values $\{\mathsf{CT}[h_1(x)] \oplus r_{\sigma(1)}, \mathsf{CT}[h_2(x)] \oplus r_{\sigma(2)}\}$ from server #2. This technique perfectly hides whether x was found at $h_1(x)$ or $h_2(x)$. While it is possible to perform a separate 2-item PSI for each PIR query, it is actually more efficient (when using the 2-party PSI protocol of [KKRT16]) to combine all of the PIR queries into a single PSI with $2n$ elements each.

Because of the random masks, this approach may introduce a false positive, where $\mathsf{CT}[j] \neq x$ but $\mathsf{CT}[j] \oplus r = x \oplus r'$ (for some masks r and r'), leading to a PSI match. In our implementation we only consider items of length 128, so the false positive probability taken over all client items is only $2^{-128+\log_2((2n)^2)}$, by a standard union bound.

9.3.4 Optimization: Binning Queries

The client probes the servers' database in positions determined by the Cuckoo hash functions. Under the reasonable assumptions that (1) the client's input items are chosen independently of the Cuckoo hash functions and (2) the Cuckoo hash functions are random functions, the client probes the Cuckoo table CT in *uniformly distributed* positions.

Knowing that the client's queries are randomly distributed in the database, we can take advantage of the fact that the queries are "well-spread-out" in some sense. Consider dividing the server's CT ($m = N \cdot e$ entries) into β bins of equal size. The client will query the database in nk random positions, so the distribution of these queries into the β bins can be modeled as a standard balls and bins problem. We can choose a number β of bins and a maximum load μ so that Pr[there exists a bin with $\geq \mu$ balls] is below some threshold (say, 2^{-40} in practice). With such parameters, the protocol can be optimized as follows.

The parties agree to divide CT into β regions of equal size. The client computes the positions of CT that he/she wishes to query, and collects them according to their region. The client adds dummy PIR queries until there are *exactly* μ queries to each region. The dummy items are necessary because revealing the number of (non-dummy) queries to each region would leak information about the client's input to the server. For each region, the server treats the relevant m/β items as a sub-database, and the client makes exactly μ PIR queries to that sub-database.

This change leads to the client making more PIR queries than before (because of the dummy queries), but each query is made to a much smaller PIR instance. Looking at specific parameters reveals that binning can give *significant performance improvements*.

It is well-known that with $\beta = O\big(nk/\log(nk)\big)$ bins, the maximum number of balls in any bin is $\mu = O\big(\log(nk)\big)$ with very high probability. The total number of PIR queries (including dummy ones) is $\beta\mu = \Theta(nk)$. That is, the binning optimization with these parameters increases the number of PIR queries by a constant factor. At the same time, the PIR database instances are all smaller by a large factor of $\beta = O\big(nk/\log(nk)\big)$. The main bottleneck in PIR-PSI is the computational cost of the server in answering PIR queries, which scales linearly with the size of the PIR database. Reducing the size of all effective PIR databases by a factor

of β has a significant performance impact. In general, tuning the constant factors in β (and corresponding μ) gives a wide trade-off between communication and computation.

9.3.5 Optimization: Larger PIR Blocks

So far we have assumed a one-to-one correspondence between the entries in the server's Cuckoo table and the server's database for purposes of PIR. That is, we invoke PIR with an v-item database corresponding to a region of the Cuckoo table with v entries.

Suppose instead that we use PIR for an $v/2$-item database, where each item in the PIR database consists of a block of 2 Cuckoo table entries. The client now queries for a single item, but the PIR returns a block of 2 Cuckoo table entries. The server will feed both entries into the 2-party PSI, so that these extra neighboring items are not leaked to the client.

This change affects the various costs in the following ways: (1) It reduces the number of cryptographic operations needed for the server to answer each PIR query by half; (2) It does not affect the computational cost of the final inner product between the expanded DPF key and PIR database entries, since this is over the same amount of data; (3) It reduces the communication cost of each PIR query by a small amount (κ bits); (4) It doubles all costs of the 2-party PSI, since the server's PSI input size is doubled.

Of course, this approach can be generalized to use a PIR blocks of size b, so that a PIR database of size v/b is used for v Cuckoo table entries. This presents a trade-off between communication and computation, discussed further in Sect. 9.5.

9.3.6 Asymptotic Performance

With these optimizations the computational complexity for the client is the generation of $\beta\mu = O(n)$ PIR queries of size $O(\log(N/\kappa\beta))$. As such they perform $O(n\log(N/\kappa\beta)) = O(n\log(N\log n/\kappa n))$ calls to a PRF and send $O(\kappa n\log(N/(\kappa n\log n)))$ bits. The servers must expand each of these queries to a length of $O(N/\beta)$ bits which requires $O(N\mu/\kappa) = O(N\log n/\kappa)$ calls to a PRF.

9.4 Security

9.4.1 Semi-Honest Security

The most basic and preferred setting for PIR-PSI is when at most one of the parties is passively corrupt (a.k.a. semi-honest). This means that the corrupt party does not deviate from the protocol. Note that restricting to a single corrupt party means that we assume *non-collusion* between the two PIR servers.

Theorem 2. ***Semi-Honest Security:*** *The $\mathcal{F}_{\text{PIR-PSI}}$ protocol (Prot. 9.1) is a realization of $\mathcal{F}_{psi}^{n,N}$ secure against a semi-honest adversary corrupting at most one party in the $\mathcal{F}_{psi}^{nk,\beta\mu}$ hybrid model.*

Proof. In the semi-honest non-colluding setting it is sufficient to show that the transcript of each party can be simulated given their input and output. That is, we consider three cases where each one of the parties is individually corrupt.

Corrupt Client: Consider a corrupt client with input X and output $Z = X \cap \mathsf{DB}$. We show a simulator that can simulate the view of the client given just X and Z. First observe that the simulator playing the role of both servers knows the permutation $\pi = \pi_2 \circ \pi_1$ and the vector of masks r. As such, response v can be computed as follows. For $x \in Z$ the simulator randomly samples one of k masks $r_{i_1}, \ldots, r_{i_k} \in r$ which the client will use to mask x and add $x \oplus r_{i_j}$ to v. Pad v with random values not contained in union u to size $\beta\mu$ and forward v to the ideal $\mathcal{F}_{\text{PSI}}^{nk,\beta\mu}$. Conditioned on no spurious collisions between v and u in the real interaction (which happen with negligible probability, following the discussion in Sect. 9.3.3) this ideal interaction perfectly simulates the real interaction.

One additional piece of information learned by the client is that Cuckoo hashing on the set DB with hash function $h_1, ..., h_k$ succeeded. However, by the choice of Cuckoo parameter, this happens with overwhelming probability and therefore the simulator can ignore the case of Cuckoo hashing failure.

Corrupt Server: Each server's view consists of:

- PIR queries (DPF keys) from the client; since a single DPF key leaks nothing about the client's query index, these can be simulated as dummy DPF keys.
- Messages in the oblivious masking step, which are uniformly distributed as discussed in Sect. 9.3.3.
- In the case of server #2, masked PIR responses from server #1, which are uniformly distributed since they are masked by the $\vec{r}$ values.

■

9.4.2 Colluding Servers

If the two servers collude, they will learn both DPF keys for every PIR query, and hence learn the locations of all client's queries into the Cuckoo table. These locations indeed leak information about the client's set, although the exact utility of this leakage is hard to characterize. The servers still learn nothing from the PSI subprotocol by colluding since only one of the servers is involved.

It is worth providing some context for private contact discovery. The state-of-the-art for contact discovery is a naïve (insecure) hashing protocol, where both parties simply hash each item of their set, the client sends its hashed set to the server, who then computes the

intersection. This protocol is insecure because the server can perform a dictionary attack on the client's inputs.

However, *any PSI protocol* (including ours) can be used in the following way. First, the parties hash all their items, and then use the hashed values as inputs to the PSI. We follow this approach in PIR-PSI. As long as the hash function does not introduce collisions, pre-hashing the inputs preserves the correctness of the PSI protocol.

A side effect of pre-hashing inputs is that the parties never use their "true" inputs to the PSI protocol. Therefore, the PSI protocol cannot leak more than the hashed inputs — identical to what the status quo naïve hashing protocol leaks. Again, this observation is true for *any* PSI protocol. In the specific case of PIR-PSI, if parties pre-hash their inputs, then even if the two servers collude (even if they are malicious), the overall protocol can never leak more about the client's inputs than naïve hashing. Relative to existing solutions implemented in current applications, that use naïve hashing, there is no extra security risk for the client to use PIR-PSI.

9.4.3 Malicious Client

Service providers may be concerned about malicious behavior (i.e., protocol deviation) by clients during contact discovery. Since servers get no output from PIR-PSI, there is no concern over a malicious client inducing inconsistent output for the servers. The only issue is therefore what unauthorized information a malicious client can learn about DB.

Overall the only information the client receives in PIR-PSI is from the PSI subprotocol. We first observe that the PSI subprotocol we use ([KKRT16]) is naturally secure against a malicious client, when it is instantiated with an appropriate OT extension protocol. This fact has been observed in [Lam16; OOS17]. Hence, in the presence of a malicious client we can treat the PSI subprotocol as an ideal PSI functionality. The malicious client can provide at most nk inputs to the PSI protocol — the functionality of PSI implies that the client therefore learns no more than nk bits of information about DB. This leakage is comparable to what an honest client would learn by having an input set of nk items.

Modifications for More Precise Leakage Characterization In DPF-based PIR clients can make malformed PIR queries to the server, by sending k_1, k_2 so that $\mathsf{DPF.Expand}(k_1) \oplus \mathsf{DPF.Expand}(k_2)$ has more than one bit set to 1. The result of such a query will be the XOR of several DB positions.

However, Boyle et al. [BGI16] describe a method by which the servers can ensure that the client's PIR queries (DPF shares) are well-formed. The technique increases the cost for the servers by a factor of roughly 3× (but adds no cost to the client). The servers agree on a pseudorandom vector $\vec{w}$ of the same length as DB. The entries of $\vec{w}$ are elements in some field with large characteristic. Let $\vec{w}^2$ denote the result of squaring every component in $\vec{w}$. Each server computes inner products $K \cdot \vec{w}$ and $K \cdot \vec{w}^2$, where K is its expanded DPF share. If the DPF query was well-formed, then $(K_1 \oplus K_2) \cdot \vec{w}$ will consist of a single element, whose square

is equal to $(K_1 \oplus K_2) \cdot \vec{w}^2$. If the DPF query is malformed, then $(K_1 \oplus K_2) \cdot \vec{w}$ will be a sum of more than one entry of $\vec{w}$, whose square will in general not be equal to the corresponding sum of squares from $(K_1 \oplus K_2) \cdot \vec{w}^2$. Hence the servers simply check whether:

$$\left([K_1 \cdot \vec{w}] \oplus [K_2 \oplus \vec{w}]\right)^2 \stackrel{?}{=} [K_1 \cdot \vec{w}^2] \oplus [K_2 \oplus \vec{w}^2].$$

Note that each bracketed term is just a single field element. The servers can use a small secure computation to test this expression, since revealing the result of $(K_1 \oplus K_2) \cdot \vec{w}$ would reveal an honest client's query location. Note that we currently do not implement this additional feature in PIR-PSI.

The client may also send malformed values in the oblivious masking phase. But since the servers use those values independently of DB, a simulator (who sees the client's oblivious masking messages to both servers) can simulate what masks will be applied to the PIR queries. Overall, if the servers ensure validity of the client's PIR values, we know that server #1's input to PSI will consist of a collection of nk individual positions from DB, each masked with values that can be simulated.

When deployed in practice, service providers can additionally use rate-limiting mechanisms that ensure that a certain account or IP-address cannot query more than a given threshold of contacts in a given time frame.

9.5 Implementation

We implemented a prototype of our $\mathcal{F}_{\text{PIR-PSI}}$ protocol described in Prot. 9.1. Our implementation uses AES as the underlying PRF (for the distributed point function of [BGI16]) and relies on the PSI implementation of [KKRT16] and the oblivious transfer in libOTe[9] with security against passive adversaries. OTs with security against malicious clients [OOS17] could be added at small additional overhead. Our implementation is publicly available on GitHub[10].

9.5.1 System-level Optimizations

We highlight here system-level optimizations that contribute to the high performance of our implementation. We analyze their impact on performance in Sect. 9.6.3.

Optimized DPF Full-domain Evaluation Recall that the DPF construction of [BGI16] can be thought of as a large binary tree of PRF evaluations. Expanding the short DPF key corresponds to computing this entire tree in order to learn the values at the leaves. The process of computing the values of all the leaves is called "full-domain evaluation" in [BGI16], and is the major computational overhead for the servers in our protocol. To limit its impact our implementation takes full advantage of instruction vectorization (SIMD). Most modern

[9] https://github.com/osu-crypto/libOTe
[10] https://github.com/osu-crypto/libPSI

Parameters: X is the client's set, DB is the set held by server #1 and #2, where $X, \mathsf{DB} \subseteq \{0,1\}^\ell$. $n = |X|, N = |\mathsf{DB}|$, k is the number of Cuckoo hash functions. The protocol uses an instance of $\mathcal{F}_{\text{PSI}}$ with input length ℓ. σ, κ are the statistical and computational security parameter.

1. **Cuckoo Hashing:** Both servers agree on the same k random hash functions $h_1, ..., h_k : \{0,1\}^\ell \to [m]$ and Cuckoo table size $m = |\mathsf{CT}|$ such that inserting N items into Cuckoo hash table CT succeeds with probability $\geq 1 - 2^{-\sigma}$.

   The servers compute Cuckoo table CT such that for all $y \in \mathsf{DB}$, $\mathsf{CT}[h_i(y)] = y$ for some $i \in k$.

2. **Query:** Upon the client receiving their set X,
   a) Send $n = |X|$ to the servers. All parties agree on the number of bins $\beta = O(n/\log n)$, and their size $\mu = O(\log n)$ (see Sect. 9.5.4). Define the region DB_i as all locations $j \in [m]$ of CT such that $(i-1)\frac{m}{\beta} < j \leq i\frac{m}{\beta}$.
   b) For $x \in X, h \in \{h_1, ..., h_k\}$, let $h(x)$ index the j-th location in bin i. The client adds (x, j) to bin $B[i]$.
   c) For bin $B[i]$,
      i. Pad $B[i]$ to size μ with the pair $(\perp, 0)$.
      ii. For $(x, j) \in B[i]$ in a random order, the client constructs the keys $k_1, k_2 = \mathsf{DPF.Gen}(j)$. Send k_s to server $\#s$.
      iii. Server $\#s$ expands their key $K_s = \mathsf{DPF.Expand}(k_s)$ and compute $v_s[\rho] = \mathsf{DB}_i \cdot K_s$ where k_s is the ρ-th DPF key received.

3. **Shuffle:** Observe that, $(v_1 \oplus v_2)[\rho] = \mathsf{CT}[j_\rho]$, where j_ρ is the ρ-th PIR location.
   a) The client samples a permutation π of $\beta\mu$ items such that for the i-th $x \in X$, and $j \in [k]$ it holds that the $\pi\big((i-1)k + j\big)$-th DPF key corresponds to the query of x at location $h_j(x)$.
   b) The client samples $w_1, w_2 \leftarrow \{0,1\}^\kappa$ and sends w_1 to server #1, and w_2 to server #2. Define shared terms $t = \text{PRG}(w_2)$, $(r||s||\pi_1) = \text{PRG}(w_1)$ where r, s, t are random vectors of the same size as v_i and π_1 is a random permutation of $\beta\mu$ items. The client sends π_2 to server #2 such that $\pi_2 \circ \pi_1 = \pi$ and sends $p_0 = t \oplus \pi(s)$ to server #1.
   c) Server #1 samples a random permutation σ of $\beta\mu$ items such that for all $i \in [\beta\mu/k]$ it holds that $\sigma(j) \in S_i$ where $j \in S_i = (i-1)k + \{1, ..., k\}$. Server #1 sends $p_1 = \pi_1\big(\sigma(r) \oplus s\big)$ and $p_2 = p_0 \oplus v_1$ to server #2.
   d) Server #2 computes $v = v_2 \oplus \pi_2(p_1) \oplus t \oplus p_2$.

4. **PSI:** The client then computes the masked versions of the $x_i \in X$ as $x_i' = \{x_i \oplus r_{\pi((i-1)k+1)}, ..., x_i \oplus r_{\pi(ik)}\}$ and computes u as the union of all these sets. The client and server #2 respectively send u, v to $\mathcal{F}_{\mathsf{psi}}^{nk,\beta\mu}$ such that the client receives $z = u \cap v$. The client outputs $\{x_i : x_i' \cap z \neq \emptyset\}$.

Protocol 9.1: Our 2-server PIR-PSI protocol $\mathcal{F}_{\text{PIR-PSI}}$.

processors are capable of performing the same instruction on multiple (e.g., 8) pieces of data. However, to fully utilize this feature, special care has to be taken to ensure that the data being operated on is in cache and contiguous. To meet these requirements, our implementation first evaluates the top 3 levels of the DPF binary tree, resulting in 8 remaining independent subtrees. We then perform SIMD vectorization to traverse all 8 subtrees simultaneously.

Combining this technique with others, such as the removal of `if` statements in favor of array indexing, our final implementation is roughly 20× faster then a straight-forward (but not careless) sequential implementation and can perform full-domain DPF evaluation at a rate of 2.6 billion bits/s on a single core.

Single-pass Processing With the high raw throughput of DPF evaluation, it may not be surprising that it was no longer the main performance bottleneck. Instead, performing many passes over the dataset (once for each PIR query) became the primary bottleneck by an order of magnitude. To address this issue we further modify the workflow to evaluate all DPFs (PIR queries) for a single bin in parallel using vectorization.

That is, for all μ DPF evaluations in a given bin, we evaluate the binary trees in parallel, and traverse the leaves in parallel. The values at the leaves are used to take an inner product with the database items, and the parallel traversal ensures that a given database item only needs to be loaded from main memory (or disk) once. This improves (up to 5×) the performance of the PIR protocol on large datasets, compared to the straightforward approach of performing multiple sequential passes of the dataset.

This technique could be combined with more efficient efficient matrix multiplication from [LG15] or the four Russians precomputation in Sect. 8.3.5.

Parallelization Beyond the optimizations listed above, we observe that our protocol simply is very amenable to parallelization. In particular, our algorithm can be parallelized both within the DPF evaluation using different subtrees and by distributing the PIR protocols for different bins between several cores/machines. In the setting where thousands of these protocols are being executed a day on a fixed dataset, distributing bin evaluations between different machines can be extremely attractive due to the fact that several protocol instances can be batched together to gain even greater benefits of vectorization and data locality. The degree of parallelism that our protocol allows can be contrasted with more traditional PSI protocols which require several global operations, such as a single large shuffle of the server's encoded set (as in [PSZ14; PSSZ15; KKRT16; PSZ18; OOS17]).

9.5.2 Cuckoo Hashing Parameters

For optimal performance it is crucial to minimize the Cuckoo table size and the number of hash functions. The table size affects how much work the servers have to perform, while the number of hash functions k affects the number of client queries. We wish to minimize both parameters while ensuring Cuckoo hashing failures are unlikely, as this leaks a single bit about DB. Several works [FMM09; DGM+10] have analyzed Cuckoo hashing from an asymptotic perspective and show that the failure probability decreases exponentially with increasing table size. However, the exact relationship between the number of hash functions, stash size, table size and security parameter is unclear from such an analysis.

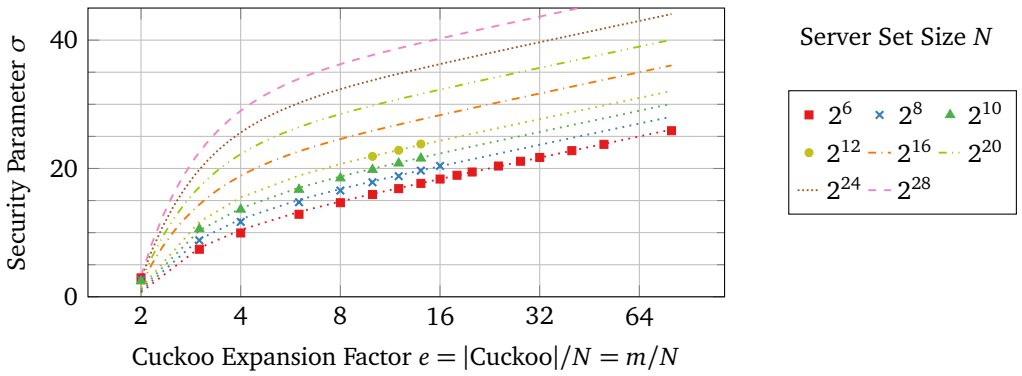


Figure 9.2: Empirical (marks) and interpolated (dashed/dotted lines) Cuckoo success probability for $k = 2$ hash functions and different set sizes N.

We solve this problem by providing an accurate relationship between these parameters through extensive experimental analysis of failure probabilities. That is, we ran Cuckoo-hashing instances totalling nearly 1 trillion items hashed, over two weeks for a variety of parameters. As a result our bounds are significantly more accurate and general than previous experiments [PSZ18; CLR17]. We analyzed the resulting distribution to derive highly predictive equations for the failure probability. We find that $k = 2$ and $k \geq 3$ behave significantly different and therefore derive separate equations for each.

Our extrapolations are graphed in Figs. 9.2 and 9.3, and the specifics of the formulas are given in Sect. 9.5.3.

9.5.3 Cuckoo Hashing Failure Probability Formula

Let $e > 1$ be the expansion factor denoting that N items are inserted into a Cuckoo table of size $m = eN$. Fig. 9.2 shows the security parameter (i.e., σ, such that the probability of hashing failure is $2^{-\sigma}$) of Cuckoo hashing with $k = 2$ hash functions. As N becomes larger, σ scales linearly with $\log_2 N$ and with the stash size s, which matches the results of [DGM+10]. For $e \geq 8$ and $k = 2$, we interpolate the relationship as the linear equation

$$\sigma = \big(1 + 0.65s\big)\big(3.3\log_2(e) + \log_2(N) - 0.8\big) \tag{9.1}$$

For smaller values of e, we observe that σ quickly converges to 1 at $e = 2$. We approximate this behavior by subtracting $\big(5\log_2(N) + 14\big)e^{-2.5}$ from Eq. (9.1). We note that these exact interpolated parameters are specific to our implementation which uses a specific eviction policy (linear walk) and re-insert bound (100). However, we observed similar bounds for other parameters and evictions strategies (e.g., random walks or 200 re-insert bound).

We also consider the case $k = 3$, shown in Fig. 9.3 and find that it scales significantly better than $k = 2$. For instance, at $e = 2$ we find $\sigma \approx 100$ for interesting set sizes while the same value of e applied to $k = 2$ results in $\sigma \approx 1$. As before we find that σ grows linearly with

the expansion factor e. Unlike in the case of $k = 2$, we observe that increasing N has a slight negative effect on σ. Namely, doubling N roughly decreases σ by 2. However, the slope at which σ increases for $k = 3$ is much larger than $k = 2$ and therefore this dependence on $\log N$ has little impact on σ. We summarize these findings for $k = 3$ as the linear equation

$$\sigma = a_N e + b_N \tag{9.2}$$

where $a_N \approx 123.5$ and $b_N \approx -130 - \log_2 N$. Here we use an approximation to hide an effect that happens for small $N \leq 512$. In this regime we find that the security level quickly falls. In particular, the slope a_N and intercept b_N go to zero roughly following the normal distribution CDF. By individually interpolating these variable we obtain accurate predictions of σ for $N \geq 4$. Our interpolations show that $a_N = 123.5 \cdot \mathrm{CDF}_{\mathrm{NORMAL}}(x = N, \mu = 6.3, \sigma = 2.3)$ and $b_N = -130 \cdot \mathrm{CDF}_{\mathrm{NORMAL}}(x = N, \mu = 6.45, \sigma = 2.18) - \log_2 N$.

For $k = 3$ we do not consider a stash due to our experiments showing it having a much smaller impact as compared to $k = 2$. Additionally, we do not compute exact parameters for $k > 3$ due to the diminishing returns. In particular, $k = 4$ follows the same regime as $k = 3$ but only marginally improves the failure probability.

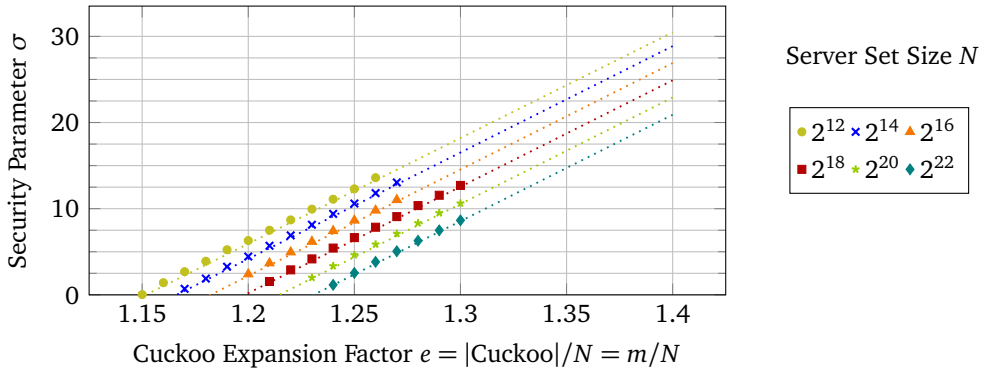


Figure 9.3: Empirical (marks) and interpolated (dashed/dotted lines) Cuckoo success probability for $k = 3$ hash functions and different set sizes N.

9.5.4 Parameter Selection for Cuckoo Hashing and Binning

Traditional use of Cuckoo hashing instructs the parties to sample new hash functions for each protocol invocation. In our setting however it can make sense to instruct the servers, which hold a somewhat static dataset, to perform Cuckoo hashing once and for all. Updates to the dataset can then be handled by periodically rebuilding the Cuckoo table once it has reached capacity. This leaves the question of what the Cuckoo hashing success probability should be. It is standard practice to set statistical failure events like this to happen with probability 2^{-40}. However, since the servers perform Cuckoo hashing only occasionally (and since hashing failure applies only to initialization, not future queries), we choose to use more efficient parameters with a security level of $\sigma = 20$ bits, i.e., $\Pr[\text{Cuckoo failure}] = 2^{-20}$. We

emphasize that once the items are successfully placed into the hash table, all future *lookups* (e.g., contact discovery instances) are error-free, unlike, say, in a Bloom filter.

We also must choose the number of hash functions to use.[11] Through experimental testing we find that overall the protocol performs best with $k = 3$ hash functions. The parameters used can be computed by solving for $e \approx 1.4$ in Eq. (9.2) given that $\sigma = 20$.

To see why this configuration was chosen, we must also consider another important parameter: the number of bins β. Due to binning being performed for each protocol invocation by the client, we must ensure that it succeeds with very high probability and thus we use $\sigma = 40$ to choose binning parameters. An asymptotic analysis shows that the best configuration is to use $\beta = O(n/\log n)$ bins, each of size $\mu = O(\log n)$. However, this hides the exact constants which give optimal performance. Upon further investigation we find that the choice of these parameters result in a communication/computation trade-off.

For the free variable c, we set the number of bins to $\beta = cn/\log_2 n$ and solve for the required bin size μ. As depicted in Fig. 9.4, the use of $k = 3$ and scaling factor $c = 4$ result in the best running time at the expense of a relatively high communication of 11 MiB. However, at the other end of the spectrum $k = 2$ and $c = 1/16$ results in the smallest communication of 2.4 MiB. The reason $k = 2$ achieves smaller communication for any fixed c is that the client sends $k = 2$ PIR queries per item instead of three. However, $k = 2$ requires that the Cuckoo table is three times larger than for $k = 3$ and therefore the computation is slower.

Varying c affects the number of bins β. Having fewer bins reduces the communication due to the bins being larger and thereby having a better real/dummy query ratio. However, larger bins also increases the overall work, since the work is proportional to the bin size μ times N. We aim to minimize both the communication and running time. We therefore decided on choosing $k = 3$ and $c = 1/4$ as our default configuration, the circled data-point in Fig. 9.4. However, we note that in specific settings it may be desirable to adjust c further.

The PIR block size b also results in a computation/communication trade-off. Having a large block size gives shorter PIR keys and therefore less work to expand the DPF. However, this also results the server having a larger input set to the subsequent PSI which makes that phase require more communication and computation. Due to the complicated nature of how all these parameters interact with each other, we empirically optimized the parameters to find that a PIR block size between 1 and 32 gives the best trade-off.

9.6 Performance

In this section we analyze the performance of PIR-PSI. We ran all experiments on a single benchmark machine which has two 18-core Intel Xeon E5-2699 2.30 GHz CPUs and 256 GB RAM. Specifically, we ran all parties on the same machine, communicating via localhost

[11] Although the oblivious shuffling procedure of Sect. 9.3.3 can be extended in a natural way to include a stash, we use a stash-free variant of Cuckoo hashing.

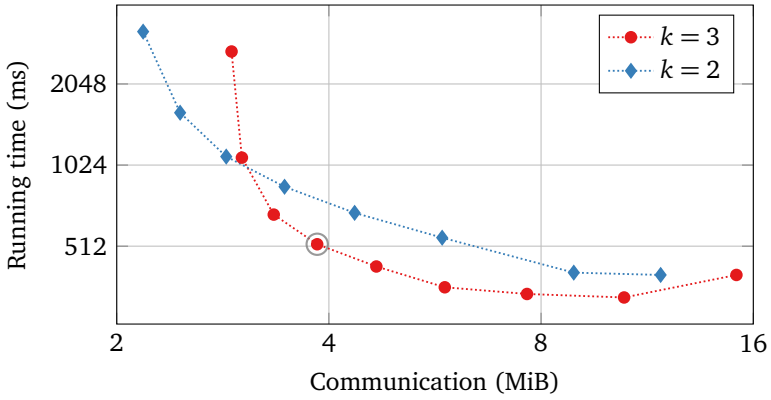


Figure 9.4: Communication and computation trade-off for $n = 2^{10}, N = 2^{24}, T = 16$ threads, k Cuckoo hash function, no stash, with the use of $\beta = cn/\log_2 n$ bins where $c \in \{2^{-5}, 2^{-4}, \ldots, 2^3\}$ and are listed left to right as seen above. The configuration $(k = 2, c = 2^{-5})$ did not fit on the plot. The highlighted point $(k = 3, c = 1/4)$ is the default parameter choice that is used.

network, and simulated a network using the Linux `tc` command: a LAN setting with 0.02 ms round-trip latency, 10 Gbps network bandwidth, and a WAN setting with a 80 ms round-trip latency, 100 Mbps network bandwidth. We process elements of size $\ell = 128$ bits.

Our protocol can be separated into two phases: the *server's init* phase when database is stored in the Cuckoo table, and the *contact discovery* phase where client and server perform the private intersection.

9.6.1 PIR-PSI Performance Results

In our *contact discovery* phase, the client and server first perform the pre-processing of PSI between n and $3n$ items which is independent of the parties' inputs. We refer to this step as pre-processing phase which specifically includes base OTs, and $O(n)$ PRFs. The online phase consists of protocol steps that depend on the parties' inputs. To understand the scalability of our protocol, we evaluate it on the range of server set size $N \in \{2^{20}, 2^{24}, 2^{26}, 2^{28}\}$ and client set size $n \in \{1, 4, 2^8, 2^{10}\}$. The small values of $n \in \{1, 4\}$ simulate the performance of incremental updates to a client's set. Tab. 9.2 presents the communication cost and total *contact discovery* time with online time for both single- and multi-threaded execution with $T \in \{1, 4, 16\}$ threads.

As discussed in Section 9.5.4, there is a communication and computation trade-off on choosing the different value c and b which effects the number of bins and how many items are selected per PIR query. The interplay between these two variable is somewhat complex and offers a variety of communication computation trade-offs. For smaller $n \in \{1, 4\}$, we set $b = 32$ to drastically reduce the cost of the PIR computation at the cost of larger PSI. For larger n, we

Table 9.2: Our protocol's total contact discovery communication cost and running time using T threads, and $\beta = cn/\log_2 n$ bins and PIR block size of b. LAN: 10 Gbps, 0.02 ms latency.

Param.				Comm.	Running time [seconds]		
N	n	c	b	[MiB]	$T=1$	$T=4$	$T=16$
2^{28}	2^{10}	4	16	28.3	4.07	1.60	0.81
		0.25	1	4.93	33.02	13.22	5.54
	2^8	3	16	7.10	3.61	1.30	0.65
		1	1	2.20	14.81	6.92	3.40
	4	1	32	0.06	1.93	–	–
	1	1	32	0.03	1.21	–	–
2^{26}	2^{10}	2	8	12.7	1.61	0.72	0.41
		0.25	1	4.28	7.22	3.65	1.36
	2^8	6	16	10.3	0.98	0.51	0.26
		0.25	4	1.36	4.36	1.90	0.97
	4	1	32	0.06	0.56	–	–
	1	1	32	0.03	0.48	–	–
2^{24}	2^{10}	1	8	8.61	0.67	0.36	0.22
		0.25	1	3.85	2.28	0.94	0.50
	2^8	4	8	4.81	0.49	0.22	0.18
		1	1	1.68	1.26	0.57	0.36
	4	1	32	0.05	0.19	–	–
	1	1	32	0.03	0.16	–	–
2^{20}	2^{10}	0.5	4	2.10	0.22	0.10	0.06
		0.25	1	2.98	0.32	0.21	0.16
	2^8	2	4	1.95	0.20	0.09	0.06
		0.25	4	1.13	0.24	0.18	0.15
	4	1	32	0.05	0.14	–	–
	1	1	32	0.03	0.13	–	–

consider parameters which optimize running time and communication separately, and show both in Tab. 9.2.

Our experiments show that our PIR-PSI is highly scalable. For a database of $N = 2^{28}$ elements and client set of $n = 2^{10}$ elements, we obtain a running time of 33.02 s and only 4.93 MiB bits of communications for the contact discovery using a single thread. Alternatively, running time can be reduced to just 4 s for the cost of 28 MiB communication. Increasing the number of threads from 1 to 16, our protocol shows a factor of 5× improvement, due to the fact that it parallelizes well. When considering the smallest server set of $N = 2^{20}$ with 16 threads, our protocol requires only 1.1 MiB of communication and 0.24 s of contact discovery time.

We point out that the computational workload for the client is small and consists only of DPF key generation, sampling random values and the classical PSI protocol with complexity linear in the size of the client set. This corresponds to 10% of the overall running time (e.g., 0.3 s of the total 3.6 s for $N = 2^{28}$ and $n = 2^8$). Despite our experiments being run on a somewhat

powerful server, the overwhelming bottleneck for performance is the computational cost for the server. Furthermore, after parameter agreement, the PIR step adds just a single round of communication between the client device and the servers before the PSI starts. Hence, our reported performance is also representative of a scenario in which the client is a computationally weaker device connected via a mobile network. Detailed numbers on the performance impact of the optimizations from Sect. 9.5.1 are provided in Sect. 9.6.3.

9.6.2 Updating the Client and Server Sets

In addition to performing PIR-PSI on sets of size n and N, the contact discovery application requires periodically adding items to these sets. In case the client adds a single item x to their set X, only the new item x needs to be compared with the servers' set. Our protocol can naturally handle this case by simply having the client use $X' = \{x\}$ as their input to the protocol. However, a shortcoming of this approach is that we cannot use binning and the PIR query spans the whole Cuckoo table.

For a database of size $N = 2^{24}$, our protocol requires only 0.16 s and 0.19 s to update 1 and 4 items, respectively. When increasing the size to $N = 2^{28}$, we need 1.9 s to update one item. Our update queries are cheap in terms of communication, roughly 30–50 kiB, and can be parallelized well since DPF.Gen and DPF.Expand can each be processed in a divide and conquer manner. Also, several update queries from different users can be batched together to offer very high throughput. However, our current implementation only supports parallelization at the level of bins/regions, and not for a single DPF query.

The case when a new item is added to the servers' set can easily be handled by performing a traditional PSI between one of the servers and the client, where the server only inputs the new item. One could also consider batching several server updates together, and then performing a larger PSI or applying our protocol to the batched server set.

9.6.3 Effect of the Optimizations

In this section, we discuss the effect of our optimizations on the performance. By far the most important optimization employed is the use of *binning*. Observe in Tab. 9.3 that the running time with all optimizations enabled is 1.0 s while the removal of binning results in a running time of 1 906 s. This can be explained by the overall reduction of asymptotic complexity to $O(N \log n)$ with binning as opposed to $O(Nn)$ without binning.

Another important optimization is the use of PIR blocks which consist of more than one Cuckoo table item. This *blocking* technique allows for a better balance between the cost of the PIR compared to the cost of the subsequent PSI. Increasing the block size logarithmically decreases the cost of the PIR while linearly increasing the cost of the PSI. Since the PIR computation is so much larger than the PSI (assuming $n \ll N$) setting the block size to be greater than 1 gave significant performance improvements. In practice we found that

setting b to be within 1 and 32 gave the best results. Tab. 9.3 shows that setting b to optimize running time gives a 3.7× improvement.

Table 9.3: Online running time in seconds of the protocol with all optimizations enabled compared with the various optimizations of Section 9.5.1 individually disabled.

N	n	All Opt. Enabled	No Batching	No Blocking	No Vectorization	No Binning
2^{24}	2^{12}	1.0	2.1	3.7	40.1	1 906

We also consider the effect that our highly optimized DPF implementation has on the overall running time. *Vectorization* refers to an implementation of the DPF with the full-domain optimization implemented similar as described by [BGI16, Figure 4]. We then improve on their basic construction to take full advantage of CPU vectorization and fixed-key AES. The result is a 40× difference in overall running-time.

The final optimization is to improve memory locality of our implementation by carefully accessing the Cuckoo table. Instead of computing each PIR query individually, which would require loading the large Cuckoo table from memory many times, our *batching* optimization runs all DPF evaluations for a given database location at the same time. This significantly reduces the amount of data that has to be fetched from main memory. For a dataset of size $N = 2^{24}$ we observe that this optimization yields 2.1× improvement, and an even bigger 5× improvement when applied to a larger dataset of $N = 2^{28}$ along with using $T = 16$ threads.

9.7 Comparison with Prior Work

In this section we give a thorough qualitative and quantitative comparison between our protocol and those of CLR [CLR17], KLSAP [KLS+17], and RA [RA18]. We obtained the implementations of CLR and KLSAP from the respective authors, but the implementation of RA is not publicly available. Because of that, we performed a comparison on inputs of size $N \in \{2^{16}, 2^{20}, 2^{24}\}$ and $n \in \{5\,535, 11\,041\}$ to match the parameters used in [RA18, Tables 1 and 2]. While the experiments of RA were performed on an Intel Haswell i7-4770K quadcore CPU with 3.4 GHz and 16 GB RAM, we ran the KLSAP and CLR protocols on our own hardware, described in Sect. 9.6. We remark that RA's benchmark machine has 3.4 GHz, which is 1.48× *faster than our machine*. The number of cores and RAM available on our hardware does not influence the results of the single-threaded benchmarks ($T = 1$). Results of the comparison are summarized in Tab. 9.4.

Table 9.4: Comparison of PIR-PSI to CLR, KLSAP, and RA with $T \in \{1, 4\}$ threads. LAN: 10 Gbps, 0.02 ms latency. WAN: 100 Mbps, 80 ms latency. Best results marked in bold. Online communication in parentheses. Cells with "–" denote that the setting is not supported, due to limitations in the respective implementation. Cells with "*" indicate that the numbers are scaled for a fair comparison of error probability. CLR and RA use $\ell = 32$ bit items, while PIR-PSI and KLSAP process 128 bit items.

Protocol	Parameters N	n	Communication Size [MiB]	Running time [seconds] LAN (10 Gbps) $T=1$	LAN (10 Gbps) $T=4$	WAN (100 Mbps) $T=1$	WAN (100 Mbps) $T=4$	Client Storage [MiB]	Server Init. [seconds] $T=1$	Server Init. [seconds] $T=4$
CLR [CLR17]	2^{24}	11 041	**21.1**	38.6	19.7	41.0	22.1	**0**	76.8	20.6
		5 535	**12.5**	34.0	16.3	36.0	18.2		71.2	18.5
	2^{20}	11 041	11.5	3.7	3.2	4.9	4.4		9.1	2.5
		5 535	5.6	3.5	1.9	4.1	2.5		5.1	1.4
	2^{16}	11 041	4.1/4.4	1.8	1.4	2.2	1.8		1.2	0.3
		5 535	2.6	0.9	0.6	1.1	0.9		0.9	0.3
KLSAP [KLS+17]	2^{24}	11 041	2 049 (43.3)	90.4	–	265.1	–	1 941	8.32	–
		5 535	1 070 (21.7)	52.3	–	128.3	–	1 016		
	2^{20}	11 041	1 968 (43.3)	82.1	–	259.9	–	1 860	0.58	–
		5 535	989 (21.7)	44.8	–	124.7	–	935		
	2^{16}	11 041	1 963 (43.3)	81.8	–	259.6	–	1 855	0.04	–
		5 535	984 (21.7)	44.0	–	121.4	–	930		
RA [RA18]	2^{24}	11 041	171.67 (0.67)*	**1.08***	–	18.39*	–	171.00*	333.62	–
		5 535	168.34 (0.34)*	**0.75***	–	17.61*	–	168.00*		
	2^{20}	11 041	**11.36 (0.67)***	0.67*	–	**3.41***	–	10.69*	20.78	–
		5 535	**10.84 (0.34)***	0.34*	–	2.89*	–	10.50*		
	2^{16}	11 041	**1.34 (0.67)***	0.66*	–	**1.33***	–	0.67*	1.30	–
		5 535	**1.00 (0.34)***	0.33*	–	**0.85***	–	0.66*		
Ours	2^{24}	11 041	32.46	2.18	**1.65**	**5.63**	**5.13**	**0**	**2.690**	–
		5 535	21.45	1.34	**1.11**	**3.72**	**2.77**			
	2^{20}	11 041	22.86	**0.37**	**0.31**	3.70	**3.59**		**0.089**	–
		5 535	11.67	**0.29**	**0.24**	**2.50**	**2.29**			
	2^{16}	11 041	12.83	**0.28**	**0.29**	2.55	**2.55**		**0.004**	–
		5 535	7.66	**0.21**	**0.20**	1.85	**1.85**			

9.7.1 The CLR protocol

The high level idea of the protocol of Chen et al. [CLR17] is to have the client encrypt each element in their dataset under a homomorphic encryption scheme, and send the result to the server. The server evaluates the intersection circuit on encrypted data, and sends back the result for the receiver to decrypt.

The CLR protocol has communication complexity $O(\ell n \log(N))$, where the items are ℓ bits long. Ours has communication complexity $O(\kappa n \log(N/(\kappa n \log n)))$, with no dependence on ℓ since the underlying PSI protocol [KKRT16] has no such dependence. For small items (e.g., $\ell = 32$ as reflected in Tab. 9.4), CLR uses less communication than our protocol, e.g., 20 MiB as opposed to 37 MiB. However, their protocol scales very poorly for string length of 128 bits as it would require significantly less efficient FHE parameters. Furthermore, CLR

can not take advantage of the fact that most contact lists have significantly fewer than 5 535 entries. That is, the cost for $n = 1$ and $n = 5\,535$ is roughly equivalent, because of the way FHE optimizations like batching are used. The main computational bottleneck in CLR is the server performing $O(n)$ homomorphic evaluations on large circuits of size $O(N/n)$. The comparable bottleneck in our protocol is performing DPF.Expand and computing the large inner products. Since these operations take advantage of hardware-accelerated AES, PIR-PSI is significantly faster than CLR, e.g., 20× for $N = 2^{24}$.

The server's initialization in CLR involves hashing the N items into an appropriate data structure (as in PIR-PSI), but also involves precomputing the many coefficients of polynomials. Hence our initialization phase is much faster than CLR, e.g., 40× for $N = 2^{24}$. The CLR protocol does not provide a full analysis of security against malicious clients. Like our protocol, the leakage allowed with a malicious client is likely to be minimal.

9.7.2 The KLSAP protocol

In the KLSAP protocol [KLS+17], the server sends a Bloom filter of size $O(\sigma N)$ to the client in an offline phase, for statistical security parameter σ and server set size N. During later contact discovery phases, the client refers to this Bloom filter, whose size is significant: nearly 2 GiB for $N = 2^{24}$ server items. This data, which must be stored by the client, may be prohibitively large for mobile client applications. By contrast, our protocol (and CLR) requires no long-term storage by the client.

While there are also other implementations in [KLS+17], we focused on the Yao-variant, as it offered the best performance. In the contact discovery phase, KLSAP runs Yao's protocol to obliviously evaluate an AES circuit for each of the client's items. Not even counting the bloom filter, this requires slightly more communication than our approach (1.5×). Additionally, it requires more computation by the (weak) client: evaluating many AES garbled circuits (thousands of AES calls per item) vs. running many instances of DPF.Gen ($\log N$ AES calls per item) followed by a specialized PSI protocol (constant number of hash/AES calls per item). Even though the server in our protocol must perform $O(N)$ computation during contact discovery, our optimizations result in a much faster discovery phase (40× for $N = 2^{24}$).

When the server makes changes to its set in KLSAP, it must either re-key its AES function (which results in re-sending the huge Bloom filter), or send incremental updates to the Bloom filter (which breaks forward secrecy, as a client can query its items in both the old and new versions of the Bloom filter).

KLSAP is easily adapted to secure against a malicious client. This stems from the fact that the contact discovery phase uses Yao's protocol with the client as evaluator. Hence it is naturally secure against a malicious client (provided malicious-secure OTs are used).

Subtleties about Hashing Errors The way that KLSAP uses a Bloom filter also leads to qualitative differences in the error probabilities compared to PIR-PSI. In KLSAP the server publishes a Bloom filter for all clients, who later query it for membership. The false-positive rate (FPR) of the Bloom filter is the probability that a single item not in the server's set is mistakenly classified as being in the intersection. Importantly, the FPR for this global Bloom filter is *per client item*. In KLSAP this FPR is set to 2^{-30}, which means after processing a combined 1 million client items the probability of some client receiving a false positive may be as high as 2^{-10}!

By contrast, the PIR-PSI server places its items in a Cuckoo table once-and-for all (with hashing error probability 2^{-20}). As long as this *one-time event* is successful, all subsequent probes to this data structure are error-free (we store the entire $\ell = 128$ bit item in the Cuckoo table, not just a short fingerprint as in [RA18]). If the hashing is unsuccessful, the server simply tries again with different hash functions. As this process happens offline, before any client queries are answered, no information about the server set is leaked. All of our other failure events (e.g., probability of a bad event within our 2-party PSI protocol) are calibrated for error probability 2^{-40} *per contact discovery instance*, not per item! To have a comparable guarantee, the Bloom filter FPR of KLSAP would have to be scaled by a factor of $\log_2(n)$.

9.7.3 The RA Protocol

The RA [RA18] protocol uses a similar approach to KLSAP, in that it uses a relatively large representation of the server's set, which is sent in an offline phase and stored by the client. The downsides of this architecture discussed above for KLSAP also apply to RA (client storage, more client computation, false-positive rate issues, forward secrecy).

RA's implementation uses a Cuckoo filter that stores for each item a 16-bit fingerprint. This choice leads to a relatively high false-positive rate of $2^{-13.4}$. To achieve the failure events with error probability 2^{-40} per contact discovery instance (in line with our protocol), the Cuckoo filter FPR of RA would be $2^{-(40+\log_2(n))}$. Therefore, their protocol would have to be modified to use 56-bit and 57-bit fingerprints for $n = 5\,535$ and $n = 11\,041$, respectively. This change increases the communication cost, transmission time, and offline storage requirements $3.44-3.5\times$, relative to the numbers reported in [RA18, Table 1]. In Tab. 9.4 we report the *scaled* communication costs, the *scaled* online running time, and the *scaled* client's storage, but refrain from trying to scale the server's initialization times. As can be seen our protocol running time is $1.2-3.2\times$ faster than RA for sufficiently large N. We also have a $100\times$ more efficient server initialization phase and achieve communication complexity of $O(n \log N)$ as compared to $O(N)$ of RA. This difference can easily be seen by how the communication of RA significantly increases for larger N.

In RA, the persistent client storage is not a Bloom filter but a more compact Cuckoo filter. This reduces the client storage, but it still remains linear in N. For $N = 2^{28}$ the storage requirement is 2.57 GiB to achieve an error probability of 2^{-40} per contact discovery instance.

The RA protocol does not provide any analysis of security against malicious clients.

9.8 Extensions and Deployment

Although this work mainly focuses on the setting of pure contact discovery with two servers, our protocol can be modified for other settings, which we discuss in this section. We also briefly go into the possibility of deploying PIR-PSI in practice.

9.8.1 PSI with Associated Data (PSI+AD)

PSI with Associated Data (PSI+AD) refers to a scenario where the client has a set A of keys and the server has a set B of key-value pairs, and the client wishes to learn $\{(k,v) \mid (k,v) \in B \text{ and } k \in A\}$. In the context of an encrypted messaging service, the keys may be phone numbers or email addresses, and the values may be the user's public key within the service.

PIR-PSI can be modified to support associated data in a natural way. The server's Cuckoo hash table simply holds key-value pairs, and the 2-party PSI protocol is replaced by a 2-party PSI+AD protocol. The client will then learn *masked* values for each item in the intersection, which it can unmask. The PSI protocol of [KKRT16] that we use is easily modified to allow associated data.

9.8.2 3- and 4-Server Variant

We described PIR-PSI in the context of two non-colluding servers, who store identical copies of the service provider's user database. Since both servers hold copies of this sensitive database, they are presumably both operated by the service provider, so the promise of non-collusion may be questionable. Using a folklore observation from the PIR literature, we can allow servers to hold only *secret shares* of the user database, at the cost of adding more servers.

Consider the case of 3 servers. The service provider can recruit two independent entities to assist with private contact discovery, without entrusting them with the sensitive user database. The main idea is to let servers #2 and #3 hold additive secret shares of the database and jointly simulate the effect of a single server that holds the database in the clear.

Recall the 2-party DPF-PIR scheme of [BGI16], that we use. The client sends DPF shares k_1, k_2 to the servers, who expand the keys to K_1, K_2 and performs an inner product with the database. The client XORs the two responses to obtain result $(K_1 \cdot \mathsf{DB}) \oplus (K_2 \cdot \mathsf{DB}) = (K_1 \oplus K_2) \cdot \mathsf{DB} = \mathsf{DB}[i]$.

In our 3-server case, we have server #1 holding DB, and servers #2 and #3 holding $\mathsf{DB}_2, \mathsf{DB}_3$ respectively, where $\mathsf{DB} = \mathsf{DB}_2 \oplus \mathsf{DB}_3$. We simply let the client send DPF share k_1 to server #1, and send k_2 to *both* of the other servers. All servers expand their DPF share and perform

an inner product with their database/share. The client will receive $K_1 \cdot \mathsf{DB}$ from server #1, $K_2 \cdot \mathsf{DB}_2$ from server #2, and $K_2 \cdot \mathsf{DB}_3$ from server #3. The XOR of all responses is indeed

$$\begin{aligned}&(K_1 \cdot \mathsf{DB}) \oplus (K_2 \cdot \mathsf{DB}_2) \oplus (K_2 \cdot \mathsf{DB}_3)\\ =\ &(K_1 \cdot \mathsf{DB}) \oplus K_2 \cdot (\mathsf{DB}_2 \oplus \mathsf{DB}_3)\\ =\ &K_1 \cdot \mathsf{DB} \oplus K_2 \cdot \mathsf{DB} = (K_1 \oplus K_2) \cdot \mathsf{DB} = \mathsf{DB}[i].\end{aligned}$$

Now the entire PIR-PSI protocol can be implemented with this 3-server PIR protocol as its basis. The computational cost of each server is identical to the 2-server PIR-PSI, and is performed in parallel by the independent servers. Hence, the total time is minimally affected. The client's total communication is unaffected since server #2 can forward k_2 to server #3. The protocol security is the same, except that the non-collusion properties hold now only if server #1 doesn't collude with any of the other servers. If servers #2 and #3 collude, then they clearly learn DB, but as far as the client's privacy is concerned, the situation simply collapses to 2-server PIR-PSI.

Similarly, server #1 can also be replaced by a pair of servers, each with secret shares (and this sharing of DB can be independent of the other sharing of DB). This results in a 4-server architecture with security for the client as long as neither of servers #1 and #2 collude with one of the servers #3 and #4, and where no single server holds DB in the clear.

9.8.3 2-Server with OPRF Variant

An alternative to the 3-server variant above is to leverage a pre-processing phase. Similar to [RA18], the idea is to have server #1 apply an oblivious PRF to their items instead of a hash function, which will ensure that the database is pseudorandom in the view of server #2, who does not know the PRF key. In particular, let server #1 sample a key k for the oblivious PRF F used in [RA18] and update the database as $\mathsf{DB}'_i := F_k(\mathsf{DB}_i)$, which is then sent to server #2. When a client wishes to compute the intersection of its set X with DB, they first perform an oblivious PRF protocol with server #1 to learn $X' = \{F_k(x) \mid x \in X\}$. Note that this protocol ensures that the client does not learn k. The client can now engage in our standard two-server PIR-PSI protocol to compute $Z' = X' \cap \mathsf{DB}'$ and thereby infer $Z = X \cap \mathsf{DB}$.

The advantage of this approach is that server #2 does not learn any information about the plaintext database DB since the PRF was applied to each record. Moreover, this holds even if server #2 colludes with one of the clients. The added performance cost of this variant has two components. First, server #1 must update its database by applying the PRF to it. As shown by [RA18], a single CPU core can process roughly 50 000 records per second, which is sufficiently fast given that this is a one-time cost. The second overhead is performing the oblivious PRF protocol with the clients. This requires three exponentiations per item in X, which represents an acceptable overhead given that $|X|$ is small.

9.8.4 Single-Server Variant

We also note that PIR-PSI could potentially be extended to the single-server setting. Several single-server PIR protocols [AS16; ACLS18; ABFK16], based on homomorphic encryption have shown to offer good performance, while at the same time removing the two-server requirement. With some modifications to our architecture, we observe that such PIR protocols can be used. The main challenge to overcome is how to secret share and shuffle the PIR results before being forwarded to the PSI protocol. First, a PIR protocol which allows the result to be secret shared is required. We observer that typical PIR protocols can support such a functionality by adding a random share to the result ciphertext. Given this, a two party variant of step 3 of Prot. 9.1 can be implemented using standard two-party shuffling protocols. We leave the optimization and exact specification of such a single-server PIR-PSI protocol to future work, but note its feasibility.

9.8.5 Practical Deployment

We now turn our attention to practical questions surrounding the real-world deployment of our multi-server PIR-PSI protocol. As briefly discussed in the previous section, the requirement that a single organization has two non-colluding servers may be hard to realize. However, we argue that the 3-server or 2-server with an OPRF variants make deployment significantly simpler. Effectively, these variants reduce the problem to finding one or two external semi-honest parties that will not collude with the service provider (server #1). A natural solution to this problem is to leverage existing cloud providers such as Microsoft Azure or Amazon EC2. Given that these companies have a significant interest to maintain their reputation, they would have a large incentive to not collude. Indeed, Microsoft has informally proposed such a setting [GLL+16] where secure computation services are provided under a non-collusion assumption. Alternatively, privacy-conscious organization such as the Electronic Frontier Foundation (EFF) could serve as the second server.

10 Conclusion

In this chapter we conclude the thesis with a summary of the included topics and provide an outlook into possible future directions of research.

It is often assumed that security and privacy are fundamentally different goals from performance and usability. This binary distinction of these goals is not only false, but also dangerous, as it puts private data at risk by suggesting that practical privacy simply cannot be realized. Furthermore, it diminishes the immense amount of work and the outstanding results that the security and privacy research community has achieved. At the same time, suggesting that users have to decide between either privacy or utility completely ignores the tremendous technical progress that has been made, which is now a key factor in enabling practical privacy solutions. This dissertation searches to find a practical combination of all goals and aims to show measures that result in efficient and privacy-preserving digital systems.

10.1 Summary

The research question that was asked in this thesis was:

> Can privacy-preserving techniques like MPC and PIR be applied to real-world applications and use-cases in order to protect the privacy of the data they process, while at the same time achieving efficiency that makes them usable in practice?

To answer this question we described the MPC tools that we have developed in Part I. With these tools we intend to provide a usable and efficient base for developers and researchers that want to implement and evaluate MPC protocols and applications. In Chapt. 3, we presented the ABY framework that serves as foundation of many of our projects. ABY has proven to be a solid foundation for many projects and is actively used by many developers and researchers worldwide. In Chapt. 4 we showed results that include optimized building blocks generated from modified hardware synthesis tools and the ability to directly compile program descriptions in a hardware definition language into MPC protocols in ABY. With the help of these features we increased the applicability of MPC in practice and made first steps towards compilation of MPC protocols from a high-level language. The HyCC compiler, presented in Chapt. 5 builds on top of ABY and CBMC-GC [HFKV12] to enable fully automated compilation of ANSI C code into hybrid MPC protocols. Especially HyCC is a crucial and important step that will help to disseminate the use of MPC in practice, as it overcomes

the requirement of having fundamental background knowledge in the complex and fast-changing research field of MPC and eliminates the burden of learning a new domain specific language.

On the foundation of these tools, we built MPC applications, which we presented in Part II. These applications are relevant real-world use cases, where it was initially unclear if the performance of the underlying MPC implementation would suffice to be practical. Chapt. 6 contains two results that show that BGP-based route computation on the Internet, and route dispatch at IXPs is possible in a privacy-preserving way, when built upon MPC. With privacy-preserving Internet routing we protect sensitive business information that influence routing decisions while at the same time allowing improved performance and greater usability compared to the status quo. In Chapt. 7, we presented a solution for private queries to outsourced, federated genome databases, which is a very crucial topic, as genomic information is highly sensitive and unchangeably connected to every individual.

Finally, in Part III, we presented improvements to existing PIR protocols and applications thereof. Chapt. 8 summarizes optimizations and the generalization of the multi-server PIR scheme of Chor et al. and an anonymous messenger as application thereof. Our results move PIR closer to practicality and might be of independent interest for similar schemes. With OnionPIR as anonymous messenger we target the protection of metadata, which is an important privacy feature. PIR-PSI, an approach for scalable private contact discovery, was presented in Chapt. 9. Contact discovery is a crucial component of modern messengers and runs frequently on every user's system, potentially exposing their social graph. Our result is also generally applicable as asymmetric PSI, where the two sets are of vastly different sizes.

Overall, from our results presented here, as well as from the large body of work in the field, one can see that MPC protocols and techniques like PIR are steadily improving and moving closer to practicality, and are thus key ingredients to practical privacy protection. Therefore, this thesis could answer the initial research question positively and contributed a small share to the important task of protecting users' privacy in a modern digital world. In the final section, I'd like to give an outlook on possible future works towards answering further aspects of the initial research question.

10.2 Future Work

This thesis has presented several techniques and systems, but there is always plenty of room for improvement and opportunities for further research. In this section we provide points that might be interesting to look at in future work, and ideas that could extend and improve the existing results. While some of these points are more important and fundamental than others, we believe that all of them will contribute to the distribution and relevance of MPC in practice, and thus benefit the protection of sensitive data.

10.2.1 From Passive to Active Security

Most of the results that we presented in this thesis are secure in the presence of semi-honest adversaries (cf. Sect. 2.2). This is an important first step to showcase practicality of our solutions and provides sufficient security in some scenarios, where the involved actors are assumed to be somewhat trustworthy, but for many real-world systems stronger adversary models must be considered. There are existing solutions that could extend our results. For the GMW protocol, there are approaches like TinyOT [NNOB12; NST17; HOSS18] that guarantee security against active adversaries, and implementations that show their feasibility [MOR16]. Arithmetic circuits can be evaluated with the SPDZ protocol with security against malicious adversaries [DPSZ12; DKL+13; KOS16]. Both approaches use information-theoretic MACs to achieve malicious security and work in the precomputation model. For Yao's garbled circuits there are two techniques that achieve malicious security. When using *dual execution* [MF06; RR16], the work is doubled and every party plays both the roles of the garbler and evaluator. In cut-and-choose [LP07; Lin13], the garbler creates multiple versions of the same circuit and the evaluator selects a subset that are opened and verified, in order to gain trust that no manipulation has happened. This technique has also been improved with several optimizations like batching [LR14; LR15] in mind. Many of these issues regarding active security, including conversion between the protocols have been approached in [MR18] for the 3-party setting.

Attestation is an important area of research that goes in the same direction. It is especially important in outsourcing scenarios, in order to ensure that computational parties actually run the code that they are supposed to. For this, hardware-based solutions can be used and efficient instantiations for an MPC use case should be found.

10.2.2 From Two-Party to Multi-Party

The use cases presented in this theses focus on the two-party case where the computation happens directly between two involved parties. This can be utilized in an outsourcing scenario, where computation on sensitive data is outsourced to two parties that must not collude to guarantee privacy. In practice there might be scenarios where more than 2 parties want to directly run a MPC protocol among themselves. Similarly, for outsourcing scenarios, there might be cases where non-collusion of only 2 parties might be too weak and more parties are desirable. To fulfill these demands, the existing solutions need to be improved and extended. While this extension can happen naturally for the GMW protocol, it is more complex to realize a version with more than 2 parties for Yao's garbled circuits. Nonetheless, there are results in that direction [MR18] that focus on a 3-party setting and we expect more to follow.

10.2.3 Implementations

It is crucial that protocols are available as implementations. While prototypical code shows the general feasibility of a concept, it often lacks the quality required for actual user data. Writing production-ready code demands careful software engineering and testing, which is often not

possible to the fullest extent in research. For the practical acceptance of privacy-preserving techniques, it would be desirable if code was more robust, documented, and tested.

We are steadily working on extending and improving our code base and especially fundamental tools like the ABY framework (cf. Chapt. 3) are under constant development. This includes incorporation of the most recent optimizations and the inclusion of interfaces to other implementations. For the future we are planning to extend the exchange formats that allow import and export of circuits to and from ABY. We believe that this in an important contribution towards exchange of ideas and the connection of diverse implementations.

Another work that goes in this direction is the MATRIX framework [BHKL18], that aims to unify performance evaluation of several heterogeneous MPC implementations, to enable fair comparison. We are currently integrating our ABY implementation into MATRIX.

Bibliography

[ABB+17] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, V. Pereira. **"A Fast and Verified Software Stack for Secure Function Evaluation"**. In: *CCS'17*. ACM, 2017, pp. 1989–2006.

[ABFK16] C. Aguilar-Melchor, J. Barrier, L. Fousse, M.-O. Killijian. **"XPIR: Private Information Retrieval for Everyone"**. In: *Privacy Enhancing Technologies Symposium (PETS'16)* 2016.2 (2016), pp. 155–174.

[ABH10] M. Albrecht, G. Bard, W. Hart. **"Algorithm 898: Efficient Multiplication of Dense Matrices over GF(2)"**. In: *ACM Transactions on Mathematical Software* 37.1 (2010), 9:1–9:14.

[ABL+04] M. J. Atallah, M. Bykova, J. Li, K. B. Frikken, M. Topkara. **"Private collaborative forecasting and benchmarking"**. In: *Workshop on Privacy in the Electronic Society (WPES'04)*. ACM, 2004, pp. 103–114.

[ABL+18] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, R. N. Wright. **"From Keys to Databases – Real-World Applications of Secure Multi-Party Computation"**. Cryptology ePrint Archive, Report 2018/450. https://ia.cr/2018/450. 2018.

[ABZS13] M. Aliasgari, M. Blanton, Y. Zhang, A. Steele. **"Secure Computation on Floating Point Numbers"**. In: *NDSS'13*. The Internet Society, 2013.

[ACF+12] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, W. Willinger. **"Anatomy of a Large European IXP"**. In: *SIGCOMM'12*. ACM, 2012.

[ACLS18] S. Angel, H. Chen, K. Laine, S. Setty. **"PIR with compressed queries and amortized query processing"**. In: *Symposium on Security and Privacy (S&P'18)*. IEEE Computer Society, 2018.

[ACM+13] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, M. V. Vyve. **"Securely Solving Simple Combinatorial Graph Problems"**. In: *Financial Cryptography and Data Security (FC'13)*. Vol. 7859. LNCS. Springer, 2013, pp. 239–257.

[ACN+17] G. Asharov, T.-H. H. Chan, K. Nayak, R. Pass, L. Ren, E. Shi. **"Oblivious Computation with Data Locality"**. Cryptology ePrint Archive, Report 2017/772. https://ia.cr/2017/772. 2017.

[ADKF70] V. Arlazarov, E. Dinic, M. Kronrod, I. Faradzev. **"On economical construction of the transitive closure of a directed graph"**. In: *USSR Academy of Sciences 134* (1970).

[ADS+17] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, M. Zohner. **"Privacy-Preserving Interdomain Routing at Internet Scale"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017). Full version: https://ia.cr/2017/393, pp. 143–163.

[AFL+16] T. Araki, J. Furukawa, Y. Lindell, A. Nof, K. Ohara. **"High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority"**. In: *CCS'16*. ACM, 2016, pp. 805–817.

[AHHK18] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, M.-O. Killijian. **"SRide: A Privacy-Preserving Ridesharing System"**. In: *WiSec'18*. ACM, 2018, pp. 40–50.

[AHLR18] G. Asharov, S. Halevi, Y. Lindell, T. Rabin. **"Privacy-Preserving Search of Similar Patients in Genomic Data"**. In: *Privacy Enhancing Technologies Symposium (PETS'18)* 2018.4 (2018), pp. 104–124.

[AHMA16] M. M. Al Aziz, M. Z. Hasan, N. Mohammed, D. Alhadidi. **"Secure and Efficient Multiparty Computation on Genomic Data"**. In: *20. International Database Engineering & Applications Symposium (IDEAS'16)*. ACM, 2016, pp. 278–283.

[AL07] Y. Aumann, Y. Lindell. **"Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries"**. In: *TCC'07*. Vol. 4392. LNCS. Springer, 2007, pp. 137–156.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, M. Zohner. **"More Efficient Oblivious Transfer and Extensions for Faster Secure Computation"**. In: *CCS'13*. Full version: https://ia.cr/2013/552. Code: https://encrypto.de/code/OTExtension. ACM, 2013, pp. 535–548.

[AMS12] **"Follow up: AMS-IX Route-Server Performance Test Euro-IX 20th"**. https://ripe64.ripe.net/presentations/49-Follow_Up_AMS-IX_route-server_test_Euro-IX_20th_RIPE64.pdf. 2012.

[AMS16] Amsterdam Internet Exchange. **"AMS-IX: Megaport and AMS-IX Partner to Provide Global SDN-Enabled Elastic Interconnection and Internet Exchange Service"**. https://ams-ix.net/newsitems/233. 2016.

[AO12] G. Asharov, C. Orlandi. **"Calling Out Cheaters: Covert Security with Public Verifiability"**. In: *ASIACRYPT'12*. Vol. 7658. LNCS. Springer, 2012, pp. 681–698.

[AS16] S. Angel, S. Setty. **"Unobservable Communication over Fully Untrusted Infrastructure"**. In: *Symposium on Operating Systems Design and Implementation, (OSDI'16)*. USENIX, 2016, pp. 551–569.

[BBD+11] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, G. Tsudik. **"Countering GATTACA: Efficient and Secure Testing of Fully-sequenced Human Genomes"**. In: *CCS'11*. ACM, 2011, pp. 691–702.

[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, M. Schwartzbach, T. Toft. **"Secure Multiparty Computation Goes Live"**. In: *FC'09*. Vol. 5628. LNCS. Springer, 2009, pp. 325–343.

[BCF+14] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, M. Zohner. **"GSHADE: Faster Privacy-Preserving Distance Computation and Biometric Identification"**. In: *2. ACM Workshop on Information Hiding and Multimedia Security (IH&MMSEC'14)*. Code: https://encrypto.de/code/GSHADE. ACM, 2014, pp. 187–198.

[BCKP01] O. Berthold, S. Clauss, S. Köpsell, A. Pfitzmann. **"Efficiency Improvements of the Private Message Service"**. In: *Information Hiding (IH'01)*. Vol. 2137. LNCS. Springer, 2001, pp. 112–125.

[BCP+17] D. BARRERA, L. CHUAT, A. PERRIG, R. M. REISCHUK, P. SZALACHOWSKI. **"The SCION Internet Architecture"**. In: *Communications of the ACM* 60.6 (2017), pp. 56–65.

[BDG15] N. BORISOV, G. DANEZIS, I. GOLDBERG. **"DP5: A Private Presence Service"**. In: *Privacy Enhancing Technologies Symposium (PETS'15)* 2015.2 (2015), pp. 4–24.

[BDK+18] 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.

[Bea91] D. BEAVER. **"Efficient Multiparty Protocols Using Circuit Randomization"**. In: *CRYPTO'91*. Vol. 576. LNCS. Springer, 1991, pp. 420–432.

[Bea95] D. BEAVER. **"Precomputing Oblivious Transfer"**. In: *CRYPTO'95*. Vol. 963. LNCS. Springer, 1995, pp. 97–109.

[Bea96] D. BEAVER. **"Correlated Pseudorandomness and the Complexity of Private Computations"**. In: *STOC'96*. ACM, 1996, pp. 479–488.

[Ber] BERKELEY LOGIC SYNTHESIS. **"ABC: a system for sequential synthesis and verification, release 70930"**. http://www.eecs.berkeley.edu/~alanmi/abc/.

[Ber09] D. J. BERNSTEIN. **"Cryptography in NaCl"**. Online: https://cr.yp.to/highspeed/naclcrypto-20090310.pdf. 2009.

[BFH+17] N. BÜSCHER, M. FRANZ, A. HOLZER, H. VEITH, S. KATZENBEISSER. **"On compiling Boolean circuits optimized for secure multi-party computation"**. In: *Formal Methods in System Design* 51.2 (2017), pp. 308–331.

[BFK+09] M. BARNI, P. FAILLA, V. KOLESNIKOV, R. LAZZERETTI, A.-R. SADEGHI, T. SCHNEIDER. **"Secure Evaluation of Private Linear Branching Programs with Medical Applications"**. In: *14. European Symposium on Research in Computer Security (ESORICS'09)*. Vol. 5789. LNCS. Full version: https://ia.cr/2009/195. Springer, 2009, pp. 424–439.

[BFL+11] M. BARNI, P. FAILLA, R. LAZZERETTI, A.-R. SADEGHI, T. SCHNEIDER. **"Privacy-Preserving ECG Classification With Branching Programs and Neural Networks"**. In: *IEEE TIFS* 6.2 (2011), pp. 452–468.

[BFMR10] K. R. B. BUTLER, T. R. FARLEY, P. MCDANIEL, J. REXFORD. **"A Survey of BGP Security Issues and Solutions"**. In: *Proceedings of the IEEE* 98.1 (2010), pp. 100–122.

[BG11] M. BLANTON, P. GASTI. **"Secure and Efficient Protocols for Iris and Fingerprint Identification"**. In: *ESORICS'11*. Vol. 6879. LNCS. Springer, 2011, pp. 190–209.

[BG12] S. BAYER, J. GROTH. **"Efficient zero-knowledge argument for correctness of a shuffle"**. In: *EUROCRYPT'12*. 2012, pp. 263–280.

[BGI15] E. BOYLE, N. GILBOA, Y. ISHAI. **"Function Secret Sharing"**. In: *EUROCRYPT'15*. Vol. 9057. LNCS. Springer, 2015, pp. 337–367.

[BGI16] E. BOYLE, N. GILBOA, Y. ISHAI. **"Function Secret Sharing: Improvements and Extensions"**. In: *CCS'16*. ACM, 2016, pp. 1292–1303.

[BGW88] M. BEN-OR, S. GOLDWASSER, A. WIGDERSON. **"Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation"**. In: *STOC'88*. ACM, 1988, pp. 1–10.

[BHKL18] A. BARAK, M. HIRT, L. KOSKAS, Y. LINDELL. **"An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for Weak Participants"**. In: *CCS'18*. ACM, 2018, pp. 695–712.

[BHKR13] M. BELLARE, V. HOANG, S. KEELVEEDHI, P. ROGAWAY. **"Efficient Garbling From a Fixed-Key Blockcipher"**. In: *Symposium on Security and Privacy (S&P'13)*. IEEE, 2013, pp. 478–492.

[BHS+17] R. BUSH, J. HAAS, J. SCUDDER, A. NIPPER, C. DIETZEL. **"Making Route Servers Aware of Data Link Failures at IXPs"**. https://tools.ietf.org/html/draft-ietf-idr-rs-bfd-05. 2017.

[BHWK16] N. BÜSCHER, A. HOLZER, A. WEBER, S. KATZENBEISSER. **"Compiling Low Depth Circuits for Practical Secure Computation"**. In: *ESORICS 2016*. Vol. 9879. LNCS. Springer, 2016, pp. 80–98.

[BJSV15] D. BOGDANOV, M. JÕEMETS, S. SIIM, M. VAHT. **"How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation"**. In: *Financial Cryptography and Data Security (FC'15)*. Vol. 8975. LNCS. Springer, 2015, pp. 227–234.

[BK15] N. BÜSCHER, S. KATZENBEISSER. **"Faster Secure Computation through Automatic Parallelization"**. In: *USENIX Security'15*. USENIX, 2015, pp. 531–546.

[BKJK16] N. BÜSCHER, D. KRETZMER, A. JINDAL, S. KATZENBEISSER. **"Scalable secure computation from ANSI-C"**. In: *IEEE International Workshop on Information Forensics and Security (WIFS'16)*. IEEE, 2016, pp. 1–6.

[BKOS07] D. BONEH, E. KUSHILEVITZ, R. OSTROVSKY, W. E. SKEITH, III. **"Public Key Encryption That Allows PIR Queries"**. In: *CRYPTO'07*. Vol. 4622. LNCS. Springer, 2007, pp. 50–67.

[BLO16] A. BEN-EFRAIM, Y. LINDELL, E. OMRI. **"Optimizing Semi-Honest Secure Multiparty Computation for the Internet"**. In: *CCS'16*. ACM, 2016, pp. 578–590.

[Blo70] B. H. BLOOM. **"Space/Time Trade-offs in Hash Coding with Allowable Errors"**. In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[BLR13] D. BOGDANOV, P. LAUD, J. RANDMETS. **"Domain-polymorphic language for privacy-preserving applications"**. In: *PETShop at CCS'13*. ACM, 2013, pp. 23–26.

[BLR14] D. BOGDANOV, P. LAUD, J. RANDMETS. **"Domain-Polymorphic Programming of Privacy-Preserving Applications"**. In: *Workshop on Programming Languages and Analysis for Security, PLAS@ECOOP'14*. 2014, p. 53.

[BLS12] D. J. BERNSTEIN, T. LANGE, P. SCHWABE. **"The Security Impact of a New Cryptographic Library"**. In: *Progress in Cryptology – LATINCRYPT'12*. Code: https://nacl.cr.yp.to. 2012, pp. 159–176.

[BLW08] D. BOGDANOV, S. LAUR, J. WILLEMSON. **"Sharemind: A Framework for Fast Privacy-Preserving Computations"**. In: *ESORICS'08*. Vol. 5283. LNCS. Springer, 2008, pp. 192–206.

[BMD+17] F. BRASSER, U. MÜLLER, A. DMITRIENKO, K. KOSTIAINEN, S. CAPKUN, A. SADEGHI. **"Software Grand Exposure: SGX Cache Attacks Are Practical"**. In: *CoRR* abs/1702.07521 (2017).

[BMR90] D. BEAVER, S. MICALI, P. ROGAWAY. **"The Round Complexity of Secure Protocols (Extended Abstract)"**. In: *22nd ACM STOC*. ACM, 1990, pp. 503–513.

[BMW+18] J. V. BULCK, M. MINKIN, O. WEISSE, D. GENKIN, B. KASIKCI, F. PIESSENS, M. SILBERSTEIN, T. F. WENISCH, Y. YAROM, R. STRACKX. **"Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution"**. In: *USENIX Security'18*. USENIX, 2018, 991–1008.

[BNP08] A. BEN-DAVID, N. NISAN, B. PINKAS. **"FairplayMP: a system for secure multi-party computation"**. In: *CCS'08*. ACM, 2008, pp. 257–266.

[BNTW12] D. BOGDANOV, M. NIITSOO, T. TOFT, J. WILLEMSON. **"High-performance secure multiparty computation for data mining applications"**. In: *Int. J. Inf. Sec.* 11.6 (2012), pp. 403–418.

[BP06] J. BOYAR, R. PERALTA. **"Concrete Multiplicative Complexity of Symmetric Functions"**. In: *Mathematical Foundations of Computer Science (MFCS'06)*. Vol. 4162. LNCS. Springer, 2006, pp. 179–189.

[BPP00] J. BOYAR, R. PERALTA, D. POCHUEV. **"On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$"**. In: *Theoretical Computer Science* 235.1 (2000), pp. 43–57.

[BPSW07] J. BRICKELL, D. E. PORTER, V. SHMATIKOV, E. WITCHEL. **"Privacy-preserving remote diagnostics"**. In: *CCS'07*. ACM, 2007, pp. 498–507.

[BS05] J. BRICKELL, V. SHMATIKOV. **"Privacy-Preserving Graph Algorithms in the Semi-honest Model"**. In: *ASIACRYPT'05*. Vol. 3788. LNCS. Springer, 2005, pp. 236–252.

[BSA13] M. BLANTON, A. STEELE, M. ALISAGARI. **"Data-oblivious Graph Algorithms for Secure Computation and Outsourcing"**. In: *ASIACCS'13*. ACM, 2013, pp. 207–218.

[BSMD10] M. BURKHART, M. STRASSER, D. MANY, X. A. DIMITROPOULOS. **"SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics"**. In: *USENIX Security'10*. USENIX, 2010, pp. 223–240.

[BTW12] D. BOGDANOV, R. TALVISTE, J. WILLEMSON. **"Deploying Secure Multi-Party Computation for Financial Data Analysis - (Short Paper)"**. In: *Financial Cryptography and Data Security (FC'12)*. Vol. 7397. LNCS. Springer, 2012, pp. 57–64.

[BV14] J. BUDURUSHI, M. VOLKAMER. **"Feasibility analysis of various electronic voting systems for complex elections"**. In: *International Conference for E-Democracy and Open Government 2014*. 2014.

[CAI16] CAIDA. **"The CAIDA AS Relationships Dataset, 20161101"**. http://www.caida.org/data/as-relationships/. 2016.

[Cap13] J. CAPPOS. **"Avoiding Theoretical Optimality to Efficiently and Privately Retrieve Security Updates"**. In: *Financial Cryptography and Data Security (FC'13)*. Vol. 7859. LNCS. Code: https://uppir.poly.edu. Springer, 2013, pp. 386–394.

[CB09] D. R. CHOFFNES, F. E. BUSTAMANTE. **"On the Effectiveness of Measurement Reuse for Performance-Based Detouring"**. In: *IEEE INFOCOM*. 2009, pp. 693–701.

[CBM15] H. CORRIGAN-GIBBS, D. BONEH, D. MAZIÈRES. **"Riposte: An anonymous messaging system handling millions of users"**. In: *IEEE Symposium on Security and Privacy (S&P'15)*. 2015, pp. 321–338.

[CCD88] D. CHAUM, C. CRÉPEAU, I. DAMGÅRD. **"Multiparty Unconditionally Secure Protocols (Extended Abstract)"**. In: *STOC'88*. ACM, 1988, pp. 11–19.

[CCL+17] G. S. CETIN, H. CHEN, K. LAINE, K. E. LAUTER, P. RINDAL, Y. XIA. **"Private Queries on Encrypted Genomic Data"**. Cryptology ePrint Archive, Report 2017/207. https://ia.cr/2017/207. 2017.

[CD16] V. COSTAN, S. DEVADAS. **"Intel SGX Explained"**. Cryptology ePrint Archive, Report 2016/086. https://ia.cr/2016/086. 2016.

[CDC+17a] M. CHIESA, D. DEMMLER, M. CANINI, M. SCHAPIRA, T. SCHNEIDER. **"Internet Routing Privacy Survey"**. http://bit.ly/2rjT7Nj. 2017.

[CDC+17b] M. CHIESA, D. DEMMLER, M. CANINI, M. SCHAPIRA, T. SCHNEIDER. **"SIXPACK: Securing Internet eXchange Points Against Curious onlooKers"**. In: *13. International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*. ACM, 2017, pp. 120–133.

[CDE+10] L. CITTADINI, G. DI BATTISTA, T. ERLEBACH, M. PATRIGNANI, M. RIMONDINI. **"Assigning AS relationships to satisfy the Gao-Rexford conditions"**. In: *International Conference on Network Protocols (ICNP)*. 2010, pp. 113–123.

[CDF+07] J. CALLAS, L. DONNERHACKE, H. FINNEY, D. SHAW, R. THAYER. **"OpenPGP Message Format"**. RFC 4880. RFC Editor, 2007.

[CDJ+17] D. CHAUM, D. DAS, F. JAVANI, A. KATE, A. KRASNOVA, J. d. RUITER, A. T. SHERMAN. **"cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations"**. In: *Applied Cryptography and Network Security (ACNS'17)*. 2017, pp. 557–578.

[CGKS95] B. CHOR, O. GOLDREICH, E. KUSHILEVITZ, M. SUDAN. **"Private Information Retrieval"**. In: *Foundations of Computer Science (FOCS'95)*. IEEE, 1995, pp. 41–50.

[CGN98] B. CHOR, N. GILBOA, M. NAOR. **"Private Information Retrieval by Keywords"**. Cryptology ePrint Archive, Report 1998/003. https://ia.cr/1998/003. 1998.

[CGR+17] N. CHANDRAN, D. GUPTA, A. RASTOGI, R. SHARMA, S. TRIPATHI. **"EzPC: Programmable, Efficient, and Scalable Secure Two-Party Computation"**. Cryptology ePrint Archive, Report 2017/1109. https://ia.cr/2017/1109. 2017.

[CH10] O. CATRINA, S. HOOGH. **"Improved Primitives for Secure Multiparty Integer Computation"**. In: *Security and Cryptography for Networks (SCN'10)*. Vol. 6280. LNCS. Springer, 2010, pp. 182–199.

[Cha81] D. L. CHAUM. **"Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms"**. In: *Communications of the ACM* 24 (2 1981), pp. 84–90.

[Cha83] D. L. CHAUM. **"Blind Signature Systems"**. In: *CRYPTO'83*. 1983, p. 153.

[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: *CT-RSA'12*. Vol. 7178. LNCS. Springer, 2012, pp. 416–432.

[CHLR18] H. CHEN, Z. HUANG, K. LAINE, P. RINDAL. **"Labeled PSI from Fully Homomorphic Encryption with Malicious Security"**. In: *CCS'18*. ACM, 2018, pp. 1223–1237.

[Cis16] CISCO. **"BGP Best Path Selection Algorithm"**. https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html. 2016.

[CJV+11] M. CANINI, V. JOVANOVIĆ, D. VENZANO, G. KUMAR, D. NOVAKOVIĆ, D. KOSTIĆ. **"Checking for Insidious Faults in Deployed Federated and Heterogeneous Distributed Systems"**. Tech. rep. 164475. EPFL, 2011.

[CLM11] D. CROCE, E. LEONARDI, M. MELLIA. **"Large-Scale Available Bandwidth Measurements: Interference in Current Techniques"**. In: *IEEE Transactions on Network and Service Management* 8.4 (2011), pp. 361–374.

[CLR17] H. CHEN, K. LAINE, P. RINDAL. **"Fast Private Set Intersection from Homomorphic Encryption"**. In: *CCS'17*. ACM, 2017, pp. 1243–1255.

[CLT14] H. CARTER, C. LEVER, P. TRAYNOR. **"Whitewash: Outsourcing Garbled Circuit Generation for Mobile Devices"**. In: *Annual Computer Security Applications Conference (ACSAC'14)*. ACSAC'14. ACM, 2014, pp. 266–275.

[CML10] D. CROCE, M. MELLIA, E. LEONARDI. **"The Quest for Bandwidth Estimation Techniques for Large-scale Distributed Systems"**. In: *SIGMETRICS Performance Evaluation Review* 37.3 (2010), pp. 20–25.

[CMO00] G. D. CRESCENZO, T. MALKIN, R. OSTROVSKY. **"Single Database Private Information Retrieval Implies Oblivious Transfer"**. In: *EUROCRYPT'00*. Vol. 1807. LNCS. Springer, 2000, pp. 122–138.

[CMS99] C. CACHIN, S. MICALI, M. STADLER. **"Computationally Private Information Retrieval with Polylogarithmic Communication"**. In: *EUROCRYPT'99*. Vol. 1592. LNCS. Springer, 1999, pp. 402–414.

[CMTB13] H. CARTER, B. MOOD, P. TRAYNOR, K. BUTLER. **"Secure Outsourced Garbled Circuit Evaluation for Mobile Devices"**. In: *USENIX Security'13*. USENIX, 2013, pp. 289–304.

[CMTB16] H. CARTER, B. MOOD, P. TRAYNOR, K. BUTLER. **"Outsourcing Secure Two-party Computation As a Black Box"**. In: *Security and Communication Networks (SCN)* 9.14 (2016), pp. 2261–2275.

[CS10] O. CATRINA, A. SAXENA. **"Secure Computation with Fixed-Point Numbers"**. In: *Financial Cryptography and Data Security (FC'10)*. Vol. 6052. LNCS. Springer, 2010, pp. 35–50.

[CT10] E. D. CRISTOFARO, G. TSUDIK. **"Practical Private Set Intersection Protocols with Linear Complexity"**. In: *Financial Cryptography and Data Security (FC'10)*. Vol. 6052. LNCS. Springer, 2010, pp. 143–159.

[DAA+11] P. DANECEK, A. AUTON, G. ABECASIS, C. A. ALBERS, E. BANKS, M. A. DEPRISTO, R. E. HANDSAKER, G. LUNTER, G. T. MARTH, S. T. SHERRY, G. MCVEAN, R. DURBIN. **"The variant call format and VCFtools"**. In: *Bioinformatics* 27.15 (2011), pp. 2156–2158.

[DCC18] A. DETHISE, M. CHIESA, M. CANINI. **"Prelude: Ensuring Inter-Domain Loop-Freedom in SDN-Enabled Networks"**. In: *APNet'18*. ACM, 2018, pp. 50–56.

[DCW13] C. DONG, L. CHEN, Z. WEN. **"When private set intersection meets big data: an efficient and scalable protocol"**. In: *CCS'13*. ACM, 2013, pp. 789–800.

[DDK+15] 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.

[DEC16] **"An IXP Route Server Test Framework"**. https://www.de-cix.net/_Resources/Persistent/fba89bc19381b6784df99d2a78d4a11ebb7583c2/DE-CIX-route-server-testframework.pdf. 2016.

[DFT13] E. DE CRISTOFARO, S. FABER, G. TSUDIK. **"Secure Genomic Testing with Size- and Position-hiding Private Substring Matching"**. In: *12. ACM Workshop on Privacy in the Electronic Society (WPES'13)*. ACM, 2013, pp. 107–118.

[DG14] C. DEVET, I. GOLDBERG. **"The Best of Both Worlds: Combining Information-Theoretic and Computational PIR for Communication Efficiency"**. In: *Privacy Enhancing Technologies Symposium (PETS'14)*. Vol. 8555. LNCS. Springer, 2014, pp. 63–82.

[DGH12] C. DEVET, I. GOLDBERG, N. HENINGER. **"Optimally Robust Private Information Retrieval"**. In: *USENIX Security'12*. USENIX, 2012, pp. 269–283.

[DGK08] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD. **"Homomorphic encryption and secure comparison"**. In: *International Journal of Applied Cryptography* 1.1 (2008), pp. 22–31.

[DGK09] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD. **"A correction to 'Efficient and secure comparison for on-line auctions'"**. In: *International Journal of Applied Cryptography* 1.4 (2009), pp. 323–324.

[DGKN09] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD, J. B. NIELSEN. **"Asynchronous Multiparty Computation: theory and implementation"**. In: *PKC'09*. Vol. 5443. LNCS. Springer, 2009, pp. 160–179.

[DGM+10] M. DIETZFELBINGER, A. GOERDT, M. MITZENMACHER, A. MONTANARI, R. PAGH, M. RINK. **"Tight thresholds for Cuckoo hashing via XORSAT"**. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2010, pp. 213–225.

[DGN10] I. DAMGÅRD, M. GEISLER, J. B. NIELSEN. **"From Passive to Covert Security at Low Cost"**. In: *TCC'10*. Vol. 5978. LNCS. Springer, 2010, pp. 128–145.

[DHC04] W. DU, Y. S. HAN, S. CHEN. **"Privacy-Preserving Multivariate Statistical Analysis: Linear Regression and Classification"**. In: *SIAM'04*. 2004, pp. 222–233.

[DHS14] D. DEMMLER, A. HERZBERG, T. SCHNEIDER. **"RAID-PIR: Practical Multi-Server PIR"**. In: *6. ACM Cloud Computing Security Workshop (CCSW'14)*. Code: https://encrypto.de/code/RAID-PIR. ACM, 2014, pp. 45–56.

[DHS17] D. DEMMLER, M. HOLZ, T. SCHNEIDER. **"OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks"**. In: *15. International Conference on Applied Cryptography and Network Security (ACNS'17)*. Vol. 10355. LNCS. Code: https://encrypto.de/code/onionPIR. Springer, 2017, pp. 599–619.

[DHSS17] D. DEMMLER, K. HAMACHER, T. SCHNEIDER, S. STAMMLER. **"Privacy-Preserving Whole-Genome Variant Queries"**. In: *16. International Conference on Cryptology And Network Security (CANS'17)*. Vol. 11261. LNCS. Springer, 2017, pp. 71–92.

[DJ01] I. DAMGÅRD, M. JURIK. **"A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System"**. In: *PKC'01*. Vol. 1992. LNCS. Springer, 2001, pp. 119–136.

[DJN10] I. DAMGÅRD, M. JURIK, J. B. NIELSEN. **"A generalization of Paillier's public-key system with applications to electronic voting"**. In: *International Journal of Information Security* 9.6 (2010), pp. 371–385.

[DKF+07] X. DIMITROPOULOS, D. KRIOUKOV, M. FOMENKOV, B. HUFFAKER, Y. HYUN, K. CLAFFY, G. RILEY. **"AS Relationships: Inference and Validation"**. In: *Computer Communication Review* 37.1 (2007), pp. 29–40.

[DKL+13] 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*. Vol. 8134. LNCS. Springer, 2013, pp. 1–18.

[DKS+17] G. DESSOUKY, F. KOUSHANFAR, A.-R. SADEGHI, T. SCHNEIDER, S. ZEITOUNI, M. ZOHNER. **"Pushing the Communication Barrier in Secure Computation using Lookup Tables"**. In: *24. Annual Network and Distributed System Security Symposium (NDSS'17)*. Full version: https://ia.cr/2018/486. Internet Society, 2017.

[DKT10] E. De Cristofaro, J. Kim, G. Tsudik. **"Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model"**. In: *ASIACRYPT 2010*. Vol. 6477. LNCS. Springer, 2010, pp. 213–231.

[DMS04] R. Dingledine, N. Mathewson, P. Syverson. **"Tor: The Second-generation Onion Router"**. In: *USENIX Security'04*. USENIX, 2004, pp. 21–21.

[DPSZ12] I. Damgård, V. Pastro, N. P. Smart, S. Zakarias. **"Multiparty Computation from Somewhat Homomorphic Encryption"**. In: *CRYPTO'12*. Vol. 7417. LNCS. Springer, 2012, pp. 643–662.

[DRRT18] D. Demmler, P. Rindal, M. Rosulek, N. Trieu. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). Code: https://github.com/osu-crypto/libPSI.

[DS18] J. P. Degabriele, M. Stam. **"Untagging Tor: A Formal Treatment of Onion Encryption"**. In: *EUROCRYPT'18*. Vol. 10822. LNCS. Springer, 2018, pp. 259–293.

[DSS14] J. Dautrich, E. Stefanov, E. Shi. **"Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns"**. In: *USENIX Security'14*. USENIX, 2014.

[DSZ14] D. Demmler, T. Schneider, M. Zohner. **"Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens"**. In: *USENIX Security'14*. Full version: https://ia.cr/2014/467. USENIX, 2014, pp. 893–908.

[DSZ15] D. Demmler, T. Schneider, M. Zohner. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. Code: https://encrypto.de/code/ABY. Internet Society, 2015.

[DSZ16] B. Dowling, D. Stebila, G. Zaverucha. **"Authenticated Network Time Synchronization"**. In: *USENIX Security'16*. USENIX, 2016, pp. 823–840.

[DT10] E. De Cristofaro, G. Tsudik. **"Practical Private Set Intersection Protocols with Linear Complexity"**. In: *Financial Cryptography and Data Security (FC'10)*. Vol. 6052. LNCS. Springer, 2010, pp. 143–159.

[DYDW10] X. Ding, Y. Yang, R. H. Deng, S. Wang. **"A new hardware-assisted PIR with O(n) shuffle cost"**. In: *International Journal of Information Security* 9.4 (2010), pp. 237–252.

[EGL85] S. Even, O. Goldreich, A. Lempel. **"A randomized protocol for signing contracts"**. In: *Communications of the ACM*. ACM 28.6 (1985), pp. 637–647.

[END15] ENDEAVOUR. **"Project ENDEAVOUR"**. https://www.de-cix.net/en/about-de-cix/research-and-development/endeavour. 2015.

[EU16] **"Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)"**. In: *Official Journal of the European Union* L119 (2016), pp. 1–88.

[FDH+13] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, H. Schröder. **"Secure computations on non-integer values with applications to privacy-preserving sequence analysis"**. In: *Information Security Technical Report* 17.3 (2013), pp. 117–128.

[FES+17] D. FROELICHER, P. EGGER, J. S. SOUSA, J. L. RAISARO, Z. HUANG, C. MOUCHET, B. FORD, J.-P. HUBAUX. **"UnLynx: A Decentralized System for Privacy-Conscious Data Sharing"**. In: *Privacy Enhancing Technologies Symposium (PETS'17)*. Vol. 4. 2017, pp. 152–170.

[FK11] M. FRANZ, S. KATZENBEISSER. **"Processing Encrypted Floating Point Signals"**. In: *ACM Multimedia and Security (MM&Sec'11)*. ACM, 2011, pp. 103–108.

[FLNW17] J. FURUKAWA, Y. LINDELL, A. NOF, O. WEINSTEIN. **"High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority"**. In: *EUROCRYPT'17*. Vol. 10211. LNCS. Springer, 2017, pp. 225–255.

[FM11] I. FETTE, A. MELNIKOV. **"The WebSocket Protocol"**. RFC 6455. RFC Editor, 2011.

[FMM09] A. FRIEZE, P. MELSTED, M. MITZENMACHER. **"An analysis of random-walk Cuckoo hashing"**. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 490–503.

[FN13] T. K. FREDERIKSEN, J. B. NIELSEN. **"Fast and Maliciously Secure Two-Party Computation Using the GPU"**. In: *Applied Cryptography and Network Security (ACNS'13)*. Vol. 7954. LNCS. Springer, 2013, pp. 339–356.

[FNP04] M. J. FREEDMAN, K. NISSIM, B. PINKAS. **"Efficient Private Matching and Set Intersection"**. In: *EUROCRYPT'04*. Vol. 3027. LNCS. Springer, 2004, pp. 1–19.

[FPRS04] J. FEIGENBAUM, B. PINKAS, R. S. RYGER, F. SAINT-JEAN. **"Secure computation of surveys"**. In: *EU Workshop on Secure Multiparty Protocols*. ECRYPT, 2004.

[FPS+11] M. FISCHLIN, B. PINKAS, A.-R. SADEGHI, T. SCHNEIDER, I. VISCONTI. **"Secure Set Intersection with Untrusted Hardware Tokens"**. In: *11. Cryptographers' Track at the RSA Conference (CT-RSA'11)*. Vol. 6558. LNCS. Springer, 2011, pp. 1–16.

[FSR11] A. FABRIKANT, U. SYED, J. REXFORD. **"There's something about MRAI: Timing diversity can exponentially worsen BGP convergence"**. In: *INFOCOM'11*. IEEE, 2011, pp. 2975–2983.

[Gam85] T. E. GAMAL. **"A public key cryptosystem and a signature scheme based on discrete logarithms"**. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[Gao01] L. GAO. **"On Inferring Autonomous System Relationships in the Internet"**. In: *IEEE/ACM Transactions on Networking* 9.6 (2001), pp. 733–745.

[GDL+14] I. GOLDBERG, C. DEVET, W. LUEKS, A. YANG, P. HENDRY, R. HENRY. **"Percy++ project on SourceForge"**. http://percy.sourceforge.net. Version 1.0.0. 2014.

[GDL+16] R. GILAD-BACHRACH, N. DOWLIN, K. LAINE, K. E. LAUTER, M. NAEHRIG, J. WERNSING. **"CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy"**. In: *ICML'16*. 2016, pp. 201–210.

[Gei10] M. GEISLER. **"Cryptographic Protocols: Theory and Implementation"**. PhD thesis. Aarhus University, 2010.

[Gen09] C. GENTRY. **"Fully Homomorphic Encryption Using Ideal Lattices"**. In: *STOC'09*. ACM, 2009, pp. 169–178.

[GGH+13] S. GARG, C. GENTRY, S. HALEVI, M. RAYKOVA, A. SAHAI, B. WATERS. **"Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits"**. In: *Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2013, pp. 40–49.

[GHS12] C. GENTRY, S. HALEVI, N. P. SMART. **"Homomorphic evaluation of the AES circuit"**. In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 850–867.

[GHSG16] Y. GILAD, A. HERZBERG, M. SUDKOVITCH, M. GOBERMAN. **"CDN-on-Demand: An Affordable DDoS Defense via Untrusted Clouds"**. In: *Network and Distributed System Security (NDSS'16)*. The Internet Society, 2016.

[GI14] N. GILBOA, Y. ISHAI. **"Distributed Point Functions and Their Applications"**. In: *EUROCRYPT'14*. Vol. 8441. LNCS. Springer, 2014, pp. 640–658.

[Gil99] N. GILBOA. **"Two Party RSA Key Generation"**. In: *CRYPTO'99*. Vol. 1666. LNCS. Springer, 1999, pp. 116–129.

[GKL10] J. GROTH, A. KIAYIAS, H. LIPMAA. **"Multi-query Computationally-Private Information Retrieval with Constant Communication Rate"**. In: *PKC'10*. Vol. 6056. LNCS. Springer, 2010, pp. 107–123.

[GKP+13] S. GOLDWASSER, Y. KALAI, R. A. POPA, V. VAIKUNTANATHAN, N. ZELDOVICH. **"Reusable Garbled Circuits and Succinct Functional Encryption"**. In: *Symposium on Theory of Computing (STOC)*. ACM, 2013, pp. 555–564.

[GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. **"More Efficient Universal Circuit Constructions"**. In: *ASIACRYPT'17*. Vol. 10625. LNCS. Full version: https://ia.cr/2017/798. Springer, 2017, pp. 443–470.

[GLL+16] R. GILAD-BACHRACH, K. LAINE, K. LAUTER, P. RINDAL, M. ROSULEK. **"Secure Data Exchange: A Marketplace in the Cloud"**. Cryptology ePrint Archive, Report 2016/620. https://ia.cr/. 2016.

[GMB+16] A. GUPTA, R. MACDAVID, R. BIRKNER, M. CANINI, N. FEAMSTER, J. REXFORD, L. VANBEVER. **"An Industrial-Scale Software Defined Internet Exchange Point"**. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX, 2016, pp. 1–14.

[GMF+16] D. GUPTA, B. MOOD, J. FEIGENBAUM, K. BUTLER, P. TRAYNOR. **"Using Intel Software Guard Extensions for Efficient Two-Party Secure Function Evaluation"**. In: *4. Workshop on Encrypted Computing and Applied Homomorphic Cryptography (WAHC'16)*. Vol. 9604. LNCS. Springer, 2016, pp. 302–318.

[GMW87] 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.

[GO96] O. GOLDREICH, R. OSTROVSKY. **"Software Protection and Simulation on Oblivious RAMs"**. In: *Journal of the ACM (JACM'96)* 43.3 (1996), pp. 431–473.

[Gol04] O. GOLDREICH. **"The Foundations of Cryptography - Volume 2, Basic Applications"**. Cambridge University Press, 2004.

[Gol07] I. GOLDBERG. **"Improving the Robustness of Private Information Retrieval"**. In: *IEEE Symposium on Security and Privacy (S&P'07)*. IEEE, 2007, pp. 131–148.

[Goo09] M. T. GOODRICH. **"The mastermind attack on genomic data"**. In: *30. IEEE Symposium on Security and Privacy (S&P'09)*. IEEE, 2009, pp. 204–218.

[GR01] L. GAO, J. REXFORD. **"Stable Internet routing without global coordination"**. In: *IEEE/ACM Transactions on Networking* 9.6 (2001), pp. 681–692.

[GSG11] P. GILL, M. SCHAPIRA, S. GOLDBERG. **"Let the market drive deployment: a strategy for transitioning to BGP security"**. In: *SIGCOMM'11*. ACM, 2011, pp. 14–25.

[GSG12] P. GILL, M. SCHAPIRA, S. GOLDBERG. **"Modeling on quicksand: dealing with the scarcity of ground truth in interdomain routing data"**. In: *Computer Communication Review* 42.1 (2012), pp. 40–46.

[GSG14] P. GILL, M. SCHAPIRA, S. GOLDBERG. **"A Survey of Interdomain Routing Policies"**. In: *Computer Communication Review* 44.1 (2014), pp. 28–34.

[GSHR10] S. GOLDBERG, M. SCHAPIRA, P. HUMMON, J. REXFORD. **"How secure are secure interdomain routing protocols"**. In: *SIGCOMM'10*. ACM, 2010, pp. 87–98.

[GSP+12] D. GUPTA, A. SEGAL, A. PANDA, G. SEGEV, M. SCHAPIRA, J. FEIGENBAUM, J. REXFORD, S. SHENKER. **"A new approach to interdomain routing based on secure multi-party computation"**. In: *Workshop on Hot Topics in Networks (HotNets'12)*. ACM, 2012, pp. 37–42.

[GSV07] J. GARAY, B. SCHOENMAKERS, J. VILLEGAS. **"Practical and Secure Solutions for Integer Comparison"**. In: *PKC'07*. Vol. 4450. LNCS. Springer, 2007, pp. 330–342.

[GSW02] T. GRIFFIN, F. SHEPHERD, G. WILFONG. **"The stable paths problem and interdomain routing"**. In: *IEEE/ACM Transactions on Networking* 10.2 (2002), pp. 232–243.

[GVS+14] A. GUPTA, L. VANBEVER, M. SHAHBAZ, S. P. DONOVAN, B. SCHLINKER, N. FEAMSTER, J. REXFORD, S. SHENKER, R. CLARK, E. KATZ-BASSETT. **"SDX: A Software Defined Internet Exchange"**. In: *SIGCOMM'14*. ACM, 2014, pp. 551–562.

[GZ11] V. GIOTSAS, S. ZHOU. **"Inferring AS Relationships from BGP Attributes"**. In: *CoRR* abs/1106.2417 (2011).

[HCE11] Y. HUANG, P. CHAPMAN, D. EVANS. **"Privacy-preserving Applications on Smartphones"**. In: *USENIX Conference on Hot Topics in Security*. HotSec'11. USENIX Association, 2011, pp. 4–4.

[HEK12] Y. HUANG, D. EVANS, J. KATZ. **"Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?"** In: *NDSS'12*. The Internet Society, 2012.

[HEKM11] Y. HUANG, D. EVANS, J. KATZ, L. MALKA. **"Faster Secure Two-Party Computation Using Garbled Circuits"**. In: *USENIX Security'11*. USENIX, 2011, pp. 539–554.

[Hen16] R. HENRY. **"Polynomial Batch Codes for Efficient IT-PIR"**. In: *Privacy Enhancing Technologies Symposium (PETS'16)* 2016 (4 2016), pp. 202–218.

[HFKV12] A. HOLZER, M. FRANZ, S. KATZENBEISSER, H. VEITH. **"Secure two-party computations in ANSI C"**. In: *CCS'12*. ACM, 2012, pp. 772–783.

[HHG13] R. HENRY, Y. HUANG, I. GOLDBERG. **"One (Block) Size Fits All: PIR and SPIR with Variable-Length Records via Multi-Block Queries"**. In: *Network and Distributed System Security (NDSS'13)*. The Internet Society, 2013.

[HHT14] K. HAMACHER, J. P. HUBAUX, G. TSUDIK. **"Genomic Privacy (Dagstuhl Seminar 13412)"**. In: *Dagstuhl Reports* 3.10 (2014), pp. 25–35.

[HKS+10] W. HENECKA, S. KÖGL, A.-R. SADEGHI, T. SCHNEIDER, I. WEHRENBERG. **"TASTY: Tool for Automating Secure Two-partY computations"**. In: *CCS'10*. Full version: https://ia.cr/2010/365. Code: https://encrypto.de/code/TASTY. ACM, 2010, pp. 451–462.

[HL10] C. HAZAY, Y. LINDELL. **"Efficient Secure Two-Party Protocols: Techniques and Constructions"**. 1st. Springer, 2010.

[HLZZ15] J. v. d. HOOFF, D. LAZAR, M. ZAHARIA, N. ZELDOVICH. **"Vuvuzela: Scalable private messaging resistant to traffic analysis"**. In: *Symposium on Operating Systems Principles (SOSP'15)*. 2015, pp. 137–152.

[HMEK11] Y. HUANG, L. MALKA, D. EVANS, J. KATZ. **"Efficient Privacy-Preserving Biometric Identification"**. In: *NDSS'11*. The Internet Society, 2011.

[HMM17] M. Z. HASAN, M. S. R. MAHDI, N. MOHAMMED. **"Secure Count Query on Encrypted Genomic Data"**. In: *International Workshop on Genome Privacy and Security (GenoPri'16)*. 2017.

[HOSS18] C. HAZAY, E. ORSINI, P. SCHOLL, E. SORIA-VAZQUEZ. **"Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT)"**. In: *ASIACRYPT'18*. LNCS. Springer, 2018.

[HR13] W. HENECKA, M. ROUGHAN. **"STRIP: Privacy-preserving vector-based routing"**. In: *International Conference on Network Protocols (ICNP'13)*. IEEE, 2013, pp. 1–10.

[HRA11] G. HUSTON, M. ROSSI, G. ARMITAGE. **"Securing BGP – A Literature Survey"**. In: *Communications Surveys Tutorials, IEEE* 13.2 (2011), pp. 199–222.

[HS13] W. HENECKA, T. SCHNEIDER. **"Faster Secure Two-Party Computation with Less Memory"**. In: *ASIACCS'13*. Code: https://encrypto.de/code/me-sfe. ACM, 2013, pp. 437–446.

[HSS17] C. HAZAY, P. SCHOLL, E. SORIA-VAZQUEZ. **"Low Cost Constant Round MPC Combining BMR and Oblivious Transfer"**. In: *ASIACRYPT 2017*. Vol. 10624. LNCS. Springer, 2017, pp. 598–628.

[IAR14] IARPA. **"Security and Privacy Assurance Research-Multiparty Computation (SPAR-MPC) Program"**. Solicitation Number: IARPA-RFI-14-03. Intelligence Advanced Research Projects Activity (IARPA). 2014.

[IEE08] IEEE. **"IEEE Standard for Floating-Point Arithmetic"**. In: *IEEE Std 754-2008* (2008), pp. 1–70.

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *CRYPTO'03*. Vol. 2729. LNCS. Springer, 2003, pp. 145–161.

[IR89] R. IMPAGLIAZZO, S. RUDICH. **"Limits on the Provable Consequences of One-Way Permutations"**. In: *21st ACM STOC*. ACM, 1989, pp. 44–61.

[JCC+13] U. JAVED, I. CUNHA, D. R. CHOFFNES, E. KATZ-BASSETT, T. E. ANDERSON, A. KRISHNAMURTHY. **"PoiRoot: investigating the root cause of interdomain path changes"**. In: *SIGCOMM'13*. ACM, 2013, pp. 183–194.

[JD08] M. JAIN, C. DOVROLIS. **"Path Selection Using Available Bandwidth Estimation in Overlay-Based Video Streaming"**. In: *Computer Networks* 52.12 (2008), pp. 2411–2418.

[JDS+16] P. JAIN, S. J. DESAI, M. SHIH, T. KIM, S. M. KIM, J. LEE, C. CHOI, Y. SHIN, B. B. KANG, D. HAN. **"OpenSGX: An Open Platform for SGX Research"**. In: *The Network and Distributed System Security Symposium (NDSS)*. 2016.

[JKS08] S. JHA, L. KRUGER, V. SHMATIKOV. **"Towards Practical Privacy for Genomic Computation"**. In: *Symposium on Security and Privacy (S&P'08)*. IEEE, 2008, pp. 216–230.

[JKSS10] K. JÄRVINEN, V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"Embedded SFE: Offloading Server and Network using Hardware Tokens"**. In: *14. International Conference on Financial Cryptography and Data Security (FC'10)*. Vol. 6052. LNCS. Full version: https://ia.cr/2009/591. Springer, 2010, pp. 207–221.

[JW05] G. JAGANNATHAN, R. N. WRIGHT. **"Privacy-preserving distributed k-means clustering over arbitrarily partitioned data"**. In: *SIGKDD'05*. ACM, 2005, pp. 593–599.

[KA99] Y.-K. KWOK, I. AHMAD. **"Static scheduling algorithms for allocating directed task graphs to multiprocessors"**. In: *ACM Computing Surveys (CSUR)* 31.4 (1999), pp. 406–471.

[KBS14] F. KERSCHBAUM, M. BECK, D. SCHÖNFELD. **"Inference Control for Privacy-Preserving Genome Matching"**. In: *CoRR* abs/1405.0205 (2014).

[Kel15] M. KELLER. **"The Oblivious Machine – or: How to Put the C into MPC"**. Cryptology ePrint Archive, Report 2015/467. https://ia.cr/2015/467. 2015.

[KHH+17] S. M. KIM, J. HAN, J. HA, T. KIM, D. HAN. **"Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments"**. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*. USENIX, 2017, pp. 145–161.

[KKK07] N. KUSHMAN, S. KANDULA, D. KATABI. **"Can You Hear Me Now?!: It Must Be BGP"**. In: *SIGCOMM'07* 37.2 (2007), pp. 75–84.

[KKRT16] V. KOLESNIKOV, R. KUMARESAN, M. ROSULEK, N. TRIEU. **"Efficient Batched Oblivious PRF with Applications to Private Set Intersection"**. In: *CCS'16*. ACM, 2016, pp. 818–829.

[KLDF16] A. KWON, D. LAZAR, S. DEVADAS, B. FORD. **"Riffle: An Efficient Communication System With Strong Anonymity"**. In: *Privacy Enhancing Technologies Symposium (PETS'16)* 2016 (2 2016), pp. 115–134.

[KLS+17] Á. KISS, J. LIU, T. SCHNEIDER, N. ASOKAN, B. PINKAS. **"Private Set Intersection for Unequal Set Sizes with Mobile Applications"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.4 (2017). Full version: https://ia.cr/2017/670. Code: https://encrypto.de/code/MobilePSI, pp. 177–197.

[KM14] V. KOLESNIKOV, A. J. MALOZEMOFF. **"Public Verifiability in the Covert Model (Almost) for Free"**. In: *ASIACRYPT'14*. Vol. 9453. LNCS. Springer, 2014, pp. 210–235.

[KMR11] S. KAMARA, P. MOHASSEL, M. RAYKOVA. **"Outsourcing Multi-Party Computation"**. Cryptology ePrint Archive, Report 2011/272. https://ia.cr/2011/272. 2011.

[KNR+17] S. KRÜGER, S. NADI, M. REIF, K. ALI, M. MEZINI, E. BODDEN, F. GÖPFERT, F. GÜNTHER, C. WEINERT, D. DEMMLER, R. KAMATH. **"CogniCrypt: Supporting Developers in Using Cryptography"**. In: *32nd International Conference on Automated Software Engineering (ASE'17)*. IEEE, 2017, pp. 931–936.

[KO62] A. A. KARATSUBA, Y. OFMAN. **"Multiplication of Many-Digital Numbers by Automatic Computers"**. In: *SSSR Academy of Sciences* 145 (1962), pp. 293–294.

[KO97] E. KUSHILEVITZ, R. OSTROVSKY. **"Replication is Not Needed: Single Database, Computationally-private Information Retrieval"**. In: *Foundations of Computer Science (FOCS'97)*. IEEE, 1997, pp. 364–373.

[KOS16] M. KELLER, E. ORSINI, P. SCHOLL. **"MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer"**. In: *CCS'16*. ACM, 2016, pp. 830–842.

[KPR+15] P. KOEBERL, V. PHEGADE, A. RAJAN, T. SCHNEIDER, S. SCHULZ, M. ZHDANOVA. **"Time to Rethink: Trust Brokerage Using Trusted Execution Environments"**. In: *Trust and Trustworthy Computing (TRUST)*. Vol. 9229. LNCS. Springer, 2015, pp. 181–190.

[KRG+16] R. KLOETI, M. ROST, P. GEORGOPOULOS, B. AGER, S. SCHMID, D. X. **"Stitching Inter-Domain Paths over IXPs"**. In: *ACM SIGCOMM Symposium on SDN Research (SOSR'16)*. 2016.

[KS08a] V. KOLESNIKOV, T. SCHNEIDER. **"A Practical Universal Circuit Construction and Secure Evaluation of Private Functions"**. In: *12. International Conference on Financial Cryptography and Data Security (FC'08)*. Vol. 5143. LNCS. Code: https://encrypto.de/code/FairplayPF. Springer, 2008, pp. 83–97.

[KS08b] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Applications"**. In: *35. International Colloquium on Automata, Languages and Programming (ICALP'08)*. Vol. 5126. LNCS. Springer, 2008, pp. 486–498.

[KS16] Á. KISS, T. SCHNEIDER. **"Valiant's Universal Circuit is Practical"**. In: *35. Advances in Cryptology – EUROCRYPT 2016*. Vol. 9665. LNCS. Full version: https://ia.cr/2016/093. Code: https://encrypto.de/code/UC. Springer, 2016, pp. 699–728.

[KSH+15] S. KIM, Y. SHIN, J. HA, T. KIM, D. HAN. **"A First Step Towards Leveraging Commodity Trusted Execution Environments for Network Applications"**. In: *ACM Workshop on Hot Topics in Networks (HotNets*. 2015.

[KSMB13] B. KREUTER, A. SHELAT, B. MOOD, K. R. B. BUTLER. **"PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation"**. In: *USENIX Security'13*. USENIX, 2013, pp. 321–336.

[KSS09] V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima"**. In: *8. International Conference on Cryptology And Network Security (CANS'09)*. Vol. 5888. LNCS. Full version: https://ia.cr/2009/411. Springer, 2009, pp. 1–20.

[KSS12] B. KREUTER, A. SHELAT, C. SHEN. **"Billion-Gate Secure Computation with Malicious Adversaries"**. In: *USENIX Security'12*. USENIX, 2012, pp. 285–300.

[KSS13a] M. KELLER, P. SCHOLL, N. P. SMART. **"An architecture for practical actively secure MPC with dishonest majority"**. In: *CCS'13*. ACM, 2013, pp. 549–560.

[KSS13b] V. KOLESNIKOV, A.-R. SADEGHI, T. SCHNEIDER. **"A Systematic Approach to Practically Efficient General Two-Party Secure Function Evaluation Protocols and their Modular Design"**. In: *Journal of Computer Security (JCS)* 21.2 (2013). Preliminary version: https://ia.cr/2010/079, pp. 283–315.

[KSS14] F. KERSCHBAUM, T. SCHNEIDER, A. SCHRÖPFER. **"Automatic Protocol Selection in Secure Two-Party Computations"**. In: *12. International Conference on Applied Cryptography and Network Security (ACNS'14)*. Vol. 8479. LNCS. Full version: https://ia.cr/2014/200. Springer, 2014, pp. 566–584.

[KW14] L. KAMM, J. WILLEMSON. **"Secure floating point arithmetic and private satellite collision analysis"**. In: *International Journal of Information Security* (2014), pp. 1–18.

[LABJ00] C. LABOVITZ, A. AHUJA, A. BOSE, F. JAHANIAN. **"Delayed Internet Routing Convergence"**. In: *SIGCOMM'00* 30.4 (2000), pp. 175–187.

[Lam16] M. LAMBÆK. **"Breaking and Fixing Private Set Intersection Protocols"**. https://ia.cr/2016/665. MA thesis. Aarhus University, 2016.

[Lan15] S. LANDAU. **"Mining the Metadata: And Its Consequences"**. In: *International Conference on Software Engineering (ICSE'15)*. 2015, pp. 4–5.

[LF80] R. E. LADNER, M. J. FISCHER. **"Parallel Prefix Computation"**. In: *Journal of the ACM* 27.4 (1980), pp. 831–838.

[LG15] W. LUEKS, I. GOLDBERG. **"Sublinear Scaling for Multi-Client Private Information Retrieval"**. In: *Financial Cryptography and Data Security (FC'15)*. LNCS. Springer, 2015, pp. 168–186.

[LHS+14] C. LIU, Y. HUANG, E. SHI, M. HICKS, J. KATZ. **"Automating Efficient RAM-Model Secure Computation"**. In: *S&P'14*. IEEE, 2014.

[Lig15] LIGHTREADING. **"Pica8 Powers French TOUIX SDN-Driven Internet Exchange"**. http://www.lightreading.com/white-box/white-box-systems/pica8-powers-french-touix-sdn-driven-internet-exchange/d/d-id/716667. 2015.

[Lin13] Y. LINDELL. **"Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries"**. In: *CRYPTO'13 (2)*. Vol. 8043. LNCS. Springer, 2013, pp. 1–17.

[LJLA17] J. LIU, M. JUUTI, Y. LU, N. ASOKAN. **"Oblivious Neural Network Predictions via MiniONN Transformations"**. In: *CCS'17*. ACM, 2017, pp. 619–631.

[LMS16] H. LIPMAA, P. MOHASSEL, S. S. SADEGHIAN. **"Valiant's Universal Circuit: Improvements, Implementation, and Applications"**. Cryptology ePrint Archive, Report 2016/017. https://ia.cr/2016/017. 2016.

[LN17] Y. LINDELL, A. NOF. **"A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority"**. In: *CCS'17*. ACM, 2017, pp. 259–276.

[LOS14] E. LARRAIA, E. ORSINI, N. P. SMART. **"Dishonest Majority Multi-Party Computation for Binary Circuits"**. In: *CRYPTO'14 (2)*. Vol. 8617. LNCS. Springer, 2014, pp. 495–512.

[LP07] Y. LINDELL, B. PINKAS. **"An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries"**. In: *EUROCRYPT'07*. Springer, 2007, pp. 52–78.

[LP09] Y. LINDELL, B. PINKAS. **"A Proof of Security of Yao's Protocol for Two-Party Computation"**. In: *Journal of Cryptology* 22.2 (2009), pp. 161–188.

[LPB+16] T. LEE, C. PAPPAS, D. BARRERA, P. SZALACHOWSKI, A. PERRIG. **"Source Accountability with Domain-brokered Privacy"**. In: *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 2016, pp. 345–358.

[LR14] Y. LINDELL, B. RIVA. **"Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings"**. In: *CRYPTO 2014*. Springer, 2014, pp. 476–494.

[LR15] Y. LINDELL, B. RIVA. **"Blazing Fast 2PC in the Offline/Online Setting with Security for Malicious Adversaries"**. In: *CCS'15*. ACM, 2015, pp. 579–590.

[LWN+15] C. LIU, X. S. WANG, K. NAYAK, Y. HUANG, E. SHI. **"ObliVM: A Programming Framework for Secure Computation"**. In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2015, pp. 359–376.

[LZ16] D. LAZAR, N. ZELDOVICH. **"Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata"**. In: *Symposium on Operating Systems Design and Implementation, (OSDI'16)*. USENIX, 2016, pp. 571–586.

[Mak10] M. X. MAKKES. **"Efficient Implementation of Homomorphic Cryptosystems"**. MA thesis. Technische Universiteit Eindhoven, 2010.

[Mal11] L. MALKA. **"VMCrypt - Modular Software Architecture for Scalable Secure Computation"**. In: *CCS'11*. ACM, 2011, pp. 715–724.

[MBC13] T. MAYBERRY, E.-O. BLASS, A. H. CHAN. **"PIRMAP: Efficient Private Information Retrieval for MapReduce"**. In: *Financial Cryptography and Data Security (FC'13)*. Vol. 7859. LNCS. Springer, 2013, pp. 371–385.

[MBC14] T. MAYBERRY, E.-o. BLASS, A. H. CHAN. **"Efficient Private File Retrieval by Combining ORAM and PIR"**. In: *Network and Distributed System Security (NDSS'14)*. The Internet Society, 2014.

[MBGR03] Z. M. MAO, R. BUSH, T. GRIFFIN, M. ROUGHAN. **"BGP Beacons"**. In: *SIGCOMM'03*. ACM, 2003, pp. 1–14.

[MCA06] M. F. MOKBEL, C.-Y. CHOW, W. G. AREF. **"The New Casper: Query Processing for Location Services Without Compromising Privacy"**. In: *International Conference on Very Large Data Bases (VLDB'06)*. 2006, pp. 763–774.

[MDL+17] M. R. DI LALLO, G. LOSPOTO, H. MOSTAFAEI, M. RIMONDINI, G. DI BATTISTA. **"PrIXP: Preserving the Privacy of Routing Policies at Internet eXchange Points"**. In: *IFIP/IEEE International Symposium on Integrated Network Management, IM*. 2017.

[MF06] P. MOHASSEL, M. FRANKLIN. **"Efficiency Tradeoffs for Malicious Two-Party Computation"**. In: *PKC'06*. Springer, 2006, pp. 458–473.

[MG08] C. A. MELCHOR, P. GABORIT. **"A fast private information retrieval protocol"**. In: *IEEE International Symposium on Information Theory (ISIT'08)*. IEEE, 2008, pp. 1848–1852.

[MGC+16] B. MOOD, D. GUPTA, H. CARTER, K. R. B. BUTLER, P. TRAYNOR. **"Frigate: A Validated, Extensible, and Efficient Compiler and Interpreter for Secure Computation"**. In: *IEEE European Symposium on Security and Privacy (EuroS&P'16)*. IEEE, 2016, pp. 112–127.

[MK04a] S. MACHIRAJU, R. H. KATZ. **"Reconciling Cooperation with Confidentiality in Multi-Provider Distributed Systems"**. Tech. rep. UCB/CSD-04-1345. EECS Department, University of California, Berkeley, 2004.

[MK04b] S. MACHIRAJU, R. H. KATZ. **"Verifying Global Invariants in Multi-Provider Distributed Systems"**. In: *ACM Workshop on Hot Topics in Networks (HotNets*. 2004.

[MK06] S. MACHIRAJU, R. H. KATZ. **"Leveraging BGP Dynamics to Reverse-Engineer Routing Policies"**. Tech. rep. UCB/EECS-2006-61. EECS Department, University of California, Berkeley, 2006.

[MKA+09] H. V. MADHYASTHA, E. KATZ-BASSETT, T. ANDERSON, A. KRISHNAMURTHY, A. VENKATARAMANI. **"iPlane Nano: Path Prediction for Peer-to-peer Applications"**. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. USENIX, 2009, pp. 137–152.

[MLB12] B. MOOD, L. LETAW, K. BUTLER. **"Memory-Efficient Garbled Circuit Generation for Mobile Devices"**. In: *Financial Cryptography and Data Security (FC'12)*. Vol. 7397. LNCS. Springer, 2012, pp. 254–268.

[MMM16] J. MAYER, P. MUTCHLER, J. C. MITCHELL. **"Evaluating the privacy properties of telephone metadata"**. In: *National Academy of Sciences* 113.20 (2016), pp. 5536–5541.

[MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. **"Fairplay – A Secure Two-Party Computation System"**. In: *USENIX Security'04*. USENIX, 2004, pp. 287–302.

[MOR16] P. MOHASSEL, O. OROBETS, B. RIVA. **"Efficient Server-Aided 2PC for Mobile Phones"**. In: *Privacy Enhancing Technologies Symposium (PETS'16)* 2016.2 (2016), pp. 82–99.

[MOT+11] P. MITTAL, F. G. OLUMOFIN, C. TRONCOSO, N. BORISOV, I. GOLDBERG. **"PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval"**. In: *USENIX Security'11*. USENIX, 2011, pp. 31–31.

[MR18] P. MOHASSEL, P. RINDAL. **"*ABY*^{3}: A Mixed Protocol Framework for Machine Learning"**. In: *CCS'18*. ACM, 2018, pp. 35–52.

[MS13] P. MOHASSEL, S. S. SADEGHIAN. **"How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation"**. In: *EUROCRYPT'13*. Vol. 7881. LNCS. Springer, 2013, pp. 557–574.

[NIS12] NIST. **"NIST Special Publication 800-57, Recommendation for Key Management Part 1: General (Rev. 3)"**. National Institute of Standards and Technology (NIST). 2012.

[NIW+13] V. NIKOLAENKO, S. IOANNIDIS, U. WEINSBERG, M. JOYE, N. TAFT, D. BONEH. **"Privacy-preserving matrix factorization"**. In: *CCS'13*. ACM, 2013, pp. 801–812.

[NNOB12] J. B. NIELSEN, P. S. NORDHOLT, C. ORLANDI, S. S. BURRA. **"A New Approach to Practical Active-Secure Two-Party Computation"**. In: *CRYPTO'12*. Vol. 7417. LNCS. Springer, 2012, pp. 681–700.

[NP01] M. NAOR, B. PINKAS. **"Efficient oblivious transfer protocols"**. In: *Symposium on Discrete Algorithms (SODA'01)*. Society for Industrial and Applied Mathematics, 2001, pp. 448–457.

[NPS99] M. NAOR, B. PINKAS, R. SUMNER. **"Privacy Preserving Auctions and Mechanism Design"**. In: *Electronic Commerce (EC'99)*. ACM, 1999, pp. 129–139.

[NST17] J. B. NIELSEN, T. SCHNEIDER, R. TRIFILETTI. **"Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO"**. In: *24. Annual Network and Distributed System Security Symposium (NDSS'17)*. Internet Society, 2017.

[NWI+13] V. NIKOLAENKO, U. WEINSBERG, S. IOANNIDIS, M. JOYE, D. BONEH, N. TAFT. **"Privacy-Preserving Ridge Regression on Hundreds of Millions of Records"**. In: *Symposium on Security and Privacy (S&P'13)*. IEEE, 2013, pp. 334–348.

[OG11] F. G. OLUMOFIN, I. GOLDBERG. **"Revisiting the Computational Practicality of Private Information Retrieval"**. In: *Financial Cryptography and Data Security (FC'11)*. Vol. 7035. LNCS. Springer, 2011, pp. 158–172.

[OOS17] M. ORRÙ, E. ORSINI, P. SCHOLL. **"Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection"**. In: *Topics in Cryptology – CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*. Springer International Publishing, 2017, pp. 381–396.

[Ope14] OPEN WHISPER SYSTEMS. **"The Difficulty Of Private Contact Discovery"**. https://whispersystems.org/blog/contact-discovery/. 2014.

[OS07] R. OSTROVSKY, W. E. SKEITH, III. **"A Survey of Single-Database Private Information Retrieval: Techniques and Applications"**. In: *PKC'07*. Vol. 4450. LNCS. Springer, 2007, pp. 393–411.

[OZPZ09] R. OLIVEIRA, B. ZHANG, D. PEI, L. ZHANG. **"Quantifying Path Exploration in the Internet"**. In: *IEEE/ACM Transactions on Networking* 17.2 (2009), pp. 445–458.

[Pai99] P. PAILLIER. **"Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"**. In: *EUROCRYPT'99*. Vol. 1592. LNCS. Springer, 1999, pp. 223–238.

[PBS12] P. PULLONEN, D. BOGDANOV, T. SCHNEIDER. **"The design and implementation of a two-party protocol suite for SHAREMIND 3"**. Tech. rep. T-4-17. CYBERNETICA Institute of Information Security, 2012.

[PC17] A. M. PERILLO, E. D. CRISTOFARO. **"PAPEETE: Private, Authorized, and Fast Personal Genomic Testing"**. Tech. rep. 770. 2017.

[PGK88] D. A. PATTERSON, G. A. GIBSON, R. H. KATZ. **"A Case for Redundant Arrays of Inexpensive Disks (RAID)"**. In: *ACM International Conference on Management of Data (SIGMOD'88)*. ACM, 1988, pp. 109–116.

[PH78] S. C. POHLIG, M. E. HELLMAN. **"An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (Corresp.)"** In: *IEEE Transactions on Information Theory* 24.1 (1978), pp. 106–110.

[PHG+17] A. M. PIOTROWSKA, J. HAYES, N. GELERNTER, G. DANEZIS, A. HERZBERG. **"AnNotify: A Private Notification Service"**. In: *WPES'17*. ACM, 2017, pp. 5–15.

[PKUM16] E. PATTUK, M. KANTARCIOGLU, H. ULUSOY, B. MALIN. **"CheapSMC: A Framework to Minimize Secure Multiparty Computation Cost in the Cloud"**. In: *Data and Applications Security and Privacy (DBSec'16)*. Vol. 9766. LNCS. Springer, 2016, pp. 285–294.

[PMH09] P. PAPAGEORGE, J. MCCANN, M. HICKS. **"Passive Aggressive Measurement with MGRP"**. In: 39.4 (2009), pp. 279–290.

[PSSW09] B. PINKAS, T. SCHNEIDER, N. P. SMART, S. C. WILLIAMS. **"Secure Two-Party Computation is Practical"**. In: *ASIACRYPT'09*. Vol. 5912. LNCS. Full version: https://ia.cr/2009/314. Springer, 2009, pp. 250–267.

[PSSZ15] B. PINKAS, T. SCHNEIDER, G. SEGEV, M. ZOHNER. **"Phasing: Private Set Intersection Using Permutation-based Hashing"**. In: *USENIX Security'15*. USENIX, 2015, pp. 515–530.

[PSZ14] B. PINKAS, T. SCHNEIDER, M. ZOHNER. **"Faster Private Set Intersection based on OT Extension"**. In: *USENIX Security'14*. Full version: https://ia.cr/2014/447. Code: https://encrypto.de/code/PSI. USENIX, 2014, pp. 797–812.

[PSZ18] B. PINKAS, T. SCHNEIDER, M. ZOHNER. **"Scalable Private Set Intersection Based on OT Extension"**. In: *ACM Transactions on Privacy and Security (TOPS)* 21.2 (2018). Preliminary version: https://ia.cr/2016/930. Code: https://encrypto.de/code/JournalPSI, 7:1–7:35.

[PTH16] A. K. PAUL, A. TACHIBANA, T. HASEGAWA. **"An Enhanced Available Bandwidth Estimation Technique for an End-to-End Network Path"**. In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 768–781.

[Pul13] P. PULLONEN. **"Actively Secure Two-Party Computation: Efficient Beaver Triple Generation"**. MA thesis. University of Tartu, 2013.

[RA18] A. C. D. RESENDE, D. F. ARANHA. **"Faster Unbalanced Private Set Intersection"**. In: *Financial Cryptography and Data Security (FC'18)*. LNCS. Springer, 2018.

[Rab81] M. O. RABIN. **"How to exchange secrets with oblivious transfer"**. Tech. rep. Aiken Computation Lab, Harvard University, https://ia.cr/2005/187. 1981.

[Ram99] B. RAMSDELL. **"S/MIME Version 3 Message Specification"**. RFC 2633. RFC Editor, 1999.

[RB89] T. RABIN, M. BEN-OR. **"Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)"**. In: *STOC'89*. ACM, 1989, pp. 73–85.

[RHH14] A. RASTOGI, M. A. HAMMER, M. HICKS. **"Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations"**. In: *Symposium on Security and Privacy (S&P'14)*. IEEE, 2014, pp. 655–670.

[RR16] P. RINDAL, M. ROSULEK. **"Faster Malicious 2-Party Secure Computation with Online/Offline Dual Execution"**. In: *USENIX Security'16*. USENIX, 2016, pp. 297–314.

[RR17] P. RINDAL, M. ROSULEK. **"Improved Private Set Intersection Against Malicious Adversaries"**. In: *EUROCRYPT'17*. Vol. 10210. LNCS. Springer, 2017, pp. 235–259.

[RSF+14] P. RICHTER, G. SMARAGDAKIS, A. FELDMANN, N. CHATZIS, J. BOETTGER, W. WILLINGER. **"Peering at Peerings: On the Role of IXP Route Servers"**. In: *Internet Measurement Conference (IMC)*. 2014.

[RWM+11] M. ROUGHAN, W. WILLINGER, O. MAENNEL, D. PEROULI, R. BUSH. **"10 Lessons from 10 Years of Measuring and Modeling the Internet's Autonomous Systems"**. In: *IEEE Journal on Selected Areas in Communications* 29.9 (2011), pp. 1810–1821.

[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: *ASIACCS'18*. Preliminary version: https://ia.cr/2017/1164. ACM, 2018, pp. 707–721.

[RZ06] M. ROUGHAN, Y. ZHANG. **"Privacy-Preserving Performance Measurements"**. In: *Workshop on Mining Network Data (MineNet)*. ACM, 2006, pp. 329–334.

[Sav97] J. E. SAVAGE. **"Models of Computation: Exploring the Power of Computing"**. 1st. Addison-Wesley Pub, 1997.

[SB15] S. S. SHRINGARPURE, C. D. BUSTAMANTE. **"Privacy Risks from Genomic Data-Sharing Beacons"**. In: *The American Journal of Human Genetics* 97.5 (2015), pp. 631–646.

[SBS08] R. SHERWOOD, A. BENDER, N. SPRING. **"Discarte: a disjunctive internet cartographer"**. In: *SIGCOMM'08*. ACM, 2008.

[SC07] R. SION, B. CARBUNAR. **"On the Practicality of Private Information Retrieval"**. In: *Network and Distributed System Security (NDSS'07)*. The Internet Society, 2007.

[SCM05] L. SASSAMAN, B. COHEN, N. MATHEWSON. **"The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval"**. In: *Workshop on Privacy in the Electronic Society (WPES'05)*. ACM, 2005, pp. 1–9.

[SDS+13] E. STEFANOV, M. v. DIJK, E. SHI, C. W. FLETCHER, L. REN, X. YU, S. DEVADAS. **"Path ORAM: an extremely simple oblivious RAM protocol"**. In: *CCS'13*. ACM, 2013, pp. 299–310.

[SFLR13] J. STRINGER, Q. FU, C. LORIER, C. E. ROTHENBERG. **"Cardigan: Deploying a Distributed Routing Fabric"**. In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13)*. ACM, 2013, pp. 169–170.

[ŠG14] J. Šeděnka, P. Gasti. **"Privacy-preserving distance computation and proximity testing on earth, done right"**. In: *ASIACCS'14*. ACM, 2014, pp. 99–110.

[SHS+15] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, F. Koushanfar. **"Tiny-Garble: Highly Compressed and Scalable Sequential Garbled Circuits"**. In: *36. IEEE Symposium on Security and Privacy (IEEE S&P'15)*. IEEE, 2015, pp. 411–428.

[SK11] A. Schröpfer, F. Kerschbaum. **"Forecasting Run-Times of Secure Two-Party Computation"**. In: *Quantitative Evaluation of Systems (QEST'11)*. IEEE, 2011, pp. 181–190.

[SKM11] A. Schröpfer, F. Kerschbaum, G. Müller. **"L1 - An Intermediate Language for Mixed-Protocol Secure Computation"**. In: *IEEE Computer Software and Applications Conference (COMPSAC'11)*. IEEE, 2011, pp. 298–307.

[SLH+17] J. S. Sousa, C. Lefebvre, Z. Huang, J. L. Raisaro, C. Aguilar-Melchor, M.-O. Killijian, J.-P. Hubaux. **"Efficient and secure outsourcing of genomic data storage"**. In: *BMC Medical Genomics* 10.2 (2017), p. 46.

[Smi14] A. Smith. **"6 new facts about Facebook"**. Pew Research Center Fact Tank. http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/. 2014.

[SN16] A. Sanatinia, G. Noubir. **"HOnions: Towards Detection and Identification of Misbehaving Tor HSDirs"**. In: *Hot Topics in Privacy Enhancing Technologies Symposium (HotPETS'16)*. 2016.

[SOA+15] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, A. Vahdat. **"Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network"**. In: *Computer Communication Review* 45.5 (2015), pp. 183–197.

[SS13] A. Shelat, C.-H. Shen. **"Fast two-party secure computation with minimal assumptions"**. In: *CCS'13*. ACM, 2013, pp. 523–534.

[ST] N. Smart, S. Tillich. **"Circuits of Basic Functions Suitable For MPC and FHE"**. https://homes.esat.kuleuven.be/~nsmart/MPC/.

[Syn10] Synopsys Inc. **"Design Compiler"**. https://www.synopsys.com/support/training/rtl-synthesis/design-compiler-rtl-synthesis.html. 2010.

[Syn15] Synopsys Inc. **"DesignWare Library - Datapath and Building Block IP"**. https://www.synopsys.com/dw/buildingblock.php. 2015.

[SZ13] T. Schneider, M. Zohner. **"GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits"**. In: *17. International Conference on Financial Cryptography and Data Security (FC'13)*. Vol. 7859. LNCS. Springer, 2013, pp. 275–292.

[TG04] S. Tao, R. Guérin. **"On-line Estimation of Internet Path Performance: An Application Perspective"**. In: *IEEE INFOCOM*. 2004, pp. 1774–1785.

[TLP+16] S. Tamrakar, J. Liu, A. Paverd, J. Ekberg, B. Pinkas, N. Asokan. **"The Circle Game: Scalable Private Membership Test Using Trusted Hardware"**. In: *CoRR* abs/1606.01655 (2016).

[UN48] United Nations. **"Universal Declaration of Human Rights"**. United Nations, 1948.

[Vah17] A. Vahdat. **"Cloud Native Networking"**. 2017. URL: https://www.youtube.com/watch?v=1xBZ5DGZZmQ.

[Val76] L. G. VALIANT. **"Universal Circuits (Preliminary Report)"**. In: *8. ACM Symposium on Theory of Computing (STOC'76)*. ACM, 1976, pp. 196–203.

[VC03] J. VAIDYA, C. CLIFTON. **"Privacy-preserving *k*-means clustering over vertically partitioned data"**. In: *SIGKDD'03*. 2003, pp. 206–215.

[Wak68] A. WAKSMAN. **"A Permutation Network"**. In: *Journal of the ACM* 15.1 (1968), pp. 159–163.

[WDDB06] S. WANG, X. DING, R. H. DENG, F. BAO. **"Private Information Retrieval Using Trusted Hardware"**. In: *European Symposium on Research in Computer Security (ESORICS'06)*. Vol. 4189. LNCS. Springer, 2006, pp. 49–64.

[WG03] F. WANG, L. GAO. **"On Inferring and Characterizing Internet Routing Policies"**. In: *Internet Measurement Conference (IMC)*. ACM, 2003, pp. 15–26.

[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: *CCS'15*. ACM, 2015, pp. 492–503.

[WLL+14] H. WANG, K. S. LEE, E. LI, C. L. LIM, A. TANG, H. WEATHERSPOON. **"Timing is Everything: Accurate, Minimum Overhead, Available Bandwidth Estimation in High-speed Wired Networks"**. In: *Internet Measurement Conference (IMC)*. 2014, pp. 407–420.

[Wol] C. WOLF. **"Yosys Open SYnthesis Suite"**. http://www.clifford.at/yosys/.

[WR04] X. WANG, M. K. REITER. **"Mitigating Bandwidth-Exhaustion Attacks Using Congestion Puzzles"**. In: *Computer and Communications Security (CCS)*. 2004, pp. 257–267.

[WRCS16] D. WALTON, A. RETANA, E. CHEN, J. SCUDDER. **"Advertisement of Multiple Paths in BGP"**. RFC 7911 (Proposed Standard). RFC Editor, 2016.

[WRK17] X. WANG, S. RANELLUCCI, J. KATZ. **"Global-Scale Secure Multiparty Computation"**. In: *CCS'17*. ACM, 2017, pp. 39–56.

[XYC09] C. XING, L. YANG, M. CHEN. **"Estimating Internet Path Properties for Distributed Applications"**. In: *WiCOM*. 2009, pp. 4179–4182.

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets"**. In: *FOCS'86*. IEEE, 1986, pp. 162–167.

[YDDB08] Y. YANG, X. DING, R. H. DENG, F. BAO. **"An Efficient PIR Construction Using Trusted Hardware"**. In: *Information Security Conference (ISC'08)*. Vol. 5222. LNCS. Springer, 2008, pp. 64–79.

[ZCD16] J. ZHOU, Z. CAO, X. DONG. **"PPOPM: More Efficient Privacy Preserving Outsourced Pattern Matching"**. In: *21. European Symposium on Research in Computer Security (ESORICS'16)*. Vol. 9878. LNCS. Springer, 2016, pp. 135–153.

[ZE15] S. ZAHUR, D. EVANS. **"Obliv-C: A Language for Extensible Data-Oblivious Computation"**. Cryptology ePrint Archive, Report 2015/1153. https://ia.cr/2015/1153. 2015.

[ZHS07] F. ZHAO, Y. HORI, K. SAKURAI. **"Two-servers PIR based DNS query scheme with privacy-preserving"**. In: *Intelligent Pervasive Computing, 2007. (IPC'07)*. IEEE. 2007, pp. 299–302.

[ZMZ04] B. ZHANG, D. MASSEY, L. ZHANG. **"Destination reachability and BGP convergence time [border gateway routing protocol]"**. In: *GLOBECOM'04*. Vol. 3. IEEE, 2004, pp. 1383–1389.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates"**. In: *EUROCRYPT'15*. Vol. 9057. LNCS. Springer, 2015, pp. 220–250.

[ZS14] L. F. ZHANG, R. SAFAVI-NAINI. **"Verifiable Multi-server Private Information Retrieval"**. In: *Applied Cryptography and Network Security (ACNS'14)*. Vol. 8479. LNCS. Springer, 2014, pp. 62–79.

[ZSB13] Y. ZHANG, A. STEELE, M. BLANTON. **"PICCO: a general-purpose compiler for private distributed computation"**. In: *CCS'13*. ACM, 2013, pp. 813–826.

[ZZG+16] M. ZHAO, W. ZHOU, A. J. T. GURNEY, A. HAEBERLEN, M. SHERR, B. T. LOO. **"Private and Verifiable Interdomain Routing Decisions"**. In: *IEEE/ACM Transactions on Networking* 24.2 (2016), pp. 1011–1024.

List of Figures

List of Tables

List of Protocols

List of Algorithms

List of Circuits

List of Abbreviations

AEAD Authenticated Encryption with Associated Data
AS Autonomous System
ASIC Application Specific Integrated Circuit
BGP Border Gateway Protocol
CDN Content Delivery Network
DAG Directed Acyclic Graph
DPF Distributed Point Function
DSL Domain Specific Language
FPGA Field Programmable Gate Array
GDPR General Data Protection Regulation
HDL Hardware Definition Language
HE Homomorphic Encryption
IXP Internet Exchange Point
LUT Look-up Table
MPC Secure Multi-Party Computation
ORAM Oblivious Random Access Memory
OT Oblivious Transfer
PAL Programmable Array Logic
PEQ Private Equality Test
PIR Private Information Retrieval
PFE Private Function Evaluation
PRG Pseudo-Random Generator
PSI Private Set Intersection
RAID Redundant Array of Inexpensive Disks
RPKI Resource Public Key Infrastructure
SDN Software Defined Networking
SIMD Single Instruction Multiple Data
SGX Software Guard Extensions
VCF Variant Call Format
VQF Variant Query Format

List of Own Publications

Peer-reviewed Conference and Workshop Publications

[ADS+17] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, M. Zohner. **"Privacy-Preserving Interdomain Routing at Internet Scale"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.3 (2017). Full version: https://ia.cr/2017/393, pp. 143–163. CORE Rank B. **Part of this thesis**.

[BDK+18] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, T. Schneider. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *25. ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018, pp. 847–861. CORE Rank A*. **Part of this thesis**.

[CDC+16] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"Towards Securing Internet eXchange Points Against Curious onlooKers (Short Paper)"**. In: *1. ACM, IRTF & ISOC Applied Networking Research Workshop (ANRW'16)*. ACM, 2016, pp. 32–34. **Part of this thesis**.

[CDC+17] M. Chiesa, D. Demmler, M. Canini, M. Schapira, T. Schneider. **"SIXPACK: Securing Internet eXchange Points Against Curious onlooKers"**. In: *13. International Conference on emerging Networking EXperiments and Technologies (CoNEXT'17)*. ACM, 2017, pp. 120–133. CORE Rank A. **Part of this thesis**.

[DDK+15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni. **"Automated Synthesis of Optimized Circuits for Secure Computation"**. In: *22. ACM Conference on Computer and Communications Security (CCS'15)*. ACM, 2015, pp. 1504–1517. CORE Rank A*. **Part of this thesis**.

[DHS14] D. Demmler, A. Herzberg, T. Schneider. **"RAID-PIR: Practical Multi-Server PIR"**. In: *6. ACM Cloud Computing Security Workshop (CCSW'14)*. Code: https://encrypto.de/code/RAID-PIR. ACM, 2014, pp. 45–56. **Part of this thesis**.

[DHS17] D. Demmler, M. Holz, T. Schneider. **"OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks"**. In: *15. International Conference on Applied Cryptography and Network Security (ACNS'17)*. Vol. 10355. LNCS. Code: https://encrypto.de/code/onionPIR. Springer, 2017, pp. 599–619. CORE Rank B. **Part of this thesis**.

[DHSS17] D. DEMMLER, K. HAMACHER, T. SCHNEIDER, S. STAMMLER. **"Privacy-Preserving Whole-Genome Variant Queries"**. In: *16. International Conference on Cryptology And Network Security (CANS'17)*. Vol. 11261. LNCS. Springer, 2017, pp. 71–92. CORE Rank B. **Part of this thesis**.

[DRRT18] D. DEMMLER, P. RINDAL, M. ROSULEK, N. TRIEU. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). Code: https://github.com/osu-crypto/libPSI. CORE Rank B. **Part of this thesis**.

[DSZ14] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens"**. In: *23. USENIX Security Symposium (USENIX Security'14)*. Full version: https://ia.cr/2014/467. USENIX, 2014, pp. 893–908. CORE Rank A*.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. Code: https://encrypto.de/code/ABY. Internet Society, 2015. CORE Rank A*. **Part of this thesis**.

[KNR+17] S. KRÜGER, S. NADI, M. REIF, K. ALI, M. MEZINI, E. BODDEN, F. GÖPFERT, F. GÜNTHER, C. WEINERT, D. DEMMLER, R. KAMATH. **"CogniCrypt: Supporting Developers in Using Cryptography"**. In: *International Conference on Automated Software Engineering (ASE)*. ASE 2017. IEEE Press, 2017, pp. 931–936.