Faster Oblivious Transfer Extension and Its Impact on Secure Computation

Vom Fachbereich Informatik der
Technischen Universität Darmstadt
genehmigte

Dissertation

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

M. Sc. Michael Zohner

geboren in Hünfeld, Deutschland

Referenten:	Dr. Thomas Schneider Technische Universität Darmstadt
	Prof. Dr. Benny Pinkas Bar-Ilan University
	Prof. Dr. Stefan Katzenbeisser Technische Universität Darmstadt
Tag der Einreichung:	28.10.2016
Tag der mdl. Prüfung:	09.12.2016
Hochschulkennziffer:	D17
Darmstadt,	2017

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 28. Oktober 2016 _____

Michael Zohner

Wissenschaftlicher Werdegang

Juli 2011 - Dezember 2016
Doktorand der Informatik an der Technischen Universität Darmstadt.

Oktober 2008 - Juni 2011
Studium der Informatik an der Technischen Universität Darmstadt.
Abschluss als Master of Science.

Oktober 2005 - August 2008
Studium der Angewandten Informatik an der Hochschule Fulda.
Abschluss als Bachelor of Science.

Summary

Secure two-party computation allows two parties to evaluate a function on their private inputs while keeping all information private except what can be inferred from the outputs. A major building block and the foundation for many efficient secure computation protocols is *oblivious transfer (OT)*. In an OT protocol a sender inputs two messages (x_0, x_1) while a receiver with choice bit c wants to receive message x_c. The OT protocol execution guarantees that the sender learns no information about c and the receiver learns no information about x_{1-c}.

This thesis focuses on the efficient generation of OTs and their use in secure computation. In particular, we show how to compute OTs more efficiently, improve *generic secure computation* protocols which can be used to securely evaluate any functionality, and develop highly efficient special-purpose protocols for *private set intersection (PSI)*. We outline our contributions in more detail next.

More Efficient OT Extensions. The most efficient OT protocols are based on public-key cryptography and require a constant number of exponentiations per OT. However, for many practical applications where millions to billions of OTs need to be computed, these exponentiations become prohibitively slow. To enable these applications, *OT extension* protocols [Bea96, IKNP03] can be used, which extend a small number of public-key-based OTs to an arbitrarily large number using cheap symmetric-key cryptography only.

We improve the computation and communication efficiency of OT extension protocols and show how to achieve security against malicious adversaries, which can arbitrarily deviate from the protocol, at low overhead. Our resulting protocols can compute several million of OTs per second and we show that, in contrast to previous belief, the local computation overhead for computing OTs is so low that the main bottleneck is the network bandwidth.

Parts of these results are published in:

- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *20th ACM Conference on Computer and Communications Security (CCS'13)*.
- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *34th Advances in Cryptology – EUROCRYPT'15*.
- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer Extensions. To appear in Journal of Cryptology. Online at http://eprint.iacr.org/2016/602.

Communication-Efficient Generic Secure Two-Party Computation. *Generic secure two-party computation techniques* allow to evaluate a function, represented as a circuit of linear (XOR) and non-linear (AND) gates. One of the most prominent generic secure two-party computation protocols is *Yao's garbled circuits* [Yao86], for which many optimizations have been proposed. Shortly after Yao's protocol, the generic secure protocol by *Goldreich-Micali-Wigderson (GMW)* [GMW87] was introduced. The GMW protocol requires a large number of OTs and was believed to be less efficient for secure two-party computation than Yao's protocol [HL10, CHK+12].

We improve the efficiency of the GMW protocol and show that it can outperform Yao's garbled circuits protocol in settings with low bandwidth. Furthermore, we utilize the flexibility of OT and outline special-purpose constructions that can be used within the GMW protocol and which improve its efficiency even further.
Parts of these results are published in:

- T. Schneider, M. Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In *17th International Conference on Financial Cryptography and Data Security (FC'13).*
- D. Demmler, T. Schneider, M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22th Network and Distributed System Security Symposium (NDSS'15).*
- G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, M. Zohner. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *24th Network and Distributed System Security Symposium (NDSS'17).*

Faster Private Set Intersection (PSI). PSI allows two parties to compute the intersection of their private sets without revealing any element that is not in the intersection. PSI is a well-studied problem in secure computation and many special-purpose protocols have been proposed. However, existing PSI protocols are several orders of magnitude slower than an insecure naive hashing solution that is used in practice. In addition, many protocols were compared in a biased fashion, which makes it difficult to identify the most promising solution for a particular scenario.

We systematize the progress made on PSI protocols by reviewing, optimizing, and comparing existing PSI protocols. We then introduce a novel PSI protocol that is based on our efficiency improvements in OT extension protocols and which outperforms existing protocols by up to two orders of magnitude.
Parts of these results are published in:

- B. Pinkas, T. Schneider, M. Zohner. Faster Private Set Intersection Based on OT Extension. In *23th USENIX Security Symposium (USENIX Security'14).*
- B. Pinkas, T. Schneider, G. Segev, M. Zohner. Phasing: Private Set Intersection using Permutation-based Hashing. In *24th USENIX Security Symposium (USENIX Security'15).*
- B. Pinkas, T. Schneider, M. Zohner. Scalable Private Set Intersection Based on OT Extension. Journal paper. In submission. Online at `http://iacr.eprint.org/2016/930`.

Zusammenfassung

Sichere Zweiparteienberechnung erlaubt es zwei Parteien eine Funktion auf ihren privaten Eingabedaten zu evaluieren ohne dass Informationen über die Eingabedaten, die nicht von der Ausgabe ableitbar sind, preisgegeben werden. Ein wichtiger Baustein und die Grundlage für viele sichere Berechnungsprotokolle ist *Oblivious Transfer (OT)*. In einem OT Protokoll interagieren ein Sender mit Nachrichten (x_0, x_1) und ein Empfänger mit Auswahlbit c, sodass der Empfänger die Nachricht x_c empfängt ohne dass der Sender Informationen über c oder der Empfänger Informationen über x_{1-c} lernt.

Diese Dissertation befasst sich mit der effizienten Generierung von OTs und deren Einsatz im Gebiet der sicheren Zweiparteienberechnung. Wir zeigen wie OTs effizienter generiert werden können, verbessern *generische sichere Berechnungsprotokolle*, die dazu genutzt werden können um jede Funktion sicher zu berechnen, und entwickeln hoch effiziente sichere Berechnungsprotokolle für *private Schnittmengenberechnung (PSI)*. Im Folgenden beschreiben wir unseren wissenschaftlichen Beitrag detaillierter.

Effizientere OT Extension Protokolle. Die meisten effizienten OT Protokolle basieren auf asymmetrischer Verschlüsselung und benötigen eine konstante Anzahl von Exponentiationen pro OT. Für praktische Anwendungen, in denen Millionen bis Milliarden von OTs benötigt werden, ist diese hohe Anzahl an Exponentiationen allerdings untragbar. Um solche Anwendungen trotzdem zu realisieren, wurden *OT Extension* Protokolle [Bea96, IKNP03] eingeführt, die eine kleine Anzahl von OTs aus Protokollen die auf asymmetrischer Verschlüsselung beruhen mittels effizienter symmetrischer Verschlüsselung auf eine beliebige Anzahl von OTs ausdehnen.

Wir verbessern die Berechnungs- und Kommunikationseffizienz von OT Extension Protokollen und zeigen auf wie Sicherheit gegen stärkere, aktive Angreifer, welche beliebig von der Protokollausführung abweichen können, mittels geringem Mehraufwand erreicht werden kann. Unsere Protokolle ermöglichen die Generierung von mehreren Millionen OTs pro Sekunde, wobei die Netzwerkkommunikation den Flaschenhals darstellt anstatt, wie allgemein vermutet, die Berechnungskomplexität.

Teile dieser Ergebnisse wurden publiziert in:

- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *20^{ter} ACM Konferenz zu Computer and Communications Security (CCS'13)*.
- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *34^{ter} Advances in Cryptology – EUROCRYPT'15*.
- G. Asharov, Y. Lindell, T. Schneider, M. Zohner. More Efficient Oblivious Transfer Extensions. Erscheint in Journal of Cryptology. Online auf `http://eprint.iacr.org/2016/602`.

Kommunikationseffiziente, Generische, Sichere Zweiparteienberechnung. *Protokolle zur generischen sicheren Zweiparteienberechnung* erlauben es eine Funktion zu evaluieren, die als Schaltkreis aus linearen (XOR) und nichtlinearen (UND) Gattern dargestellt ist. Eines der prominentesten Protokolle in diesem Umfeld ist *Yao's Garbled Circuits* [Yao86], für welches viele Optimierungen vorgeschlagen wurden. Kurz nach Yao's Protokoll wurde das generische, sichere Protokoll von *Goldreich-Micali-Wigderson (GMW)* [GMW87] vorgeschlagen, welches aber aufgrund seiner hohen Abhängigkeit von OT als weniger effizient für generisch, sichere Zweiparteienberechnung als Yao's Protokoll angesehen wurde [HL10, CHK+12].

Wir verbessern die Effizienz des GMW Protokolls und zeigen, dass es besser in Szenarien mit niedriger Bandbreite abschneidet als Yao's Protokoll. Zudem nutzen wir die Flexibilität von OT um Protokolle für spezielle Funktionen zu entwickeln, die mit dem GMW Protokoll kombinierbar sind und dessen Effizienz weiter steigern.
Teile dieser Ergebnisse wurden publiziert in:

- T. Schneider, M. Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In *17ter Internationaler Konferenz zu Financial Cryptography and Data Security (FC'13).*
- D. Demmler, T. Schneider, M. Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22ter Network and Distributed System Security Symposium (NDSS'15).*
- G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, M. Zohner. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *24ter Network and Distributed System Security Symposium (NDSS'17).*

Schnellere Private Schnittmengenberechnung (PSI). Mittels PSI können zwei Parteien die Schnittmenge ihrer privaten Mengen berechnen ohne Elemente preiszugeben, die nicht in der Schnittmenge sind. Obwohl zahlreiche PSI Protokolle vorgeschlagen wurden, sind alle derzeit existierenden Protokolle mehrere Größenordnungen langsamer als eine unsichere Lösung, welche in der Praxis eingesetzt wird. Zudem wurden viele Protokolle unter ungleichen Bedingungen verglichen, was die Identifizierung des vielversprechendsten Protokolls für ein bestimmtes Szenario erschwert.

Wir systematisieren existierende PSI Protokolle indem wir einen Überblick über existierende Protokolle geben und diese dann optimieren und vergleichen. Zudem entwickeln wir ein neues PSI Protokoll, das auf unseren Verbesserungen im Bereich der OT Extension Protokolle basiert und die Leistung bestehender PSI Protokolle um zwei Größenordnungen übertrifft.
Teile dieser Ergebnisse wurden publiziert in:

- B. Pinkas, T. Schneider, M. Zohner. Faster Private Set Intersection Based on OT Extension. In *23ter USENIX Security Symposium (USENIX Security'14).*
- B. Pinkas, T. Schneider, G. Segev, M. Zohner. Phasing: Private Set Intersection using Permutation-based Hashing. In *24ter USENIX Security Symposium (USENIX Security'15).*
- B. Pinkas, T. Schneider, M. Zohner. Scalable Private Set Intersection Based on OT Extension. Journal Einreichung. Im Bewertungsprozess. Online auf http://iacr.eprint.org/2016/930.

Acknowledgements

The road that has lead to the completion of this thesis has been long and would have been impossible without the help and support of many people. First and foremost, I am grateful for my supervisor Thomas Schneider, who has always been open for discussion and relentless in pointing out shortcomings. I would also like to thank Benny Pinkas for many exciting and fruitful discussions on various topics of secure computation as well as Stefan Katzenbeisser for agreeing to review this thesis.

Before joining Thomas' group I spent a year as a Ph.D. candidate within the CASCADE group together with Annelie Heuser, Michael Kasper, and Marc Stöttinger. Even though I worked on a different topic at that time, this year has been a great experience and I am very grateful for the opportunity of working in such a diverse and highly motivated team.

Special thanks goes to all the people who have accompanied me through all these years: to Kevin Falzon for always reminding me that there is somebody who is worse off, to Andreas Follner for his constant praise of David Guetta, and to Tommaso Gagliardoni for introducing me to the fantastic flavor of pineapple pizza. An extended and equal thanks goes to my colleagues within the ENCRYPTO group: Daniel Demmler, Ágnes Kiss, and Christian Weinert, our student assistants: Oleksandr Tkachenko and Sreeram Sadasivam, and the people behind the scenes whom I could always turn to if I had organizational questions: Karina Köhres, Michael Kreutzer, Heike Meissner, and Cornelia Reitz.

I am very fortunate to have worked within the secure computation community, which has always been helpful, friendly, and open for discussion. Special thanks for in-depth discussions and challenging my understanding of the subject goes to Gilad Asharov, Yehuda Lindell, Claudio Orlandi, and Roberto Trifiletti. I thank all my coauthors without whom this work would have not been possible: Martin Albrecht, Julien Bringer, Hervé Chabanne, Ghada Dessouky, Mélanie Favre, Farinaz Koushanfar, Alain Patey, Christian Rechberger, Ahmad-Reza Sadeghi, Michael Schapira, Gil Segev, Scott Shenker, Tyge Tiessen, and Shaza Zeitouni. Although I have never met him personally, I would also like to thank Seung Geol Choi for open-sourcing his implementations, which have been the starting point of this thesis.

I would like to thank my family and all my friends who have supported me throughout these years. Finally, I am most grateful for my soon-to-be wife Ann-Christine Krieger for her unending patience and support during good and bad times.

Short Contents

Contents

1 Introduction

In the setting of secure two-party computation, two parties P_0 and P_1 with respective inputs x and y wish to compute a joint function f on their inputs without revealing anything but the output $f(x, y)$. This captures a large variety of tasks, including privacy-preserving data mining, anonymous transactions, private database search, and many more. Protocols for secure computation can be divided in two categories: *Generic secure computation protocols* and *special-purpose secure computation protocols*. Generic secure computation protocols, such as *Yao's garbled circuits* [Yao86] or the protocol by Goldreich-Micali-Wigderson (*GMW*) [GMW87], are powerful techniques that can be used to securely evaluate any function, represented as Boolean circuit. In contrast, special-purpose protocols are tailored to one particular functionality but often achieve a much higher efficiency than generic secure computation protocols.

Until the last decade [MNPS04], the bulk of research on secure computation was mostly theoretical. Many held the opinion that secure computation will never be practical since carrying out cryptographic operations for every Boolean gate in a circuit will never be fast enough to be of use. Due to many works that pushed secure computation further towards practical applications, this conjecture has been proven wrong and it is possible to carry out secure computation at speeds that ten years ago would have been inconceivable. In particular, protocols that provide security against *semi-honest* (or *passive*) adversaries, which honestly follow the protocol but try to learn as much information as possible from the execution, process several million Boolean gates per second [LWN+15]. Protocols that provide security against stronger *malicious* (or *active*) adversaries which can arbitrarily deviate from the protocol execution, process several hundred thousand Boolean gates per second [LR15b, RR16].

A fundamental primitive in secure computation and the focus of this thesis is *oblivious transfer (OT)*. In an OT, a sender P_S holds two input messages (x_0, x_1) of which a receiver P_R with choice bit $c \in \{0, 1\}$ wants to obliviously obtain x_c such that P_S learns no information about c and P_R learns no information about x_{1-c}. OT is a major building block for many secure computation protocols. However, in [IR88, IR89] it was shown that OT requires public key cryptography, which limits its efficiency. *OT extension* protocols [Bea96, IKNP03] greatly improve the efficiency of OT by extending a small number of public-key-based OTs, called *base-OTs*, to an arbitrary number of OTs using cheap symmetric-key cryptography only. However, even though OT extension protocols greatly improve the efficiency of OT, their potential was underestimated and protocols that were based on OT were believed to be less efficient than protocols based on other techniques (e.g., OT-based GMW vs. Yao's garbled circuits in [CHK+12]).

1.1 Outline and Scope

This thesis analyzes the following research question:

Can OT extension enable more efficient semi-honest secure two-party computation?

To answer this research question, we derive three sub-questions that form the scope and methodology of this work:

- **§3: How efficient can OT extension protocols become?**
  Prior to this thesis, the most efficient OT extension protocol was due to [IKNP03]. Although the [IKNP03] protocol already required only symmetric operations and few communication per OT, it was often overlooked as a building block for efficient protocols [HL10] or practical implementations that used it were found to be less efficient than related techniques [CHK+12]. We revisit the OT extension protocol of [IKNP03] and perform a detailed protocol and algorithmic analysis to identify and resolve bottlenecks. We then focus on features of protocols that use OT as a building block and analyze how to improve OT extension protocols based on these features.

- **§4: Can OT extension enable more efficient semi-honest generic secure two-party computation?**
  Most efficient generic secure two-party computation implementations focused on Yao's garbled circuits [Yao86]. Implementations of the GMW protocol [GMW87], which heavily relies on OT, were believed to be less efficient in the two-party case [CHK+12] due to higher computation, communication, and round complexity. Using the results from the efficiency analysis of OT extension, we re-evaluate the efficiency of the GMW protocol. We then analyze further possibilities how OT can improve the performance of generic secure two-party computation.

- **§5: Can OT extension enable more efficient semi-honest secure special-purpose protocols?**
  One of the most prominent secure computation functionalities is private set intersection (PSI), which allows two parties to compute the intersection of their private sets without revealing elements that are not in the intersection. Various special-purpose PSI protocols have been proposed, making it difficult to determine a suitable protocol for a particular setting. Furthermore, existing protocols add several orders of magnitude overhead over insecure solutions, used in practice. We systematize the knowledge in the area of PSI by reviewing, categorizing, optimizing, and evaluating existing PSI protocols. We then utilize our insights in OT extension to further improve the efficiency of PSI.

We give the necessary background for this work in §2 and conclude this thesis and outline directions for future work in §6.

1.2 Summary of Our Results

Our results show that OT extension can indeed enable more efficient secure computation. In particular, we improve the run-time of OT extension protocols by at least factor 2×. Using our efficient OT extension protocols, we show that OT-based generic secure computation techniques achieve better communication than the state-of-the-art Yao's garbled circuits protocol. We show that our OT-based special-purpose PSI protocol achieves two orders of magnitude better run-time than existing solutions and is only slightly slower than insecure solutions, used in practice. More detailed, we obtain the following results:

Faster OT Extension (§3).
We improve the OT extension protocol of [IKNP03] by factor 3× in computation and 2× in communication. For $\kappa = 128$-bit symmetric security, our most efficient protocol generates OTs at a rate of up to 7 million per second and thread where each OT requires 128 bit of communication. Furthermore, we improve the OT extension protocol of [KK13], which generates OTs on single bits at a rate of up to 2.5 million per second and thread where each OT requires 73 bit of communication whereas the previous most efficient [KK13] instantiation generated OTs at a rate of 0.6 million per second and thread with 80 bit of communication per OT. We then show how to achieve malicious security for the [IKNP03] OT extension protocol at a cost overhead of 150% compared to the semi-honest version.

Communication-Efficient Generic Secure Two-Party Computation (§4).
We achieve more communication-efficient secure two-party computation. In particular, we show that each party in the GMW protocol each party has to send the same data as in Yao's garbled circuits protocol using our improved OT extension protocol of [IKNP03] and only half of the data using our improved OT extension protocol of [KK13]. We then design special-purpose protocols that are based on OT and which further reduce the communication for certain functions by factor 1.5× to 30×. Finally, we outline a framework that combines our special-purpose protocols with generic secure computation techniques and allows us to combine the efficiency of special-purpose protocols with the generality of generic secure computation protocols.

Faster Private Set Intersection (§5).
We review, categorize, and optimize existing works on PSI and perform an extensive performance evaluation. Our optimized protocols outperform their original versions by factor 2× to 5×. We then introduce a novel OT-based PSI protocol that utilizes our efficient OT extension protocols and improves the run-time of existing protocols by two orders of magnitude. Finally, we identify practical applications of PSI and compare the performance of our OT-based PSI protocol with state-of-the-art insecure solutions. Overall, our secure solutions reduce the overhead over insecure solutions in practice to factor 8× and, in settings with unequal set sizes, even achieve the same performance.

2 Background

In this chapter, we present the necessary background for this thesis. We first give our definitions (§2.1) and outline cryptographic primitives (§2.2). Then, we review oblivious transfer (§2.3) and generic secure computation techniques (§2.4). Finally, we discuss hashing elements to smaller domains (§2.5).

Remark. Parts of this chapter have been published in [SZ13, ALSZ16, PSZ16].

2.1 Definitions

In the following, we give our notation (§2.1.1), review the Boolean circuit representation (§2.1.2), present our security parameters (§2.1.3), outline our adversary definitions (§2.1.4), and discuss the pre-computation model (§2.1.5).

2.1.1 Notation

We denote the parties as P_0 and P_1 or sender P_S and receiver P_R. We write $b[i]$ for the i-th element of a list b, denote the bitwise-AND between two bit strings a and b of equal length as $a \wedge b$ and the bitwise-XOR as $a \oplus b$. We denote a constant string of m zeros (or ones) as 0^m (or 1^m). For the PSI protocols, we denote the input sets of P_0 and P_1 as X and Y with $|X| = n_0$ and $|Y| = n_1$. We refer to elements from X as x and elements from Y as y and each element has bit-length σ (cf. §2.5 for the relation between n_0, n_1, and σ). We denote m 1-out-of-N OT executions on strings with n bit as $\binom{N}{1}\text{OT}_n^m$. We often abbreviate the output of protocols as (x, y), where P_0 obtains x as output while P_1 obtains y as output.

2.1.2 Circuit Representations and Evaluation Metrics

In this thesis, we discuss protocols that securely evaluate functions represented as arithmetic circuit, which consists of addition and multiplication gates, or a Boolean circuit, which consists of XOR and AND gates. Motivated by the fact that we mostly focus on secure computation protocols that operate on Boolean circuits and provide free XORs, we consider the *(multiplicative) size* $\mathbf{S}(\text{C})$ of a Boolean circuit C as the number of AND gates in C and the *(multiplicative) depth* $\mathbf{D}(\text{C})$ as the maximum number of AND gates on any path from any input to any output of C. An example circuit C_{ex} with $\mathbf{S}(\text{C}_{ex}) = 3$ and $\mathbf{D}(\text{C}_{ex}) = 2$ can be seen in Figure 2.1. For later

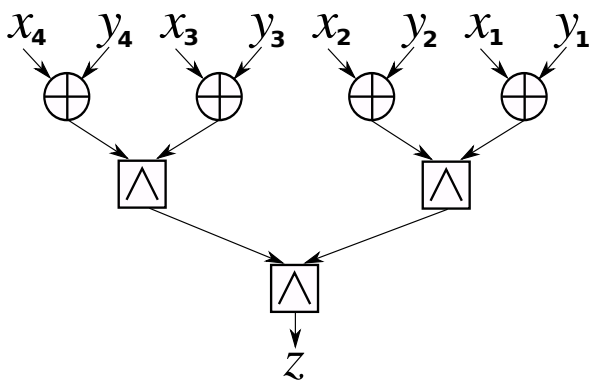


Figure 2.1: Example circuit C_{ex} with $\mathbf{S}(\mathrm{C}_{ex}) = 3$ and $\mathbf{D}(\mathrm{C}_{ex}) = 2$.

analyses, we bound the size $\mathbf{S}(\mathrm{C})$ and depth $\mathbf{D}(\mathrm{C})$ of a Boolean circuit C. For many functionalities, a low size and a low depth are two mutually exclusive goals. Hence, we first outline the case for Boolean circuits with $\ell \geq 1$ input bits and $o = 1$ output bit, since this allows us to set tighter upper bounds, and then examine the case for $o > 1$.

Boolean Circuits with One Output Bit. It was shown in [TP14] that any functionality with $\ell \leq 5$ input bits can be realized by a Boolean circuit with at most $\ell - 1$ AND gates. For functions with $\ell > 5$ inputs, a lower bound on the maximum number of AND gates is still unknown but, according to [TP14], "no specific ℓ-variable function has yet been proven to have multiplicative complexity larger than $\ell - 1$ for any ℓ". We bound the number of AND gates in a Boolean circuit C with ℓ inputs by $\mathbf{S}(\mathrm{C}) \leq \ell - 1$. In [BB94] it was shown that every Boolean circuit of multiplicative size n has an equivalent Boolean circuit of multiplicative depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n^{\alpha})$ for arbitrary $\alpha > 1$. We bound the multiplicative depth of a circuit C with ℓ inputs by $\mathbf{D}(\mathrm{C}) \leq \log_2(\ell)$.

Boolean Circuits with Multiple Output Bits. Finding a size- or depth-optimal Boolean circuit for functionalities with $o > 1$ outputs is a hard problem for a larger number of inputs ℓ [BP12] and determining a minimal upper bound is a complex task out of scope of this thesis. A more tractable approach to find a possible upper bound is to build optimal Boolean circuits for each output bit separately. In this thesis, we take this approach and assume that a Boolean circuit C with ℓ input and o output bits has at most size $\mathbf{S}(\mathrm{C}) \leq o(\ell - 1)$ if optimized for size and $\mathbf{D}(\mathrm{C}) \leq \log_2 \ell$ if optimized for depth.

2.1.3 Security Parameters

Our protocols use the following security parameters, summarized in Table 2.1: A *computational (symmetric)* security parameter κ, a *statistical* security parameter λ, a *prime* of bit-length ψ, an *elliptic curve* of field size ξ, and a *hashing failure* parameter ϕ.

Description	Parameter	Value
Symmetric security	κ	128
Statistical security	λ	40
Integer factorization prime bit length	ψ	3 072
Elliptic curve bit length	ξ	283
Hashing failure (affects correctness of PSI protocols)	ϕ	40

Table 2.1: Security parameters and values, used throughout this thesis.

We fix the security parameters to achieve security beyond 2030.[1], i.e., set $\kappa = 128$, $\lambda = 40$, $\psi = 3\ 072$, $\xi = 283$, and $\phi = 40$. The hashing failure parameter ϕ is used for some PSI protocols in §5 and specifies that the protocols yield the correct output except with probability $2^{-\phi}$. For ECC we use the Koblitz curve K-283, which had the best performance in our experiments (cf. [EFLL12]).

2.1.4 Adversaries

Protocols for secure computation provide security in the presence of adversarial behavior. A number of adversary models have been considered in the literature (cf. [HL10] for more details). The most common adversaries are: *Passive* or *semi-honest adversaries* who follow the protocol specification but attempt to learn more than allowed by inspecting the protocol transcript, and *active* or *malicious adversaries* who run any arbitrary strategy in an attempt to break the protocol. In both these cases, the security of a protocol guarantees that nothing is learned by an adversary beyond its legitimate output. Another notion is that of security in the presence of *covert adversaries*; in this case the adversary may follow any arbitrary strategy, but is guaranteed to be caught with good probability if it attempts to cheat. The ultimate goal in designing efficient protocols is to construct protocols that are secure against strong (active or covert) adversaries while adding very little overhead compared to the passive variant. Within this goal, optimizing the efficiency of protocols in the semi-honest model serves as an important stepping stone. In this thesis, we optimize protocols in the semi-honest model and show how to achieve covert and malicious security for OT extension protocols at low additional cost.

2.1.5 Pre-Computation Model

Most of our protocols work in the *pre-computation model*, where the computation is divided into a *setup* and an *online* phase. In the setup phase, the parties do not know the actual inputs (or not necessarily even the function to be evaluated) but are allowed

[1] According to the summary of cryptographic key length recommendations at http://keylength.com.

to generate helper data. This helper data can then be used to achieve a more efficient online phase, where the inputs (or the function) become known.

The advantage of the pre-computation model is threefold. Firstly, the parties can pre-compute computation- and communication-intensive operations when resources are available (e.g., at night) to allow a faster online evaluation when required. Secondly, pre-computation allows to batch operations, which amortizes costs and allows for faster evaluation. Finally, by separating the protocol into multiple phases, the protocol design becomes more modular, which allows to flexibly exchange the underlying algorithms.

Note, that setup and online phase can be pipelined in order to reduce the memory footprint and overall evaluation time [HEKM11, HS13].

2.2 Cryptographic Primitives

The following section lists cryptographic primitives that are commonly used in secure computation protocols and presents the instantiations used in this work: The *pseudo-random generator* (§2.2.1), the *random oracle* (§2.2.2), the *pseudo-random function* (§2.2.3), and the *correlation-robust function* (§2.2.4).

2.2.1 Pseudo-Random Generator

A *pseudo-random generator (PRG)* $G : \{0,1\}^\kappa \mapsto \{0,1\}^*$ takes as input a truly random κ-bit string and generates an arbitrary-length output that is computationally indistinguishable from a truly random sequence [Gol04]. In this thesis, we instantiate the PRG using AES in counter mode (AES-CTR), as outlined in [NIS15]. More detailed, to generate a 128-bit pseudo-random sequence, we compute $AES_k(i)$, where k is a sequence of truly random bits that is used as key and i is a monotone counter that is initialized with a fixed value and incremented after generating a 128-bit block. The AES-CTR-based instantiation is especially efficient on machines that are equipped with the AES-NI operations, which have a dedicated AES encryption operation [Gue12].

2.2.2 Random Oracle

A *random oracle (RO)* $H : \{0,1\}^\ell \mapsto \{0,1\}^\ell$ is a black box function that, upon being queried by an input for the first time, maps it to a truly random output, which it consistently returns for the same input from then on [BR93]. Cryptographic constructions often distinguish between security proofs that use the standard or the random oracle model. There are many criticisms about the random oracle model, and in the theory of cryptography proofs, this model is considered heuristic. However, as most previous works on efficient secure computation, we often use the random oracle model to achieve more efficient implementations [BR93]. Throughout this thesis, we instantiate the RO using SHA256.

2.2.3 Pseudo-Random Function

A *pseudo-random function (PRF)* $F_s : \{0,1\}^\ell \mapsto \{0,1\}^\ell$ that is parameterized by a κ-bit seed s takes as input an ℓ-bit argument x and computes $y = F_k(x)$ such that y is indistinguishable from the output of truly random function for any $s \in \{0,1\}^\kappa$ [Gol04]. A PRF can be instantiated by using a specific sequence in the output of a PRG [Gol04]. In this thesis, we instantiate the PRF using the AES encryption by computing $F_s(x) = AES_s(x)$, i.e., encrypting x using the key s, since the AES-NI operations provide us with a very fast AES encryption and key scheduling operation [Gue12].

2.2.4 Correlation-Robust Function

A *correlation robust one-way function* (CRF) $H : \{0,1\}^\kappa \mapsto \{0,1\}^\ell$ is a function for which, given uniformly and randomly $x_1, ..., x_m, s$, an adversary is unable to computationally distinguish the outputs chosen $x_1, ..., x_m, H(x_1 \oplus s)$, ..., $H(x_m \oplus s)$ from a uniform distribution. It is a weaker assumption than the random oracle model and is used in OT extension and, with an additional circularity property, in Yao's garbled circuits protocol with state-of-the-art optimizations [ZRE15].

Instantiations of a CRF based on number-theoretic or lattice-based assumptions were given in [AHI11]. However, many implementations use a hash function (e.g., SHA) to increase the performance. An instantiation of the CRF in Yao's garbled circuits protocol which uses fixed-key AES and greatly improves performance was proposed in [BHKR13] and refined in [ZRE15] for use in the half-gates scheme. Using the fixed-key AES instantiation, a value x is processed as $AES_k(2x \oplus t) \oplus 2x \oplus t$, where k is a fixed AES key, $2x$ refers to doubling in GF(2^κ), and t is a unique identificator (i.e., a monotone counter). The fixed-key AES instantiation, however, imposes the strong ideal permutation assumption [Bla06] on AES. To achieve more relaxed assumptions, [GLNP15] propose several different constructions that use a pipelined AES-NI evaluation to achieve speeds similar to fixed-key AES.

In this thesis, we first instantiate the CRF using SHA-256 to ensure a fair comparison between OT extension protocols in §3. Since the focus of this thesis is on practical efficiency, we then instantiate the CRF for up to 128-bit inputs using the fixed-key AES [BHKR13], for up to 256-bit inputs using AES-256 with key schedule (cf. §3.2.4.2), and for up to 512-bit inputs using SHA-256. We empirically evaluate the performance improvement of AES-based over SHA-based CRF instantiations for OT extension in §3.5.5.

2.3 Oblivious Transfer

OT was first introduced by Rabin [Rab81] as a function where a receiver receives a message, sent by a sender, with probability 1/2, while the sender remains oblivious

whether the message was received. It was later re-defined by [EGL85] to the 1-out-of-2 OT ($\binom{2}{1}$ OT) functionality more commonly used today, where the sender inputs two messages (x^0, x^1) and the receiver inputs a choice bit c and obliviously receives x^c without learning any information about x^{1-c}. In this thesis we focus on the general $\binom{N}{1}$ OT_n^m functionality, which is equivalent to m invocations of the $\binom{N}{1}$ OT functionality on n-bit strings. That is, the sender inputs m tuples of n-bit strings $(x_j^1, ..., x_j^N)$ and the receiver inputs m choice strings $\mathbf{r} = (r_1, \ldots, r_m)$ with $r_j \in [1...N]$ and for $1 \leq j \leq m$. The output of the receiver is $(x_1^{r_1}, \ldots, x_m^{r_m})$ while the sender has no output. Throughout this thesis, we sometimes omit $\binom{2}{1}$ and write OT instead of $\binom{2}{1}$ OT.

Several protocols for OT based on different cryptographic assumptions and attacker models were introduced. Most notable are the passive secure OT protocol of [NP01] and the active secure OT protocols of [PVW08] and [CO15], which are among the most efficient today. However, the impossibility result of [IR88, IR89] showed that OT protocols require costly asymmetric cryptography, which greatly limits their efficiency.

Efficiency. The most efficient OT protocols in the semi-honest and malicious model are [NP01] and [CO15], which both have an amortized cost of 3 exponentiations and one public-key ciphertext sent per OT.

2.3.1 1-Out-of-2 OT Extension

In his seminal work, Beaver [Bea96] introduced *OT extension* protocols, which extend few costly public-key *base-OTs* using symmetric cryptography only. While the first construction of [Bea96] was inefficient and mostly of theoretical interest, the protocol of [IKNP03] showed that OT can be extended efficiently and with very little overhead. We give the semi-honest secure $\binom{2}{1}$ OT extension protocol of [IKNP03] in Protocol 1. An extension to malicious adversaries was also given in [IKNP03] but is omitted.

Efficiency. We give the complexity for the $\binom{2}{1}$ OT extension of [IKNP03] in Table 2.2. For the complexity analysis we omit the base-OTs, which present a linear overhead in the security parameter and amortize fairly quickly.

Complexity	**Sender** P_S	**Receiver** P_R
PRG evaluations	1	2
CRF evaluations	2	1
Sent [bits]	$2n$	2κ

Table 2.2: Computation and communication complexity per party for one $\binom{2}{1}$ OT on n-bit messages using the OT extension of [IKNP03] (excluding base-OTs).

PROTOCOL 1 (Semi-Honest Secure $\binom{2}{1}$ OT Extension Protocol of [IKNP03]).

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \leq j \leq m$.
- **Input of P_R:** m choice bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and number of base-OTs $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the OT_κ^ℓ functionality and use a pseudo-random generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a correlation-robust function $H : [m] \times \{0,1\}^\ell \to \{0,1\}^n$ (cf. §2.2.4).

1. *Initial OT Phase:*
   a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in_R \{0,1\}^\ell$ and P_R randomly chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   b) The parties invoke the OT_κ^ℓ-oracle, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

   Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ be a random $m \times \ell$ bit matrix that is generated by P_R where its ith column is $\mathbf{t}^i$ for $1 \leq i \leq \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \leq j \leq m$.
2. *OT Extension Phase:*
   a) P_R computes $\mathbf{u}^{(i,0)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^0)$ and $\mathbf{u}^{(i,1)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $(\mathbf{u}^{i,0}, \mathbf{u}^{i,1})$ to P_S for every $1 \leq i \leq \ell$.
   b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = \mathbf{u}^{(i,s_i)} \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   c) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   d) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

   e) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- **Output:** P_S has no output; P_R outputs $(x_1, \ldots, x_m)$.

2.3.2 1-Out-of-N OT Extension

In the $\binom{2}{1}$ OT extension protocol of [IKNP03], the parties use multiple base-OTs to obliviously transfer shares of the receiver's choice bits. The authors of [KK13] observed that this approach can be generalized to have both parties share a ρ-bit codeword from a code Γ^ρ with codewords of Hamming distance κ to efficiently realize $\binom{N}{1}$ OT extension. These codewords encode the receiver's choices and constitute the main component of the communication workload of the OT extension protocol. We give a full description of the protocol in Protocol 2.

For $N = 2$, a repetition code can be used, which has 2 codewords of size $\rho = \kappa$. In this

PROTOCOL 2 (Semi-Honest Secure $\binom{N}{1}$ OT Extension Protocol of [KK13])**.**

- **Input of P_S:** m N-tuples $(x_j^1, ..., x_j^N)$ of n-bit strings, $1 \leq j \leq m$.
- **Input of P_R:** m choice integers $\mathbf{r} = (r_1, \ldots, r_m)$ with $r_j \in [1...N]$.
- **Common Input:** Symmetric security parameter κ, code $\Gamma^\rho = (\gamma_1, ..., \gamma_N)$ with N ρ-bit codewords that have κ-bit relative distance, and number of base-OTs $\ell = \rho$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the OT_κ^ℓ functionality and use a pseudo-random generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a correlation-robust function $H : [m] \times \{0,1\}^\ell \to \{0,1\}^n$ (cf. §2.2.4).

1. *Initial OT Phase:*
   a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in_R \{0,1\}^\ell$ and P_R randomly chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   b) The parties invoke the OT_κ^ℓ-oracle, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

   P_R generates two $m \times \ell$ bit matrices: A random matrix $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$, where its ith column is $\mathbf{t}^i$ for $1 \leq i \leq \ell$, and a choice-code bit matrix $C = [\mathbf{c}_1 | \ldots | \mathbf{c}_\ell]$, where its jth row is $\mathbf{c}_j = \gamma_{r_j}$ for $1 \leq j \leq m$. Let $\mathbf{t}_j$ denote the jth row of T and $\mathbf{c}^i$ denote the ith column of C for $1 \leq i \leq \ell$ and $1 \leq j \leq m$.
2. *OT Extension Phase:*
   a) P_R computes $\mathbf{u}^{(i,0)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^0)$ and $\mathbf{u}^{(i,1)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{c}^i$, and sends $(\mathbf{u}^{i,0}, \mathbf{u}^{i,1})$ to P_S for every $1 \leq i \leq \ell$.
   b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = \mathbf{u}^{(i,s_i)} \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot c^i) \oplus \mathbf{t}^i$.)
   c) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot c^i) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (c_j \wedge \mathbf{s}) \oplus \mathbf{t}_j$.)
   d) For every $p \in [1...N]$, P_S computes:

   $$y_j^p = x_j^p \oplus H(j, \mathbf{q}_j \oplus \gamma_p)$$

   and sends $(y_j^1, ..., y_j^N)$ for every $1 \leq j \leq m$.
   e) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- **Output:** P_S has no output; P_R outputs $(x_1, \ldots, x_m)$.

case, the $\binom{N}{1}$ OT protocol of [KK13] is identical to the $\binom{2}{1}$ OT protocol of [IKNP03].

For $2 < N \leq 2\kappa$, the authors propose to use a Walsh-Hadamard code which has codewords of size $\rho = 2\kappa$ to achieve a relative Hamming distance of κ.

For $N > 2\kappa$, a linear error-correcting code achieves the best performance. In particular, when $N = poly(\kappa)$, the communication cost for the OT extension part of $\binom{N}{1}$ OT invocations decreases asymptotically from $\mathcal{O}(\kappa \log N)$ to $\mathcal{O}(\kappa)$ compared to a $\binom{2}{1}$ OT instantiation.

Efficiency. We give the complexity for one $\binom{N}{1}$ OT extension protocol of [KK13] in Table 2.3. We again omit the base-OTs, which present a linear overhead in the code size and amortize fairly quickly.

Complexity	**Sender** P_S	**Receiver** P_R
PRG evaluations	1	2
CRF evaluations	N	1
Sent [bits]	Nn	2ρ

Table 2.3: Computation and communication complexity for $\binom{N}{1}$ OT on n-bit messages using the OT extension of [KK13] (excluding base-OTs).

2.3.3 Converting Between 1-Out-of-2 OT and 1-Out-of-N OT

Converting from $\binom{N}{1}$ OT to $\binom{2}{1}$ OT and vice versa is trivial and can be done with only some additional overhead. We outline in Protocol 3 how to obtain $\binom{N}{1}$ OT from $\binom{2}{1}$ OT and in Protocol 4 how to obtain $\binom{2}{1}$ OT from $\binom{N}{1}$ OT and give a high level discussion of both protocols in the following.

$\binom{N}{1}$ $\mathbf{OT}_n^1$ from $\binom{2}{1}$ $\mathbf{OT}_\kappa^{\log_2 N}$ (Protocol 3). The high-level idea of converting to $\binom{N}{1}$ OT_n^1 is to let both parties compute $\binom{2}{1}$ $\mathrm{OT}_\kappa^{\log_2 N}$ on random κ-bit key pairs and send each n-bit input message masked using a different combination of keys [NP05]. In terms of efficiency, this conversion involves the cost of $\binom{2}{1}$ $\mathrm{OT}_\kappa^{\log_2 N}$ plus $2N$ additional PRF evaluations to mask the values and $\log_2 N$ additional PRF evaluations to unmask the correct value.

$\binom{2}{1}$ $\mathbf{OT}_n^{\log_2 N}$ from $\binom{N}{1}$ $\mathbf{OT}_{n\log_2 N}^1$ (Protocol 4). To break $\binom{N}{1}$ OT down to $\log_2 N$ $\binom{2}{1}$ OT, P_S builds all combinations of message pairs (x_j^0, x_j^1) for $1 \leq j \leq \log_2 N$ and obliviously transfers these combinations via $\binom{N}{1}$ $\mathrm{OT}_{n\log_2 N}^1$. The cost for the conversion to $\binom{2}{1}$ $\mathrm{OT}_n^{\log_2 N}$ is equal to the cost of a $\binom{N}{1}$ $\mathrm{OT}_{n\log_2 N}^1$.
It was shown in [KK13] that when computing $\binom{2}{1}$ OT_1^4, the $\binom{N}{1}$ OT extension protocol of [KK13] requires less communication than the OT extension protocol of [IKNP03]. In particular, while computing $\binom{2}{1}$ OT_1^4 using the OT extension from [IKNP03] requires sending $8(\kappa+2) = 1\,040$ bits (cf. Table 2.2), a conversion from $\binom{16}{1}$ OT_4^1 using the OT extension of [KK13] requires sending only $2\rho + 64 = 576$ bits (cf. Table 2.3), which is factor 1.8× improvement in communication.

2.3.4 OT Pre-Computation

OT pre-computation [Bea95] allows to pre-compute a $\binom{N}{1}$ OT_n^m on random inputs in the setup phase and later in the online phase use these pre-computed values as one-time

PROTOCOL 3 ($\binom{N}{1}$ OT_n^1 From $\binom{2}{1}$ $\mathrm{OT}_\kappa^{\log_2 N}$ [NP05]).

- **Input of P_S:** A N-tuple $(x^1, ..., x^N)$ of n-bit strings.
- **Input of P_R:** A choice integer r with $r \in [1...N]$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the $\binom{2}{1}$ $\mathrm{OT}_\kappa^{\log_2 N}$ functionality and use a PRF $F_k : \{0,1\}^{\log_2 N} \mapsto \{0,1\}^n$ with a κ-bit seed k.

1. P_S chooses $\log_2 N$ pairs of random κ-bit keys $(k_i^0, k_i^1) \in_R \{0,1\}^{2\kappa}$, for $1 \leq i \leq \log_2 N$.
2. P_S and P_R invoke the $\binom{2}{1}$ $\mathrm{OT}_\kappa^{\log_2 N}$ functionality where, in the i-th OT, P_S plays the sender with random inputs (k_i^0, k_i^1) while P_R plays the receiver and inputs the i-th bit of r. P_S obtains no output while P_R obtains $t_i = k_i^{r[i]}$, for $1 \leq i \leq \log_2 N$.
3. P_S then computes for $1 \leq j \leq N$:

$$y^j = x^j \oplus \bigoplus_{i=1}^{\log_2 N} F_{k_i^{j[i]}}(j),$$

   and sends $(y^1, ..., y^N)$ to P_R, where $j[i]$ is the i-th bit of j.
4. P_R computes $x = y^r \oplus \bigoplus_{i=1}^{\log_2 N} F_{t_i}(r)$.

- **Output:** P_S has no output; P_R outputs x.

PROTOCOL 4 ($\binom{2}{1}$ $\mathrm{OT}_n^{\log_2 N}$ From $\binom{N}{1}$ $\mathrm{OT}_{n \log_2 N}^1$).

- **Input of P_S:** $\log_2 N$ tuples (x_j^0, x_j^1) of n-bit strings, for $1 \leq j \leq \log_2 N$.
- **Input of P_R:** $\log_2 N$ choice bits $\mathbf{r} = (r_1, ..., r_{\log_2 N})$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the $\binom{N}{1}$ $\mathrm{OT}_{n \log_2 N}^1$ functionality.

1. P_S computes a sequence $(s^0, ..., s^{N-1})$ of $n \log_2 N$-bit strings s^i for $0 \leq i < N$ as:

$$s^i = (x_1^{i[1]}, x_2^{i[2]}, ..., x_{\log_2 N}^{i[\log_2 N]}),$$

   where $i[j]$ is the j-th bit of i for $1 \leq j \leq \log_2 N$.
2. P_S and P_R invoke the $\binom{N}{1}$ $\mathrm{OT}_{n \log_2 N}^1$ functionality where P_S plays the sender with input $(s^0, ..., s^{N-1})$ while P_R plays the receiver with input $\mathbf{r}$. P_S obtains no output while P_R obtains $s^{\mathbf{r}} = (x_1^{r_1}, ..., x_{\log_2 N}^{r_{\log_2 N}}) = (x_1, ..., x_{\log_2 N})$.

- **Output:** P_S has no output; P_R outputs $(x_1, ..., x_{\log_2 N})$.

pads to run the OT on the actual inputs. We give a full protocol of OT pre-computation in Protocol 5. Regarding efficiency in the online phase, the parties perform two rounds of communication, where P_R first sends one message of size $m \log_2 N$ bits to P_S who replies with a message of size Nmn bits.

PROTOCOL 5 (OT Pre-Computation of [Bea95])**.**

- **Input of P_S:** m N-tuples $(x_j^1, ..., x_j^N)$ of n-bit input messages as well as m random N-tuples $(k_j^1, , ..., k_j^N)$ of n-bit strings from the pre-computed $\binom{N}{1}$ OT_n^m, for $1 \leq j \leq \log_2 N$.
- **Input of P_R:** m choice integers $\mathbf{r} = (r_1, ..., r_m)$ with $r_j \in [1...N]$ as well as m random choice integers $\mathbf{c} = (c_1, ..., c_m)$ and m n-bit strings k_j^c from the pre-computed $\binom{N}{1}$ OT_n^m, for $1 \leq j \leq \log_2 N$.

1. P_R sends $u_j = r_j \oplus c_j$ to P_S, for $1 \leq j \leq m$.
2. P_S computes y_j^i for $1 \leq i \leq N$ as:

$$y_j^i = k_j^{i \oplus u_j} \oplus x_j^i,$$

   and sends $(y_j^1, ..., y_j^N)$ to P_R, for $1 \leq j \leq m$.
3. P_R computes $x_j = y_j^{r_j} \oplus k_j^c$.

- **Output:** P_S has no output; P_R outputs $(x_1, ..., x_m)$.

2.4 Generic Secure Two-Party Computation

In this section, we present the two most prominent generic secure two-party computation protocols for Boolean circuits: *Yao's garbled circuits* (§2.4.1) and the *GMW* protocol (§2.4.2) and compare various aspects of both protocols (§2.4.3).

2.4.1 Yao's Garbled Circuits Protocol

In his seminal work [Yao86], Yao introduced the garbled circuits protocol[2], which enables two parties to securely evaluate any functionality that can be represented as Boolean circuit. Yao's garbled circuits protocol is run between two parties called *garbler* and *evaluator*, which perform the following steps: *Circuit garbling*, *input encoding*, and *circuit evaluation*, which we describe in more detail in §2.4.1.1. Several optimizations for Yao's garbled circuits have been proposed, of which we describe the most prominent ones in §2.4.1.2. A first proof of security for Yao's garbled circuits protocol

[2]Even though [Yao86] is used as source for Yao's garbled circuits protocol, it was, in fact, only mentioned in an oral presentation [Gol01, BHR12].

appeared in [LP04, LP09] and a generalization of garbled circuits protocols to garbling schemes appeared in [BHR12].

2.4.1.1 Unoptimized Garbled Circuits

In the following, we describe the unoptimized Yao's garbled circuits protocol, as outlined in [LP04].

Circuit Garbling. The garbler selects the function f to be computed and represents f as Boolean circuit C. For each wire A in the circuit, the garbler selects two random κ-bit keys $(k_0^A, k_1^A) \in_R \{0,1\}^{2\kappa}$ that represent the logical 0 and 1 on the wire. For each gate g with input wires A and B, output wire C, corresponding wire keys $(k_0^A, k_1^A, k_0^B, k_1^B, k_0^C, k_1^C)$, and gate semantic $s : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$, the garbler then selects a random permutation $\pi_g : [0...3] \mapsto [0...3]$ and creates a garbled table T_g with four entries as:

$$
\begin{aligned}
T_g[\pi_g(0)] &= E_{k_0^A}(E_{k_0^B}(k^C_{s(0,0)})) \\
T_g[\pi_g(1)] &= E_{k_0^A}(E_{k_1^B}(k^C_{s(0,1)})) \\
T_g[\pi_g(2)] &= E_{k_1^A}(E_{k_0^B}(k^C_{s(1,0)})) \\
T_g[\pi_g(3)] &= E_{k_1^A}(E_{k_1^B}(k^C_{s(1,1)})).
\end{aligned}
$$

The symmetric encryption function E_k uses a secret-key k and the corresponding decryption function D_k guarantees correct decryption only if the ciphertext was indeed encrypted with the same key k and otherwise outputs an error (for more details see [LP04]). Each of the 4 entries in the garbled table has a size of 2κ [LP04]. When all gates have been garbled, the garbler then sends the garbled circuit, which consists of the garbled tables of all gates, to the evaluator.

Input Encoding. After the circuit has been garbled and transmitted, the evaluator needs to obtain the keys for the input wires that correspond to both parties' input bits. For each of the garbler's input wires, the garbler simply sends the wire key that corresponds to its input to the evaluator. For each of the evaluator's input wires W with corresponding input bit w, the parties run an OT^1_κ, where the garbler acts as sender with messages (k_0^W, k_1^W) and the evaluator acts as receiver with choice bit $w \in \{0,1\}$ and obliviously receives k_w^W.

Circuit Evaluation. Upon obtaining the garbled circuit and the input wire keys, the evaluator can evaluate the garbled circuit. For each gate g with input wires A and B, and output wire C, the evaluator holds two input keys k_a^A and k_b^B (the evaluator remains oblivious about the exact values of a and b) and the garbled table T_g. The evaluator then decrypts all entries in the garbled table and obtains a valid key $k_c^C = D_{k_a^A}(D_{k_b^B}(T_g[i]))$ for one entry i. Note that the decryption function has to guarantee that the evaluator only obtains a valid output if the encryption and decryption

procedure use the same key and otherwise yields an error symbol ($\perp$). After decrypting all gates, the evaluator either sends the keys on the output wires to the garbler or the garbler provides a mapping from output wire keys to plaintext bits, depending on which party obtains the output.

Efficiency. In the unoptimized Yao's garbled circuits protocol, described in [LP04], the garbler has to evaluate 4 double encryption functions and send 8κ-bit per gate in the Boolean circuit while the evaluator has to evaluate 2.5 double decryption functions on average until a successful decryption occurs. Note that Yao's garbled circuits provides one-sided malicious security against a malicious evaluator when using a malicious secure OT protocol, since the only message that the evaluator sends to the garbler consists of the keys on the output wires, which the evaluator can only correctly decrypt for the corresponding keys on the input wires. A malicious garbler, however, could encode any function in the garbled circuit and thereby retrieve the inputs of the evaluator.

2.4.1.2 Optimizations for Yao's Garbled Circuits

Several optimizations for Yao's garbled circuits have been proposed, among which the most prominent are: *Point and permute (PnP)* [BMR90], *garbled row reduction (GRR-3)* [NPS99], *free-XOR* [KS08], *fixed-key AES garbling* (Fixed-Key AES) [BHKR13], *half-gates* [ZRE15], and *inter party parallelization (IPP)* [BK15]. We give their performance improvement in terms of run-time for symmetric cryptographic operations and communication for the AES circuit of [HEKM11] with the S-Box circuit of [BP10] in Figure 2.2. Overall, this circuit has 24 848 total gates of which 5 120 are AND gates. We used the S-Box circuit of [BP10], since it is optimized for both, small number of Boolean gates and small number of AND gates. For optimizations prior to fixed-key AES of [BHKR13], we assume that SHA-256 is used to encrypt and decrypt an entry in the garbled table (cf. [LPS08] for the instantiation in the random oracle model), which requires amortized 248.7 ns computation on a desktop PC with Intel AES-NI support, while fixed-key AES requires amortized 15.8 ns computation (cf. Table 3.3 on page 36). Using all optimizations, the parties can process XOR gates for free and, for each AND gate, the garbler needs to send a garbled table with two κ-bit entries while the local computation overhead of 4 symmetric operations for the garbler and 2 symmetric operations for the evaluator is close to negligible. It was shown in [ZRE15] that the communication of 2κ bit presents a lower bound for most of the existing garbling schemes. Using IPP, the parties can then evenly distribute the computation and communication overhead, given that the circuit allows for parallelization. A more detailed description of each optimization is given next.

Point And Permute (PnP) [BMR90]. In [LP04], the garbler permutes the garbled table to hide from the evaluator, which entry in the garbled table corresponds to which plaintext bit. The evaluator then decrypts all entries in the garbled table and selects the correctly decrypted key. In order to identify the correctly decrypted key, the

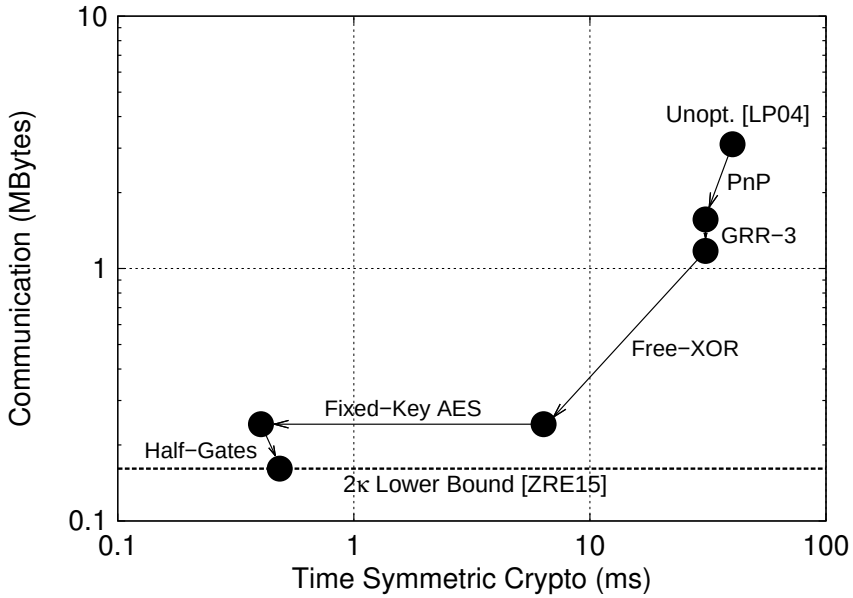


Figure 2.2: Time for symmetric cryptographic operations (x-axis) and communication (y-axis) for different Yao's garbled circuit optimizations when evaluating the AES circuit of [HEKM11] using the S-Box circuit of [BP10] (24 848 total gates, 5 120 AND gates). Each optimization includes previous optimizations. Time for symmetric crypto was measured using an Intel Haswell i7-4770K CPU with AES-NI support.

garbler needs to increase the garbled table entries to size 2κ (or $\kappa + \lambda$ for statistical security parameter λ) to ensure that the correct key is chosen except with negligible probability $2^{-\kappa}$ (or $2^{-\lambda}$).

The point and permute technique reduces the size per entry in the garbled table from 2κ to $\kappa + 1$ and allows the evaluator to decrypt only one table entry per garbled table instead of 2.5 on average. This is done by introducing a special bit per key, called *permutation bit*, which "points" to the correct entry in the garbled table but "permutes" the semantics of the key in such a way that the evaluator gains no information on the processed value. Note that, in the remainder of this thesis, we use the common approach of replacing the least significant bit of the wire key by the permutation bit, which allows for more efficient evaluation but decreases the symmetric security by one bit.

Garbled Row Reduction (GRR-3) [NPS99]. While garbling the circuit, the garbler assigns two symmetric keys to each wire in the circuit and, for each gate,

creates a garbled table with four entries.

The garbled row reduction technique reduces the number of entries in the garbled table from four to three by fixing the first entry in the garble table to a constant value (e.g., 0^κ). Thereby, one of the keys on the output wire is determined by the input keys. An additional row-reduction technique was proposed in [PSSW09], which reduces the number of entries in the garbled table to two but is incompatible with the free-XOR optimization, outlined next. A technique that switches between garbling using the free-XOR technique and the row-reduction technique of [PSSW09] was outlined in [KMR14].

Free-XOR [KS08]. During circuit garbling, the two keys on each wire are chosen independently and the garbler needs to garble every gate.

The free-XOR technique allows to evaluate XOR (linear) gates "for free" by choosing all wire key pairs in the circuit such that they are XOR-correlated by the same constant offset. This allows the parties to process XOR gates locally by computing the XORs of the keys instead of generating and sending a garbled table for each gate. The efficiency gains of free-XOR vary, depending on the circuit. E.g., for the example AES circuit, where the number of XOR gates is 24 848 and the number of AND gates is 5 120, the free-XOR decreases the number of garbled gates by approximately factor 5×. Note, however, that the free-XOR optimizations imposes the circular-2 correlation robustness assumption to CRF [CKKZ12].

Fixed-Key AES Garbling [BHKR13]. The main computational workload in Yao's garbled circuits protocol comes from encryption and decryption functions that are used to create and evaluate the garbled table. In [LP04], these functions were instantiated using a double encryption scheme with a PRF. Later works instantiated these functions using a hash function [KS08, HEKM11], or using a block cipher with 2κ-bit key length [KSS12].

The work of [BHKR13] proposes a fixed-key AES instantiation, which utilizes the AES-NI operations to reduce the computation time for garbling a gate close to negligible at the cost of modeling the encryption function as an ideal permutation (cf. §2.2.4). Several further garbling function instantiations with weaker security requirements that make use of the pipelining features of the AES-NI operations were given in [GLNP15].

Half-Gates [ZRE15]. The half-gates technique reduces the number of entries in the garbled table from three to two at the cost of one additional decryption for the evaluator. The idea of the half-gates garbling scheme is to split each AND gate into two parts: A garbler half-gate, where the garbler knows the plaintext bit corresponding to one input key, and an evaluator half-gate, where the evaluator knows the plaintext bit corresponding to one input key. The authors then show how to garble each half-gate using only one κ-bit ciphertext each and how to securely evaluate an AND gate by combining both half-gates. In addition, [ZRE15] introduced the notion of linear garbling schemes, which covers most known garbling schemes, and proved that there

is a lower-bound of two κ-bit ciphertexts (plus two permutation bits) per non-linear gate for such schemes.

Inter Party Parallelization (IPP) [BK15]. The garbler in Yao's garbled circuits has a higher computation and communication workload than the evaluator, since the garbler has to construct and send the garbled circuit. Furthermore, in the pre-computation model, where the garbling and evaluation are separated into the setup and online phase, one party waits idly until the other party is done.

In order to reduce this idle time, [BK15] introduces the inter party parallelization technique (IPP), where the garbler and evaluator switch roles for parts of the garbled circuit. To merge parts with different role assignments, [BK15] proposes a role transformation protocol, which is a two-round protocol, similar to the input encoding routine for client inputs (cf. §2.4.1.1), where the parties perform a OT^1_κ. The garbler of the old garbled circuit (and hence evaluator of the new garbled circuit) plays the receiver and inputs the permutation bit on the output key of the old garbled circuit and obtains the input key to the new garbled circuit. The evaluator of the old garbled circuit (and hence garbler of the new garbled circuit) plays the sender with two input keys to the new garbled circuit, which are swapped depending on the least significant bit of the output key of the old garbled circuit, and obtains no output.

IPP was evaluated on several functionalities and was shown to achieve approximately factor 1.1× local computation speed-up for a pipelined Yao's garbled circuits evaluation and a factor 2× communication improvement. In our empirical evaluations of generic secure computation protocols in §4.6, we use Yao's garbled circuits in the pre-computation model without pipelining and show that IPP achieves a performance gain for functions that consist of many independent sub-circuits of nearly factor 2×. Note, that IPP removes the malicious client security guarantee (cf. §2.4.1.1), since both parties act as garbler. However, since the focus of this work is on semi-honest adversaries, we use IPP in our evaluation.

2.4.2 Goldreich-Micali-Wigderson (GMW)

In the GMW protocol [GMW87], two parties interactively compute a function using secret-shared values. The parties share the value of each input and intermediate wire using a 2-out-of-2 secret sharing scheme such that each party holds a random-looking share x_i with $x = x_0 \oplus x_1$. As XOR is an associative operation, XOR gates can be securely evaluated locally by XORing the shares.

For secure evaluation of AND gates, the parties run an interactive protocol using OT or multiplication triples (MTs) as discussed further below. Note that AND gates of the same layer in the circuit can be evaluated in parallel. To obtain the output, the parties exchange their shares of the respective output wire.

AND via OT [CHK+12, LLXX05]. To securely evaluate an AND gate $z = x \wedge y$ on input shares $x = x_0 \oplus x_1$ and $y = y_0 \oplus y_1$, the two parties can run a $\binom{4}{1}$ OT^1_1

protocol. Here, the sender P_0 chooses a random output share z_0 and provides four inputs (s_0, s_1, s_2, s_3) to the OT protocol using x_0, y_0 with:

$$\begin{aligned} s_0 &= z_0 \oplus (x_0 \wedge y_0) \\ s_1 &= z_0 \oplus (x_0 \wedge (y_0 \oplus 1)) \\ s_2 &= z_0 \oplus ((x_0 \oplus 1) \wedge y_0) \\ s_3 &= z_0 \oplus ((x_0 \oplus 1) \wedge (y_0 \oplus 1)). \end{aligned}$$

The receiver P_1 inputs $x_1||y_1$ as choice bits and obliviously obtains its share $z_1 = s_{x_1||y_1} = z_0 \oplus (x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$. As described in §2.3, all OTs can be moved into a pre-processing phase such that the online phase is highly efficient (only two message rounds of 2 and 4 bit and inexpensive one-time-pad operations).

AND via Multiplication Triples [Bea91]. An alternative method to securely evaluate an AND gate $z = x \wedge y$ on input shares $x = x_0 \oplus x_1$ and $y = y_0 \oplus y_1$ are *multiplication triples (MTs)* [Bea91]. MTs are random shares a_i, b_i, c_i satisfying $(c_0 \oplus c_1) = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$ where P_i holds the shares labeled with i. They can be generated in the setup phase using a $\binom{4}{1}$ OT_1^1 protocol in a similar way to the OT-based AND evaluation described above. In the online phase, the parties then use the pre-generated MTs to compute and exchange $d_i = x_i \oplus a_i$ and $e_i = y_i \oplus b_i$ and set their output shares as $z_0 = (d \wedge e) \oplus (b_0 \wedge d) \oplus (a_0 \wedge e) \oplus c_0$ and $z_1 = (b_1 \wedge d) \oplus (a_1 \wedge e) \oplus c_1$. The advantage of MTs over the OT-based AND evaluation is that, per AND gate in the online phase, the parties only need to perform one interaction round (instead of two interaction rounds for OT pre-computation) and the size of the messages is slightly smaller ($2 + 2$ bits instead of $2 + 4$ bits).

2.4.3 GMW vs. Yao

In this section, we give a conceptual and performance comparison of Yao's garbled circuits and the GMW protocol. In our comparison, we assume a state-of-the-art Yao's garbled circuits instantiation, using the optimizations summarized in §2.4.1.2 excluding IPP, and a GMW instantiation similar to [CHK+12] that pre-computes the $\binom{4}{1}$ OT extension protocol of [LLXX05] (which is similar to 2 invocations of the [IKNP03] OT extension protocol) but uses MTs to evaluate AND gates. We first review the ability for pre-computation of both schemes, then compare their complexity, and finally discuss implementation features. An overview of the complexity of both schemes assuming that both can fully pre-compute all necessary data can be found in Table 2.4.

Pre-Computability. Yao's garbled circuits and the GMW protocol can both be pre-computed but differ in the information that is necessary to allow pre-computation. The GMW protocol allows to pre-compute OT extension and hence all symmetric cryptographic operations and the majority of the communication if the function size or even only an upper bound on the function size is known. Yao's garbled circuits allows

Properties	Garbled Circuits	GMW [CHK+12]
per AND gate:		
setup computation	P_0: 4×CRF	P_0: 1×PRG, 4×CRF P_1: 4×PRG, 1×CRF
setup communication [bit]	$P_0 \rightarrow P_1$: $2(\kappa+1)$	$P_0 \rightarrow P_1$: 4 $P_0 \leftarrow P_1$: 4κ
online computation	P_1: 2×CRF	negligible
online communication [bit]	-	$P_0 \rightarrow P_1$ &$P_0 \leftarrow P_1$: 2
per wire storage P_1 & P_0 [bit]	κ	1
per input: comm. [bit]	P_0: κ; P_1: 4κ	1
Online communication rounds	$\mathcal{O}(1)$	**D**(C)

Table 2.4: Conceptual comparison between state-of-the-art Yao's garbled circuits (using the optimizations in §2.4.1.2) and the GMW protocol (using $\binom{4}{1}$ OT extension of [LLXX05] to compute MTs). P_0: garbler, P_1: evaluator. Assuming that function and input sizes are known during setup phase.

to pre-compute and send the garbled circuit in case the function and the number of inputs are known in advance. A garbling scheme that relaxes the necessary information for pre-computing Yao's garbled circuits is given in [GLMY16]. Instead of garbling the whole function at once, [GLMY16] garbles individual modular blocks, which can be connected arbitrarily at an additional communication overhead in the online phase per connected wire.

Implementation Features. The memory required for storing the pre-computed data is smaller for the GMW protocol as for every AND gate in the circuit, each party only needs to store 3 bits instead of 2κ-bits that the evaluator in Yao's garbled circuits needs to store. On the other hand, Yao's garbled circuits is able to efficiently evaluate much larger circuits than GMW by pipelining the circuit generation and evaluation [HEKM11, Mal11, HS13, KMSB13, SHS+15], which allows the parties to only keep in memory a small number of gates at any point during the computation. The GMW protocol, on the other hand, needs to store the full circuit description in memory at one point during the computation in order to schedule the gates layer-wise to identify the critical path that defines the depth of the circuit. Finally, the setup phase of the GMW protocol can be easily parallelized, independently of the function, while the parallelizability of Yao's garbled circuits protocol depends on the function that is evaluated. A compiler that schedules high level functions and allows automatic parallelization for Yao's garbled circuits is given in [BK15].

Efficiency. An implementation of the GMW protocol in the multi-party setting is given in [CHK+12]. The authors of [CHK+12] expect their multi-party GMW implementation to be roughly factor 2× slower in the two-party setting than the Yao's garbled circuits implementation of [HEKM11]. Our complexity comparison supports

this statement, since the total communication and computation per AND gate for the GMW protocol is approximately twice as large as for Yao's garbled circuits. Another disadvantage of the GMW protocol is its need for interaction for evaluating AND gates which becomes a performance bottleneck for networks with high latency. On the other hand sharing the inputs is cheaper in the GMW protocol as only one random bit needs to be chosen and sent to the other party, whereas in garbled circuits a random κ-bit key needs to be chosen and sent to the evaluator (using OT for evaluator's inputs).

2.5 Hashing Inputs to a Smaller Domain

The performance of some PSI protocols depends on the length of the representation of their inputs, e.g., the protocols described in §5.3.2 and §5.4.2. In some settings, inputs come from a small and densely populated domain, such as national identity numbers (e.g., social security numbers in the US, which can be represented by 30 bits) or credit card numbers (54 bits). In other settings, the input representation is sparse, for instance when a person is identified by an ASCII string containing his or her name, or when the number of inputs is very small, say 100.

When the original input representation is sparse, we can first use a hash function to map the identities of the input items to identities from a smaller domain with a shorter representation. We then run the original protocol on that representation, resulting in a more efficient execution. The size of the new domain should be large enough so that no two different input items are mapped to the same value. The theoretical analysis of this mapping, related to the birthday paradox, shows that when n items are mapped to a domain of size D using a random hash function, the probability of experiencing a collision is $p = 1 - e^{-n\cdot(n-1)/(2D)}$, and can be approximated as $p \approx n^2/(2D)$ (see [MR95], p. 45).

Let us denote the length of the representation of items in D as $d = \log D$. Then $p \approx n^2/(2 \cdot 2^d)$, and therefore

$$d = 2\log(n) - 1 - \log(p).$$

For example, for an input of $n = 90$ items and a collision probability of $p = 2^{-20}$, the representation can be $d = 32$ bits long. If we need the collision probability to be $p = 2^{-40}$ then we should set $d = 52$. Similarly, for $n = 10^6$, and collision probability $p = 2^{-20}$ the representation can be $d = 59$ bits long, and can fit in a single long word. For a collision probability of 2^{-40} we must use $d = 79$. Throughout this thesis, we use this observation to reduce the bit-length of hashes to achieve collision probability $p = 2^{-40}$.

3 Faster OT Extension

Practical applications in secure computation can require millions to billions of OTs. Efficient OT extension protocols are of great importance to meet this large demand. In this chapter, we show how to improve the efficiency of OT extension protocols in the semi-honest and malicious model.

Remark. Parts of this chapter have and will be published in [ALSZ13, ALSZ15, ALSZ16, DKS+17]. The author of this thesis has significantly contributed to the research results of these publications, in particular the protocols, optimizations, and evaluation. The security proof in §3.3.2 was done in collaboration with Gilad Asharov (IBM T.J. Watson Research Center, USA) and Yehuda Lindell (Bar-Ilan University, Israel) during a visit of the authors from TU Darmstadt at Bar-Ilan University. The implementations are available online at `http://encrypto.de/code/OTExtension`.

3.1 Motivation

OT is an extremely powerful tool and the foundation for almost all efficient protocols for secure computation. Notably, Yao's garbled-circuits protocol [Yao86] requires OT for every input bit of one party (cf. §2.4.1.1), and the GMW protocol [GMW87] requires OT for every AND gate of the circuit (cf. §2.4.2). Accordingly, the efficient instantiation of OT is of crucial importance as is evident in many recent works that focus on efficiency, e.g., [MNPS04, HKS+10, KSS12, NNOB12, GKK+12, CHK+12, NNOB12, LOS14, DLT14, FKOS15, BLN+15, KOS16]. The best known public-key-based OT protocol in the semi-honest and malicious case is that of [CO15], which achieves around 10 000 1-out-of-2 OTs per second using one thread. However, if millions or even billions of OTs need to be carried out, this becomes prohibitively expensive. We give concrete examples for typical applications requiring a large number of OTs next:

Example 3.1.1. *The AES circuit has* $\sim 5\,000$ *AND gates (cf. §4.2.1.12) and requires* $10\,000$ *passive secure OTs when evaluated with GMW and* $\sim 200\,000$ *active secure OTs when evaluated with TinyOT* (≥ 40 *OTs (aBits) per AND gate [LOS14]).*

Example 3.1.2. *The PSI circuit (Sort-Compare-Shuffle) of [HEK12] has* $\mathcal{O}(\sigma n \log n)$ *AND gates and for* $n = 65\,536$ *elements with* $\sigma = 32$ *bits the circuit has* 2^{25} *AND gates and requires* 2^{26} *passive secure OTs when evaluated with GMW and* $\sim 2^{30}$ *active secure OTs when evaluated with TinyOT.*

Example 3.1.3. *The PSI protocol of [DCW13] needs* $1.44\kappa n$ *OTs. For* $n = 1\ 000\ 000$ *elements and symmetric security parameter* $\kappa = 128$*, this amounts to* $\sim 2^{27}$ *OTs* (~ 180 *OTs per element).*

To meet this large-scale demand of OTs, *OT extensions* [Bea96, IKNP03] can be used. An OT extension protocol works by running a small number of base-OTs (say 128) that are used as a base for obtaining many OTs via the use of cheap symmetric cryptographic operations only. This is conceptually similar to hybrid encryption where instead of encrypting a large message using RSA, which would be too expensive, only a single RSA computation is carried out to encrypt a symmetric key and then the long message is encrypted using symmetric operations only. Such an OT extension can actually be achieved with extraordinary efficiency; specifically, the protocol of [IKNP03] requires only three evaluations of a symmetric cryptographic function per OT (beyond the initial base-OTs, cf. §2.3.1). For active adversaries, OT extensions are somewhat more expensive. Prior to this work, the best known protocol for OT extensions with security against active adversaries was introduced by [NNOB12], which added an overhead of approximately $\frac{8}{3}$ ($= 266\%$) to the passive secure OT extension protocol of [IKNP03].

3.1.1 Our Contributions

In this chapter, we present more efficient protocols for OT extensions in the semi-honest and malicious model. Our improvements in the semi-honest model (§3.2) seem somewhat surprising since the protocol of [IKNP03] sounds optimal given that only three symmetric cryptographic function computations are needed per transfer. Interestingly, our protocols do not lower the number of symmetric cryptographic operations. However, we observe that significant cost is incurred due to other factors than the symmetric cryptographic operations. We propose several optimizations that improve computation and communication and outline how to parallelize the semi-honest OT extension. We build on the efficiency improvements of the semi-honest OT extension protocol of [IKNP03] and outline how to extend the protocol to malicious adversaries at a lower cost than the previously best malicious secure OT extension protocol of [NNOB12] (§3.3). In short, our protocol improves the overhead that comes with extending the passive secure OT extension protocol of [IKNP03] to malicious adversaries from 266% to 150%. Finally, we outline different OT flavors that are specifically designed to be used in secure computation protocols and which reduce the communication and computation even further (§3.4). We apply our optimizations to the OT extension implementation of [CHK+12] and demonstrate the improvements by extensive experiments (§3.5). After presenting related work in §3.1.2, this chapter is structured as follows:

Faster Semi-Honest OT Extensions (§3.2). We present an improved version of the original OT extension protocol of [IKNP03] with reduced communication and com-

putation complexity. We demonstrate how the OT extension protocol can be processed in independent blocks, allowing OT extension to be parallelized and yielding a much faster run-time (§3.2.1). In addition, we describe how to implement the matrix transpose operation using a cache-efficient algorithm that operates on multiple entries at once (§3.2.2); this significantly reduces the run-time of the protocol to 41% as can be seen in the LAN experiments in Table 3.1. We then show how to reduce the communication from the receiver to the sender to 50% (§3.2.3). This is of great importance since local computations of the OT extension protocol are so fast that the communication is often the bottleneck, especially when running the protocol over the Internet or even wireless networks (cf. WAN results in Table 3.1 and Figure 3.2). Parallel to and independently of our work, [KK13] introduced an efficient 1-out-of-N OT ($\binom{N}{1}$ OT) extension protocol that achieves good communication when performing $\binom{2}{1}$ OT on short strings at the cost of increased computation (cf. §2.3.2). We improve the communication complexity of the [KK13] protocol for $\binom{2}{1}$ OT by approximately 10% (§3.2.4).

Faster Malicious OT Extensions (§3.3). We present our improved malicious OT extension protocol which improves on the previously best malicious OT extension protocol of [NNOB12]. We first present the basic protocol (§3.3.1) and prove its security (§3.3.2). The basic protocol adds very low communication overhead to the semi-honest version but incurs a high computation overhead. We then use the results from [ALSZ16] to reduce the computation at the cost of increased communication, which results in better overall efficiency. The resulting protocol decreases the communication overhead for obtaining active secure OT extension from 266% for [NNOB12] to 150%. In a subsequent work, [KOS15] outlined an OT extension protocol that achieves nearly the same communication and computation overhead as our passive secure OT extension protocol.

Extended OT Functionalities (§3.4). Our improved protocols can be used in any setting that regular OT can be used. However, with a mind on the application of secure computation, we further optimize the protocol by taking into account its use in secure computation. We outline four OT flavors that are specifically designed to be used in secure computation protocols and which reduce the communication and computation even further: *Correlated OT*, *Sender Random OT*, *Receiver Random OT*, and *Random OT*. Correlated OT (C-OT, §3.4.1) is suitable for secure computation protocols that require varying correlated inputs, such as Yao's garbled circuits protocol with the free-XOR technique [KS08]. Sender Random OT (SR-OT, §3.4.2) and Receiver Random OT (RR-OT, §3.4.3) are suitable where the input of the sender (or receiver) can be random but the input of the receiver (sender) needs to be non-random. Finally, Random OT (R-OT §3.4.4) is a combination of Sender Random and Receiver Random OT and can be used where the inputs of sender and receiver can be random, such as GMW with multiplication triples [GMW87, Bea91] (cf. §2.4.2). In most cases, the communication from the sender to the receiver is reduced to 50% (or even less) of the original protocol of [IKNP03].

Experimental Evaluation (§3.5). We experimentally verify the performance improvements of our proposed optimizations for OT extension and special-purpose OT functionalities in a LAN and a WAN setting. A summary of our results for 2^{24} random OT extensions on 1-bit strings using 4 threads is given in Table 3.1. Overall, our optimizations improve the run-time and communication of the passive secure OT extension protocol of [IKNP03] by factor $2\times - 3\times$ and $2\times$, respectively, and the run-time and communication for active secure OT extension by factor $1.3\times - 1.7\times$ and $1.7\times$, respectively.

Protocol	**Comm. [MB]**	**Run-Time [s]**		**Base-OTs**	**Security**
		LAN	**WAN**		
Semi-Honest					
[IKNP03]	508	9.2	39.9	128	CRF
[KK13]	160	-	-	256	RO
Opt. [KK13] **(§3.2.4)**	146	7.8	20.8	240	RO
This (§3.2)	254	3.8	18.8	128	CRF
Malicious					
[Lar14]	196 688*	-	-	323	CRF
[NNOB12]	682	9.1	50.4	342	RO
This (§3.3)	378	7.3	30.5	190	CRF / RO
[KOS15]	256*	-	-	128	RO

Table 3.1: Empirical communication and run-time for 2^{24} random OT extensions on 1-bit strings with κ=128-bit security evaluated using 4 threads in a LAN and WAN setting (cf. §3.5). The security assumption is given as correlation robust function assumption (CRF) or random oracle assumption (RO) cf. §2.2.2. Numbers with * are estimated.

3.1.2 Previous Works

In the following, we review works on OT extension in the semi-honest (§3.1.2.1) and malicious model (§3.1.2.2).

3.1.2.1 Semi-Honest OT Extension

In the semi-honest model, the protocol of [IKNP03] was implemented by the FastGC framework [HEKM11]. In [HS13], the memory footprint of the OT extension implementation in [HEKM11] was improved by splitting the OT extension protocol sequentially into multiple rounds and speedups were obtained by instantiating the pseudo-random generator with AES instead of SHA-1. In [KK13], a $\binom{N}{1}$ OT extension protocol was introduced that is based on the OT extension protocol of [IKNP03] and, for $\binom{2}{1}$ OT

on short strings, achieves sub-linear communication in the number of OTs. In particular, for $\binom{2}{1}$ OT on 1-bit strings, their protocol improves communication compared to [IKNP03] by factor 1.8× (cf. §2.3.3). This improvement in communication comes with an increased cost in computation, since the number of evaluations of the random oracle H for the sender is increased from $2\log_2(N)$ to N. In addition, [KK13] independently and in parallel introduced the same optimization for reducing the communication from the receiver to the sender by 50% that we propose in §3.2.3. In §3.2.4 we improve the [KK13] OT and empirically compare the optimized $\binom{N}{1}$ OT extension protocol to our $\binom{2}{1}$ OT extension protocol for $\binom{2}{1}$ OT on 1-bit strings in order to evaluate this computation / communication trade-off.

The above works all consider the concrete efficiency of OT extensions. The theoretical feasibility of OT extensions was established in [Bea96], and further theoretical foundations were laid in [LZ13]. [IKOS08] introduced a non-black-box technique for extending OTs with asymptotic constant computation / communication overhead. Their protocol assumes the existence of a polynomial stretch pseudo-random generator in NC^0, i.e., the set of functions that can be computed by a constant depth circuit with bounded fan-in where each output bit depends on a constant number of input bits. The high level idea of the protocol is to use the PRG in the scheme for extending OTs of [Bea96]. However, their scheme is extremely costly in concrete terms and the security of the PRG in NC^0 requires non-standard assumptions.

3.1.2.2 Malicious OT Extension

Due to its importance, a number of previous works have tackled the question of OT extensions with security for malicious/active adversaries. All of the known constructions build on the semi-honest protocol of [IKNP03], and add *consistency checks* of different types to the OT extension protocol, to ensure that the receiver sent consistent values. (Note that in [IKNP03], the sender cannot cheat and so it is only necessary to enforce honest behavior for the receiver.)

The first active secure version of OT extension used a cut-and-choose technique and was already given in [IKNP03]. This cut-and-choose technique achieves a security of $2^{-\rho}$ by performing ρ parallel evaluations of the basic OT extension protocol.

This was improved on by [Nie07, HIKN08], who show that active security can be achieved at a much lower cost. Their approach works in the random oracle model and ensures security against a malicious receiver by adding a low-cost check per extended OT, which uses the uncertainty of the receiver in the choice bit of the sender. As a result, a malicious receiver who wants to learn p choice bits of the sender risks being caught with probability 2^{-p}. However, this measure allows a malicious sender to learn information about the receiver's choice bits. They prevent this attack by combining $S \in \{2, 3, 4\}$ OTs and ensuring the security of one OT by sacrificing the remaining $S - 1$ OTs. Hence, their approach adds an overhead of at least $S \geq 2$ compared to the semi-honest OT extension protocol of [IKNP03] for a reasonable number of OTs (with

$S = 2$ and approximately 10^7 OTs, they achieve security except with probability 2^{-25}, cf. [Nie07]).

An alternative approach for achieving active secure OT extension was presented in [NNOB12]. Their approach also works in the random oracle model but, instead of performing checks per extended OT as in [Nie07, HIKN08], they perform consistency checks per base-OT. Their consistency check method involves hashing the strings that are transferred in the base-OTs and is highly efficient. In their approach, they ensure the security of a base-OT by sacrificing another base-OT, which adds an overhead of factor $2\times$. In addition, a malicious receiver is able to learn p choice bits of the sender in the base-OTs with probability 2^{-p}. [NNOB12] shows that this leakage can be tolerated by increasing the number of base-OTs from κ to $\lceil \frac{8}{3}\kappa \rceil$. The [NNOB12] protocol has been optimized and implemented on a GPU in [FN13].

An approach for achieving active secure OT extension that works in the standard model has been introduced in [Lar14]. Their approach achieves less overhead in the number of base-OTs at the expense of substantially more communication during the check routine (cf. Table 3.1 on page 28) and is therefore considerably less efficient. Nevertheless, we point out that the work of [Lar14] is of independent interest since it is based on the original correlation robustness assumption only.

Since it is the previous best, we compare our protocol to that of [NNOB12]. Our approach reduces the number of base-OTs by removing the "sacrifice" step of [NNOB12] (where one out of every 2 base-OTs are opened) but increases the workload in the consistency check routine. Indeed, we obtain an additive factor of a statistical security parameter, instead of the multiplicative increase of [NNOB12]. This can be seen as a trade-off between reducing communication through fewer base-OTs while increasing computation through more work in the consistency check routine. We empirically show that this results in a more efficient active secure OT extension protocol, which only has $60\% - 90\%$ more time and 50% more communication than the passive secure OT extension protocol of [IKNP03] in the LAN- and WAN setting compared to $90\% - 175\%$ more time and 166% more communication for [NNOB12] (cf. Table 3.1).

In [IPS08] it was shown how to achieve active secure OT extension with constant overhead from the passive secure protocol of [IKNP03]. Their approach involves the sender and receiver "simulating" additional parties and then running an outer secure computation protocol with security against honest majority. In addition, they show that their transformation can make black-box use of any passive secure OT protocol. Overall, this approach improves on the asymptotic communication of [HIKN08] but the exact constants involved in this approach have not been analyzed.

3.1.3 Follow-Up Works

Our work on semi-honest secure OT extension is considered state-of-the-art in works on practical secure computation protocols with over 70 citations according to Google Scholar. Our open source implementation has been used in works such as [BCP+14,

SLPR15, WFNL16, GLNP15, BK15, BB16, PSS17] and is a major building block of our ABY secure computation framework, presented in §4, and our PSI protocols, presented in §5.

Subsequently to our work on active secure OT extension in [ALSZ15], a new active secure OT extension protocol has been introduced [KOS15] which achieves nearly the same communication and computation overhead as our passive secure OT extension protocol protocol in §3.2. Their protocol is conceptually similar to ours (and to that of [NNOB12]) but performs the checks on the base-OTs in parallel instead of checking individual pairs. Thereby, the protocol of [KOS15] reduces the communication per extended OT to κ bit plus a small additive term that is independent of the number of OTs. Furthermore, their check routine can be implemented very efficiently using the AES new instructions (AES-NI), resulting in very little computational overhead over the passive secure variant. Note that there is a difference in security guarantees between the protocol of [KOS15] and our active secure OT extension protocol in §3.3. In both protocols, a malicious receiver can learn at most c bits of the sender's secret base-OT choices but gets caught except with probability 2^{-c}. If the receiver is successful, it can then issue q queries to an oracle and break the protocol with probability $q/2^{\kappa-c}$. In our malicious OT extension protocol, we therefore increase the number of base-OTs to $\kappa + \lambda$ while the protocol of [KOS15] keeps the number of base-OTs at κ. Hence, our active secure OT extension protocol provides κ-bit symmetric security while the active secure OT extension protocol of [KOS16] provides $(\kappa - \lambda)$-bit symmetric security against a malicious receiver that successfully guesses λ bits. When aligning the security of both protocols by performing $\kappa + \lambda$ base-OTs, the [KOS16] protocol would perform better than our protocol since it's checks are more computationally efficient.

Recently, two independent works [OOS17, PSS17] outlined how to achieve active security for the $\binom{N}{1}$ OT extension protocol of [KK13] at only a small communication overhead that is independent of the number of OTs. Previous works on active secure OT extension utilize certain features of the [IKNP03] protocol that do not hold for the more general protocol of [KK13] (cf. §2.3.2). The works of [OOS17, PSS17] both showed how to generalize the existing techniques for achieving active security for OT extension using ideas of [FJNT16].

3.2 Faster Semi-Honest OT

In the following we describe algorithmic optimizations that improve the scalability and computational complexity of OT extension protocols. We identified computational bottlenecks in OT extension by micro-benchmarking the $\binom{2}{1}$ OT extension [IKNP03] implementation of [CHK+12]. We found that the combined computation time of P_S and P_R was mostly spent on two operations: The matrix transposition (61%) and the evaluation of the correlation-robust function (CRF, cf. §2.2.4) in Step 2d and Step 2e in Protocol 1 on page 11, instantiated with SHA-256 (32%). The remaining time

was mostly spent on XOR operations (5%) and the evaluation of the pseudo-random generator (PRG, cf. §2.2.1) in Step 2a and Step 2b, instantiated with AES (2%). Furthermore, for networks with low bandwidth, the communication of OT quickly became the bottleneck. To speed up OT extension, we propose to use parallelization (§3.2.1), an efficient algorithm for bit-matrix transposition (§3.2.2), and a protocol optimization that allows to reduce the communication from P_R to P_S by half (§3.2.3). Note that these implementation optimizations are of general nature and can be applied to our, but also to other OT extension protocols with security against stronger active adversaries. Finally, we show how to optimize the computation and communication of the $\binom{N}{1}$ OT extension protocol of [KK13] using more efficient instantiations of the underlying primitives (§3.2.4).

3.2.1 Blockwise Parallelized OT Extension

The previous OT extension implementation of [CHK+12] improved the performance of OT extension by using a *vertical* pipelining approach, i.e., one thread is associated to each step of the protocol: The first thread evaluates the PRG and the second thread evaluates the CRF (cf. §2.2.4). However, since the PRG, which is instantiated using fixed-key AES, is much faster than the CRF, which is instantiated using SHA-256, the workload between the two threads is distributed unequally, causing idle time for the first thread. Additionally, this method for pipelining is designed to run exactly two threads and thus cannot easily be scaled to a larger number of threads.
As observed in [IKNP03, HS13], a large number of OT extensions can be performed by *sequentially* running the OT extension protocol on blocks of fixed size, which reduces the total memory consumption. We propose to use a *horizontal* pipelining approach that splits the matrices processed in the OT extension protocol into *independent* blocks that can be processed in parallel using multiple threads with equal workload, i.e., each of the T threads evaluates the OT extension protocol for $\frac{m}{T}$ of the m inputs in parallel.

3.2.2 Efficient Bit-Matrix Transposition

The computational complexity of cryptographic protocols is often measured by counting the number of invocations of cryptographic primitives, since their evaluation often dominates the overall run-time. However, non-cryptographic operations can also have a high impact on the overall run-time although they might seem insignificant in the protocol description. Matrix transposition is an example for such an operation. It is required during the OT extension protocol to transpose the $m \times \ell$ bit-matrices T and Q in Step 2d and Step 2e in Protocol 1 on page 11, which are created column-wise but processed row-wise. Although transposition is a seemingly trivial operation, it has to be performed individually for each entry in T and Q, making it a very costly operation.
We propose to efficiently implement the matrix transposition using Eklundh's algo-

rithm [Ekl72], which uses a divide-and-conquer approach to recursively swap elements of adjacent rows (cf. Figure 3.1). This decreases the number of swap operations for transposing a $n \times n$ matrix from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2 n)$. Additionally, since we process a bit-matrix, we can perform multiple swap operations in parallel by loading multiple bits into one register. Thereby, we again reduce the number of swap operations from $\mathcal{O}(n \log_2 n)$ to $\mathcal{O}(\lceil \frac{n}{r} \rceil \log_2 n)$, where r is the register size of the CPU ($r = 64$ for the machines used in our experiments). Jumping ahead to the evaluation in §3.5.2, this reduces the total time for the matrix transposition by approximately factor 22× from 17.4 s to 0.8 s per party for 2^{24} OTs and reduces the total time for the OTs from 30.4 s to 13.2 s when using a single thread.

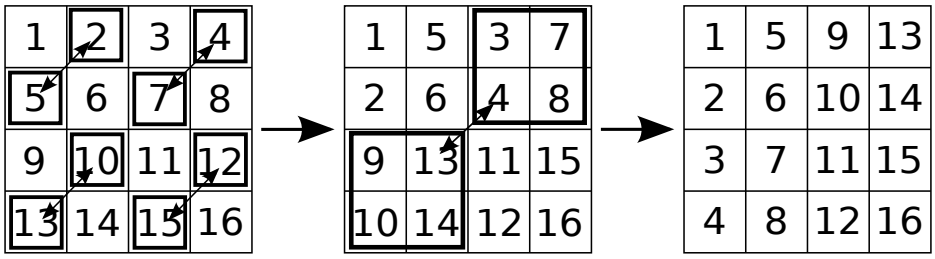


Figure 3.1: Efficient matrix transposition of a 4×4 matrix using Eklundh's algorithm.

3.2.3 Optimized Semi-Honest OT Extension

In the following, we optimize the OT_n^m extension protocol of [IKNP03], described in §2.3.1. Note that our optimization was independently outlined in [KK13]. Recall, that in the first step of the protocol in [IKNP03], P_R chooses a huge $m \times \ell$ matrix $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\kappa]$ while P_S waits idly (for the semi-honest OT extension protocol we can set $\ell = \kappa$; for the malicious OT extension protocol, ℓ needs to be increased). The parties then engage in a OT_m^ℓ protocol, where the inputs of the receiver are $(\mathbf{t}^i, \mathbf{t}^i \oplus \mathbf{r})$ where $\mathbf{r}$ is its input in the outer OT_n^m protocol (m choice bits). After the OT, P_S holds $\mathbf{t}^i \oplus (s_i \cdot \mathbf{r})$ for every $1 \leq i \leq \ell$. As described in the appendices of [IKNP03, HMEK11], the protocol can be modified such that P_R only needs to choose two small $\ell \times \kappa$ matrices $K_0 = [\mathbf{k}_1^0 | \dots | \mathbf{k}_\ell^0]$ and $K_1 = [\mathbf{k}_1^1 | \dots | \mathbf{k}_\ell^1]$ of seeds. These κ-bit seeds are used as input to the ℓ base-OTs; specifically P_R's input as sender in the i-th OT is $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ and, as in [IKNP03], the input of P_S is s_i. To transfer the m-bit tuple $(\mathbf{t}^i, \mathbf{t}^i \oplus \mathbf{r})$ in the i-th OT, P_R expands $\mathbf{k}_i^0$ and $\mathbf{k}_i^1$ using a PRG G, sends $(\mathbf{u}^{(i,0)}, \mathbf{u}^{(i,1)}) = (G(\mathbf{k}_i^0) \oplus \mathbf{t}^i, G(\mathbf{k}_i^1) \oplus \mathbf{t}^i \oplus \mathbf{r})$, and P_S recovers $G(\mathbf{k}_i^{s_i}) \oplus \mathbf{u}^{(i,s_i)}$.

Our main observation is that, instead of choosing $\mathbf{t}^i$ randomly, we can set $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Now, P_R needs to send only one m-bit element $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ to P_S (whereas in previous protocols of [IKNP03, HMEK11] two m-bit elements were sent). Observe that if P_S had input $s_i = 0$ in the i-th OT, then it can just define its output $\mathbf{q}^i$ to

PROTOCOL 6 (Our Optimized Semi-Honest Secure OT Extension Protocol)**.**

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \leq j \leq m$.
- **Input of P_R:** m choice bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and number of base-OTs $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the OT_κ^ℓ functionality and use a pseudo-random generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a correlation-robust function $H : [m] \times \{0,1\}^\ell \to \{0,1\}^n$ (cf. §2.2.4).

1. *Initial OT Phase:*
   a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in_R \{0,1\}^\ell$ and P_R randomly chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   b) The parties invoke the OT_κ^ℓ-oracle, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

   For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \leq i \leq \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \leq j \leq m$.
2. *OT Extension Phase*[a]:
   a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \leq i \leq \ell$.
   b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   c) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   d) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

   e) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- **Output:** P_S has no output; P_R outputs $(x_1, \ldots, x_m)$.

[a] This phase can be iterated. Specifically, P_R can compute the next κ bits of $\mathbf{t}^i$ and $\mathbf{u}^i$ (by applying G to get the next κ bits from the PRG for each of the seeds and using the next κ bits of its input in $\mathbf{r}$) and send the block of $\ell \times \kappa$ bits to P_S.

be $G(\mathbf{k}_i^0) = G(\mathbf{k}_i^{s_i})$. In contrast, if P_S had input $s_i = 1$ in the i-th OT, then it can define its output $\mathbf{q}^i$ to be $G(\mathbf{k}_i^1) \oplus \mathbf{u}^i = G(\mathbf{k}_i^{s_i}) \oplus \mathbf{u}^i$. Since $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, we have that $G(\mathbf{k}_i^1) \oplus \mathbf{u}^i = G(\mathbf{k}_i^0) \oplus \mathbf{r} = \mathbf{t}^i \oplus \mathbf{r}$, as required. The full description of our protocol is given in Protocol 6. This optimization is significant in applications of OT_n^m extension where m is very large and n is short, such as in GMW. In typical use-cases for GMW, m is in the size of several millions to a billion (cf. examples in §3.1), while n is one. Thereby, the communication complexity of GMW is almost reduced by half. In addition, observe that the initial OT phase in Protocol 6 is completely independent

of the actual inputs of the parties. Thus, the parties can compute the initial base-OTs before their inputs are determined.
Finally, another problem that arises in the original protocol of [IKNP03] is that the entire $m \times \ell$ matrix is transmitted together and processed. This means that the number of OTs to be obtained must be predetermined and, if m is very large, this results in considerable latency as well as memory management issues. As in [HS13], splitting the matrix into blocks which are processed in a pipelined fashion reduces latency, computation time, and avoids memory management problems. In addition, it is possible to continually extend OTs, with no a priori bound on m. This is very useful in a secure computation setting, where parties may interact many times together with no a priori bound. We state and prove security of our optimized protocol in [ALSZ16].

3.2.4 Faster 1-Out-of-N OT Extension

In the following, we improve the efficiency of the $\binom{N}{1}$ OT extension protocol of [KK13]. We improve the communication using size-optimized codes (§3.2.4.1) and the computation using pipelined AES implementations (§3.2.4.2).

3.2.4.1 Size-Optimized Codes

The communication efficiency of the $\binom{N}{1}$ OT extension protocol of [KK13] for $N > 2$ is directly related to the codeword bit-length ρ of the code Γ^ρ (cf. §2.3.2). For $2 < N \leq 2\kappa$, [KK13] uses a Walsh-Hadamard code, which has codewords of size $\rho = 2\kappa = 256$ bits to achieve a Hamming distance of $\kappa = 128$ between codewords. However, for $N = 2^i$, with $i \in [2, 8]$, the Walsh-Hadamard code is not size-optimal with regard to the codeword size ρ. Hence, we propose to use more size-efficient codes in order to further decrease the communication. We base our code choices on the list of efficient codes in [SS06] and give the codeword sizes for $N = 2^i$ for $i \in [1, 12]$ in Table 3.2. In particular, for $N = 4$ we use a parity check code, for $N \in \{8, 16, 32, 64, 128, 256\}$ we use a Simplex code, for $N = 512$ we use a Reed-Muller code, for $N \in [1\,024, 2\,048]$ we use a narrow-sense BCH-code, and for $N = 4\,096$, we use the concatenation of a Denniston code and a Simplex code (see [SS06] for more details). The OT communication improvements achieved by adopting our reduced codeword sizes are the largest for $N = 4$ (reduced by 64 bits, i.e., 25%) and decrease with N growing towards 256 (reduced by 1 bit, i.e., 0.4%).

In our work on private set intersection in §5, we encounter $N = 2^{70}$. In this case, we use the linear BCH code $[2^{77}, 512, 129]$, generated by [MZ06], which allows us to set $N = 2^{77}$ with $\rho = 512$ bit codewords and relative Hamming distance $\kappa = 128$.

N	**2**	**4**	**8**	**16**	**32**	**64**
Our Size-Efficient Codes [bits]	128	192	224	240	248	252
[KK13] Codes [bits]	128	256	256	256	256	256
N	**128**	**256**	**512**	**1024**	**2048**	**4096**
Our Size-Efficient Codes [bits]	254	255	256	264	268	270
[KK13] Codes [bits]	256	256	-	-	-	-

Table 3.2: Communication for $\binom{N}{1}$ OT with size-optimal codes [SS06] compared to those used in [KK13].

3.2.4.2 Pipelined AES-256 with Key Schedule

In OT extension [IKNP03, KK13], both parties process several value tuples that are correlated by a constant XOR offset using a CRF (Step 2d and Step 2e in Protocol 2 on page 12). While the CRF has traditionally been instantiated with a hash function, more efficient AES-based constructions have replaced it (cf. §2.2.4). When using the most efficient, fixed-key AES instantiation [BHKR13], the input is restricted to the block-length of AES, i.e., 128-bit, which suffices for the $\binom{2}{1}$ OT extension protocol of [IKNP03] when $\kappa = 128$-bit. However, in the $\binom{N}{1}$ OT extension of [KK13], we need to process codewords of size $\rho > 128$ for $N > 2$, which prevents the use of fixed-key AES. Falling back to a hash function or AES-256 with key schedule (AES-256+KS) [KSS12] greatly decreases performance by about an order of magnitude, as depicted in Table 3.3. Furthermore, the $\binom{N}{1}$ OT protocol requires N invocations of an expensive CRF (instantiated via AES-256+KS or SHA-256) as opposed to $2 \log N$ invocations of a cheaper CRF (instantiated via AES-128) when using $\binom{2}{1}$ OT. In particular, for our protocol in §4.4.2.3 we use $N = 256$, which requires 256 CRF invocations when using $\binom{N}{1}$ OT compared to 16 invocations when using $\binom{2}{1}$ OT. Using the AES-256+KS instantiation for $\binom{N}{1}$ OT and the pipelined AES-128 instantiation of [GLNP15] for $\binom{2}{1}$ OT, this results in a computational overhead of 480x.

Primitive	**Width**	**Sequential [ms]**	**Pipelined [ms]**
AES-128 [BHKR13]	128	158	54
AES-128+KS [GLNP15]	128	1 460	358
AES-256+KS [KSS12]	256	1 625	476
SHA-256	arbitrary	2 487	-

Table 3.3: Instantiations of a correlation-robust function with input width in bits, sequential, and pipelined run-time for 10^7 invocations, where KS denotes the key schedule.

We improve the performance of the CRF instantiation based on AES-256 with key schedule by pipelining the AES-256 key expansion and encryption as well as pipelining

multiple invocations of AES, as was done for AES-128 in [GLNP15]. Thereby, we manage to decrease the computation time for AES-256 by factor 4×, which reduces the computation overhead compared to $\binom{2}{1}$ OT from 480× to 140×. We show in §3.5.5 that when evaluating $\binom{2}{1}$ $\text{OT}_1^{2^{24}}$ using the [KK13] protocol, instantiating the CRF with pipelined AES-256 with key schedule reduces the run-time from 39 s to 12 s compared to a SHA-256 instantiation. For $\rho > 256$, we instantiate the CRF with SHA-256.

A promising line of research is given in [GM16], which outlines how to obtain cryptographic permutations with larger block sizes based on fixed-key AES-128. Due to security concerns, however, we refrain from using their instantiations but point it out as a future alternative to explore.

3.3 Faster Malicious OT

The key insight to understanding how to secure OT extension against malicious adversaries is to understand that a malicious party only has very limited possibilities for an attack. In fact, the original OT extension protocol of [IKNP03] already provides security against a malicious P_S. In addition, the only attack for a malicious P_R is in Step 2a of Protocol 6 on page 34, where P_R computes and sends $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ (cf. [IKNP03]). A malicious P_R could choose a different $\mathbf{r}$ for each $\mathbf{u}^i$ (for $1 \leq i \leq \ell$), and thereby extract P_S's choice bits $\mathbf{s}$. Hence, malicious security can be obtained if P_R can be forced to use the same choice bits $\mathbf{r}$ in all messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$.

3.3.1 Overview of Our Malicious Secure Protocol

All we add to the semi-honest protocol in Protocol 6 is a consistency check for the values $\mathbf{r}$ that are sent in Step 2a, and increase the number of base-OTs. Let $\mathbf{r}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{u}^i$, i.e., the value that is implicitly defined by $\mathbf{u}^i$. We observe that if the receiver P_R uses the same choice bits $\mathbf{r}^i$ and $\mathbf{r}^j$ for some distinct $i, j \in [\ell]^2$, they cancel out when computing their XOR, i.e., $\mathbf{u}^i \oplus \mathbf{u}^j = (\mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i) \oplus (\mathbf{t}^j \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j) = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1)$. After the base-OTs, P_S holds $G(\mathbf{k}_i^{s_i})$ and $G(\mathbf{k}_j^{s_j})$ and in Step 2a of Protocol 6, P_R computes and sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i$ and $\mathbf{u}^j = G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j$. Now note that P_S can compute the XOR of the strings it received in the base-OTs $G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j})$ as well as the "inverse" XOR of the strings received in the base-OTs $G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) = G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j$ if and only if P_R has correctly used $\mathbf{r}^i = \mathbf{r}^j$.

However, P_S cannot check whether the "inverse" XOR is correct, since it has no information about $G(\mathbf{k}_i^{\overline{s_i}})$ and $G(\mathbf{k}_j^{\overline{s_j}})$ (this is due to the security of the base-OTs that guarantees that P_S receives the keys $\mathbf{k}_i^{s_i}, \mathbf{k}_i^{s_j}$ only, and learns nothing about $\mathbf{k}_i^{\overline{s_i}}, \mathbf{k}_j^{\overline{s_j}}$). P_R cannot give these values to P_S since this will reveal its choice bits. However, P_R can send the hashes of these inverse values. Specifically, P_R commits to the XORs

of all strings $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q))$, for all combinations of $p, q \in \{0,1\}$. Now, given $h_{i,j}^{s_i,s_j}$, $h_{i,j}^{\overline{s_i},\overline{s_j}}$, P_S checks that $h_{i,j}^{s_i,s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$, and that $h_{i,j}^{\overline{s_i},\overline{s_j}} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_i^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j) = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}))$. This check passes if $\mathbf{r}^i = \mathbf{r}^j$ and $h_{i,j}^{p,q}$ were set correctly.

If a malicious P_R tries to cheat and has chosen $\mathbf{r}^i \neq \mathbf{r}^j$, it has to convince P_S by computing $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$ for all $p, q \in \{0,1\}$. However, P_S can check the validity of $h_{i,j}^{s_i,s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$ while P_R remains oblivious to s_i, s_j. Hence, P_R can only convince P_S by guessing s_i, s_j, computing $h_{i,j}^{s_i,s_j}$ correctly and setting $h_{i,j}^{\overline{s_i},\overline{s_j}} = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$, which P_R cannot do better than with probability $1/2$. This means that P_R can only successfully learn ρ bits but will be caught except with probability $2^{-\rho}$. The full description of our new protocol is given in Protocol 7. We give some more explanations regarding the possibility of the adversary to cheat during the consistency check in §3.3.2.

We note that learning few bits of the secret $\mathbf{s}$ does not directly break the security of the protocol once $|\mathbf{s}| > \kappa$. In particular, the values $\{H(j, \mathbf{t}_j \oplus \mathbf{s})\}_j$ are used to mask the inputs $\{x_j^{\overline{r_j}}\}_j$. Therefore, when H is modeled as a random oracle and enough bits of $\mathbf{s}$ remain hidden from the adversary, each value $H(j, \mathbf{t}_j \oplus \mathbf{s})$ is random, and the adversary cannot learn the input $x_j^{\overline{r_j}}$. For simplicity we first prove security of our protocol in the random oracle model. We show in [ALSZ16] that H can be replaced with a variant of a correlation-robustness assumption.

The advantage of our protocol over [NNOB12] is that P_S does not need to reveal any information about s_i, s_j when checking the consistency between r^i and r^j (as long as P_R does not cheat, in which case it risks getting caught). Hence, it can force P_R to check that $\mathbf{r}^i$ equals any $\mathbf{r}^j$, for $1 \leq j \leq \ell$ without disclosing any information.

3.3.2 The Security of Our Protocol

In the following, we prove the security of our protocol.

Malicious Sender. The original OT extension protocol of [IKNP03] already provides security against a malicious P_S. Our checks do not add any capabilities for a malicious sender, since they consist of messages from the receiver to the sender only. Thus, by a simple reduction to the original protocol, one can show that our protocol is secure in the presence of a malicious sender.

Simulating a Malicious Receiver. In the case of a malicious receiver, the adversary may not use the same $\mathbf{r}$ in the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$, and as a result learn some bits from the secret $\mathbf{s}$. Therefore, we add a consistency check of $\mathbf{r}$ to the semi-honest protocol of [IKNP03]. However, this verification of consistency of $\mathbf{r}$ is not perfectly sound, and the verification may still pass even when the receiver sends few $\mathbf{u}$'s that do not define the same $\mathbf{r}$. This makes the analysis a bit more complicated.

PROTOCOL 7 (Our Active Secure OT Extension Protocol).

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \le j \le m$.
- **Input of P_R:** m choice bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ, statistical security parameter λ, and number of base-OTs $\ell = \kappa + \lambda$.
- **Oracles and cryptographic primitives:** The parties have oracle access to the active secure OT_κ^ℓ functionality, and use a pseudo-random generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a random oracle H.

1. *Initial OT Phase:*
   a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in_R \{0,1\}^\ell$ and P_R randomly chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   b) The parties invoke the OT_κ^ℓ oracle, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \le i \le \ell$.

   For every $1 \le i \le \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \le i \le \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \le j \le m$.
2. *OT Extension Phase (Part I):*
   a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \le i \le \ell$.
3. *Consistency Check of $\mathbf{r}$:* (the main change from the passive secure protocol)
   a) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_R defines the four values:

$$h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0)) \ , \qquad h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)) \ ,$$
$$h_{\alpha,\beta}^{1,0} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0)) \ , \qquad h_{\alpha,\beta}^{1,1} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^1)) \ .$$

      It then sends $\mathcal{H}_{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$ to P_S.
   b) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_S knows $s_\alpha, s_\beta, \mathbf{k}_\alpha^{s_\alpha}, \mathbf{k}_\beta^{s_\beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta$ and checks that:
      i. $h_{\alpha,\beta}^{s_\alpha,s_\beta} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$.
      ii. $h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$ $\quad (= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
      iii. $\mathbf{u}^\alpha \ne \mathbf{u}^\beta$.

      In case one of these checks fails, P_S aborts and outputs $\perp$.
4. *OT Extension Phase (Part II):*
   a) For every $1 \le i \le \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   b) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   c) P_S sends (y_j^0, y_j^1) for every $1 \le j \le m$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

   d) For $1 \le j \le m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.

- **Output:** P_S has no output; P_R outputs $(x_1, \ldots, x_m)$.

For every $1 \leq i \leq \ell$, let $\mathbf{r}^i \stackrel{\text{def}}{=} \mathbf{u}^i \oplus G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1)$ that is, the "input" $\mathbf{r}^i$ which is implicitly defined by $\mathbf{u}^i$ and the base-OTs.

We now explore how the matrices Q, T are changed when the adversary uses inconsistent $\mathbf{r}$'s. Recall that when the receiver uses the same $\mathbf{r}$, then $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$. However, in case of inconsistent $\mathbf{r}$'s, we get that $\mathbf{q}^i = (s_i \cdot \mathbf{r}^i) \oplus \mathbf{t}^i$. The case of $\mathbf{q}_j$ is rather more involved; let $R = \left[\mathbf{r}^1 \mid \ldots \mid \mathbf{r}^\ell\right]$ denote the $m \times \ell$ matrix where its ith column is $\mathbf{r}^i$, and let $\mathbf{r}_j$ denote the jth row of the matrix R. For two strings of the same length $\mathbf{a} = (a_1, \ldots, a_k), \mathbf{b} = (b_1, \ldots, b_k)$, let $\mathbf{a} * \mathbf{b}$ define the entry-wise product, that is $\mathbf{a} * \mathbf{b} = (a_1 \cdot b_1, \ldots, a_k \cdot b_k)$. We get that $\mathbf{q}_j = (\mathbf{r}_j * \mathbf{s}) \oplus \mathbf{t}_j$ (note that in an honest execution, $\mathbf{r}_j$ is the same bit everywhere). The sender masks the inputs (x_j^0, x_j^1) with $(H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s}))$.

In order to understand better the value $\mathbf{q}_j$, let $\mathbf{r} = (r_1, \ldots, r_m)$ be the string that occurs the most from the set $\{\mathbf{r}^1, \ldots, \mathbf{r}^\ell\}$, and let $\mathcal{U} \subset [\ell]$ be the set of all indices for which $\mathbf{r}^i = \mathbf{r}$ for all $i \in \mathcal{U}$. Let $B = [\ell] \setminus \mathcal{U}$ be the complementary set, that is, the set of all indices for which for every $i \in B$ it holds that $\mathbf{r}^i \neq \mathbf{r}$. As we will see below, except with some negligible probability, the verification phase guarantees that $|\mathcal{U}| \geq \ell - \lambda$. Thus, for every $1 \leq j \leq m$, the vector $\mathbf{r}_j$ (which is the jth row of the matrix R), can be represented as $\mathbf{r}_j = (r_j \cdot \mathbf{1}) \oplus \mathbf{e}_j$, where $\mathbf{1}$ is the all one vector of size ℓ, and $\mathbf{e}_j$ is some error vector with Hamming distance at most λ from $\mathbf{0}$. Note that the non-zero indices in $\mathbf{e}_j$ are all in B. Thus, we conclude that:

$$\mathbf{q}_j = (\mathbf{s} * \mathbf{r}_j) \oplus \mathbf{t}_j = (\mathbf{s} * (r_j \cdot \mathbf{1} \oplus \mathbf{e}_j)) \oplus \mathbf{t}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j) \ .$$

Recall that in an honest execution $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$, and therefore the only difference is the term $(\mathbf{s} * \mathbf{e}_j)$. Moreover, note that $\mathbf{s} * \mathbf{e}_j$ completely hides all the bits of $\mathbf{s}$ that are in $\mathcal{U}$, and may expose only the bits that are in B. Thus, the consistency check of $\mathbf{r}$ guarantees two important properties: First, that almost all the inputs are consistent with some implicitly defined string $\mathbf{r}$, and thus the bits r_j are uniquely defined. Second, the set of inconsistent inputs (i.e., the set B) is small, and thus the adversary may learn only a limited amount of bits of $\mathbf{s}$.

The Consistency Checks of r. We now examine what properties are guaranteed by our consistency check, for a single pair (α, β). The malicious receiver P_R first sends the set of keys $\mathcal{K} = \{\mathbf{k}_i^0, \mathbf{k}_i^1\}$ to the base-OT protocol, and then sends all the values $(\mathbf{u}^1, \ldots, \mathbf{u}^\ell)$ and the checks $\mathcal{H} = \{\mathcal{H}_{\alpha,\beta}\}_{\alpha,\beta}$. In the simulation, the simulator can choose $\mathbf{s}$ only after it receives all these messages (this is because the adversary gets no output from the invocation of the OT primitive). Thus, for a given set of messages that the adversary outputs, we can ask what is the number of secrets $\mathbf{s}$ for which the verification will pass, and the number for which it will fail. If the verification passes for some given $\mathcal{T} = (\mathcal{K}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \mathcal{H})$ and some secret $\mathbf{s}$, then we say that $\mathcal{T}$ is consistent with $\mathbf{s}$; In case the verification fails, we say that $\mathcal{T}$ is inconsistent.

In the following, let $\mathcal{T}_{\alpha,\beta}$ denote all messages that the receiver sends and which are rel-

evant for the verification of the pair (α, β), that is, $\mathcal{T}_{\alpha,\beta} = \left(\mathbf{k}_\alpha^0, \mathbf{k}_\alpha^1, \mathbf{k}_\beta^0, \mathbf{k}_\beta^1, \mathbf{u}^\alpha, \mathbf{u}^\beta, \mathcal{H}_{\alpha,\beta}\right)$. Note that $\mathcal{T}$, the set of all messages that the receiver sends, is defined as $\mathcal{T} = \bigcup_{\alpha,\beta} \mathcal{T}_{\alpha,\beta} = (\mathcal{K}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \mathcal{H})$, exactly as considered above.

The following Lemma considers the values that the adversary has sent regarding some pair (α, β), and considers the relation to the pair of bits (s_α, s_β) of the secret $\mathbf{s}$. We have:

Lemma 3.3.1. *Let $\mathcal{T}_{\alpha,\beta} = \{\{\boldsymbol{k}_\alpha^b\}_b, \{\boldsymbol{k}_\beta^b\}_b, \boldsymbol{u}^\alpha, \boldsymbol{u}^\beta, \mathcal{H}_{\alpha,\beta}\}$ and H be a collision-resistant hash-function. Then, the following holds, except with negligible probability:*

1. *If $\boldsymbol{r}^\alpha \neq \boldsymbol{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β), then it is inconsistent with $(\overline{s_\alpha}, \overline{s_\beta})$.*

2. *If $\boldsymbol{r}^\alpha = \boldsymbol{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β), then it is consistent also with $(\overline{s_\alpha}, \overline{s_\beta})$.*

Proof: For the first item, assume that $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and that $\mathcal{T}_{\alpha,\beta}$ is consistent both with (s_α, s_β) and $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, from the check of consistency of (s_α, s_β):

$$h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})\right), \qquad h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} = H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) ,$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. In addition, from the check of consistency of $(\overline{s_\alpha}, \overline{s_\beta})$ it holds that:

$$h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}})\right), \qquad h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) ,$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. This implies that:

$$H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})\right) = h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) ,$$

and from the collision resistance property of H, except for some negligible probability we get that:

$$G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) = G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta .$$

Recall that $\mathbf{r}^\alpha \stackrel{\text{def}}{=} \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$, and $\mathbf{r}^\beta \stackrel{\text{def}}{=} \mathbf{u}^\beta \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$. Combining the above, we get that $\mathbf{r}^\alpha = \mathbf{r}^\beta$, in contradiction.

For the second item, once $\mathbf{r}^\alpha = \mathbf{r}^\beta$, we get that $\mathbf{u}^\alpha \oplus \mathbf{u}^\beta = G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$ and it is easy to see that if the consistency check of (s_α, s_β) holds, then the consistency check of $(\overline{s_\alpha}, \overline{s_\beta})$ holds also. ■

Lemma 3.3.1 implies what attacks the adversary can do, and what bits of $\mathbf{s}$ it can learn from each such an attack. In the following, we consider a given partial transcript $\mathcal{T}_{\alpha,\beta} = ((\mathbf{k}_\alpha^0, \mathbf{k}_\alpha^1, \mathbf{k}_\beta^0, \mathbf{k}_\beta^1), (\mathbf{u}^\alpha, \mathbf{u}^\beta), \mathcal{H}_{\alpha,\beta})$ and analyze what the messages might be, and what the adversary learns in case the verification passes. Let $\mathbf{r}^\alpha = \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$ and $\mathbf{r}^\beta$ defined analogously. We consider 4 types:

1. **Type 1: $\mathbf{r}^{\alpha} = \mathbf{r}^{\beta}$ and $\mathcal{H}_{\alpha,\beta}$ is correct.** That is, for every $(a,b) \in \{0,1\}^2$: $h_{\alpha,\beta}^{a,b} = H\left(G(\mathbf{k}_{\alpha}^{a}) \oplus G(\mathbf{k}_{\beta}^{b})\right)$. In this case, the verification passes for every possible value of (s_{α}, s_{β}).

2. **Type 2: $\mathbf{r}^{\alpha} = \mathbf{r}^{\beta}$, but $\mathcal{H}_{\alpha,\beta}$ is incorrect.** In this case, the adversary sent $\mathbf{u}^{\alpha}, \mathbf{u}^{\beta}$ that define the same $\mathbf{r}$. However, it may send hashes $\mathcal{H}_{\alpha,\beta}$ that are incorrect (i.e., for some $(a,b) \in \{0,1\}^2$, it may send: $h_{\alpha,\beta}^{a,b} \neq H(G(\mathbf{k}_{\alpha}^{a}) \oplus G(\mathbf{k}_{\beta}^{b})))$. However, from Lemma 3.3.1, if $\mathbf{r}^{\alpha} = \mathbf{r}^{\beta}$ and $\mathcal{H}_{\alpha,\beta}$ is consistent with (s_{α}, s_{β}) then it is also consistent with $(\overline{s_{\alpha}}, \overline{s_{\beta}})$.

   Thus, a possible attack of the adversary, for instance, is to send correct hashes for some bits $(0,0)$ and $(1,1)$, but incorrect ones for $(0,1)$ and $(1,0)$. The verification will pass with probability $1/2$, exactly if (s_{α}, s_{β}) are either $(0,0)$ or $(1,1)$, but it will fail in the other two cases (i.e., $(1,0)$ or $(0,1)$). We therefore conclude that the adversary may learn the relation $s_{\alpha} \oplus s_{\beta}$, and gets caught with probability $1/2$.

3. **Type 3: $\mathbf{r}^{\alpha} \neq r^{\beta}$ and $\mathcal{H}_{\alpha,\beta}$ is incorrect in two positions.** In this case, for instance, the adversary can set the values $h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}$ correctly (i.e., $h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_{\alpha}^{0}) \oplus G(\mathbf{k}_{\beta}^{0}))$ and $h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_{\alpha}^{0}) \oplus G(\mathbf{k}_{\beta}^{1})))$ and set values $h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1}$, accordingly, such that the verification will pass for the cases of $(s_{\alpha}, s_{\beta}) = (0,0)$ or $(0,1)$. That is, it sets:

$$h_{\alpha,\beta}^{1,0} = H(G(\mathbf{k}_{\alpha}^{0}) \oplus G(\mathbf{k}_{\beta}^{1}) \oplus \mathbf{u}^{\alpha} \oplus \mathbf{u}^{\beta})$$

   (and it sets $h_{\alpha,\beta}^{1,1}$ in a similar way). In this case, the adversary succeeds with probability $1/2$ and learns that $s_{\alpha} = 0$ in case the verification passes. Similarly, it can guess the value of s_{β} and set the values accordingly. In conclusion, the adversary can learn whether its guess was correct, and in which case it learns exactly one of the bits s_{α} or s_{β} but does not learn anything about the other bit.

   In case where $\mathcal{H}_{\alpha,\beta}$ is correct in only one position but $\mathbf{r}^{\alpha} \neq \mathbf{r}^{\beta}$, the probability of success is even smaller. For instance, assume that $h_{\alpha,\beta}^{a,b} = H(G(\mathbf{k}_{\alpha}^{a}) \oplus G(\mathbf{k}_{\beta}^{b}))$ for $(a,b) = (0,0), (0,1), (1,0)$, but the adversary sends $h_{\alpha,\beta}^{1,1}$ incorrectly as above. In this case, the verification will fail for $(s_{\alpha}, s_{\beta}) = (1,1)$ and, in addition, also for the cases where $(s_{\alpha}, s_{\beta}) = (0,1)$ or $(1,0)$, since $\mathbf{r}^{\alpha} \neq \mathbf{r}^{\beta}$. Similarly, for the case where $\mathcal{H}_{\alpha,\beta}$ is incorrect in only one position, for which the adversary only succeeds with probability $1/2$. Therefore, it is more beneficial for the adversary to send two positions incorrectly.

4. **Type 4: $\mathbf{r}^{\alpha} \neq \mathbf{r}^{\beta}$ and $\mathcal{H}_{\alpha,\beta}$ is incorrect in three positions.** In this case, the adversary may guess both bits $(s_{\alpha}, s_{\beta}) = (a,b)$ and set $h_{\alpha,\beta}^{a,b}$ correctly, set $h_{\alpha,\beta}^{\overline{a},\overline{b}}$ accordingly (i.e., such that the verification will pass for (a,b)), but will fail for

any one of the other cases. In this case, the adversary learns the values (s_α, s_β) entirely, but succeeds with probability at most 1/4.

Note that whenever $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$, the adversary may pass the verification of the pair (α, β) with probability at most 1/2. This is because it cannot send consistent hashes for all possible values of (s_α, s_β), and must, in some sense, "guess" either one of the bits, or both (i.e., Type 3 or Type 4). However, an important point that makes the analysis more difficult is the fact that the two checks are not necessarily independent. That is, in case where $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathbf{r}^\beta \neq \mathbf{r}^\gamma$, although the probability to pass each one of the verification of (α, β) and (β, γ) separately is at most 1/2, the probability to pass both verifications together is higher than 1/4, and these two checks are not independent. This is because the adversary can guess the bit s_β, and set the hashes as in Type 3 in both checks. The adversary will pass these two checks if it guesses s_β correctly, with probability 1/2.

Proving Security of the Protocol. Before proceeding to the full proof of security, we first provide a proof sketch. The simulator $\mathcal{S}$ invokes the malicious receiver and plays the role of the base-OT trusted party and the honest sender. It receives from the adversary its inputs to the base-OTs, and thus knows the values $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell$. Therefore, it can compute all the values $\mathbf{r}^1, \ldots, \mathbf{r}^\ell$ when it receives the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$. It computes the set of indices $\mathcal{U}$, and extracts $\mathbf{r}$. It then performs the same checks as an honest sender, in Step 3 of Protocol 7, and aborts the execution if the adversary is caught cheating. Then, it sends the trusted party the value $\mathbf{r}$ that it has extracted, and learns the inputs $x_1^{r_1}, \ldots, x_m^{r_m}$. It computes $\mathbf{q}_j$ as instructed in the protocol (recall that these $\mathbf{q}_j$ may contain the additional "shift" $\mathbf{s} * \mathbf{e}_j$) and use some random values for all $\{y_j^{\overline{r_j}}\}_{j=1}^m$.

Since the values $\{y_j^{\overline{r_j}}\}_{j=1}^m$ are random in the ideal execution, and equal $\{x_j^{\overline{r_j}} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})\}$ in the real execution, a distinguisher may distinguish between the real and ideal execution once it makes a query of the form $(j, \mathbf{q}_j \oplus \mathbf{s})$ to the random oracle. We claim, however, that the probability that the distinguisher will make such a query is bounded by $(t+1)/|S|$, where t is the number of queries it makes to the random oracle, and S is the set of all possible secrets $\mathbf{s}$ that are consistent with the view that it receives. Thus, once we show that $|S| > 2^\kappa$, the probability that it will distinguish between the real and ideal execution is negligible in κ.

However, the above description is too simplified. First, if the adversary performs few attacks of Type 2, it learns information regarding $\mathbf{s}$ from the mere fact that the verification has passed. Moreover, recall that $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$, and that the adversary can control the values $\mathbf{t}_j$ and $\mathbf{e}_j$. Recall that $\mathbf{e}_j$ is a vector that is all zero in positions that are in $\mathcal{U}$, and may vary in positions that are in B. This implies that by simple queries to the random oracle, and by choosing the vectors $\mathbf{e}_j$ cleverly, the adversary can totally reveal the bits $\mathbf{s}_B$ quite easily. We therefore have to show that the set B is small, while also showing that the set of consistent secrets is greater

than 2^κ (that is, $|S| \geq 2^\kappa$). We now proceed to a formal statement of the theorem and formal proof of security, where there we prove the two informal claims that were just mentioned.

Theorem 3.3.2. *Assume that H is a random oracle and that G is a pseudo-random generator. Then, Protocol 7 with $\ell = \kappa + \lambda$ securely computes the OT_n^m functionality in the OT_κ^ℓ-hybrid model in the presence of a static malicious adversary, where κ is the symmetric security parameter and λ is the statistical security parameter.*

Proof:

Recall that security against a malicious sender can be proven by a simple reduction to the original OT extension protocol of [IKNP03], which is already secure against malicious sender, using the fact that our checks consist of messages that go from the receiver to the sender only. In the following we will give the proof for a malicious receiver. Since we already gave some proof sketch, we start directly with a formal description of the simulator $\mathcal{S}$:

The Simulator $\mathcal{S}$.

1. The simulator invokes the adversary $\mathcal{A}$ on the auxiliary input z.
2. **Initial OT phase:** The adversary $\mathcal{A}$ outputs ℓ pairs of κ-bits each $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell$ as input to the $\ell \times OT_\kappa$-functionality. It receives no output from this invocation.
3. **First part of OT extension phase:** The adversary $\mathcal{A}$ outputs ℓ strings $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$.
4. **Consistency check of r:**
   a) For every $\alpha, \beta \in [\ell]^2$, the adversary $\mathcal{A}$ outputs the quadruple $H^{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$.
   b) The simulator chooses a string $\mathbf{s}$ uniformly at random from $\{0,1\}^\ell$.
   c) Given the values $\{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \{H^{\alpha,\beta}\}_{\alpha,\beta}\}$ and the chosen secret $\mathbf{s}$, the simulator can perform all the needed checks as the honest sender in the real execution. In case where one of the verification fails, the simulator halts.
5. **Second part of the OT extension phase:**
   a) The simulator computes the matrices T, Q and R, where for every i, $\mathbf{t}^i = G(\mathbf{k}_i^0)$, $\mathbf{q}^i = (s_i \cdot \mathbf{r}^i) \oplus \mathbf{t}^i$ and $\mathbf{r}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{u}^i$.
   b) From all the vectors $\mathbf{r}^1, \ldots, \mathbf{r}^\ell$, let $\mathbf{r}$ be the vector that is mostly repeated (as we will see, the verification process guarantees that there exists a vector that is repeated at least $\ell - \lambda$ times).

c) Send $\mathbf{r}$ to the trusted party, and receive $x_1^{r_1}, \ldots, x_m^{r_m}$. Define the values $\mathbf{e}_j$ for every $1 \leq j \leq m$ (explicitly, define the matrix R as the matrix for which its ith column is $\mathbf{r}^i$, and let $\mathbf{r}_j$ denote its jth row. Then, $\mathbf{e}_j = (r_j \cdot \mathbf{1}) \oplus \mathbf{r}_j$. Then, for every $1 \leq j \leq m$, set $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$, and set $y_j^{\overline{r_j}}$ uniformly at random. Send $\{(y_j^0, y_j^1)\}_{j=1}^m$ to the adversary $\mathcal{A}$, output whatever it outputs and halt.

Let $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \{H_{\alpha,\beta}\}_{\alpha,\beta}\}$, i.e., the values that the adversary gives during the execution of the protocol. Observe that the simulator chooses the secret $\mathbf{s}$ only after $\mathcal{T}$ is determined (since the adversary receives no output from the execution of the OT primitive, we can assume that). We divide all possible choices of $\mathcal{T}$ into two sets, $\mathcal{T}_{\mathsf{good}}$ and $\mathcal{T}_{\mathsf{bad}}$, defined as follows:

$$\mathcal{T}_{\mathsf{good}} = \left\{\mathcal{T} \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] > 2^{-\lambda}\right\}$$
$$\text{and}$$
$$\mathcal{T}_{\mathsf{bad}} = \left\{\mathcal{T} \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] \leq 2^{-\lambda}\right\},$$

where $\mathsf{consistent}(\mathcal{T}, \mathbf{s})$ is a predicate that gets 1 when the verification passes for the transcript $\mathcal{T}$ and the secret $\mathbf{s}$, and 0 otherwise. The probability is taken over the choice of $\mathbf{s}$. For a given $\mathcal{T}$, let $\mathcal{S}(\mathcal{T})$ be the set of all possible secrets $\mathbf{s} \in \{0,1\}^\ell$, that are consistent with $\mathcal{T}$. That is: $\mathcal{S}(\mathcal{T}) = \{\mathbf{s} \in \{0,1\}^\ell \mid \mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\}$. Therefore, it holds that:

$$\Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] = \frac{|\mathcal{S}(\mathcal{T})|}{2^\ell}$$

and thus $|\mathcal{S}(\mathcal{T})| = 2^\ell \cdot \Pr\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right]$. As a result, for every $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, it holds that $|\mathcal{S}(\mathcal{T})| > 2^\ell \cdot 2^{-\lambda} = 2^{\ell-\lambda} = 2^\kappa$. That is, in case a transcript $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ passes the consistency check of $\mathbf{r}$, there are at least 2^κ different secrets $\mathbf{s}$ that are consistent with the given transcript, each are likely with the same probability, and thus the adversary may guess $\mathbf{s}$ with probability at most $2^{-\kappa}$.

Let $\mathcal{U}$ be the largest set of indices such that for every $i, j \in \mathcal{U}$, $\mathbf{r}^i = \mathbf{r}^j$. Let B be the complementary set, that is, $B = [\ell] \setminus \mathcal{U}$. From the definition of the sets, for every $\alpha \in \mathcal{U}$ and $\beta \in B$, it holds that $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$.

We claim that if $|\mathcal{U}| < \ell - \lambda$ (i.e., $|B| > \lambda$), then it must hold that $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$ and the adversary gets caught with high probability. That is:

Claim 3.3.3. *Let $\mathcal{T}$ be as above, and let $\mathcal{U}$ be the largest set of indices such that for every $\alpha, \beta \in \mathcal{U}$, $\boldsymbol{r}^\alpha = \boldsymbol{r}^\beta$. Assume that $|\mathcal{U}| < \ell - \lambda$. Then:*

$$\Pr_{\boldsymbol{s}}\left[\mathit{consistent}(\mathcal{T}, \boldsymbol{s}) = 1\right] \leq 2^{-\lambda}$$

and thus, $\mathcal{T} \in \mathcal{T}_{\mathit{bad}}$.

We will prove the claim below. Let $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, and let $\mathcal{U}$ and B be as above. Using the claim above, we have that $|B| < \lambda$. We now focus on the set of secrets $\mathbf{s}$ that are

consistent with this transcript $\mathcal{T}$, i.e., the set $S(\mathcal{T})$. For a set of indices A, we let $\mathbf{s}_A$ denote all the bits in $\mathbf{s}$ with indices from A, that is $\mathbf{s}_A = (\mathbf{s}_a)_{a\in A}$. We now claim that the bits $\mathbf{s}_B$ are common to all the secrets in $S(\mathcal{T})$, and thus, even when we give the adversary the bits $\mathbf{s}_B$ in the clear once the verification is completed, the adversary still has to guess $\mathbf{s}$ from a set of at least 2^κ secrets. Formally:

Claim 3.3.4. *Let $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, and let $\mathcal{U}$ and B as above. Then, there exists a string $w \in \{0,1\}^{|B|}$, such that for every $\boldsymbol{s}' \in S(\mathcal{T})$, it holds that: $\boldsymbol{s}'_B = w$.*

Proof: From the definition of the sets B and $\mathcal{U}$, it holds that for every $\alpha \in \mathcal{U}$ and $\beta \in B$, $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$. Consider two secrets $\mathbf{s}, \mathbf{s}'$ that are consistent with $\mathcal{T}$ (since $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, there are many such $\mathcal{T}$'s). We show the following:

- If there exists an index $\beta \in B$ such that $s_\beta \neq s'_\beta$, then for every $\alpha \in \mathcal{U}$ it holds that $\mathbf{s}_\alpha = s'_\alpha$ (that is, $\mathbf{s}_\mathcal{U} = \mathbf{s}'_\mathcal{U}$).
- Similarly, if there exists an index $\alpha \in \mathcal{U}$ such that $s_\alpha \neq s'_\alpha$ then for every $\beta \in B$ it holds that $s_\beta = s'_\beta$ (that is, $\mathbf{s}_B = \mathcal{S}'_B$).

We show the first claim. Assume that $\mathbf{s}_B \neq \mathbf{s}'_B$, thus, there must exist an index $\beta \in B$ such that $s_\beta \neq s'_\beta$, i.e., $s'_\beta = \overline{s_\beta}$. Now, consider some $\alpha \in \mathcal{U}$, we show that $s_\alpha = s'_\alpha$ and thus $\mathbf{s}_\mathcal{U} = \mathbf{s}'_\mathcal{U}$. Recall that $\mathcal{T}$ is consistent with $\mathbf{s}$, and therefore is consistent with (s_α, s_β). From Lemma 3.3.1, it is *inconsistent* with $(\overline{s_\alpha}, \overline{s_\beta}) = (\overline{s_\alpha}, s'_\beta)$. However, recall that $\mathcal{T}$ is consistent also with $\mathbf{s}'$, and therefore it is consistent with (s'_α, s'_β). We therefore conclude that it must hold that $\overline{s_\alpha} \neq s'_\alpha$ and thus $s_\alpha = s'_\alpha$. The second claim is proven analogously.

We now claim that the set $S(\mathcal{T})$ either shares the same $\mathbf{s}_B$ for all its elements, or shares the same $\mathbf{s}_\mathcal{U}$ for all elements (and of course, not both). In order to see this, let $S(\mathcal{T}) = \{\mathbf{s}^1, \ldots, \mathbf{s}^n\}$. Assume, without loss of generality, that $\mathbf{s}^1_\mathcal{U} \neq \mathbf{s}^2_\mathcal{U}$ (and so, from above, $\mathbf{s}^1_B = \mathbf{s}^2_B$). We now claim that all the other elements share the same bits in B. If not, that is, if there exists an element $\mathbf{s}^i \in \mathcal{S}(\mathcal{T})$ such that $\mathbf{s}^i_B \neq \mathbf{s}^1_B$, it must hold that $\mathbf{s}^i_\mathcal{U} = \mathbf{s}^1_\mathcal{U}$. However, $\mathbf{s}^1_\mathcal{U} \neq \mathbf{s}^2_\mathcal{U}$, which implies that $\mathbf{s}^i_\mathcal{U} \neq \mathbf{s}^2_\mathcal{U}$ and thus $\mathbf{s}^i_B = \mathbf{s}^2_B = \mathbf{s}^1_B$, in contradiction.

We further claim that the set $S(\mathcal{T})$ must share the same $\mathbf{s}_B$ and cannot share the same $\mathbf{s}_\mathcal{U}$. This is a simple counting argument: Since $|B| < \lambda$, a set $S(\mathcal{T})$ that shares the same $\mathbf{s}_\mathcal{U}$ has size of at most $2^{|B|} \leq 2^\lambda$. However, since $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, it holds that $|S(\mathcal{T})| > 2^\kappa$. Therefore, the set must share the same $\mathbf{s}_B$, and the claim follows. ■

We now show that the distinguisher distinguishes between the ideal and real executions with relatively small probability, even when it asks the oracle H as (polynomially) many queries as it wishes.

First, assume that the distinguisher cannot make any queries to H. We claim that the distributions of the real and ideal executions are statistically close. Intuitively, if the adversary outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$, then clearly the distinguisher may succeed only

if the consistency check fails, which happens with probability at most $2^{-\lambda}$. On the other hand, in case where the adversary outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, then, except for negligible probability (in λ) it holds that $|\mathcal{U}| \geq \ell - \lambda = \kappa$, and $|\mathcal{S}(\mathcal{T})| \geq 2^{\kappa}$, where all the elements in $S(\mathcal{T})$ share the same bits $\mathbf{s}_B$. Thus, even if the distinguisher receives $\mathbf{s}_B$ in the clear, the values $H(j, \mathbf{q}_j)$, $H(j, \mathbf{q}_j \oplus \mathbf{s})$ that are used for masking the inputs are uniformly random and independent of each other. Therefore, the simulation is indistinguishable from the real-execution.

Now, assume that the distinguisher can also make queries to the random oracle H. In this case, we claim that the distinguisher can distinguish only if it makes a "critical query", where:

Definition 3.3.5. *For every $1 \leq j \leq m$, a query made by the distinguisher or the receiver to the random oracle is a* critical query *if it is of the form $(j, ((1 - r_j) \cdot \boldsymbol{s}) \oplus \boldsymbol{q}_j)$ for some j.*

Note that a critical query can also be represented as $H(j, (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$. Clearly, a critical query totally reveals $x_j^{\overline{r_j}}$. Conditioned on the event that the distinguisher (or the receiver) never queries such a critical query and that $\mathbf{s}_{\mathcal{U}} \neq 0$, the distributions of the real and ideal executions are statistically close. On the other hand, as long as $\mathbf{s}_{\mathcal{U}} \neq 0$, and as long as no such a critical query is made, the answers to the queries are independent of the value of $\mathbf{s}$, and the distinguisher does not learn anything new from the queries themselves. Any query to H is distributed uniformly and independently at random, and since $\mathbf{s}$ is distributed uniformly in $\mathcal{S}(\mathcal{T})$, the probability to "hit" a critical query is bounded by $1/|\mathcal{S}(\mathcal{T})|$. We therefore conclude the following claim:

Claim 3.3.6. *The probability that the distinguisher or the receiver make a* critical query *is bounded by: $(t+1)/|\mathcal{S}(\mathcal{T})| \leq (t+1) \cdot 2^{-\kappa}$, where t is an upper-bound on the number of their queries.*

This completes the proof. ■

We now restate Claim 3.3.3 and prove it. This claim is in fact, an analysis of the consistency check phase of the protocol.

Claim 3.3.7 (Claim 3.3.3, restated)**.** *Let $\mathcal{T}$ be as above, and let $\mathcal{U}$ be the largest set of indices such that for every $\alpha, \beta \in \mathcal{U}$, $\boldsymbol{r}^{\alpha} = \boldsymbol{r}^{\beta}$. Assume that $|\mathcal{U}| < \ell - \lambda$. Then:*

$$\Pr_{\boldsymbol{s}}\left[\mathit{consistent}(\mathcal{T}, \boldsymbol{s}) = 1\right] \leq 2^{-\lambda}$$

and thus, $\mathcal{T} \in \mathcal{T}_{\mathit{bad}}$.

Proof: Let $\mathcal{T}$ be the values that the adversary outputs, i.e., the values

$$\mathcal{T} = \left\{ \{\mathbf{k}_i^0, \mathbf{k}_i^1\}_i, \{\mathbf{u}^i\}_i, \{\mathcal{H}_{\alpha,\beta}\}_{\alpha,\beta} \right\}.$$

For a pair $\alpha \in \mathcal{U}$, $\beta \in B$, we claim that the adversary passes the verification of the pair (α, β) with probability at most $1/2$. This is because $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and due to Lemma 3.3.1, if $\mathcal{T}$ is consistent with some (s_α, s_β) then it is inconsistent with $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, there are at most 2 possible values (s_α, s_β) that are consistent with $\mathcal{T}$, and the adversary gets caught for the 2 other values.

We define the inconsistency graph $\Gamma = (V, E)$ as follows. The set of vertices is the set $[\ell]$. The set of edges contains all the pairs that define different $\mathbf{r}$'s, that is, there exists an edge (α, β) if $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$. Note that since (α, β) are not consistent, the adversary gets caught in the check (α, β) with probability at least $1/2$. We sometimes consider the complement graph (or, the consistency graph) $\overline{\Gamma} = (V, \overline{E})$. In $\overline{\Gamma}$, each edge represents that the two vertices define the same implicit input $\mathbf{r}$.

We now analyze the size of the set $\mathcal{U}$.

1. **Case 1: $\boldsymbol{\lambda \leq |\mathcal{U}| < \ell - \lambda}$.** In this case, we have a large enough set which is consistent within itself, but is inconsistent with all the others. We claim that in this case, the adversary will get caught with probability $1 - 2^{-\lambda}$.

   In order to see this, consider the set $B = [\ell] \setminus \mathcal{U}$. Since $B \cup \mathcal{U} = [\ell]$, we have that $\lambda < |B| \leq \ell - \lambda$ as well.

   Moreover, consider the inconsistency graph Γ, and remove all the edges that are between two elements of B (this can be interpreted as follows: Although there is some possibility that the adversary gets caught because of these checks, we ignore them and do not consider these inconsistencies as cheating). As a result, we have a bipartite graph where the set of vertices is divided to B and $\mathcal{U}$. Moreover, when considering the complement graph for the resulting graph, we have two cliques, B and $\mathcal{U}$, and the maximal clique in this graph is at most $\ell - \lambda$.

   According to König's theorem [LP86], in any bipartite graph the number of edges in the maximal matching equals the minimal vertex cover. Moreover, it is easy to see that the sum of the minimum vertex cover in some graph, and the maximal clique of its complement graph equals to the overall number of vertices ℓ. We therefore conclude that the maximal matching in the graph Γ is at least λ.

   Each edge in the graph represents a check where the adversary gets caught with probability at least $1/2$. Since there are at least λ edges in the maximal matching in the inconsistency graph, there are at least λ pairs for which the adversary gets caught with probability at least $1/2$. Moreover, since we have a matching, each pair and check are independent. We therefore conclude that the adversary succeeds in its cheating with probability at most $2^{-\lambda}$, and therefore it gets caught with probability at least $1 - 2^{-\lambda}$.

2. **Case 2: $\boldsymbol{|\mathcal{U}| < \lambda}$.** Similarly to the previous case, we can just find a superset $\mathcal{U}'$ that contains $\mathcal{U}$ of size at least λ for which we assume (artificially) that it is all consistent. That is, for this set $\mathcal{U}'$ we just ignore the checks between the

elements in this set and assume that they are all consistent. After we obtain this clique (by ignoring some of the checks), we are back to the previous case.

For conclusion, we have the following: If $\mathcal{T}$ is such that $|\mathcal{U}| < \ell - \lambda$, then:

$$\Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] < 2^{-\lambda}.$$

■

Reducing the Number of Checks. Protocol 7 uses $\ell = \kappa + \lambda$ base-OTs and performs $\ell(\frac{\ell-1}{2})$ checks, i.e., a check between all possible combinations of base-OTs. In [ALSZ16], we show that the number of base-OTs ℓ and the number of checks can be traded which improves the performance of the protocol. We give possible parameter choices from [ALSZ16] for $\kappa = 128$ and $\lambda = 40$ in Table 3.4.

# base-OTs ℓ	190	177	174	172	171	170	169	168
#-checks	380	531	696	860	1 026	1 360	2 535	14 028

Table 3.4: Concrete choice of parameters for Protocol 7 that achieve $\kappa = 128$ and $\lambda = 40$ from [ALSZ16].

Achieving Covert Security. In [ALSZ16] we show that the number of base-OTs and number of checks can be chosen to achieve covert instead of active security. We show that using $\ell = 166$ base-OTs and 7 checks allows to detect a malicious receiver with probability 1/2.

3.4 Special-Purpose OT Functionalities

The protocols described up until now implement the OT_n^m functionality. In the following, we present further optimizations that are specifically tailored to the use of OT extensions in secure computation protocols summarized in Table 3.5 on page 52: Correlated OT (§3.4.1), Sender Random OT (§3.4.2), Receiver Random OT (§3.4.3), and Random OT (§3.4.4). We give the intuition and overview of the functionalities next and then summarize the resulting complexities (§3.4.5). The definitions and proofs of security are given in [ALSZ16].

3.4.1 Correlated OT (C-OT)

When performing OT extension, often the sender does not need to transfer two independent n-bit strings (x_j^0, x_j^1). In some protocols, x_j^0 and x_j^1 only need to be correlated by a value Δ_j and a correlation function f_{Δ_j}, while one of the two strings can be

constant and publicly known or random. For instance, Yao's garbled circuits with the free-XOR optimization (cf. §2.4.1.2) chooses a key k_0^j per wire at random and computes the second key for the wire as $k_1^j = k_0^j \oplus \Delta$ for a fixed global offset Δ (hence we can set $k_0^j = x_j^0$ and $k_1^j = f_\Delta(x_j^0) = x_j^0 \oplus \Delta$), the PSI protocol of [DCW13] fixes $x_j^0 = 0$ and transfers only x_j^1 (hence, we can set $\Delta_j = x_j^1$ and $f_{\Delta_j}(x_j^0) = \Delta_j$), and the Hamming distance protocol of [BCP13] requires a random x_j^0 and a correlated $x_j^1 = f_{\Delta_j}(x_j^0) = x_j^0 + \Delta_j$. We can alter the functionality of our OT extension protocols to compute correlated OT as follows. Since x_j^0 is just a random value, P_S can set $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$ and can send the *single* value $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$. P_R defines its output as $H(j, \mathbf{t}_j)$ if $r_j = 0$ or as $y_j \oplus H(j, \mathbf{t}_j)$ if $r_j = 1$. For $\binom{2}{1}$ OT on n-bit strings, we thereby reduce the communication from P_S to P_R from $2n + \ell$ to $n + \ell$ per OT, where ℓ is the number of base-OTs. The protocol description can be found in Protocol 8. Note that the C-OT functionality can be generalized to $\binom{N}{1}$ OT by generating one message at random and sending correlations for the remaining $N - 1$ messages.

PROTOCOL 8 (Implementing Correlated OT (C-OT)).
We follow Protocol 6 but assume that the sender has inputs $f_{\Delta_1}, \ldots, f_{\Delta_m}$ instead of $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$. In Step 2d and Step 2e, we have the following modifications:

1. P_S defines $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$.
2. P_S sends y_j for every $1 \leq j \leq m$, where: $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$.
3. For $1 \leq j \leq m$, P_R computes $x_j = H(j, \mathbf{t}_j)$ if $r_j = 0$, and $x_j = y_j \oplus H(j, \mathbf{t}_j)$ otherwise.

Output: P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$; P_R outputs $(x_1, \ldots, x_m)$.

3.4.2 Sender Random OT (SR-OT)

When using OT extensions for implementing the OT-based PSI protocol of [DCW13], the efficiency can be improved even further. In this case, the inputs for P_S in every OT are *independent random* strings x^0 and x^1. Thus, the sender can allow the $\binom{2}{1}$ OT extension protocol (functionality) Sender Random OT (SR-OT) to determine *both* of its inputs randomly. This is achieved in the OT extension protocol by having P_S define $x^0 = H(j, \mathbf{q}_j)$ and $x^1 = H(j, \mathbf{q}_j \oplus \mathbf{s})$. Then, P_R computes x^{r_j} just as $H(j, \mathbf{t}_j)$. The protocol description can be found in Protocol 9. The SR-OT functionality can be generalized to $\binom{N}{1}$ OT by generating all N messages as output of the CRF. With this optimization, we obtain that the entire communication in the OT extension protocol consists only of the initial base-OTs, together with the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$, and there are *no* y_j messages. This is a dramatic improvement of bandwidth. In particular, for our OT-PSI protocol in §5.4.2, which performs $\mathcal{O}(n)$ $\binom{2^\sigma}{1}$ OT on $\lambda + 2\log_2(n)$-bit strings, where n is the number of elements in both parties sets and σ is the bit-length

of each element, the communication from P_S to P_R is reduced from $\mathcal{O}(2^\sigma \cdot n \log_2(n))$ to $\mathcal{O}(n \log_2(n))$.

PROTOCOL 9 (Implementing Sender Random OT (SR-OT)).
We follow Protocol 6, where the sender does not have any input. In Step 2d and Step 2e, we have the following modification:

1. P_S defines $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = H(j, \mathbf{q}_j \oplus \mathbf{s})$ for every $1 \leq j \leq m$.
2. P_R defines $x_j = H(j, \mathbf{t}_j)$ for every $1 \leq j \leq m$. Note that there is no interaction between the parties in this step.

Output: P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$; P_R outputs $x_1, \ldots, x_m$.

3.4.3 Receiver Random OT (RR-OT)

Analogously to the Sender Random OT, in the Receiver Random OT (RR-OT), P_R obtains its input choice bits $\mathbf{r}$ as random output of the protocol execution. Our instantiation of RR-OT in OT extension allows P_R to save one bit of communication per OT. Recall that in Step 2(a) in Protocol 6, P_R sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ for $1 \leq i \leq \ell$. However, if we allow $\mathbf{r}$ to be randomly chosen, we can set $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$ and $\mathbf{t}^1 = G(\mathbf{k}_1^0)$ and only need to transfer $\mathbf{u}^{i'} = G(\mathbf{k}_{i'}^0) \oplus G(\mathbf{k}_{i'}^1) \oplus \mathbf{r}$ for $2 \leq i' \leq \ell$. P_S can then compute $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$ and, as before, $\mathbf{q}^{i'} = (s_{i'} \cdot \mathbf{u}^{i'}) \oplus G(\mathbf{k}_{i'}^{s_{i'}})$. Thereby, the communication from P_R to P_S is reduced by one bit per OT. The protocol description can be found in Protocol 10.

In order to randomly sample the choice bits in the $\binom{N}{1}$ OT protocol of [KK13], we transform the code $\mathcal{C}^\rho$ into a *systematic form*, similar to [FJNT16]. In the systematic form, the input data is embedded into the codeword, i.e., the integer $s \in \{0,1\}^{\log_2 N}$ is a sub-string of codeword c_s. This reduces the communication per $\binom{N}{1}$ OT by $\log_2 N$ bits.

3.4.4 Random OT (R-OT)

In a random OT, both P_S and P_R obtain their input as random output of the protocol. The random OT functionality can be obtained by combining the SR-OT protocol with the RR-OT protocol. Random OT can be used in the GMW protocol when precomputing MTs (cf. §4.3.2). The protocol description can be found in Protocol 11.

3.4.5 Summary

The original OT extension protocol of [IKNP03] and our proposed improvements for OT_n^m are summarized in Tab. 3.5. We compare the communication complexity of P_R and P_S for m parallel $\binom{2}{1}$ OT extensions of n-bit strings, with security parameter κ and ℓ base-OTs (we omit the cost of the initial OT_κ^κ). We also compare the assumption

PROTOCOL 10 (Implementing Receiver Random OT (RR-OT)).
We follow Protocol 6 with the following modifications:

1. P_R has no input.
2. Given the chosen keys $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$, P_R sets $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$.
3. For every $2 \leq i \leq \ell$, P_R sets $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$ to P_S. Note that $\mathbf{u}^1$ is not sent.
4. In case of our active secure OT extension protocol, the parties check consistency as previously.
5. P_R defines $T = [\mathbf{t}^1 \mid \ldots \mid \mathbf{t}^\ell]$ where $\mathbf{t}^i = G(\mathbf{k}_i^0)$ for every $1 \leq i \leq \ell$ as in Protocol 6.
6. P_S defines $Q = [\mathbf{q}^1 \mid \ldots \mid \mathbf{q}^\ell]$ where $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$, and for every $2 \leq i \leq \ell$, $\mathbf{q}^i$ is defined as in Protocol 6, i.e., $\mathbf{q}^i = G(\mathbf{k}_i^0)$ if $s_i = 0$; otherwise, set $\mathbf{q}^i = \mathbf{u}^i \oplus G(\mathbf{k}_i^1)$.
7. The parties proceed with the execution as in Protocol 6.

Output: P_S has no output; P_R outputs $\mathbf{r}$ and $x_1, \ldots, x_m$.

PROTOCOL 11 (Implementing Random-OT (R-OT)).
This is a simple combination of Protocol 9 and Protocol 10. Specifically, P_R defines its input as $G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$, and P_S defines its inputs x_j^0, x_j^1 according to $H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s})$, respectively, for every $1 \leq j \leq m$. There is no transmission of $\mathbf{u}^1$ from P_R to P_S, and there is no transmission of y_j^0, y_j^1 from P_S to P_R for every $1 \leq j \leq m$.

on the function H needed in each protocol, where CR denotes correlation-robustness and RO denotes random oracle.

Protocol	**Applicability**	$P_R \to P_S$	$P_S \to P_R$	H
Orig. [EGL85]	All applications	$m\ell$	$2mn$	CR
C-OT §3.4.1	x_j^0 random; x_j^1 correlated with Δ_j	$m\ell$	mn	RO
SR-OT §3.4.2	x_j^0, x_j^1 random, r_j chosen	$m\ell$	0	RO
RR-OT §3.4.3	x_j^0, x_j^1 chosen, r_j random	$m(\ell-1)$	$2mn$	RO
R-OT §3.4.4	x_j^0, x_j^1, r_j random	$m(\ell-1)$	0	RO

Table 3.5: Bits sent for sender P_S and receiver P_R for $\binom{2}{1}$ OT_n^m using the semi-honest OT extension protocol of [IKNP03] with our optimizations.

3.5 Evaluation

In this section, we empirically evaluate our optimizations and proposed protocols. We first describe our benchmarking environment and implementation (§3.5.1). Then, we evaluate the optimizations on the passive secure OT extension protocol of [IKNP03] from §3.2 (§3.5.2) as well as the special-purpose OT functionalities from §3.4 (§3.5.3).

We then compare the performance of our active secure OT extension protocol of §3.3 to related protocols (§3.5.4). Finally, we evaluate the performance gains that can be achieved when using AES-based instead of SHA-based CRF instantiations (§3.5.5).

3.5.1 Benchmark Setting

Parameters and Instantiation. In all our experiments, we assume long-term security beyond 2030 (cf. §2.1.3). We instantiate the PRG using AES-CTR (cf. §2.2.1) and the RO using SHA256 (cf. §2.2.2). For the evaluation in §3.5.2, §3.5.3, and §3.5.4, we instantiate the CRF using SHA256. For the evaluation in §3.5.5, we instantiate the CRF using fixed-key AES-128 [BHKR13] for the passive secure [IKNP03] protocol and pipelined AES-256 with key schedule for the passive secure [KK13] protocol (cf. §3.2.4.2). We process the OTs blockwise with blocks of size $w = 2^{19}$ and run two additional threads for network send and receive operations. We use the OT protocol of [NP01] in the random oracle model as base-OT protocol for the passive secure OT extension protocols and the OT protocol of [CO15] as base-OT protocol for the active secure OT extension protocols. As parameters for our active secure protocol we use 190 base-OTs and 380 checks and for our covert secure protocol we use 166 base-OTs and 7 checks as these parameters resulted in the best performance. For the active secure protocol of [NNOB12], we use the parameters in the paper, i.e. 342 base-OTs and 171 checks.

3-step OT Extension. To generate large numbers $m > w = 2^{19}$ of OTs for the active secure OT extension protocols, we perform a 3-step OT, where P_S and P_R first perform ℓ base-OTs, then extend these ℓ OTs to $\lceil m\ell/w \rceil$ OTs using the respective OT extension protocols, and finally split these $\lceil m\ell/w \rceil$ OTs into m/w blocks of ℓ OTs and extend each block to w OTs again using the respective OT extension protocol to obtain the m OTs. In case $m > w\lfloor w/\ell \rfloor$, one can simply extend this approach again and do a 4-step OT.

$\binom{2}{1}$ R-OT on bits via $\binom{N}{1}$ R-OT [KK13]. For the passive secure $\binom{N}{1}$ OT extension protocol of [KK13], we use $N = 16$, since this resulted in the lowest communication, and hence convert one $\binom{16}{1}$ R-OT to four $\binom{2}{1}$ R-OT (cf. §2.3.3). In particular, we convert the i-th $\binom{16}{1}$ R-OT with 4-bit output values $(z_i^0, ..., z_i^{15}) \in \{0,1\}^{64}$ to the $4i$-th to $(4i+3)$-th $\binom{2}{1}$ OT with single bit output values $(x_{4i}^0, x_{4i}^1), ..., (x_{4i+3}^0, x_{4i+3}^1)$ as: $(x_{4i}^0 || x_{4i+1}^0 || x_{4i+2}^0 || x_{4i+3}^0) = z_i^0$ and $(x_{4i}^1 || x_{4i+1}^1 || x_{4i+2}^1 || x_{4i+3}^1) = z_i^{15}$. For the remaining values $(z_i^1, ..., z_i^{14})$, P_S sends 4-bit correction values $y_i^j = z_i^j \oplus (x_{4i}^{j_0} || x_{4i+1}^{j_1} || x_{4i+2}^{j_2} || x_{4i+3}^{j_3})$ for $1 \leq j \leq 14$ and where $j = j_0 || j_1 || j_2 || j_3$ with j_0 being the least significant bit of j. Thereby, we do not need to send the correction values for z_i^0 and z_i^{15} which reduces the cost per conversion from 64 to 56 bit. Note, that the [KK13] OT can also be instantiated with $N \in \{4, 8\}$, which would increase communication but reduce the computation complexity.

Benchmark Environment. We perform our experiments in two settings: A *LAN* setting and a *WAN* setting. In the LAN setting, we run the sender and receiver routines on two Desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 4 cores and AES-NI support and 16 GB RAM that are connected by Gigabit Ethernet. In the WAN setting, we run the sender on an Amazon EC2 m3.xlarge instance with a 2.5 GHz Intel Xeon E5-2670v2 CPU with 4 virtual CPUs (vCPUs) and AES-NI support and 15 GB RAM, located in North Virginia (US EAST) and the receiver routine on one of our Desktop PCs in Europe. The average bandwidth between these two machines was 120MBit/s and the average ping latency (round-trip time) was 100 ms.

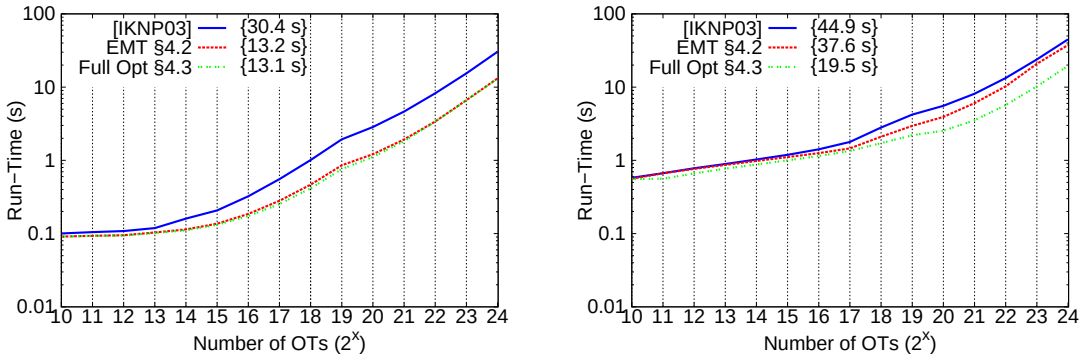
3.5.2 Evaluation of Semi-Honest OT Extension

In the following, we evaluate the performance gains from our optimizations on the passive secure OT extension protocol of [IKNP03] described in §3.2. We benchmark the protocol in three versions: The original passive secure OT extension protocol of [IKNP03] with naive matrix transposition, the protocol of [IKNP03] with the Eklundh matrix transposition (cf. §3.2.2), and our improved passive secure OT extension protocol (cf. §3.2.3), including the Eklundh matrix transposition. We evaluate all three versions using the Random OT functionality (cf. §3.4.4) as this functionality reduces the overhead for the last step in the protocol and hence lets us evaluate the core-efficiency of the protocol more precisely. We vary the number of OTs from 2^{10} (=1 024) to 2^{24} (=16 777 216) and fix the bit-length of the transferred strings to 128. The results in the LAN and WAN setting are given in Figure 3.2.

In both the LAN and WAN setting, we were able to decrease the run-time by factor 2× to 3×. In the LAN setting, the efficient matrix transposition from §3.2.2 had the highest impact while our protocol optimization from §3.2.3 only slightly decreased the run-time. This can be explained by the computation being the bottleneck in the LAN setting, hence the communication improvement from our protocol optimization had only little effect. In the WAN setting, on the other hand, the communication improvement from our protocol optimization resulted in a higher run-time decrease than the efficient matrix transposition because in this setting communication is the main bottleneck. For smaller number of OTs, the base-OTs have a high impact on the total run-time, hence the run-time of all protocols is similar. However, the base-OTs amortize for higher number of OTs and the improvements on the OT extension protocols can be seen more clearly. Note that the dent for 2^{19} OTs for both the LAN and WAN setting is due to the block size of our implementation. More than 2^{19} OTs are processed in multiple blocks, resulting in a better amortization.

3.5.3 Evaluation of Special-Purpose OT Functionalities

Next we evaluate the performance of the special-purpose OT functionalities, outlined in §3.4. We use the performance of the Random OT (R-OT) (cf. §3.4.4) as base-



(a) LAN Setting

(b) WAN Setting

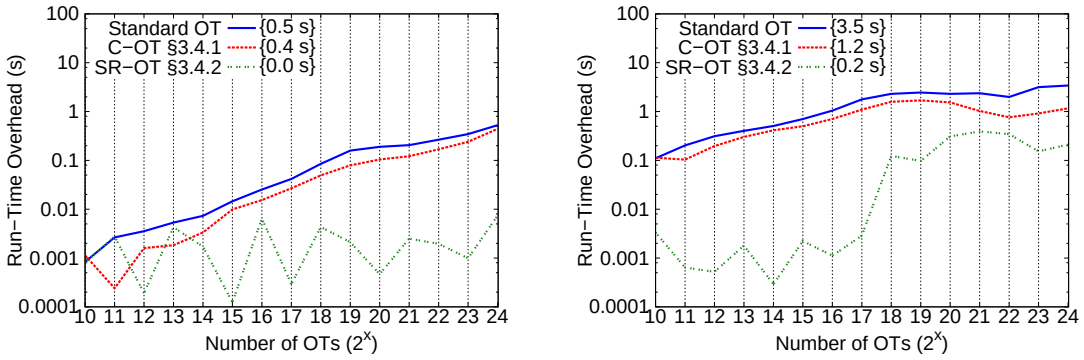
Figure 3.2: Run-time for passive secure R-OT extension on 128-bit strings in the LAN (a) and WAN (b) setting. Time for 2^{24} OTs is given in {}.

line and evaluate the overhead that is added when using the original OT, Correlated OT (C-OT) (cf. §3.4.1), and Sender Random OT (SR-OT) (cf. §3.4.2) functionalities. Similar to the evaluation of semi-honest protocol optimizations, we vary the number of OTs from 2^{10} (=1 024) to 2^{24} (=16 777 216) and fix the bit-length of the transferred strings to 128. The results for the LAN and WAN scenario are given in Figure 3.3.

From the results we can observe that the standard OT functionality and the C-OT functionality are both slower than the R-OT functionality. The SR-OT, on the other hand, has a similar performance as the R-OT since R-OT only reduces the communication by a single bit per OT. In the LAN setting, the performance difference is nearly negligible (2^{24} R-OTs need 13.1 s while the same number of OTs require 13.6 s), since the improvements from R-OT mainly affect the communication complexity which is not the bottleneck in the LAN setting. In the WAN setting, however, the performance improvements of (S)R-OT are higher, since the communication is the bottleneck and the C-OT and standard OT functionality have to send messages from the sender to the receiver. Evaluating 2^{24} OTs in the WAN setting requires 23.0 s for the standard OT functionality, 20.7 s for C-OT, 19.7 s for SR-OT, and 19.5 s for R-OT.

3.5.4 Evaluation of Active Secure OT Extension

Here we evaluate our active secure OT extension protocol of §3.3 and compare its performance to the active secure protocol of [NNOB12], our optimized versions of the passive secure $\binom{2}{1}$ OT protocol of [IKNP03] and $\binom{N}{1}$ OT protocol of [KK13], and our covert secure OT protocol in [ALSZ16]. We benchmark all five protocols on the 1-bit R-OT functionality and vary the number of OTs from 2^{10} (=1 024) to 2^{30} (=1 073 741



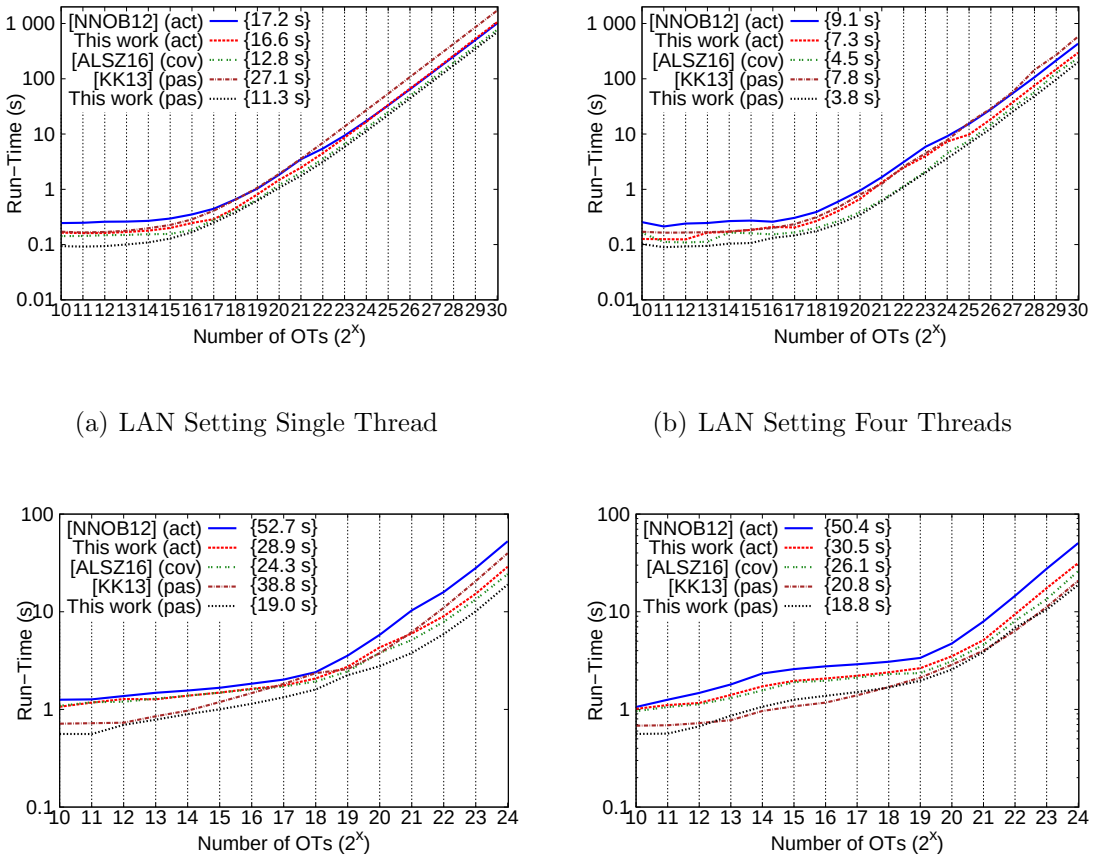
(a) LAN Setting

(b) WAN Setting

Figure 3.3: Run-time overhead over R-OT for different OT flavors using the semi-honest OT extension on 128-bit strings in the LAN (a)- and WAN (b) setting. Run-time overhead for 2^{24} OTs is given in {}.

824) in the LAN setting and to 2^{24} (=16 777 216) in the WAN setting. We evaluate the protocols once using one thread for both parties and once using four threads for both parties to highlight the effect of increased computing power. The single- and multi-thread results are given in Figure 3.4. To evaluate the improvement when using multiple threads in parallel, we benchmark all protocols in the LAN setting on a fixed number of 2^{26} random OTs with increasing number of threads from 1 to 4 and give the results in Table 3.6.

Single Thread. As expected, we can observe that the run-time increases with the provided security as our passive secure OT extension protocol outperforms the covert secure protocol which again outperforms both active secure protocols in both LAN and WAN. The only exception to this is the passive secure $\binom{N}{1}$ OT extension of [KK13], which is slowest in the LAN setting and second slowest in the WAN setting due to its higher computational overhead. In the LAN setting, the active secure protocol of [NNOB12] outperforms our active secure protocol since our protocol has a larger computational overhead for the check routine. In the WAN setting, however, the communication becomes the bottleneck and the overhead for the communication of [NNOB12] outweighs the computational overhead for the check routine of our protocol. In fact, in the WAN setting, the run-time overhead of the covert- and active secure OT extension protocols over the passive secure protocol is proportional to their communication overhead. Compared to our passive secure $\binom{2}{1}$ OT protocol, our covert secure protocol has a communication and run-time overhead of 130%, our active secure protocol has a communication overhead of 148% and a run-time overhead of 152%, and the active secure protocol of [NNOB12] has a communication overhead of 267% and a



(a) LAN Setting Single Thread

(b) LAN Setting Four Threads

(c) WAN Setting Single Thread

(d) WAN Setting Four Threads

Figure 3.4: Run-time for single thread (a,c) and multi thread (b,d) passive, covert, and active secure R-OT extension protocols on 1-bit strings in the LAN (a,b)- and WAN (c,d) setting. Time for 2^{24} OTs is given in {}.

run-time overhead of 277%.

Multiple Threads. The main improvement when increasing the number of threads can be seen in the LAN setting, were the run-time of all protocols was improved. In particular the passive secure OT extension protocol of [KK13] and our active secure protocol benefit most from the increased number of threads (cf. Table 3.6). The better scaling of these protocols can again be explained by their lower communication, which becomes the bottleneck when using multiple threads even in the LAN setting. In the WAN setting, the run-times for nearly all protocols remain unchanged even when using multiple threads since already a single thread is able to utilize the full bandwidth. The only exception to this is the passive secure protocol of [KK13], which nearly achieves

the same run-times as our passive secure protocol.

Protocol	**1 Thread**	**2 Threads**	**3 Threads**	**4 Threads**	**Improvement** $4 \mapsto 1$ **Threads**
[NNOB12] (act)	65.8 s	34.6 s	27.2 s	27.3 s	41%
This work (act)	64.7 s	33.1 s	23.7 s	18.8 s	29%
[ALSZ16] (cov)	49.9 s	25.8 s	18.2 s	15.1 s	30%
This work (pas)	44.1 s	22.7 s	15.9 s	13.2 s	30%
[KK13] (pas)	107.7 s	54.6 s	37.4 s	29.5 s	27%

Table 3.6: Run-time for increasing number of threads and time improvement of 4 threads over 1 thread when evaluating R-OT$_1^{2^{26}}$ extension in the LAN setting.

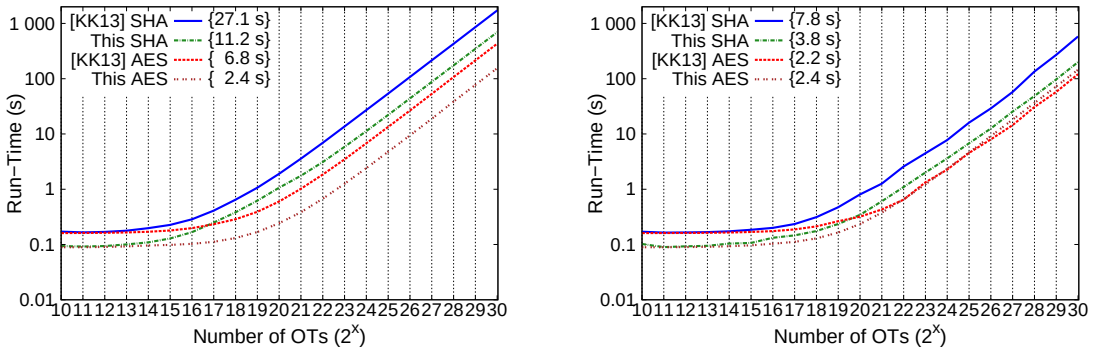
3.5.5 AES-Based CRF Instantiations

The main computational workload for the passive secure protocols comes from the CRF evaluation, which is instantiated with SHA-256. In this section, we evaluate the performance benefits when relaxing the security assumption by using an AES-based CRF instantiation. In §3.2.4.2, we discuss different AES-based instantiations, which utilize the AES-NI operations to improve the evaluation time of the CRF. For our passive secure OT extension protocol, the most efficient instantiation is the fixed-key AES instantiation from [BHKR13], which models AES as ideal permutation. For the passive secure $\binom{N}{1}$ OT extension protocol of [KK13], the most efficient instantiation uses pipelined AES-256 with key schedule (cf. §3.2.4.2), which requires a related-key assumption. We implement both CRF instantiations and compare their run-time to an instantiation using SHA-256 in the same setting as the previous section §3.5.4 and give the results in Figure 3.5.

In the LAN setting, the AES-based CRF instantiations improve the time by factor 5× for our passive secure protocol and by factor 4× for the protocol of [KK13] compared to a SHA-256-based instantiation. The most notable observation is that when using the AES-based CRF instantiation within our protocol, its run-time does not decrease at all when increasing the number of threads. This means that even a single-threaded execution of our $\binom{2}{1}$ OT extension implementation generates data at a faster rate than can be sent over a Gigabit network. The run-time of the [KK13] OT extension protocol, on the other hand, is decreased by factor 3× and thereby achieves a better run-time than our protocol when increasing the number of threads to 4, which is due to its reduced communication overhead. The results in the WAN setting confirm this observation: The run-time of our passive secure protocol only decreases slightly for the AES-based CRF instantiation, even though the computation complexity is drastically reduced. The run-time of the [KK13] protocol, on the other hand, is decreased by

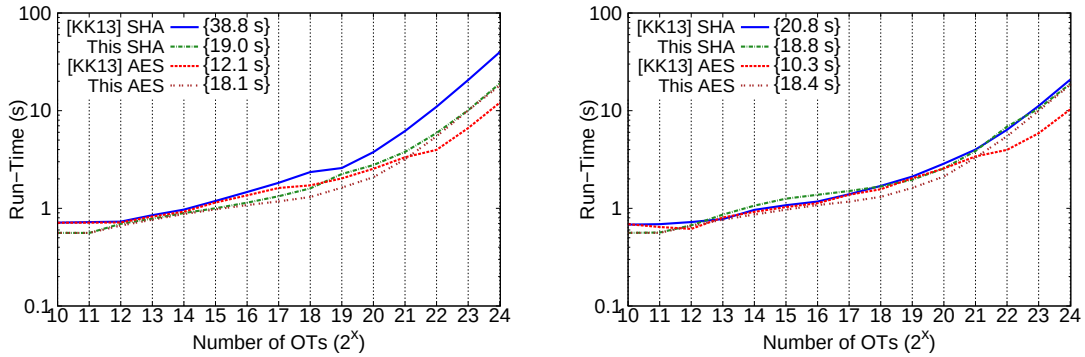
factor 3× when using the AES-based CRF instantiation and is slightly more decreased when using multiple threads, outperforming our $\binom{2}{1}$ OT protocol by nearly factor 2×.

In the following chapter on generic secure two-party computation protocols, we use the AES-based CRF instantiations due to their high efficiency.



(a) LAN Setting Single Thread

(b) LAN Setting Four Threads



(c) WAN Setting Single Thread

(d) WAN Setting Four Threads

Figure 3.5: Run-time for passive secure R-OT extension on 1-bit strings using a SHA-256 and AES-based CRF instantiation in the LAN (a,b) and WAN (c,d) settings using 1 (a,c) and 4 (b,d) threads. The time for 2^{24} OTs is given in {}.

4 Communication-Efficient Generic Secure Two-Party Computation

So far, research on practical secure two-party computation has focused mostly on Yao's garbled circuits protocol. The GMW protocol, which was proposed around the same time as Yao's garbled circuits protocol, was mostly neglected since it requires communication rounds linear in the function's multiplicative depth and heavily relies on OTs which were believed to be expensive. In this chapter, we refute this belief and outline protocols and optimizations that can make GMW a valid alternative to Yao's garbled circuits protocol. We then go a step further and introduce several OT-based special-purpose protocols that we combine with generic secure computation techniques using mixed-protocols.

Remark. Parts of this chapter have and will be published in [SZ13, ALSZ13, DSZ15, ALSZ16, DKS+17]. The author of this thesis has significantly contributed to those research results of these publications that are given in this thesis. For contributions of our co-authors such as the public-key-based multiplications and their comparison in [DSZ15] and automatically compiling high-level function representations into LUT-based representations in [DKS+17], we refer to the respective papers. The implementations are available online at `http://encrypto.de/code/ABY` and form the basis for the compiler for privacy-preserving applications within project E3 of the collaborative research center CROSSING.

4.1 Motivation

Generic secure two-party computation allows two parties to jointly compute any function on their private inputs without revealing anything but the result. Interestingly, two different approaches have been introduced at around the same time in the late eighties: Yao's garbled circuits [Yao86] and the protocol of Goldreich-Micali-Wigderson (GMW) [GMW87, Gol04]. Both protocols allow secure evaluation of a function that is represented as a Boolean circuit and, in their basic form, provide security against semi-honest adversaries who honestly follow the protocol but try to learn additional information from the observed messages (cf. §2.1.4) – this widely used model allows to construct highly efficient protocols and is the focus of this chapter.

Many subsequent works presented improvements to Yao's protocol (cf. §2.4.1.2) and showed that it can be made practical and applied to a large variety of privacy-

preserving applications, e.g., [MNPS04, LPS08, KS08, PSSW09, HKS+10, SKM11, HEKM11, Mal11, HCE11, SK11, HEK12, KSS12, BHKR13, HS13, KMR14, ZRE15, GLNP15]. These optimizations have shifted the bottleneck in Yao's garbled circuits towards the communication side. For instance, the implementation of [BHKR13] computes at the speed of 2Gbit/s per thread [BK15], which is twice the speed of even a Gigabit Ethernet connection. However, the work of [ZRE15] has shown that nearly all recent garbling techniques have hit a lower bound on the communication of two κ-bit ciphertexts per AND gate, where κ is the symmetric security parameter.

The practical improvements on the two-party GMW protocol have been less numerous. In [CHK+12], it was shown that by implementing the OT extension protocol of [IKNP03], securely evaluating a function represented as Boolean circuit using the GMW protocol can outperform protocols that use an arithmetic circuit representation for $n \geq 3$ parties. However, for the two-party case, the authors of [CHK+12] state that they expect their GMW implementation to be roughly a factor $2\times$ slower than the comparable Yao's garbled circuits implementation of [HEKM11]. In fact, since Yao's protocol has a constant number of rounds and requires OTs only for the inputs of one of the parties it was believed to be more efficient than the GMW protocol, which requires an interactive OT for each AND gate (see for example [HL10, Sect. 1.2]). In addition, GMW needs to send twice the data and perform nearly twice the number of symmetric cryptographic operations per AND gate as Yao's garbled circuits (cf. §2.4.3).

4.1.1 Our Contributions

In light of our efficiency improvements on OT extension from §3 we re-evaluate the practical efficiency of the GMW protocol and propose several optimizations that reduce the communication and round complexity of GMW. Our optimizations are tailored to different aspects of the GMW protocol: Improving the online phase using efficient *circuit constructions* (§4.2), improving the setup phase using efficient *pre-computation* protocols (§4.3), and improving the complexity for dedicated operations using *special-purpose* protocols (§4.4). We combine our special-purpose and optimize generic secure computation protocols in a framework, called *ABY*, which allows to "mix" secure computation techniques (§4.5). Finally, we evaluate our protocols on several standard operations (§4.6) and typical secure computation applications (§4.7).

We summarize our contributions by contrasting the local computation time and communication complexity of some protocols on a single machine and compare them to Yao's garbled circuits with and without IPP (cf. §2.4.1.2) when evaluating the AES circuit in Figure 4.1. 2-*MT* (§4.3.2) and N-*MT* (§4.3.3) are two pre-computation methods for GMW that pre-compute multiplication triples (MTs, cf. §2.4.2) based on $\binom{2}{1}$ OT extension and $\binom{N}{1}$ OT extension, respectively, and present a trade-off between communication and computation. *Setup-LUT (SP-LUT)* (§4.4.2.3) is a special-purpose protocol that abstracts from 2-input Boolean gates to multi-input gates and reduces communication and round complexity while similar computation complexity as N-MT.

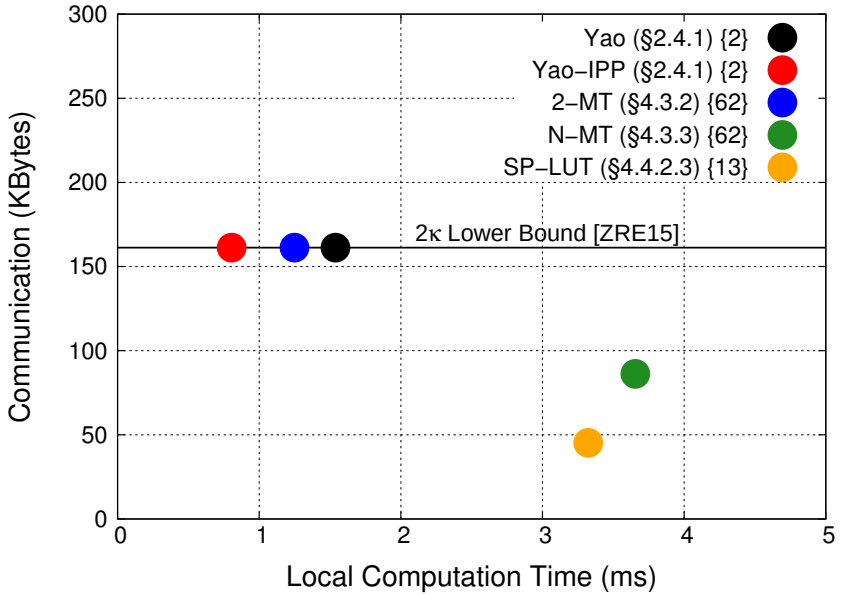


Figure 4.1: Local computation time (x-axis) and communication (y-axis) per AES circuit (5 120 AND gates) for state-of-the-art Yao's garbled circuits with and without inter party parallelization (IPP) [BK15], our 2-MT and N-MT pre-computation methods for GMW, and our SP-LUT special-purpose protocol, amortized over 8 192 parallel evaluations of the AES circuit. The number of communication rounds in the online phase is given in {}. Local computation time was measured on a Desktop PC with an Intel Haswell i7-4770K CPU and AES-NI support.

From the results we can observe that Yao's garbled circuits and GMW with 2-MT require the same communication but Yao's garbled circuits with IPP has faster local computation than 2-MT which again has faster local computation than Yao's garbled circuits without IPP. GMW using our N-MT protocol for pre-computation has $4\times$ more computation overhead but requires only approximately half of the communication. Finally, our special purpose SP-LUT protocol has slightly less computation complexity than the generic N-MT and again approximately half of the communication of N-MT. We give our contributions in more detail next.

Circuit-based Optimizations (§4.2). The main performance overhead in Yao's garbled circuits and the GMW protocol comes from evaluating AND gates in the circuit. In order to reduce this overhead for Yao's garbled circuits, several works have introduced circuits with small size (i.e., small number of AND gates) for various standard functionalities [KSS09, HEKM11, HEK12]. However, in contrast to Yao's

garbled circuits, where only the circuit size affects the performance, the performance in the GMW protocol is also affected by the circuit's depth, since both parties need to communicate for each layer. The additional low depth requirement is disadvantageous in two ways. Firstly, most of the previous size-optimized circuit constructions have high depth, achieving very poor performance when evaluated using GMW. Secondly, the whole circuit must be held in memory to group gates that can be evaluated in the same communication round, which drastically reduces the scalability of GMW.

We tackle both problems using *circuit-based optimizations*. We give an overview of *size- and depth-optimized circuit constructions* for several standard functionalities (§4.2.1), which achieve better performance when evaluated using GMW. To improve the scalability of the GMW protocol, we show how to build *Single-Instruction Multiple Data (SIMD)* circuits (§4.2.2) that drastically reduce the memory footprint of the circuit.

Optimized Pre-Computation (§4.3). The GMW protocol allows to pre-compute the computation and communication-intensive operations in the form of multiplication triples (MTs) during the setup phase, leaving only very efficient one-time pad operations in the online phase. In the work of [CHK+12], this pre-computation is done via the $\binom{4}{1}$ OT_1^1 extension protocol of [LLXX05], which requires $4(\kappa+1)$ bits communication – twice as much as Yao's garbled circuits.

We improve the communication overhead in the setup phase by utilizing our efficient OT extension protocols from §3.2. We first observe that the roles in the GMW protocol are symmetric, which allows to evenly *balance the workload* during pre-computation (§4.3.1). Next, we outline our 2-*MT* pre-computation method that is based on $\binom{2}{1}$ R-OT_1^2 and reduces the communication per AND gate from $4(\kappa+1)$ to $2(\kappa-1)$ (§4.3.2). We then introduce N-*MT*, which utilizes the $\binom{N}{1}$ OT extension protocol to further reduce the communication for MT pre-computation to nearly a single ciphertext (138 bit for $\kappa = 128$) at the cost of increased computation (§4.3.3).

Special-Purpose Optimizations (§4.4). While representing the functionality as Boolean circuit indeed allows to capture a large variety of functions, it adds a high overhead for some functionalities such as integer multiplication ($\mathcal{O}(\ell^{1.585})$ AND gates for ℓ-bit inputs using the Karatsuba multiplication circuit [KO62, HKS+10]).

In order to avoid this overhead, we introduce three special-purpose protocols that improve the performance for certain functions: *Vector MTs* (§4.4.1), *Setup-LUT (SP-LUT)* (§4.4.2) and *OT-based multiplication* (§4.4.3). Vector MTs allow to combine multiple AND gates which have one input wire in common and thereby reduce the pre-computation costs for multiple such AND gates to that of a single AND gate. SP-LUT is based on $\binom{N}{1}$ OT extension and abstracts from 2-input Boolean gates to multi-input lookup tables (LUTs) which can be evaluated in a single communication round and whose complexity only depends on the number of inputs and outputs and not on the internal logic representation. The OT-based multiplication protocol achieves efficient ℓ-bit integer multiplication using $\mathcal{O}(\ell)$ OTs and can be used to evaluate functions

represented as arithmetic circuit consisting of addition and multiplication gates.

Mixed-Protocol Secure Computation (§4.5). The protocols that we introduce have different features, making them favorable in specific settings and for specific functions. In order to achieve the benefits of different protocols, the TASTY framework [HKS+10, KSS13b] proposed to combine different secure computation protocols based on Yao's garbled circuits and homomorphic encryption.

We improve and extend the work of [HKS+10, KSS13b]. We first formalize and categorize the underlying secret-sharing schemes of GMW, Yao's garbled circuits, and our special-purpose protocols (§4.5.1). Then, we give improved protocols for transforming between different secret-sharing schemes (§4.5.2). We implement our protocols in a framework called ABY, which allows a developer of secure computation protocols to arbitrarily mix these protocols and to flexibly change the assignment of operations to underlying protocols.

Empirical Evaluation (§4.6 and §4.7). We perform an extensive empirical evaluation of our protocols. We first evaluate and compare our depth-optimized circuit constructions with their size-optimized counterparts (§4.6.1). We then compare our optimized pre-computation methods for the GMW protocol to a state-of-the-art Yao's garbled circuits implementation (§4.6.2). Next, we compare our special-purpose protocols against generic secure techniques on standard operations (§4.6.3) and evaluate the performance of our protocols for transforming between secure computation protocols (§4.6.4). Finally, we use our resulting ABY framework to give protocol instantiations for two different applications in secure computation: Private set intersection (§4.7.1) and biometric matching (§4.7.2).

4.1.2 Previous Works

In the following, we review previous work on generic secure computation frameworks. We first describe frameworks that implement Yao's garbled circuits (§4.1.2.1) and secret-sharing techniques (§4.1.2.2). Then, we outline mixed protocol frameworks, which allow to combine different secure computation techniques (§4.1.2.3). Finally, we give an overview of secure computation protocol compilers, which translate a high-level function description into a Boolean circuit (§4.1.2.4)

4.1.2.1 Secure Computation Frameworks based on Yao's Garbled Circuits

We present secure computation frameworks based on Yao's garbled circuits in the semi-honest model and in the malicious model separately.

Semi-Honest Adversaries. The first practical framework that implemented Yao's garbled circuits was Fairplay [MNPS04], which compiled functions from a high-level language into a Boolean circuit. Fairplay was extended from two to multiple parties by FairplayMP [BNP08]. The FastGC [HEKM11] framework greatly improved

the efficiency and scalability of Yao's garbled circuits using pipelining techniques, and evaluates circuits with billions of gates at rates of up to $\sim$ 100 000 AND gates per second. The JustGarble [BHKR13] framework introduced fixed-key AES garbling techniques (cf. §2.2.4) and showed that the local computation of circuits alone can be done at the speed of several million AND gates per second.

Covert- and Malicious Adversaries. In parallel to the improvements on semi-honest Yao's garbled circuits, several works improved the efficiency of Yao's garbled circuits with security against covert- and malicious adversaries. Most schemes are based on the cut-and-choose paradigm [LP07, Lin13], where several copies of the same circuit are generated, a fraction of them is opened, and the remaining circuits are combined and evaluated. The framework of [KSS12, SS13] used a cluster with hundreds of machines to demonstrate the practicality of evaluating billion gate circuits with security against malicious adversaries. The work of [FN13, FJN14] introduced several protocol optimizations and used the GPU to parallelize the computational work in Yao's garbled circuits protocol. The BlazingFast framework [HKK+14, LR14, LR15b] yields further efficiency improvements when multiple evaluations of the same function are performed and reduces the overhead compared to semi-honest secure computation to approximately factor 3×. LEGO-style protocols [NO09, FJN+13, FJNT15, NST17] perform the cut-and-choose approach on the gate level instead of the circuit level, which allows a batch computation even if the functionality is evaluated only once.
Another technique that achieves security against covert- and malicious adversaries is the dual execution technique [MF06], where a malicious garbler is able to learn some bits of its choice in the circuit. The framework of [HKE12] implemented the dual execution technique and showed that it achieves very little overhead compared to a semi-honest evaluation at the cost of a single bit leakage. The framework of [RR16, KMRR15] combined the dual execution technique, the cut-and-choose approach, and the batch-execution model of [LR14, HKK+14] and allows secure evaluation of an AES circuit in only single milliseconds.

4.1.2.2 Secure Computation Frameworks based on Secret Sharing

We present secure computation frameworks based on secret sharing techniques in the semi-honest model and in the malicious model separately.

Semi-Honest Adversaries. The Sharemind [BLW08, BJL12] framework allows secure three-party computation on arithmetic circuits with up to one corrupted party and has been used for various practical secure computation projects [BoSV15, BKK+16, BBG+16]. The VIFF framework [DGKN09] implements secure $t \geq 3$-party computation on arithmetic circuits in asynchronous networks, and achieves semi-honest security if $n < t/2$ of the parties are corrupted and active security if $n < t/3$ parties are corrupted. The SEPIA framework [BSMD10] implements secure multi-party computation on arithmetic circuits based on the BGW protocol [BGW88] in the semi-honest model.

The multi-party GMW framework of [CHK+12] operates on Boolean circuits, achieves semi-honest security even if *all* except one party are corrupted, and shows that for certain functions a Boolean circuit can be more efficient than an arithmetic circuit. The three-party computation framework of [AFL+16] proposes a more efficient of non-linear gates and shows that secure evaluation of the AES circuit can be done in the order of millions per second. Finally, [BLO16] shows how to efficiently instantiate the BMR protocol [BMR90] for constant-round secure multi-party computation using the BGW protocol [BGW88] and GMW protocol [GMW87] for pre-computation.

Covert- and Malicious Adversaries. The following multi-party frameworks all achieve active security, even if all except one party have been corrupted, and work in the pre-computation model (cf. §2.1.5). SPDZ [DPSZ12, DKL+13, KSS13a] works on arithmetic circuits and uses somewhat-homomorphic encryption in order to pre-compute arithmetic MTs. In [LPSY15], the SPDZ framework has been used to implement the constant round generic secure multi-party protocol of [BMR90], which works by emulating the role of the garbler in Yao's garbled circuits using a multi-party secure computation protocol. TinyOT [NNOB12, LOS14, BLN+15] works on Boolean circuits and uses active secure OT extension in order to pre-compute MTs. Finally, the MiniMac [DLT14, DZ13, DZ16] protocol also works on Boolean circuits, uses error-correcting codes to encode multiple shares, and a implementation, tailored to AES, has been shown to achieve very good performance. In [FKOS15] it was shown how to use OT extension for pre-computation in TinyOT, SPDZ, and MiniMac, which all use message authentication codes (MACs) to achieve security against malicious adversaries.

4.1.2.3 Mixed-Protocol Secure Computation

In order to evaluate a function, many secure computation protocols represent the function either as arithmetic or Boolean circuit. However, for some functions, one of these circuit representations can be much more efficient than the other, which greatly benefits the respective secure computation protocols. I.e., the multiplication of two ℓ-bit values requires a Boolean circuit with at least $\mathcal{O}(\ell^{1.585})$ AND gates (cf. [HKS+10]) but an arithmetic circuit with only a single multiplication gate. In order to achieve the benefits of both representations, *mixed-protocols* were introduced that allow to combine secure computation protocols working on different representations. The TASTY framework [HKS+10, KSS13b] outlines how to transfer from a Boolean circuit, evaluated using Yao's garbled circuits protocol, to an arithmetic circuit, evaluated using additively homomorphic encryption, and vice versa, and shows that secure computation protocols that mix Yao's garbled circuits with homomorphic encryption can achieve better overall performance for some function classes. The ObliVM [LWN+15] framework outlines how to combine Yao's garbled circuits protocol with oblivious RAM (ORAM) techniques and, for functions that require many oblivious data accesses on large arrays, achieves much better performance than a pure Yao's garbled circuits

evaluation.

4.1.2.4 Compilers for Secure Computation

The Fairplay framework [MNPS04] allows a developer to specify the function to be computed in a high-level language called Secure Function Definition Language (SFDL), which is compiled into a Boolean circuit. Optimization techniques and a compiler that optimizes programs written in SFDL by automatically inferring which parts of the computation can be performed on plaintext values, were presented in [Ker11]. A memory-efficient compiler that allows to compile SFDL programs into circuits even on resource-constrained mobile phones was presented in [MLB12]. The VIFF framework [DGKN09] provides a secure computation language and uses a scheduler, which evaluates gate when the inputs become available. The CBMC-GC compiler [HFKV12] allows to compile a C program into a size-optimized Boolean circuit. The Billion-Gate compiler [KSS12] uses unused gate removal, constant propagation, and gate deduplication techniques to reduce the size of Boolean circuits and improve scalability. The Portable Circuit Format (PCF) [KMSB13] represents Boolean circuits as a sequence of instructions and can be compiled from a C program. The PICCO compiler [ZSB13] performs a source-to-source compilation, supports parallelization of operations to decrease the number of communication rounds, and generates secure multi-party computation protocols based on linear secret sharing. Wysteria [RHH14] is a strongly typed high-level language for the specification of secure multi-party computation protocols. A mixed-protocol secure computation compiler was given in [KSS14], which performs automatic partitioning of a function by expressing it as sequence of primitive operations that are then assigned to different secure computation protocols using a method based on integer programming and another method based on a heuristic. The TinyGarble compiler [SHS+15] uses hardware-synthesis tools generate Boolean circuits from high level descriptions. The ParCC compiler [BK15] detects parallelizable operations in high-level function descriptions and translates these to parallel sub-circuits. The Sharemind framework [BLW08] represents functionalities in a domain-specific language [BLR13, LR15a] which can be optimized using compilers. The Frigate compiler [MGC+16] uses automatic verification techniques to ensure the correctness of the compiled circuits.

4.1.3 Follow-Up Works

Our ABY framework has been used for evaluation in [ARS+15, DDK+15, KS16, CDC+16, PKUM16, BHWK16]. The work of [DDK+15] extends the TinyGarble compiler [SHS+15] with our depth-optimized circuit constructions in §4.2, automatically generates and optimizes circuits with regards to depth, and evaluates them using our ABY framework. The CheapSMC [PKUM16] framework uses our ABY framework to automatically generate mixed-protocols that achieve low cost overhead when

evaluated on virtual machines in a cloud setting. The depth-efficient compiler ShallowCC [BHWK16] showed that the Ladner-Fischer adder, outlined in §4.2.1.2, can be built more efficiently, reducing its depth by half from $2\lceil \log_2 \ell \rceil + 1$ to $\lceil \log_2 \ell \rceil + 1$. An active secure OT-based multiplication protocol was given in [KOS16], which, in its passive secure form, equals our OT-based protocol in §4.4.3.

4.2 Circuit Optimizations

Finding an efficient Boolean circuit representation for the function to be computed is a crucial task in secure computation. In this section, we outline size- and depth efficient circuit constructions for standard operations (§4.2.1) and describe how to reduce the memory footprint of the Boolean circuit, which allows secure computation protocols to scale to process much larger functions (§4.2.2).

4.2.1 Circuit Constructions with Low Depth and Size

In this section, we give circuit constructions with low depth and size. These circuit constructions are later used as building blocks in our protocols. We summarize the circuit constructions and their corresponding size and depth in Table 4.1. In the following, we write x^ℓ to say that a variable x has ℓ bits and $\text{OP}^{(\ell,n)}(x_1^\ell, ..., x_n^\ell)$ to say that an operation OP is evaluated on n inputs $x_1^\ell, ..., x_n^\ell$ of ℓ-bit length. If the number of inputs is clear from the context, we shorten the notation to OP^ℓ.

4.2.1.1 Multiplexer

The multiplexer circuit $z^\ell = \text{MUX}^\ell(x^\ell, y^\ell, c)$ chooses one of two inputs x^ℓ, y^ℓ as output z^ℓ depending on selection bit c:

$$\text{MUX}^\ell(x^\ell, y^\ell, c) = \begin{cases} x^\ell & \text{, if } c = 0 \\ y^\ell & \text{, if } c = 1 \end{cases}$$

A MUX circuit was outlined in [KS08] that computes: $z[i] = x[i] \oplus c \wedge (x[i] \oplus y[i])$. This MUX circuit construction has size $\mathbf{S}(\text{MUX}^\ell) = \ell$ and depth $\mathbf{D}(\text{MUX}^\ell) = 1$.

4.2.1.2 Addition

We describe three adders: The *ripple-carry adder* with linear size and linear depth, the *Ladner-Fischer adder* with log-linear size and logarithmic depth, and the *carry-save adder* which represents the sum of three values as the sum of two with linear size and constant depth.

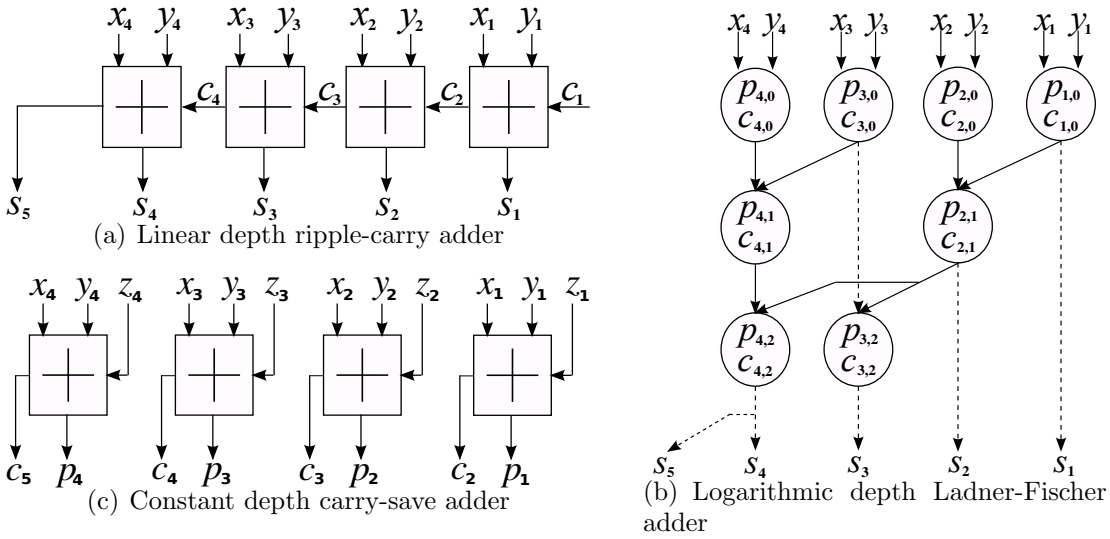
Ripple-Carry Adder. The ripple-carry adder ADD_{RC}^ℓ adds two ℓ-bit unsigned integers x^ℓ, y^ℓ and is composed from a chain of 1-bit full-adders, as depicted in Fig-

Circuit	Size **S**	Depth **D**
Addition		
Ripple-carry ADD/SUB$^{\ell}_{RC}$	ℓ	ℓ
Ladner-Fischer ADD$^{\ell}_{LF}$	$1.5\ell\lceil\log_2 \ell\rceil + 1$	$2\lceil\log_2 \ell\rceil + 1$
LF subtraction SUB$^{\ell}_{LF}$	$1.5\ell\lceil\log_2 \ell\rceil + \ell$	$2\lceil\log_2 \ell\rceil + 2$
Carry-save ADD$^{(\ell,3)}_{CSA}$	$\ell + \mathbf{S}(\text{ADD}^{\ell})$	$\mathbf{D}(\text{ADD}^{\ell})$+1
RC network ADD$^{(\ell,n)}_{RC}$	$\ell n - \ell + n - \lceil\log_2 n\rceil - 1$	$\lceil\log_2 n - 1\rceil + \ell$
CSA network ADD$^{(\ell,n)}_{CSA}$	$\ell n - 2\ell + n - \lceil\log_2 n\rceil$ $+\mathbf{S}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$	$\lceil\log_2 n - 1\rceil$ $+\mathbf{D}(\text{ADD}^{\ell+\lceil\log_2 n\rceil}_{LF})$
Multiplication		
RCN school method MUL$^{\ell}_{RC}$	$2\ell^2 - \ell$	$2\ell - 1$
CSN school method MUL$^{\ell}_{CSN}$	$2\ell^2 + 1.5\ell\lceil\log_2 \ell\rceil - 2\ell + 3$	$3\lceil\log_2 \ell\rceil + 4$
RC squaring SQR$^{\ell}_{RC}$	$\ell^2 - \ell$	$2\ell - 3$
LF squaring SQR$^{\ell}_{LF}$	$\ell^2 + 1.5\ell\lceil\log_2 \ell\rceil - 2.5\ell - 1$	$3\lceil\log_2 \ell\rceil + 3$
Comparison		
Equality EQ$^{\ell}$	$\ell - 1$	$\lceil\log_2 \ell\rceil$
Sequential greater than GT$^{\ell}_{S}$	ℓ	ℓ
D&C greater than GT$^{\ell}_{DC}$	$3\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil + 1$
Selection		
Multiplexer MUX$^{\ell}$	ℓ	1
Minimum MIN$^{(\ell,n)}$	$(n-1)(\mathbf{S}(\text{GT}^{\ell})+\ell)$	$\lceil\log_2 n\rceil(\mathbf{D}(\text{GT}^{\ell})+1)$
Minimum index MIN$^{(\ell,n)}_{IDX}$	$(n-1)(\mathbf{S}(\text{GT}^{\ell})+\ell+\lceil\log_2 n\rceil)$	$\lceil\log_2 n\rceil(\mathbf{D}(\text{GT}^{\ell})+1)$
Set Operations		
Set union $\cup^{\ell}$	ℓ	1
Set intersection $\cap^{\ell}$	ℓ	1
Set inclusion $\subseteq^{\ell}$	$2\ell - 1$	$\lceil\log_2 \ell\rceil + 1$
Count		
Full Adder count CNT$^{\ell}_{FA}$	$2\ell - \lceil\log_2 \ell\rceil - 2$	$\lceil\log_2 \ell\rceil$
Boyar-Peralta count CNT$^{\ell}_{BP}$	$\ell - d_H(\ell)$	$\lfloor\log_2 \ell\rfloor$
Distances		
Manhattan distance DST$^{\ell}_{M}$	$2\mathbf{S}(\text{SUB}^{\ell})+\mathbf{S}(\text{ADD}^{(\ell,3)})+1$	$\mathbf{D}(\text{SUB}^{\ell})+\mathbf{D}(\text{ADD}^{(\ell,3)})+1$
Euclidean distance DST$^{\ell}_{E}$	$2\mathbf{S}(\text{SUB}^{\ell})+2\mathbf{S}(\text{SQR}^{\ell})$ $+\mathbf{S}(\text{ADD}^{(2\ell,4)})+2\mathbf{S}(\text{MUX}^{\ell})$	$\mathbf{D}(\text{SUB}^{\ell})$ $+\mathbf{D}(\text{SQR}^{\ell})+3$
AES S-Box		
Size-efficient SBox$_S$	32	6
Depth-efficient SBox$_D$	34	4

Table 4.1: Size and depth of circuit constructions (d_H: Hamming weight).

ure 4.2(a). Each full-adder takes as input three bits $x[i]$, $y[i]$, and $c[i]$ and produces a parity bit $p[i] = x[i] \oplus y[i] \oplus c[i]$ and a carry bit $c[i+1] = c[i] \oplus ((x[i] \oplus c[i]) \wedge (y[i] \oplus c[i]))$ [BP06, KSS09], with $c[1] = 0$. The sum $s^{\ell+1} = x^{\ell} + y^{\ell}$ is obtained by setting $s[i] = p[i]$ for $1 \leq i \leq \ell$ and $s[\ell+1] = c[\ell+1]$. The ripple-carry adder has linear size $\mathbf{S}(\text{ADD}^{\ell}_{RC}) = \ell$ and depth $\mathbf{D}(\text{ADD}^{\ell}_{RC}) = \ell$.

Ladner-Fischer Adder. The Ladner-Fischer adder ADD$^{\ell}_{LF}$ [LF80] (also referred to



(a) Linear depth ripple-carry adder

(c) Constant depth carry-save adder

(b) Logarithmic depth Ladner-Fischer adder

Figure 4.2: Size - and depth efficient addition circuits for two (three) four-bit values.

as the Sklansky adder [Skl60]) is a so-called parallel prefix adder that adds two ℓ-bit values x^ℓ and y^ℓ in logarithmic depth. The idea of parallel prefix adders is to evaluate multiple carry-bits in parallel. An example of a Ladner-Fischer adder can be seen in Figure 4.2(b). During the computation of the sum, the Ladner-Fischer adder computes a parity bit $p_j[i]$ and a carry bit $c_j[i]$ in each node at bit position i ($1 \leq i \leq \ell$) and level j ($0 \leq j \leq \lceil \log_2 \ell \rceil$). At level 0, the Ladner-Fischer adder computes $p_0[i] = x[i] \oplus y[i]$ and $c_0[i] = x[i] \wedge y[i]$. Then, for every node at level $j > 0$, the parity is $p_j[i] = p_{j-1}[i] \wedge p_{j-1}[k]$ and the carry bit is $c_j[i] = (p_{j-1}[i] \wedge c_{j-1}[k]) \vee c_j[i]$, where k is the node that propagates the carry-bit to position i. Lastly, at level $\lceil \log_2 \ell \rceil + 1$, the sum $s^{\ell+1}$ is computed as $s[\ell+1] = c_{\lceil \log_2 \ell \rceil}[\ell]$, $s_i = p_0[i] \oplus c_{\lceil \log_2 (i-1) \rceil}[i-1]$ for $1 < i \leq \ell$, and $s[1] = p_0[1]$. The Ladner-Fischer adder has size $\mathbf{S}(\mathrm{ADD}^\ell_{LF}) = 1.5\ell\lceil \log_2 \ell \rceil + 1$ and depth $\mathbf{D}(\mathrm{ADD}^\ell_{LF}) = 2\lceil \log_2 \ell \rceil + 1$.

Carry-Save Adder. The carry-save adder $\mathrm{CSA}^{(\ell,3)}$ converts the sum of three ℓ-bit unsigned integers x^ℓ, y^ℓ, z^ℓ into two $(\ell+1)$-bit unsigned integers $p^{\ell+1}$ and $c^{\ell+1}$ such that $p^{\ell+1} + c^{\ell+1} = x^\ell + y^\ell + z^\ell$. An example of the carry-save adder can be seen in Figure 4.2(c). The carry-save adder is composed from ℓ 1-bit full-adders that compute the parity $p[i]$ and the carry $c[i+1]$ of three bits $x[i]$, $y[i]$, $z[i]$ in parallel for every position i. Finally, $p[\ell+1]$ and $c[1]$ are set to 0. The carry-save adder has linear size $\mathbf{S}(\mathrm{CSA}^{(\ell,3)}) = \ell$ and constant depth $\mathbf{D}(\mathrm{CSA}^{(\ell,3)}) = 1$. The result of the addition of x^ℓ, y^ℓ, and z^ℓ can then be obtained by adding $p^{\ell+1}$ and $c^{\ell+1}$ using the ripple-carry adder or the Ladner-Fischer adder. This results in a circuit $\mathrm{ADD}^{(\ell,3)}_{CSA}(x^\ell, y^\ell, z^\ell) = \mathrm{ADD}^\ell(\mathrm{CSA}^{(\ell,3)}(x^\ell, y^\ell, z^\ell))$[1] with $\mathbf{S}(\mathrm{ADD}^{(\ell,3)}_{CSA}) = \ell + \mathbf{S}(\mathrm{ADD}^\ell)$ and $\mathbf{D}(\mathrm{ADD}^{(\ell,3)}_{CSA}) = \mathbf{D}(\mathrm{ADD}^\ell) + 1$.

[1] Due to the appended 0 in the result of the CSA, we only require a Ladner-Fischer adder for ℓ bit.

4.2.1.3 Addition Networks

While the addition of two numbers requires a rather high depth, the addition of $n > 2$ values can be done at a much lower increase in depth and linear size. In the following, we first describe the *carry-save adder* for the addition of $n = 3$ ℓ-bit unsigned integers and then summarize the *ripple-carry network* and the *carry-save network* for the addition of an arbitrary number of unsigned integers. The difference between the two addition networks is that the carry-save network can be extended using a Ladner-Fischer adder, which allows the addition of multiple values in sublinear depth.

Ripple-Carry Network. Multiple values can be added in a straight forward way using multiple ripple-carry adders, assembled as a tree. Hence, a ripple-carry network $\text{ADD}_{RC}^{(\ell,n)}$ for adding n unsigned integers $x_{1\ldots n}^{\ell}$ of ℓ-bits can be built as $\text{ADD}_{RC}^{(\ell,n)}(x_{1\ldots n}^{\ell}) = \text{ADD}_{RC}^{(\ell+\lceil\log_2 \frac{n}{2}\rceil)}(\text{ADD}_{RC}^{(\ell,\lceil\frac{n}{2}\rceil)}(x_L^{\ell}), \text{ADD}_{RC}^{(\ell,\lfloor\frac{n}{2}\rfloor)}(x_H^{\ell}))$, where $x_L^{\ell} = x_{1\ldots\lceil\frac{n}{2}\rceil}^{\ell}$ and $x_H^{\ell} = x_{\lceil\frac{n}{2}\rceil+1\ldots n}^{\ell}$. The overall circuit consists of $\frac{n}{2}$ ADD_{RC}^{ℓ} circuits, $\frac{n}{4}$ $\text{ADD}_{RC}^{\ell+1}$ circuits, $\frac{n}{8}$ $\text{ADD}_{RC}^{\ell+2}$ circuits, and so on. This yields size $\mathbf{S}(\text{ADD}_{RC}^{(\ell,n)}) = (\ell-1)(n-1)+\sum_{i=0}^{\lceil\log_2 n\rceil} i\frac{n}{2^i} = (\ell-1)(n-1)+n(2-\frac{\lceil\log_2 n\rceil+2}{2^{\lceil\log_2 n\rceil}}) \approx \ell n-\ell+n-\lceil\log_2 n\rceil-1$. The depth of $\text{ADD}_{RC}^{(\ell,n)}$ is determined by the longest path from the carry bit of the least significant bit to the carry bit of the most significant bit in the result, i.e., $\mathbf{D}(\text{ADD}_{RC}^{(\ell,n)}) = \ell+\lceil\log_2 n-1\rceil$.

Carry-Save Network. Using the carry-save adder, a carry-save network $\text{CSN}^{(\ell,n)}$ can be built that converts n unsigned ℓ-bit integers to two unsigned $(\ell+\lceil\log_2 n\rceil)$-bit unsigned integers. The standard construction method [Sav97] results in a carry-save network $\text{CSN}_{SAV}^{(\ell,n)}$ of size $\mathbf{S}(\text{CSN}^{(\ell,n)}) = (\ell-1)(n-2)+\sum_{i=0}^{\lceil\log_2 n-1\rceil} i\frac{n-1}{2^i} \approx \ell n-2\ell+n-\lceil\log_2 n\rceil$ and depth $\mathbf{D}(\text{CSN}^{(\ell,n)}) = \frac{\lceil\log_2 n\rceil}{\lceil\log_2 1.5\rceil}+1 \approx \lceil\log_{1.5} n\rceil+1$ (see [Sav97] for more details). The depth of the carry-save network can be further decreased by using the outputs of the same depth as inputs into subsequent carry-save adders. This optimized interconnection method reduces the depth of the carry-save network to $\mathbf{D}(\text{CSN}^{(\ell,n)}) = \lceil\log_2 n-1\rceil$, while maintaining the same size.

The carry-save network can be extended to compute the sum of the addition using a Ladner-Fischer adder to add the last two values. This yields a carry-save network addition circuit ADD_{CSA} with size $\mathbf{S}(\text{ADD}_{CSA}^{(\ell,n)}) = \mathbf{S}(\text{CSN}^{(\ell,n)})+\mathbf{S}(\text{ADD}_{LF}^{(\ell+\lceil\log_2 n\rceil)})$ and depth $\mathbf{D}(\text{ADD}_{CSA}^{(\ell,n)}) = \mathbf{D}(\text{CSN}^{(\ell,n)})+\mathbf{D}(\text{ADD}_{LF}^{(\ell+\lceil\log_2 n\rceil)})$. Note that using a ripple-carry adder for the addition of the last two values would result in a circuit which is larger and deeper than $\text{ADD}_{RC}^{(\ell,n)}$. We therefore only use the Ladner-Fischer adder to instantiate ADD_{CSA} and use ADD_{RC} as size-optimized addition network.

4.2.1.4 Subtraction

A circuit $\text{SUB}^{\ell}(x^{\ell}, y^{\ell})$ for subtracting a value y^{ℓ} from a value x^{ℓ} can be constructed using the 2-complement of y^{ℓ} (i.e., $(y^{\ell}\oplus 1^{\ell})+1$) by computing $x^{\ell}+(y^{\ell}\oplus 1^{\ell})+1$. Note that the addition of 1 can be included into the addition of x^{ℓ} and $(y^{\ell}\oplus 1^{\ell})$ for the ripple-carry

adder (cf. §4.2.1.2) and the carry-save adder (cf. §4.2.1.3) by setting the least significant carry bit $c[1] = 1$. Thus, we have $\mathbf{S}(\text{SUB}_T^{\ell)})$=$\mathbf{S}(\text{ADD}_T^\ell)$ and $\mathbf{D}(\text{SUB}_T^\ell)$=$\mathbf{D}(\text{ADD}_T^\ell)$, where $T \in \{RC, CSA\}$.

A Ladner-Fischer subtracter SUB_{LF}^ℓ requires the evaluation of an additional carry-save adder that converts $x^\ell + (y^\ell \oplus 1^\ell) + 1$ to $c^{\ell+1} + p^{\ell+1}$. This yields $\text{SUB}_{LF}^\ell(x^\ell, y^\ell)$= $\text{ADD}_{LF}^\ell(\text{CSA}^{(\ell,3)}(x^\ell, (y^\ell \oplus 1^\ell), 0^{\ell-1}||1))$ with $\mathbf{S}(\text{SUB}_{LF}^\ell) = 1.5\ell\lceil\log_2 \ell\rceil + \ell + 1$ and $\mathbf{D}(\text{SUB}_{LF}^\ell) = 2\lceil\log_2 \ell\rceil + 2$.

4.2.1.5 Multiplication

In the following, we summarize the school method for multiplying and a compact circuit for squaring two unsigned integers.

School Method. The *school method* multiplication $\text{MUL}^\ell(x^\ell, y^\ell)$ multiplies two ℓ-bit unsigned integers x^ℓ, y^ℓ by computing $x^\ell y^\ell = \sum_{i=1}^{\ell} 2^{i-1}(x^\ell y[i])$, i.e. (1) multiplying each $y[i]x^\ell = z_i^\ell$, (2) shifting each z_i^ℓ left by $i-1$ positions, and (3) adding all resulting z_i^ℓ. The multiplications (1) can be done using ℓ^2 AND gates. Shifting (2) the bit sequences is done by rewiring the outputs. Adding the resulting bit products (3) can be done using an addition network (cf. §4.2.1.3).

The values in step (3) can either be added using a ripple-carry network ADD_{RC} or a carry-save network ADD_{CSA}. Note that both networks can be optimized to add the shifted ℓ-bit values more size-efficiently. Using a ripple-carry network results in the circuit MUL_{RC} with $\mathbf{S}(\text{MUL}_{RC}^\ell) = 2\ell^2 - \ell$ and $\mathbf{D}(\text{MUL}_{RC}^\ell) = 2\ell - 1$ [KS08]. The carry-save network multiplication circuit MUL_{CSA} is constructed similarly, requiring $(\ell-2)$ carry-save adders of $(\ell-1)$-bit length, ℓ half adders, and a final addition of two ℓ-bit values. Thus we have $\mathbf{S}(\text{MUL}_{CSA}^\ell) = 2\ell^2 + 1.5\ell\lceil\log_2 \ell\rceil - 2\ell + 3$ and $\mathbf{D}(\text{MUL}_{CSA}^\ell)$ $= 3\lceil\log_2 \ell\rceil + 4$. The school multiplication method can be used within the Karatsuba multiplier [KO62, HKS+10], which breaks the multiplication of larger numbers down to the multiplication of smaller numbers and scales with $\mathcal{O}(\ell^{\log_2 3})$ for ℓ-bit inputs. It was shown in [HKS+10] that for $\ell \geq 19$ bit integers, the Karatsuba multiplication circuit has fewer AND gates than the school method circuit. However, we omit the Karatsuba multiplication method due to its complex design when considering depth-efficiency.

Squaring. Although the square of a number can be computed with a multiplication circuit, a squaring circuit is smaller by a factor of about 2×. The school method multiplication circuit MUL^ℓ computes the product $x^\ell x^\ell$ as $\sum_{i=1}^{\ell} 2^{i-1}(x^\ell x[i])$. Since each $x_j x_i$ with $i \neq j$ is computed twice, $x[j]x[i] + x[i]x[j]$ can be simplified to $2x[i]x[j]$ and $x[i]x[i]$ can be replaced by $x[i]$ [YSG95].

We obtain a squaring circuit SQR^ℓ with $\mathbf{S}(\text{SQR}^\ell) = \frac{\ell^2-\ell}{2}$+$\mathbf{S}(\text{ADD}^{(2\ell,\ell/2)})$ and depth $\mathbf{D}(\text{SQR}^\ell)$=$\mathbf{D}(\text{ADD}^{(2\ell,\ell/2)})$+1. Similar to the multiplication circuit, the size and depth of the addition network can be improved since not all operands are 2ℓ-bit long. The ripple-carry version SQR_{RC}^ℓ evaluates to $\mathbf{S}(\text{SQR}_{RC}^\ell) = \ell^2 - \ell$ and $\mathbf{D}(\text{SQR}_{RC}^\ell) = 2\ell - 3$. The corresponding depth-efficient squarer circuit SQR_{LF}^ℓ has size $\mathbf{S}(\text{SQR}_{LF}^\ell) =$

$\ell^2 + 1.5\ell\lceil\log_2 \ell\rceil - 2.5\ell - 1$ and depth $\mathbf{D}(\text{SQR}^\ell_{LF})$=$\mathbf{D}(\text{ADD}^{(2\ell,\lceil\ell/2\rceil)}_{CSA})$+$\mathbf{D}(\text{ADD}^{2\ell}_{LF})$+1 = $3\lceil\log_2 \ell\rceil + 3$.

4.2.1.6 Integer Equality

The EQ^ℓ circuit computes whether two bit strings x^ℓ and y^ℓ are equal:

$$\text{EQ}^\ell(x^\ell, y^\ell) = \begin{cases} 1 & \text{, if } x^\ell = y^\ell \\ 0 & \text{, if } x^\ell \neq y^\ell. \end{cases}$$

For this, the circuit checks whether the bits are equal in each position using $z[i] = x[i] \oplus y[i] \oplus 1$ and outputs $\bigwedge_{i=1}^{\ell} z[i]$. To reduce the depth of the circuit, the AND between the $z[i]$ can be computed in a pairwise tournament fashion. The overall circuit has size $\mathbf{S}(\text{EQ}^\ell) = \ell - 1$ and depth $\mathbf{D}(\text{EQ}^\ell) = \lceil\log_2 \ell\rceil$. We note that inequality can be computed with the same complexity as $1 \oplus \text{EQ}^\ell(x^\ell, y^\ell)$.

4.2.1.7 Integer Comparison

In the following, we describe a size-optimized greater-than circuit with linear size and depth and a depth-optimized greater-than circuit with larger size but logarithmic depth. The greater than operation $\text{GT}^\ell(x^\ell, y^\ell)$ takes as input two bit sequences x^ℓ and y^ℓ and computes:

$$\text{GT}^\ell(x^\ell, y^\ell) = \begin{cases} 1 & \text{, if } x^\ell > y^\ell \\ 0 & \text{, if } x^\ell \leq y^\ell. \end{cases}$$

We note that the greater equals operation can be computed with the same size and depth complexity as $\text{GE}^\ell(x^\ell, y^\ell) = 1 \oplus \text{GT}^\ell(y^\ell, x^\ell)$.

Sequential Greater Than. A size-optimized greater than circuit GT_S was introduced in [KSS09]. The circuit computes the greater than operation using only the carry bits of a ripple-carry subtraction circuit for $x^\ell - y^\ell - 1$, i.e. $g = c_{\ell+1}$, $c_{i+1} = x_i \oplus ((x_i \oplus c_i) \wedge (y_i \oplus c_i))$ for $1 \leq i \leq \ell$, and $c_1 = 0$. Since this circuit is the same as the subtraction circuit without the parity bits, it has the same size $\mathbf{S}(\text{GT}^\ell_S) = \ell$ and depth $\mathbf{D}(\text{GT}^\ell_S) = \ell$.

Divide-And-Conquer Greater Than. A depth optimized greater than circuit GT^ℓ_{DC} was described in [GSV07] and is depicted in Figure 4.3. GT^ℓ_{DC} utilizes the divide-and-conquer approach to reduce the computation of the greater than for two ℓ-bit variables to the computation of two greater than operations for $\frac{\ell}{2}$-bit variables. The reduction is recursively applied until it can be performed on bit-level. More precisely, let $x^\ell = (x_H || x_L)$ and $y^\ell = (y_H || y_L)$ be two ℓ-bit integers with x_H, y_H being $\lceil\frac{\ell}{2}\rceil$-bit

and x_L, y_L being $\lfloor \frac{\ell}{2} \rfloor$-bit unsigned integers. GT^{ℓ}_{DC} is then recursively computed as:

$$\mathrm{GT}^{\ell}_{DC}(x^{\ell}, y^{\ell}) = \mathrm{GT}^{\lceil \frac{\ell}{2} \rceil}_{DC}(x_H, y_H) \oplus \mathrm{EQ}^{\lceil \frac{\ell}{2} \rceil}(x_H, y_H) \wedge \mathrm{GT}^{\lfloor \frac{\ell}{2} \rfloor}_{DC}(x_L, y_L)$$

until x and y are one bit values on which GT^{1}_{DC} can be computed as:

$$\mathrm{GT}^{1}_{DC}(x_i, y_i) = x_i \wedge (y_i \oplus 1).$$

Note that the EQ circuit can be computed independently of and in parallel to GT and that it can re-use parts of values that have already been computed in a deeper level of recursion. As the EQ circuits require $\ell - \lceil \log_2 \ell \rceil - 1$ AND gates, the bitwise comparisons in the leaf nodes require ℓ AND gates, and the recursions together also require $\ell - 1$ AND gates, the total size of the circuit is increased to $\mathbf{S}(\mathrm{GT}^{\ell}_{DC}) = 3\ell - \lceil \log_2 \ell \rceil - 2$ while its depth is reduced to $\mathbf{D}(\mathrm{GT}^{\ell}_{DC}) = \lceil \log_2 \ell \rceil + 1$.

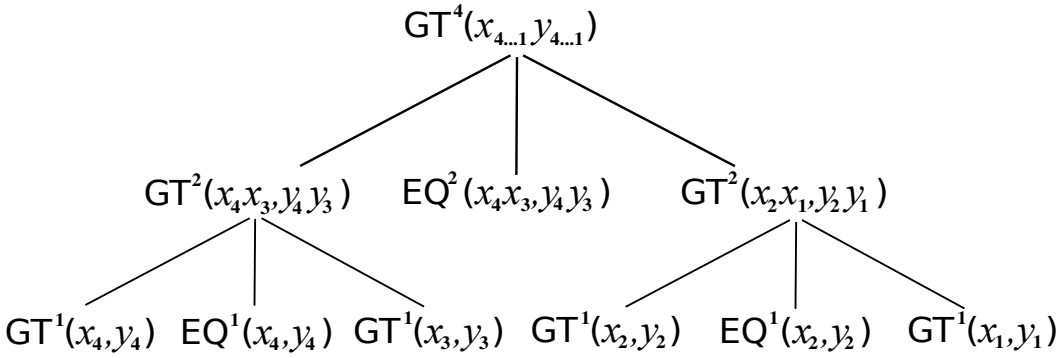


Figure 4.3: Depth-optimized greater than circuit on two 4-bit values.

4.2.1.8 Minimum/Maximum

The minimum operation $\mathrm{MIN}^{(\ell,n)}(x^{\ell}_{1\ldots n})$ takes as input a sequence of n unsigned ℓ-bit integers $x^{\ell}_{1\ldots n}$ and outputs their minimum value $z^{\ell} = \min(x^{\ell}_{1\ldots n})$. The circuit can be constructed recursively as described in [KSS09]:

$$\mathrm{MIN}^{(\ell,n)}(x^{\ell}_{1\ldots n}) = \mathrm{MIN}^{(\ell,2)}(\mathrm{MIN}^{(\ell,\frac{n}{2})}(x^{\ell}_L)), \mathrm{MIN}^{(\ell,\frac{n}{2})}(x^{\ell}_H)),$$

where $x^{\ell}_L = x^{\ell}_{1\ldots\lceil \frac{n}{2} \rceil}$ and $x^{\ell}_H = x^{\ell}_{\lceil \frac{n}{2} \rceil + 1\ldots n}$. At the base of the recursion the minimum of two inputs x^{ℓ}_i, x^{ℓ}_j is computed as:

$$\mathrm{MIN}^{(\ell,2)}(x^{\ell}_i, x^{\ell}_j) = \mathrm{MUX}^{\ell}(x^{\ell}_i, x^{\ell}_j, \mathrm{GT}^{\ell}(x^{\ell}_i, x^{\ell}_j)).$$

The circuit has size $\mathbf{S}(\text{MIN}^{(\ell,n)}) = (n-1)(\mathbf{S}(\text{GT}^\ell) + \ell)$ and depth $\mathbf{D}(\text{MIN}^{(\ell,n)}) = \lceil \log_2 n \rceil (\mathbf{D}(\text{GT}^\ell) + 1)$. When using the size-optimized GT^ℓ_S for GT^ℓ we obtain a circuit $\text{MIN}^{(\ell,n)}_S$ with small size $\mathbf{S}(\text{MIN}^{(\ell,n)}_S) = 2\ell(n-1)$ but large depth $\mathbf{D}(\text{MIN}^{(\ell,n)}_S) = \lceil \log_2 n \rceil (\ell+1)$. When using the depth-optimized GT^ℓ_{DC} for GT^ℓ we obtain $\text{MIN}^{(\ell,n)}_{DC}$ with slightly larger size $\mathbf{S}(\text{MIN}^{(\ell,n)}_{DC}) = (4\ell - \lceil \log_2 \ell \rceil - 2)(n-1)$ but smaller depth $\mathbf{D}(\text{MIN}^{(\ell,n)}_{DC}) = \lceil \log_2 n \rceil (\lceil \log_2 \ell \rceil + 1)$. A similar circuit that computes the maximum can be obtained by swapping the order of the inputs into GT^ℓ.

Some applications additionally require the index of the minimum. For these applications, we build the index of minimum circuit $\text{MIN}^{(\ell,n)}_{IDX}$, where k is the bit-length of the index by building an additional pairwise tournament circuit for the index, using again the output of each GT^ℓ circuit. The index of minimum circuit evaluates to $\mathbf{S}(\text{MIN}^{(\ell,n)}_{IDX}) = \mathbf{S}(\text{MIN}^{(\ell,n)}) - \ell + (n-1)\lceil \log_2 \ell \rceil$ and $\mathbf{D}(\text{MIN}^{(\ell,n)}_{IDX}) = \mathbf{D}(\text{MIN}^{(\ell,n)})$. A more efficient construction for the MIN_{IDX} circuit which requires the indices to correspond to their positions in the sequence has been given in [KSS09].

4.2.1.9 Count

The count circuit CNT^ℓ computes the Hamming weight d_H of a bit sequence x^ℓ, i.e., $\text{CNT}^\ell(x^\ell) = d_H(x^\ell) = \sum_{i=1}^{\ell} x_i$.

Addition Tree. In [CHK+12], a circuit CNT^ℓ_{AT} was described that computes the Hamming weight of x^ℓ using a ripple-carry network on ℓ 1-bit values $\text{ADD}^{(1,\ell)}_{RC}(x_{1\ldots\ell})$. The size of this circuit is $\mathbf{S}(\text{CNT}^\ell_{AT}) = \mathbf{S}(\text{ADD}^{(1,\ell)}_{RC}) = 2\ell - \lceil \log_2 \ell \rceil - 2$ and its depth is $\mathbf{D}(\text{CNT}^\ell_{AT}) = \mathbf{D}(\text{ADD}^{(1,\ell)}_{RC}) = \lceil \log_2 \ell \rceil$.

Improved CNT. Boyar and Peralta [BP06] propose an improved circuit CNT^ℓ_{BP} for computing the Hamming weight of x^ℓ and prove that its multiplicative size matches the theoretical lower bound. The structure of CNT^ℓ_{BP} also follows a divide-and-conquer approach, however they split a value x^ℓ into three parts of length $m = \lceil \frac{\ell-1}{2} \rceil$, $n = \lfloor \frac{\ell-1}{2} \rfloor$, and one bit, respectively: $x^\ell = (x^m || x^n || x_1)$. CNT^ℓ_{BP} is then recursively computed as:

$$\text{CNT}^\ell_{AT}(x^\ell) = \text{ADD}^{\lceil \log_2 \ell \rceil}_{RC}(\text{CNT}^m_{AT}(x^m), \text{CNT}^n_{AT}(x^n), x_1).$$

As the addition of x_1 can be performed using it as carry-input of the addition circuit (instead of fixing this to be 0), the size of the circuit is reduced to $\mathbf{S}(\text{CNT}^\ell_{BP}) = \ell - d_H(\ell)$ and the depth is reduced to $\mathbf{D}(\text{CNT}^\ell_{BP}) = \lfloor \log_2 \ell \rfloor$.

4.2.1.10 Set Operations

Set operations are commonly used in the area of privacy preserving computation, e.g. for computing common contacts [CMP11, CHK+12], privately scheduling a meeting [BJH+11], or privacy preserving genome testing [BBC+11]. As proposed in [HEK12], a set whose elements are chosen from a small domain of ℓ elements can be represented

as a binary sequence x^ℓ where x_i denotes whether the element i is contained in the set ($x_i = 1$) or not ($x_i = 0$). In the following, we explain how to use the described building blocks in order to construct depth-efficient set operations.

Set Intersection and Union. Set intersection $\cap^\ell(x^\ell, y^\ell)$ (set union $\cup^\ell(x^\ell, y^\ell)$) between two sets x^ℓ and y^ℓ of size ℓ can be computed in parallel using the bitwise AND (OR). I.e., we compute $\cap^\ell(x^\ell, y^\ell) = x^\ell \wedge y^\ell$ and $\cup^\ell(x^\ell, y^\ell) = x^\ell \vee y^\ell$. Thus, we have $\mathbf{S}(\cap^\ell) = \mathbf{S}(\cup^\ell) = \ell$ and $\mathbf{D}(\cap^\ell) = \mathbf{D}(\cup^\ell) = 1$.

Set Inclusion. Set inclusion $\subseteq^\ell(x^\ell, y^\ell)$ outputs a bit c that indicates whether a set x^ℓ is a subset of y^ℓ ($c = 1$) or not ($c = 0$). We use the variant of [CHK+12] that computes the set inclusion by computing $s_i = 1 \oplus (x_i \wedge (1 \oplus y_i))$. The resulting s_i are then used as leafs of a tree of AND gates, where the root node is the output of $\subseteq^\ell(x^\ell, y^\ell)$. The size and depth of $\subseteq^\ell$ evaluate to $\mathbf{S}(\subseteq^\ell) = 2\ell - 1$ and $\mathbf{D}(\subseteq^\ell) = \lceil \log_2 \ell \rceil + 1$.

Set Size. The size of a set x^ℓ can be computed using the CNT^ℓ circuit (cf. §4.2.1.9).

4.2.1.11 Distance Metrics

Another application scenario for privacy preserving protocols are proximity-based services [MBF+09]. An example for a proximity-based service is the identification of friends in close proximity without revealing the actual position of the user. There exist several distances that can be used to measure the proximity: The Hamming distance, the Manhattan distance, and the Euclidean distance. In the following, we describe the distance circuits assuming a two-dimensional space and identify a point in this space as $p = (x^\ell, y^\ell)$.

Hamming Distance. The Hamming distance HD^ℓ between two ℓ-bit sequences x^ℓ and y^ℓ can be computed as $\mathrm{HD}^\ell(x^\ell, y^\ell) = \mathrm{CNT}^\ell(x^\ell \oplus y^\ell)$ (cf. §4.2.1.9).

Manhattan Distance. The Manhattan distance DST^ℓ_M between two points $p_1 = (x^\ell_1, y^\ell_1)$ and $p_2 = (x^\ell_2, y^\ell_2)$ is the distance in a two dimensional space allowing only horizontal and vertical moves and is computed as $|x^\ell_1 - x^\ell_2| + |y^\ell_1 - y^\ell_2|$. [CHK+12] give such a circuit $\mathrm{DST}^\ell_{M,C}$ with size $\mathbf{S}(\mathrm{DST}^\ell_{M,C}) = 9\ell$ and depth $\mathbf{D}(\mathrm{DST}^\ell_{M,C}) = 2\ell + 2$. They use 4 multiplexer circuits MUX^ℓ (cf. §4.2.1.1), 2 GT^ℓ_S circuits (cf. §4.2.1.7), 2 SUB^ℓ_{RC} circuits (cf. §4.2.1.4), and one ADD^ℓ_{RC} circuit (cf. §4.2.1.2).

We build a more efficient Manhattan distance circuit DST^ℓ_M as:

$$
\begin{aligned}
&x^{\ell+1} = \mathrm{SUB}^\ell(x^\ell_1, x^\ell_2), && y^{\ell+1} = \mathrm{SUB}^\ell(y^\ell_1, y^\ell_2)\\
&b^\ell_1 = (x_{\ell+1}||x_{\ell+1}||...)^\ell \oplus (x_{\ell...1})^\ell, && b^\ell_2 = (y_{\ell+1}||y_{\ell+1}||...)^\ell \oplus (y_{\ell...1})^\ell\\
&\mathrm{DST}^\ell_M(p_1, p_2) = \mathrm{ADD}^{(\ell,3)}(b^\ell_1, b^\ell_2, 0^{\ell-2}||x_{\ell+1} \wedge y_{\ell+1}||x_{\ell+1} \oplus y_{\ell+1}).
\end{aligned}
$$

We can choose between the size-optimized ripple-carry and the depth-optimized Ladner-Fischer instantiations of SUB^ℓ and ADD^ℓ. Using the ripple-carry adder yields $\mathrm{DST}^\ell_{M,RC}$ with $\mathbf{S}(\mathrm{DST}^\ell_{M,RC}) = 4\ell + 1$ and $\mathbf{D}(\mathrm{DST}^\ell_{M,RC}) = 2\ell + 2$. The Ladner-Fischer variant

$\text{DST}^\ell_{M,LF}$ has approximately $\mathbf{S}(\text{DST}^\ell_{M,LF}) = 4.5\ell\lceil\log_2 \ell\rceil + 3\ell + 4$ and $\mathbf{D}(\text{DST}^\ell_{M,LF}) = 4\lceil\log_2 \ell\rceil + 6$.

Euclidean Distance. The Euclidean distance DST^ℓ_E between two points $p_1 = (x^\ell_1, y^\ell_1)$ and $p_2 = (x^\ell_2, y^\ell_2)$ is computed as $\sqrt{(x^\ell_1 - x^\ell_2)^2 + (y^\ell_1 - y^\ell_2)^2}$. Since computing the square root is very inefficient, the square of the Euclidean distance is often used as output instead (cf. [EFG+09]).

We propose an efficient (squared) Euclidean distance circuit DST^ℓ_E as:

$$
\begin{aligned}
x^{\ell+1} &= \text{SUB}^\ell(x^\ell_1, x^\ell_2), & y^{\ell+1} &= \text{SUB}^\ell(y^\ell_1, y^\ell_2)\\
a^\ell &= (x_{\ell+1}||x_{\ell+1}||...)^\ell \oplus (x_{\ell...1})^\ell, & b^\ell &= (y_{\ell+1}||y_{\ell+1}||...)^\ell \oplus (y_{\ell...1})^\ell\\
c^\ell &= \text{MUX}^\ell((0||0||...)^\ell, a^\ell, x_{\ell+1}), & d^\ell &= \text{MUX}^\ell((0||0||...)^\ell, b^\ell, y_{\ell+1})\\
\text{DST}^\ell_E(p_1, p_2) &= \text{ADD}^{(2\ell,4)}(\text{SQR}^\ell(a^\ell), c^\ell||x_{\ell+1}, \text{SQR}^\ell(b^\ell), d^\ell||y_{\ell+1}).
\end{aligned}
$$

Note that adding $c^\ell||x_{\ell+1}$ and $d^\ell||y_{\ell+1}$ in the last step can be done as part of an addition network (cf. §4.2.1.3), requiring only $2\ell + 2$ additional AND gates and a constant overhead in depth. DST^ℓ_E can be instantiated with size- or depth-efficient addition circuits. The size-efficient variant $\text{DST}^\ell_{E,RC}$ uses the ripple-carry adder and has size $\mathbf{S}(\text{DST}^\ell_{E,RC}) = 2\ell^2 + 6\ell + 2$ and depth $\mathbf{D}(\text{DST}^\ell_{E,RC}) = 3\ell + 2$. The depth-efficient variant $\text{DST}^\ell_{E,LF}$ uses the Ladner-Fischer adder and has $\mathbf{S}(\text{DST}^\ell_{E,LF}) = 2\ell^2 + (9\ell + 2)\lceil\log_2 \ell\rceil + 5\ell + 1$ and $\mathbf{D}(\text{DST}^\ell_{E,LF}) = 5\lceil\log_2 \ell\rceil + 9$.

The Euclidean distance circuit can be extended to multiple dimensions by instantiating the SUB^ℓ and MUX^ℓ in parallel and building one big addition network for adding the squares and correction values.

4.2.1.12 AES S-Box

AES consists of the four operations AddRoundKey, SubBytes, ShiftRows, and MixColumns that are repeated 10 times and operates on a state of 16×8 bits. Among all operations, the only operation that contains AND gates is the AES S-Box substitution in SubBytes. The remaining operations can be performed using only XOR gates [HEKM11].

There exist two efficient S-Box circuits: A size-efficient circuit SBox_S [BP10] and a depth-efficient circuit SBox_D [BP12]. The size-efficient SBox_S circuit has size $\mathbf{S}(\text{SBox}_S) = 32$ and depth $\mathbf{D}(\text{SBox}_S) = 6$, resulting in an AES circuit AES_S with size $\mathbf{S}(\text{AES}_S) = 5\ 120$ and depth $\mathbf{D}(\text{AES}_S) = 60$. The depth-efficient SBox_D circuit has size $\mathbf{S}(\text{SBox}_D) = 34$ and depth $\mathbf{D}(\text{SBox}_D) = 4$, resulting in an AES circuit AES_D with size $\mathbf{S}(\text{AES}_D) = 5\ 440$ and $\mathbf{D}(\text{AES}_D) = 40$. For both circuits, we assume that the expanded key is input as plaintext, as was done in [HEKM11].

4.2.2 Single Instruction Multiple Data (SIMD) Circuits

The Sharemind framework [BLW08, BJL12] for secure three-party computation showed that Single Instructions Multiple Data (SIMD) circuits can result in substantially reduced memory footprint. The idea of SIMD circuits is to replace the evaluation of n identical copies of the same sub-circuit on one-bit values by one evaluation of the sub-circuit on n-bit values. This optimization reduces the overall computation time and the memory footprint as the circuit needs to be generated only once. SIMD circuits are especially beneficial in data mining applications [BJL12], but can also speed up other applications, e.g., AES where the same S-Box is applied in parallel (cf. §4.2.1.12) or PSI (cf. §5.3).

We implement the SIMD evaluation by introducing two new virtual gate types that only require re-wiring of values, depicted in Figure 4.4: A *combiner* gate and a *splitter* gate. The combiner gate combines n one-bit input wires to a n-bit output wire. Subsequently, AND and XOR gates can be placed as usual to process the n-bit values. When the SIMD evaluation is done, a splitter gate can be used to convert n-bit wires back to n one-bit wires (or alternatively to an arbitrary subset or permutation of the output values).

Efficiency. The efficiency improvements of the SIMD programming style greatly depends on the function that is evaluated. The biggest efficiency improvements can be observed if the same function is evaluated on multiple independent inputs in parallel. In this case, for n independent function evaluations, the memory requirement of the circuit is reduced by factor $n\times$. For functions where the data needs to be re-arranged more often, the memory improvement greatly varies. For instance, when evaluating the sort-compare-shuffle PSI circuit of [HEK12] on sets of 65 536 elements with 32 bit, the SIMD instructions reduce the memory requirement by factor 30× from 382 million to 12 million gates. The biggest disadvantage of the SIMD programming style, however, is that it is more complicated than a regular single instruction single data (SISD) programming style, since the programmer has to consider data-flow dependencies.

4.3 Optimized Pre-Computation

In this section, we show how to improve the pre-computation complexity of GMW using OT extension. We first show how to equally balance the computation and communication complexity among both parties (§4.3.1). Then, we show how to pre-compute MTs in a more communication-efficient manner using $\binom{2}{1}$ R-OT extension (§4.3.2). Finally, we outline how to further improve communication for pre-computing MTs at the cost of increased computation complexity using $\binom{N}{1}$ R-OT extension (§4.3.3).

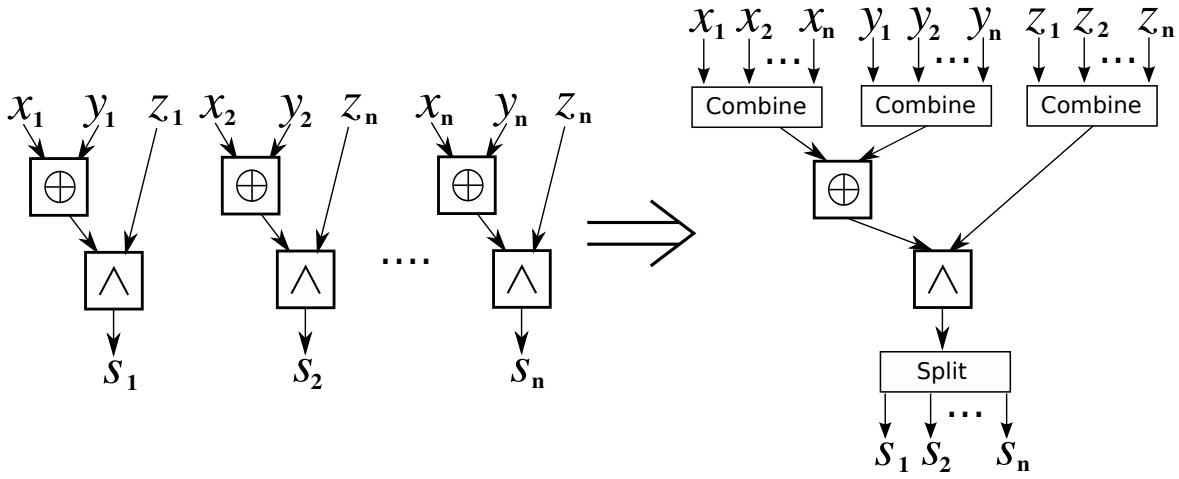


Figure 4.4: One-time evaluation of identical circuits using SIMD operations.

4.3.1 Load Balancing

The GMW implementation of [CHK+12] implements the $\binom{4}{1}$ OT extension protocol of [LLXX05] in the setup phase, where the majority of the data is sent by the receiver (cf. §2.4.3). As the MTs used in the online phase are symmetric (cf. §2.4.2), we can run the OT extensions to generate them in the setup phase in either direction. Hence, to balance the communication, we run two instantiations of the $\binom{4}{1}$ OT protocol (each for half of the AND gates) in parallel with the roles reversed. With this optimization, each party has the same workload per AND gate. Note that now we also need to run the base-OT protocol for the seed OTs twice, which however amortizes fairly quickly (35 ms computation time and 10 KBytes to be transferred).

4.3.2 2-MT: MTs via $\binom{2}{1}$ R-OT

An AND gate in the GMW protocol can be computed efficiently using MTs (cf. §2.4.2), which are bits $a_0, a_1, b_0, b_1, c_0, c_1$ under the constraint that $c_0 \oplus c_1 = (a_0 \oplus a_1)(b_0 \oplus b_1)$. Each P_i receives the shares labeled with index i. To pre-compute one MT, the work of [CHK+12] evaluates $\binom{4}{1}$ OT_1^1 using the OT extension protocol of [LLXX05].
In the following, we present our new 2-MT approach for generating MTs that is outlined in Protocol 12 and uses $\binom{2}{1}$ R-OT_1^2 (cf. §3.4.4). The R-OT functionality is exactly the same as OT, except that the parties have no inputs but the sender gets two random messages and the receiver gets a random choice bit and corresponding message as *output* (cf. §3.4.4).
To understand the high-level idea of our protocol, note that a MT can be re-written as $c_0 \oplus c_1 = (a_0 \oplus a_1)(b_0 \oplus b_1) = a_0b_0 \oplus a_0b_1 \oplus a_1b_0 \oplus a_1b_1$. Both, P_0 and P_1 can compute the terms a_0b_0 and a_1b_1 locally from their shares. We then compute the mixed-terms a_0b_1 and a_1b_0 using a secure evaluation via $\binom{2}{1}$ R-OT_1^1 for each term, where the parties

hold no inputs. From the first OT they receive $((b_0, v_0), (a_1, u_1))$ and from the second OT they receive $((a_0, u_0), (b_1, v_1))$, under the constraints that $a_1 b_0 = u_1 \oplus v_0$ and $a_0 b_1 = u_0 \oplus v_1$. Finally, each P_i sets $c_i = a_i b_i \oplus u_i \oplus v_i$.

PROTOCOL 12 (Generating Random MTs from $\binom{2}{1}$ R-OT$_1^2$).

- **Oracles:** The parties have an oracle access to the $\binom{2}{1}$ R-OT$_1^1$ functionality.

1. P_0 and P_1 invoke the $\binom{2}{1}$ R-OT$_1^1$ functionality where P_0 plays the sender and P_1 plays the receiver. P_0 receives as outputs two random bits (x_0, x_1) while P_1 receives a random choice bit a_1 and x_{a_1} as output. P_0 sets $b_0 = x_0 \oplus x_1$ and $v_0 = x_0$; P_1 sets $u_1 = x_{a_1}$.
   [Note that $a_1 b_0 = u_1 \oplus v_0$ as $a_1 b_0 = a_1(x_0 \oplus x_1) = (a_1(x_0 \oplus x_1) \oplus x_0) \oplus x_0 = x_{a_1} \oplus x_0 = u_1 \oplus v_0$.]
2. P_0 and P_1 again invoke the $\binom{2}{1}$ R-OT$_1^1$ functionality, with reverse roles where P_1 plays the sender and P_0 plays the receiver. P_1 receives as outputs two random bits (y_0, y_1) while P_0 receives a random choice bit a_0 and y_{a_0} as output. P_1 sets $b_1 = y_0 \oplus y_1$ and $v_1 = y_0$; P_0 sets $u_0 = y_{a_0}$.
   [Note that $a_0 b_1 = u_0 \oplus v_1$ as $a_0 b_1 = a_0(y_0 \oplus y_1) = (a_0(y_0 \oplus y_1) \oplus y_0) \oplus y_0 = y_{a_0} \oplus y_0 = u_0 \oplus v_1$.]
3. P_0 computes $c_0 = a_0 b_0 \oplus u_0 \oplus v_0$; P_1 computes $c_1 = a_1 b_1 \oplus u_1 \oplus v_1$.

- **Output:** P_0 outputs (a_0, b_0, c_0); P_1 outputs (a_1, b_1, c_1).

Efficiency. As we have shown in §3.4, R-OT can be instantiated more efficiently than OT: In comparison to performing $\binom{4}{1}$ OT$_1^1$ using the protocol of [LLXX05], using our $\binom{2}{1}$ R-OT$_1^2$ protocol only slightly increases the computation complexity per party from 2.5 to 3 PRG and CRF evaluations (and one additional matrix transposition), but improves the total communication complexity by a factor of approximately $2\times$ from $4(\kappa + 1)$ to $2(\kappa - 1)$.

Correctness. For correctness, observe that $c_0 \oplus c_1 = (a_0 b_0 \oplus u_0 \oplus v_0) \oplus (a_1 b_1 \oplus u_1 \oplus v_1) = a_0 b_0 \oplus (u_0 \oplus v_1) \oplus (u_1 \oplus v_0) \oplus a_1 b_1 = a_0 b_0 \oplus a_0 b_1 \oplus a_1 b_0 \oplus a_1 b_1 = (a_0 \oplus a_1)(b_0 \oplus b_1)$, as required.

Security. In Protocol 12, a_1, x_0, and x_1 are generated randomly by the first R-OT and a_0, y_0, and y_1 are generated randomly by the second R-OT. By the definition of OT, P_0 gains no information on (a_1, y_{1-a_0}) and hence $b_1 = y_0 \oplus y_1$ and P_1 gains no information on (a_0, y_{1-a_1}) and hence $b_0 = x_0 \oplus x_1$.

4.3.3 *N*-MT: MTs via $\binom{N}{1}$ OT

To improve the communication in secure computation, the work of [KK13] proposed to use their $\binom{N}{1}$ OT protocol to reduce $\binom{N}{1}$ OT$^1_{\log_2 N}$ to $\binom{2}{1}$ OT$_1^{\log_2 N}$ (cf. §2.3.3 for how

to convert from $\binom{N}{1}$ OT to $\binom{2}{1}$ OT). They achieved a communication saving of up to 1.6× per $\binom{2}{1}$ OT_1^2, from 256 bits to 160 bits, when setting $\kappa = 128$ and $N = 16$.

In the following, we introduce our N$-MT$ protocol which further improves on their communication savings by using our optimized $\binom{N}{1}$ OT protocol from §3.2.4 to directly compute a MT, which corresponds to a $\binom{4}{1}$ OT_1^1. For this reduction, we evaluate $\binom{N}{1}$ $\text{OT}_{\log_4(N)}^1$ which we can be directly transformed to $\binom{4}{1}$ $\text{OT}_1^{\log_4(N)}$. We vary possible choices for N in Table 4.2 and observe that the highest improvement of 1.9× is obtained for $N = 16$, where one MT can be computed at the cost of 134 bits in the setup phase (2 MTs at the cost of 268 bits) as shown in Table 4.2. Adding the 4 bits for the evaluation of AND gates in the online phase, the total communication cost of a single AND gate is 138 bits.

N	**4**	**8**	**16**	**32**	**64**	**128**	**256**
#MTs	*1*	*1.5*	*2*	*2.5*	*3*	*3.5*	*4*
2-MT	256	384	**512**	640	768	896	1 024
***N*-MT**	194	223	**268**	339	438	759	1 271
Improvement	1.32	1.72	**1.91**	1.89	1.75	1.18	0.81

Table 4.2: Communication for generating MTs using 2-MT (cf. §4.3.2) and our N-MT, based on our optimized $\binom{N}{1}$ OT protocol (cf. §3.2.4) of [KK13]. Best results marked in **bold**.

Efficiency. To pre-compute one MT using the N-MT technique, each party has to perform approximately 0.75 PRG evaluations and 4.25 CRF evaluations, instantiated using AES-256 with key-schedule (cf. §3.2.4.2), and both parties have to send 134 bits. In contrast, for 2-MT each party has to perform only 3 CRF evaluations, instantiated using the more efficient fixed-key AES-128, but both parties have to send 254 bits. Thus, we obtain a computation vs. communication trade-off, similar as for $\binom{2}{1}$ OT extension vs. $\binom{N}{1}$ OT extension (cf. §3.5.5).

4.4 Special-Purpose Protocols

Special-purpose protocols that were tailored specifically to one operation often perform better than generic techniques such as GMW or Yao's garbled circuits. In this section, we outline special-purpose protocols that improve the performance for certain operations. We first introduce vector MTs that can be used to improve the computation and communication complexity for the multiplexer operation (§4.4.1). We then outline the Setup-LUT (SP-LUT) protocol that can be used to evaluate multi-input lookup tables at much lower communication and round complexity than an equivalent Boolean circuit evaluation using GMW (§4.4.2). Finally, we give an integer multiplica-

tion protocol that is based on OT and achieves much better overall performance than a secure evaluation of a Boolean multiplication circuit using GMW (§4.4.3).

4.4.1 Vector MTs

Many Boolean circuit representations of functions compute the AND between one fixed wire and many different wires. Consider, for instance, the multiplexer circuit (cf. §4.2.1.1), which takes as input two ℓ-bit values x and y and one selection bit s and outputs $z = x$ if $s = 0$ or $z = y$ if $s = 1$. Internally, the circuit is of the form $z[i] = x[i] \oplus s \wedge (x[i] \oplus y[i])$, for all $1 \leq i \leq \ell$. Observe that the circuit computes the AND between one wire for s and ℓ wires for $(x[i] \oplus y[i])$. When using GMW with 2-MT, these ℓ AND gates would be evaluated using ℓ MTs, which requires R-OT$_1^{2\ell}$ (cf. §4.3.2). In this section, we describe how to evaluate such AND gates at a much lower cost using *vector MTs*. A vector MT consists of shares $a_0, a_1 \in \{0,1\}$ and $b_0, b_1, c_0, c_1 \in \{0,1\}^\ell$ with $c_0[j] \oplus c_1[j] = (a_0 \oplus a_1) \wedge (b_0[j] \oplus b_1[j])$, where P_i holds the shares labeled with index i, and for $1 \leq j \leq \ell$. As described next, these vector MTs reduce the pre-computation costs for such circuits from R-OT$_1^{2\ell}$ to R-OT$_\ell^2$, making the pre-computation cost independent of ℓ and reducing the online communication by half. A similar construction in the context of private function evaluation was outlined in [MS13].

Generating Vector MTs. Vector MTs can be pre-computed analogously to regular MTs using 2-MT (cf. §4.3.2), but using R-OT$_\ell^2$ instead of R-OT$_1^{2\ell}$ (cf. §4.3.2). The parties perform a R-OT$_\ell^1$ with P_0 acting as sender and P_1 acting as receiver, and a second R-OT$_\ell^1$ with P_1 acting as sender and P_0 acting as receiver. From these random OTs, the sender P_i obtains random $(s_i^0, s_i^1) \in \{0,1\}^{2\ell}$ and sets $b_i = s_i^0 \oplus s_i^1$ and $v_i = s_i^0$ the receiver P_{1-i} obtains random choice bit a_{1-i} and $u_{1-i} = s_{1-i}^{a_{1-i}}$. Finally, both parties locally compute $c_i = a_i b_i \oplus u_i \oplus v_i$. Note that while it is possible to pre-compute vector MTs similar to N-MT using $\binom{N}{1}$ OT extension, the communication increases with $\mathcal{O}(N\ell)$ and hence performs worse for larger ℓ than using $\binom{2}{1}$ OT extension.

Evaluating Vector MTs. To evaluate an ℓ-bit vector AND gate $z[j] = x \wedge y[j]$ using a vector MT, both parties compute $d_i = a_i \oplus x_i$ and $e_i[j] = b_i[j] \oplus y_i[j]$, exchange d_i and $e_i[j]$, set $d = d_1 \oplus d_2$, $e[j] = e_1[j] \oplus e_2[j]$, and compute $z_i[j] = (d \wedge e[j]) \oplus (d \wedge b_i[j]) \oplus (e[j] \wedge a_i) \oplus c_i[j]$.

Efficiency. Generating one ℓ-bit vector MT reduces the communication cost in the setup phase by factor $\ell\times$, i.e., from $2\ell\kappa$ bit for R-OT$_1^{2\ell}$ to 2κ bit for R-OT$_\ell^2$. Evaluating one ℓ-bit vector MT reduces the communication cost in the online phase from 4ℓ to $2\ell + 2$.

Generalization to arbitrary circuits. Note that our vector MTs can be used in every circuit where wires are used as input in two or more AND gates. In our evaluation

in §4.6.3, we apply the vector MT optimization to improve circuits for multiplication, Ladner-Fischer addition, and the AES S-Box.

4.4.2 Lookup Tables

In this section, we discuss how to abstract from circuits with 2-input Boolean gates and model the functionality as network of interconnected lookup tables (LUTs) with multiple input bits (§4.4.2.1). We then give two protocols that allow to evaluate these LUTs in a constant number of rounds. The first protocol for evaluating LUTs that we describe is the one-time truth table (OTTT) approach of [IKM+13] with pre-computation of [DZ16] (§4.4.2.2). We then present our protocol, called Setup-LUT (SP-LUT), that substantially reduces the communication in the setup phase but increases the communication in the online phase (§4.4.2.3). Finally, we show how to optimize the online phase of the SP-LUT protocol to achieve better round complexity and communication complexity and how to compute LUTs with overlapping inputs more efficiently (§4.4.2.4). We give a summary of the communication costs for a ℓ-input LUT using these protocols in Table 4.3. From the table, we can observe that the online phase of the OTTT protocol, which scales linear with $\mathcal{O}(\ell)$, is much more efficient than the online phase of our SP-LUT protocol, which scales with $\mathcal{O}(2^\ell)$. The total communication of our SP-LUT protocol, on the other hand, is much better, since it scales only with $\mathcal{O}(2^\ell)$ while the OTTT protocol scales with $\mathcal{O}(2^{2\ell})$.

# **Inputs** ℓ	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Setup Communication [bits]							
OTTT [IKM+13]	$\lesssim 2^9 o$	$\lesssim 2^{11} o$	$\lesssim 2^{13} o$	$\lesssim 2^{14} o$	$\lesssim 2^{16} o$	$\lesssim 2^{17} o$	$\lesssim 2^{18} o$
SP-LUT (§4.4.2.3)	**190**	**221**	**236**	**243**	**246**	**247**	**247**
Online Communication [bits]							
OTTT	**4**	**6**	**8**	**10**	**12**	**14**	**16**
SP-LUT	$4o+2$	$8o+3$	$16o+4$	$32o+5$	$64o+6$	$128o+7$	$256o+8$
Total Communication (Setup + Online) [bits]							
OTTT [IKM+13]	$\lesssim 2^9 o$	$\lesssim 2^{11} o$	$\lesssim 2^{13} o$	$\lesssim 2^{14} o$	$\lesssim 2^{16} o$	$\lesssim 2^{17} o$	$\lesssim 2^{18} o$
SP-LUT (§4.4.2.3)	$4o+192$	$8o+224$	$16o+240$	$32o+248$	$64o+252$	$128o+254$	$256o+255$

Table 4.3: Setup, Online and Total communication for a ℓ-input LUT with o outputs ((ℓ, o)-LUT) for OTTT and Setup-LUT (SP-LUT). Best results marked in **bold**.

4.4.2.1 Lookup Tables

For our protocols in this section, we assume that the parties have XOR secret-shared their private inputs (as in GMW) and represent the functionality as network of lookup tables (LUTs) and XOR gates (cf. Figure 4.5). In our context, a ℓ-input bit LUT with o output bits is a table that maps an ℓ-bit secret-shared input to an o-bit secret-shared

output and can thereby be used to represent any function $f : \{0,1\}^\ell \mapsto \{0,1\}^o$. In contrast to Boolean circuits based on 2-input gates, LUT-based circuits do not use internal logic operations to map inputs to outputs and their evaluation costs depend only on the number of inputs and outputs. We show how to pre-compute and evaluate a ℓ-bit input LUT in the next sections. XOR gates can be evaluated locally by both parties XORing their respective shares. Similarly, we can reduce the number of output bits if one output bit can be computed as a linear combination of two other outputs.

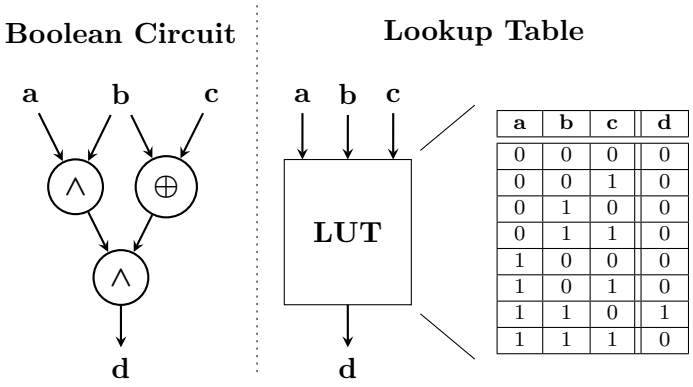


Figure 4.5: A function with $\ell = 3$ input and $o = 1$ output bits represented as 2-input Boolean gate circuit and $(3,1)$-LUT.

4.4.2.2 One-Time Truth Tables (OTTT)

In this section we describe the OTTT protocol of [IKM+13] with circuit-based pre-computation from [DZ16], which is given in Protocol 13. The high-level idea behind the OTTT protocol is that two parties hold secret shares T_0 and T_1 of a lookup table T, whose entries were randomly rotated across both dimensions using r, s such that $T_0[i] \oplus T_1[i] = T[r \oplus s \oplus i]$, for all $0 \leq i < 2^\ell$. Each of the parties knows a secret share of the truth-table as well as the rotation value, i.e., P_0 knows (T_0, r) and P_1 knows (T_1, s).

Pre-Computation. During the setup phase, the truth-table T needs to be shared such that P_0 holds (T_0, r) and P_1 holds (T_1, s). A possible method for pre-computing the table was outlined in [DZ16]: Both parties evaluate a Boolean circuit representing the table once for each of the 2^ℓ possible inputs, resulting in an overhead of factor $2^\ell\times$ compared to a Boolean circuit evaluation.[2] In more detail, the parties represent the table T as Boolean circuit $C : \{0,1\}^\ell \mapsto \{0,1\}^o$. Then, P_0 and P_1 choose their random rotation values $r, s \in_R \{0,1\}^\ell$, securely evaluate $C(r \oplus s \oplus i) = z_0^i \oplus z_1^i$ and set $T_0[i] = z_0^i$ and $T_1[i] = z_1^i$ for all $i \in [0...2^\ell - 1]$.

[2]Note that the evaluated circuit can be optimized by removing duplicate gates [KSS12]. Assuming that the last gate in the circuit is an AND gate (otherwise, one could remove that last gate from the LUT), we expect the circuit after the duplicate removal to have at least 2^ℓ gates.

Online Evaluation. In the online phase, the OTTT protocol of [IKM+13] takes as input two ℓ-bit shared values x_0 and x_1 such that $x = x_0 \oplus x_1$ and evaluates a function f, represented as a lookup table T. The parties hold shares (T_0, r) and (T_1, s) of a permuted lookup table T such that $T_0[i] \oplus T_1[i] = T[r \oplus s \oplus i]$, where $r, s \in_R \{0,1\}^\ell$ and for all $0 \leq i < 2^\ell$. To evaluate T, the parties exchange $u = x_0 \oplus r$ and $v = x_1 \oplus s$ and compute the shared result $z_0 = T_0[u \oplus v]$ and $z_1 = T_1[u \oplus v]$. To see that $z = T[x] = z_0 \oplus z_1$, observe that $z_0 \oplus z_1 = T_0[u \oplus v] \oplus T_1[u \oplus v] = T_0[r \oplus s \oplus x] \oplus T_1[r \oplus s \oplus x] = T[x]$.

Efficiency. To pre-compute an OTTT for a ℓ-input to o-out LUT ((ℓ, o)-LUT), the parties need to securely evaluate 2^ℓ copies of the same Boolean circuit C. Assuming the upper bound of $(\ell - 1)o$ AND gates for C (cf. §2.1.2 for a discussion on an upper bound on AND gates for arbitrary Boolean circuits) and using the N-MT pre-computation with GMW with 138-bits per AND gate from §4.3.3, this results in an overall communication of at most $138 \cdot 2^\ell(\ell - 1)o$ bits in the setup phase. The online phase, on the other hand, is highly efficient, since the parties only need to send 2ℓ bits.

PROTOCOL 13 (Evaluating (ℓ, o)-LUT Using the OTTT Protocol of [IKM+13])**.**

- **Common Input:** Input bit-length ℓ; Output bit-length o; $N = 2^\ell$; Truth-table $T : \{0,1\}^\ell \mapsto \{0,1\}^o$.

Pre-Computation:

1. The parties represent T as Boolean circuit $C : \{0,1\}^\ell \mapsto \{0,1\}^o$.
2. P_0 chooses $r \in_R \{0,1\}^\ell$ and P_1 chooses $s \in_R \{0,1\}^\ell$.
3. P_0 and P_1 securely compute $z_0^i \oplus z_1^i = C(s \oplus r \oplus i)$ and set $T_0[i] = z_0^i$ and $T_1[i] = z_1^i$ for all $0 \leq i < N$.

- **Output:** P_0 outputs (T_0, r); P_1 outputs (T_1, s).
  Note: $\forall i \in [0...N)$ *it holds that* $T_0[i] \oplus T_1[i] = T[r \oplus s \oplus i]$.

Online Evaluation:

- **Input of P_0:** $x_0 \in \{0,1\}^\ell$.
- **Input of P_1:** $x_1 \in \{0,1\}^\ell$.

1. P_0 sends $u = x_0 \oplus r$ to P_1; P_1 sends $v = x_1 \oplus s$ to P_0.
2. P_0 sets $z_0 = T_0[u \oplus v]$; P_1 sets $z_1 = T_1[u \oplus v]$.

- **Output:** P_0 outputs z_0; P_1 outputs z_1 with $z_0 \oplus z_1 = T[x_0 \oplus x_1]$.

4.4.2.3 Setup-LUT (SP-LUT)

While the OTTT approach achieves a good online communication, its pre-computation cost scales with at least $\mathcal{O}(2^\ell \kappa)$, where ℓ is the number of input bits of a LUT. This

greatly hinders its applicability when pre-computation is not negligible, i.e., when the parties do not have a pre-established communication channel or when they wish to perform secure computation ad-hoc. In order to enable LUT-based secure computation even in settings with no pre-computation, we suggest a new protocol for securely pre-computing and evaluating LUTs. This protocol, called *Setup-LUT (SP-LUT)*, achieves much better total communication but increases the online communication compared to the OTTT protocol. The general idea of SP-LUT is simple: Pre-compute $\binom{2^\ell}{1}$ OT in the setup phase and obliviously transfer all possible outcomes of the lookup table in the online phase. We give a full description of the protocol in Protocol 14.

Compared to the OTTT approach, the SP-LUT protocol only requires correlated randomness in the form of a pre-computed $\binom{2^\ell}{1}$ OT, which requires only little communication in the setup phase at the cost of $2^\ell o$ bits of communication during the online phase. However, the total communication of SP-LUT is much lower than that of OTTT, since only single bits need to be transferred instead of pre-computing MTs to evaluate the Boolean circuit (cf. Table 4.3 on page 84). The security of the SP-LUT protocol is similar to that of the GMW protocol [GMW87]: Both parties operate on secret-shared data by sacrificing a pre-computed OT on random data.

PROTOCOL 14 (Evaluating (ℓ, o)-LUT Using Our Setup-LUT (SP-LUT) Protocol).
Inputs and Oracles:

- **Common Input:** Symmetric security parameter κ; Input bit-length ℓ; Output bit-length o; $N = 2^\ell$; Truth-table $T : \{0,1\}^\ell \mapsto \{0,1\}^o$.
- **Oracles:** Both parties have access to the $\binom{N}{1}$ R-OT^1_o functionality.

Pre-Computation:

1. P_0 and P_1 invoke the $\binom{N}{1}$ R-OT^1_o functionality where P_0 plays the sender and P_1 plays the receiver. From the OT, P_0 receives random bits $(m_0, ..., m_{N-1})$ and P_1 receives a random choice $s \in \{0,1\}^\ell$ and message m_s.
2. **Output:** P_0 outputs $(m_0, ..., m_{N-1})$; P_1 outputs (m_s, s).

Online Evaluation:

- **Input of P_0:** $x_0 \in \{0,1\}^\ell$.
- **Input of P_1:** $x_1 \in \{0,1\}^\ell$.

1. P_1 sends $u = s \oplus x_1$ to P_0.
2. P_0 chooses $z_0 \in_R \{0,1\}^o$ and computes and sends $V = (v_0, ..., v_{N-1})$, where $v_i = T[i \oplus x_0] \oplus m_{i \oplus u} \oplus z_0$.
3. P_1 computes $z_1 = v_{x_1} \oplus m_s$.

- **Output:** P_0 outputs z_0; P_1 outputs z_1 with $z_0 \oplus z_1 = T[x_0 \oplus x_1]$.

4.4.2.4 Optimizations

In the following we discuss two optimizations that we use in our SP-LUT protocol: *Switching roles* to reduce the round complexity and *combining LUTs* with overlapping inputs.

Reducing the Online Round Complexity. The SP-LUT protocol in §4.4.2.3 pre-computes $\binom{N}{1}$ OT in a setup phase and then uses these pre-computed values in the online phase to securely evaluate the function. In its vanilla version, the online phase consists of two rounds. In the first round, the receiver sends its updated choice bits to the sender. In the second round, the sender rotates its pre-computed masks and sends the updated correlations to the receiver. Thereby, we overall require $2\mathbf{D}_L(\mathrm{C})$ communication rounds in the online phase, where $\mathbf{D}_L(\mathrm{C})$ is the highest number of lookup tables from any input to any output value.

In order to reduce the number of communication rounds, we let both parties switch roles in the online phase after each communication round, similar to [Hua12] and as shown in Figure 4.6. More specifically, assume P_0 plays the sender and P_1 plays the receiver in the first round. P_1 first sends its updated choice bits u_1 to P_0 (Step 1 of the online evaluation in Protocol 14), who plays the receiver in the second round and replies with the updated correlations V_1 and the updated choice bits of the second round u_2 (Step 2 followed by Step 1 of the next round). P_1 then updates its local shares using V_1, switches to the role of the sender and replies with its updated correlations V_2, and then again switches to the role of the receiver and sends its updated choice bits u_3 for the third communication round, etc. Overall, this reduces the number of communication rounds from $2\mathbf{D}_L(\mathrm{C})$ to $\mathbf{D}_L(\mathrm{C}) + 1$.

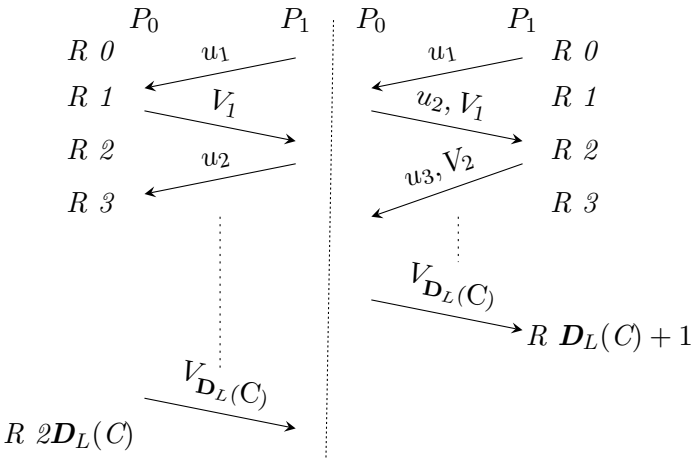


Figure 4.6: Reducing the number of communication rounds for SP-LUT from $2\mathbf{D}_L(\mathrm{C})$ to $\mathbf{D}_L(\mathrm{C}) + 1$ by flipping roles.

Multi-Out LUTs. Note that in our LUT-based approach, we can efficiently combine

two or more LUTs that have the same or even only some inputs in common. Consider a functionality which has o LUTs with one output bit each and the same ℓ input bits. When naively applying our approach, we would generate o ℓ-input LUTs, one for each output bit. However, since we build on a $\binom{N}{1}$ OT protocol, we can amortize the cost for computing the OT protocol by sending outputs of o bits during the OT protocol. More specifically, instead of performing $\binom{N}{1}$ OT_1^o, we perform $\binom{N}{1}$ OT_o^1, thereby saving $o-1$ executions of the OT protocol. This optimization naturally extends to an arbitrary number of output bits o. Overall, for a functionality with ℓ input bits and o output bits, we thereby decrease the required communication from $o(256+2^\ell)$ to $256+o2^\ell$. We use this optimization to decrease the communication for the 8-input to 8-output AES S-Box by a factor of 1.8× from 4 096 bit to 2 304 bit. Similarly, we can combine two or more LUTs which share a sub-set of inputs. For instance, consider the case where one LUT has $\ell_1 = 3$ inputs (x_0, x_1, x_2) and a second LUT has $\ell_2 = 4$ inputs (x_0, x_1, x_3, x_4). In this case, we can combine both LUTs to one LUT with $\ell = 5$ inputs and thereby reduce $\binom{N_1}{1}$ OT_1^1 and $\binom{N_2}{1}$ OT_1^1 to $\binom{N}{1}$ OT_2^1, where $N_1 = 2^{\ell_1}$, $N_2 = 2^{\ell_2}$, and $N = 2^\ell$. Concretely, this reduces the communication by factor 1.56× from 488 bits for a $(3,1)$-LUT and a $(4,1)$-LUT to 312 bits for a $(5,2)$-LUT (cf. Table 4.3 on page 84).

4.4.3 Multiplication via OT

In generic secure two-party computation, one can either represent the function as Boolean or arithmetic circuit. While Boolean circuits can be evaluated using Yao's garbled circuits or the GMW protocol, arithmetic circuits consist of addition and multiplication gates and can be evaluated using additively homomorphic encryption [FNP04, EFG+09, BBC+10, BG11, HKS+10, KSS13b, KSS14]. More detailed, values are shared additively between both parties such that addition can be performed locally while multiplications use homomorphic encryption schemes such as Paillier [DJ01, DJN10, Pai99] or DGK [DGK08, DGK09]. However, homomorphic encryption becomes prohibitively slow for larger security parameters, compared Boolean circuit-based techniques [KSS14].

Instead of using homomorphic encryption to evaluate multiplications in arithmetic circuits, we describe a protocol that is based on OT extension and which utilizes our efficiency improvements, outlined in §3.2. A similar variant of the protocol was proposed in [Gil99, Sect. 4.1] and used in [BCP+14]. It allows to efficiently compute the product of two secret-shared values using OT. The protocol we describe in the following uses more efficient correlated OT extension (C-OT, cf. §3.4.1). Overall, an ℓ-bit multiplication requires C-$\text{OT}_\ell^{2\ell}$ (or even on shorter strings, as described below). In the following, we describe how to compute arithmetic MTs (cf. §2.4.2), which can be batch pre-computed before the function evaluation.

Generating Arithmetic MTs. The generation of arithmetic MTs $c = a \cdot b$ is similar to the generation of a Boolean MT using 2-MT (cf. §4.3.2). Observe that we can

write $a \cdot b = (a_0 + a_1) \cdot (b_0 + b_1) = a_0b_0 + a_0b_1 + a_1b_0 + a_1b_1$, where all operations are performed in $\mathbb{Z}_{2^\ell}$. Let P_0 randomly generate $a_0, b_0 \in_R \mathbb{Z}_{2^\ell}$ and P_1 randomly generate $a_1, b_1 \in_R \mathbb{Z}_{2^\ell}$. The terms a_0b_0 and a_1b_1 can be computed locally by P_0 and P_1, respectively. The mixed-terms a_0b_1 and a_1b_0 are computed as described next. We detail only the computation of a_0b_1, since a_1b_0 can be computed symmetrically by reversing the parties' roles.

Note that, since a_0b_1 leaks information if known in plain by a party, we compute the shares $u_0 + v_1 = a_0b_1$ securely, such that P_0 holds u_0 and P_1 holds v_1. We have P_0 and P_1 engage in a C-OT$_\ell^\ell$, where P_0 is the sender and P_1 is the receiver. In the i-th C-OT, P_1 inputs $b_1[i]$ as choice bit and P_0 inputs the correlation function $f_{\Delta_i}(x) = (a_0 \cdot 2^i - x) \mod 2^\ell$. As output from the i-th C-OT, P_0 obtains (s_i^0, s_i^1) with $s_i^0 \in_R \mathbb{Z}_{2^\ell}$ and $s_i^1 = f_{\Delta_i}(s_i^0) = (a_0 \cdot 2^i - s_i^0) \mod 2^\ell$ and P_1 obtains $s_i^{b_1[i]} = (b_1[i] \cdot a_0 \cdot 2^i - s_i^0) \mod 2^\ell$. P_0 sets $u_0 = (\sum_{i=1}^{\ell} s_i^0) \mod 2^\ell$ and P_1 sets $v_1 = (\sum_{i=1}^{\ell} s_i^{b_1[i]}) \mod 2^\ell$.

Analogously, P_0 and P_1 compute $v_0 + u_1 = a_1b_0$. Finally, P_i sets $c_i = a_ib_i + u_i + v_i$.

Correctness and security of the protocol directly follow from the protocol and proof in [Gil99, Sect. 4.1].

Evaluating Arithmetic MTs. Arithmetic MTs can be used to evaluate a multiplication gate $z = x \cdot y$ with $x, y, z \in \mathbb{Z}_{2^\ell}$ in a similar fashion as Boolean MTs can be used to evaluate AND gates (cf. §2.4.2): The parties compute and exchange $d_i = x_i - a_i$ and $e_i = y_i - b_i$, compute $d = d_0 + d_1$ and $e = e_0 + e_1$, and set their output shares as $z_0 = (d \cdot e) + (b_0 \cdot d) + (a_0 \cdot e) + c_0$ and $z_1 = (b_1 \cdot d) + (a_1 \cdot e) + c_1$. Note that all operations are performed in $\mathbb{Z}_{2^\ell}$.

Efficiency. To generate an ℓ-bit MT, P_0 and P_1 run C-OT$_\ell^{2\ell}$, where each party evaluates 6ℓ symmetric cryptographic operations and sends $2\ell(\kappa + \ell)$ bits. The communication can be further decreased by sending only the $\ell - i$ least significant bits in the i-th C-OT, since the i most significant bits are cut off by the modulo operation anyway. This reduces the communication to C-OT$_\ell^2$ + C-OT$_{\ell-1}^2$... + C-OT$_1^2$, which averages to C-OT$_{(\ell+1)/2}^{2\ell}$ and requires $2\ell\kappa + \frac{\ell+1}{2}$ bits of communication. In contrast, a Boolean multiplication circuit has $2\ell^2 - \ell$ AND gates (cf. §4.2.1.5) and hence requires R-OT$_1^{4\ell^2-2\ell}$ or $(4\ell^2 - 2\ell)\kappa$ bits of communication using the GMW protocol, which is factor $\sim 2\ell\times$ more than our OT-based multiplication protocol. Furthermore, we shown in [DSZ15] that our OT-based multiplication protocol outperforms additively homomorphic encryption-based multiplication protocols by one to three orders of magnitude.

4.5 Mixed Protocol Secure Computation

In the following, we show how to combine our special-purpose protocols in §4.4 with generic secure computation protocols to a *mixed protocol* secure computation framework called ABY. We first categorize the underlying classes of secret-sharing: Arith-

metic, Boolean, and Yao's garbled circuits-based (§4.5.1). Then, we show how to transform between these classes of secret-sharing (§4.5.2), which achieves better runtime for secure evaluation of certain operations. An outline of our ABY framework can be found in Figure 4.7 and our ABY framework is available as open source project on GitHub at https://github.com/encryptogroup/aby. The code includes most of the protocols and optimizations, referred to in this section as well as most of the circuit constructions, outlined in §4.2.1. The remaining N-MT and SP-LUT protocols will be made available upon publication of the respective paper [DKS+17].

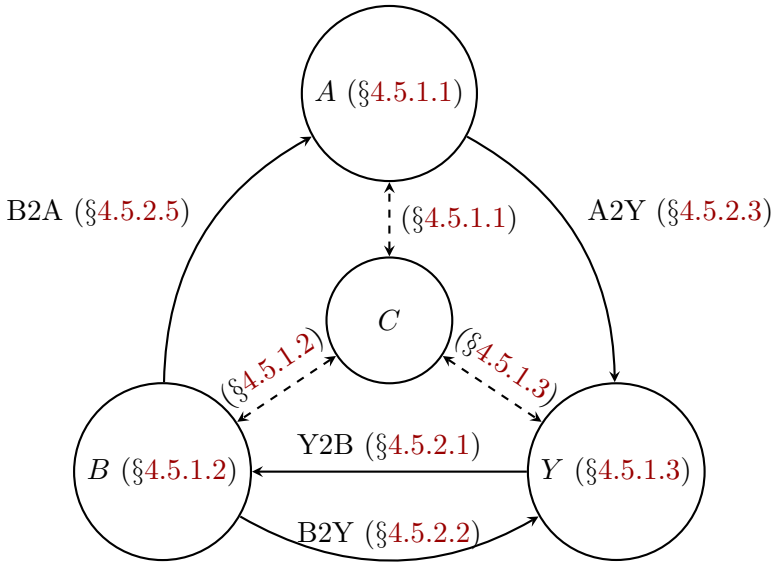


Figure 4.7: Overview of our ABY framework that allows efficient conversions between **C**leartext values and secure computation protocols that use **A**rithmetic or **B**oolean secret-sharing or **Y**ao's garbled circuits.

4.5.1 Representations

In order to combine different secure computation schemes, we formalize the format in which the values are shared. In the following, we categorize protocols for three different sharing types: **A**rithmetic sharing (§4.5.1.1), **B**oolean sharing (§4.5.1.2), and **Y**ao sharing (§4.5.1.3). For each sharing type we describe the semantics of the sharing and how standard operations are evaluated. Throughout this section and §4.5.2, we denote a value x that is shared with an arithmetic, Boolean, or Yao's garbled circuits-based secret-sharing scheme as $\langle x \rangle^A$, $\langle x \rangle^B$, or $\langle x \rangle^Y$ and the respective shares of P_i as $\langle x \rangle_i^A$, $\langle x \rangle_i^B$, and $\langle x \rangle_i^Y$, for $i \in \{0, 1\}$. We denote by $\mathsf{Shr}_i^S(x)$ the process of secret-sharing a variable x that is held by P_i and by $\mathsf{Rec}_i^S(x)$ the process of disclosing the plaintext value of x to P_i in the respective secret-sharing scheme $S \in \{A, B, Y\}$.

4.5.1.1 Arithmetic Sharing

In the 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 protocols described in the following are based on [ABL+04, PBS12, KSS14]. First we define the sharing semantics and then the operations. We assume all arithmetic operations to be performed in the ring $\mathbb{Z}_{2^\ell}$, i.e., all operations are $(\text{mod } 2^\ell)$.

Sharing Semantics. Arithmetic sharing is based on additively sharing private values between the parties as follows.

- *Shared Values.* For an ℓ-bit arithmetic sharing $\langle x \rangle^A$ of x we have $\langle x \rangle_0^A + \langle x \rangle_1^A \equiv x \pmod{2^\ell}$ with $\langle x \rangle_0^A, \langle x \rangle_1^A \in \mathbb{Z}_{2^\ell}$.
- *Sharing.* $\mathsf{Shr}_i^A(x)$: 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 $\langle x \rangle_{1-i}^A = r$.
- *Reconstruction.* $\mathsf{Rec}_i^A(x)$: P_{1-i} sends its share $\langle x \rangle_{1-i}^A$ to P_i who computes $x = \langle x \rangle_0^A + \langle x \rangle_1^A$.

Operations. Every arithmetic circuit is a sequence of addition and multiplication gates, evaluated as follows:

- *Addition.* $\langle z \rangle^A = \langle x \rangle^A + \langle y \rangle^A$: P_i locally computes $\langle z \rangle_i^A = \langle x \rangle_i^A + \langle y \rangle_i^A$.
- *Multiplication.* $\langle z \rangle^A = \langle x \rangle^A \cdot \langle y \rangle^A$: Multiplication is performed using a pre-computed arithmetic MT that can be pre-computed using our OT-based multiplication protocol from §4.4.3.

4.5.1.2 Boolean Sharing

The Boolean sharing uses an XOR-based secret sharing scheme to share a variable. We represent functions as a network of Boolean gates, evaluated using GMW with 2-MT and N-MT pre-computation (cf. §4.3.2 and §4.3.3) and our vector MTs (cf. §4.4.1), and LUTs, evaluated using our SP-LUT protocol (cf. §4.4.2.3). We first define the sharing semantics, and then describe how operations are evaluated.

Sharing Semantics. Boolean sharing uses an XOR-based secret sharing scheme. To simplify presentation, we assume single bit values; for ℓ-bit values each operation is performed ℓ times in parallel.

- *Shared Values.* A Boolean share $\langle x \rangle^B$ of a bit x is shared between the two parties, such that $\langle x \rangle_0^B \oplus \langle x \rangle_1^B = x$ with $\langle x \rangle_0^B, \langle x \rangle_1^B \in \mathbb{Z}_2$.
- *Sharing.* $\mathsf{Shr}_i^B(x)$: 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 $\langle x \rangle_{1-i}^B = r$.

- *Reconstruction.* $\mathsf{Rec}_i^B(x)$: P_{1-i} sends its share $\langle x\rangle_{1-i}^B$ to P_i who computes $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$.

Operations. Every efficiently computable function can be expressed as a Boolean circuit consisting of XOR and AND gates which can be combined into LUTs for more complex functionalities. We detail the evaluation of these operations in the following.

- *XOR.* $\langle z\rangle^B = \langle x\rangle^B \oplus \langle y\rangle^B$: P_i locally computes $\langle z\rangle_i^B = \langle x\rangle_i^B \oplus \langle y\rangle_i^B$.
- *AND.* $\langle z\rangle^B = \langle x\rangle^B \wedge \langle y\rangle^B$: AND is evaluated using a Boolean MT that can be pre-computed using our 2-MT (cf. §4.3.2) or N-MT (cf. §4.3.3) pre-computation methods.
- *LUTs.* $\langle z\rangle^B = T[\langle x\rangle^B]$: A LUT is evaluated using SP-LUT (cf. §4.4.2).
- *MUX.* For multiplexer operations we use vector MTs (cf. §4.4.1).
- *Others.* For standard functionalities we use the depth-optimized circuit constructions summarized in §4.2.1.

4.5.1.3 Yao Sharing

In Yao's garbled circuits protocol, the garbler encrypts a Boolean function to a garbled circuit, which is evaluated by the evaluator (cf. §2.4.1). 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 [KS08], 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 k_w^W without knowing the actual value of w. To simplify presentation, we assume single bit values; for ℓ-bit values each operation is performed ℓ times in parallel.

- *Shared Values.* A garbled circuits share $\langle x\rangle^Y$ of a value x is shared as $\langle x\rangle_0^Y = k_0$ and $\langle x\rangle_1^Y = k_x = k_0 \oplus xR$.
- *Sharing.* $\mathsf{Shr}_0^Y(x)$: P_0 samples $\langle x\rangle_0^Y = k_0 \in_R \{0,1\}^\kappa$ and sends $k_x = k_0 \oplus xR$ to P_1. $\mathsf{Shr}_1^Y(x)$: Both 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 $\langle x\rangle_1^Y = k_x$.

- *Reconstruction.* $\mathsf{Rec}_i^Y(x)$: 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:

- *XOR.* $\langle z \rangle^Y = \langle x \rangle^Y \oplus \langle y \rangle^Y$ is evaluated using the free-XOR technique [KS08]: P_i locally computes $\langle z \rangle_i^Y = \langle x \rangle_i^Y \oplus \langle y \rangle_i^Y$.
- *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 $\langle y \rangle_1^Y$.
- *Others.* For standard functionalities we use the size-optimized circuit constructions summarized in §4.2.1.

4.5.2 Transformations

In this section, we detail methods to convert between different secure computation sharings that we denote as $s2d$ with $s, d \in \{A, B, Y\}$ and $s \neq d$. We start by explaining already existing or straight-forward conversions: $Y2B$ (§4.5.2.1), $B2Y$ (§4.5.2.2), $A2Y$ (§4.5.2.3), and $A2B$ (§4.5.2.4). We then detail our improved constructions for $B2A$ (§4.5.2.5) and $Y2A$ (§4.5.2.6). We summarize the complexities of the sharing, reconstruction, and conversion operations in Table 4.4.

	Comp. [# sym]	**Comm. [bits]**	**# Msg**
Y2B	0	0	0
$\mathsf{Shr}_*^{A/B}$, Rec_*^*	0	ℓ	1
Shr_0^Y	ℓ	$\ell\kappa$	1
B2A, Y2A	6ℓ	$\ell\kappa + (\ell^2 + \ell)/2$	2
B2Y, Shr_1^Y	6ℓ	$2\ell\kappa$	2
A2Y, A2B	12ℓ	$6\ell\kappa$	2

Table 4.4: Total computation (# symmetric cryptographic operations), communication, and number of messages in online phase for sharing, reconstruction, and conversion operations on ℓ-bit values.

4.5.2.1 Yao to Boolean Sharing (Y2B)

Converting a Yao share $\langle x \rangle^Y$ to a Boolean share $\langle x \rangle^B$ is the easiest conversion and comes essentially for free. The key insight is that the permutation bits of $\langle x \rangle_0^Y$ and $\langle x \rangle_1^Y$ already form a valid Boolean sharing of x. Thus, P_i locally sets $\langle x \rangle_i^B = Y2B(\langle x \rangle_i^Y) = \langle x \rangle_i^Y[0]$.

4.5.2.2 Boolean to Yao Sharing (B2Y)

Converting a Boolean share $\langle x \rangle^B$ to a Yao share $\langle x \rangle^Y$ is very similar to the Shr_1^Y operation (cf. §4.5.1.3): Assume that x is a single bit; for ℓ-bit values, each operation is done ℓ times in parallel. Let $x_0 = \langle x \rangle_0^B$ and $x_1 = \langle x \rangle_1^B$. P_0 samples $\langle x \rangle_0^Y = k_0 \in_R \{0,1\}^\kappa$. Both parties run OT_κ^1 where P_0 acts as sender with inputs $(k_0 \oplus x_0 R; k_0 \oplus (1 - x_0)R)$, whereas P_1 acts as receiver with choice bit x_1 and obliviously obtains $\langle x \rangle_1^Y = k_0 \oplus (x_0 \oplus x_1)R = k_x$. Note that we also use this protocol for switching roles in IPP [BK15].

4.5.2.3 Arithmetic to Yao Sharing (A2Y)

Converting an arithmetic share $\langle x \rangle^A$ to a Yao share $\langle x \rangle^Y$ was outlined in [HKS+10, KSS13a, KSS14] and can be done by securely evaluating an addition circuit. More precisely, the parties secret share their arithmetic shares $x_0 = \langle x \rangle_0^A$ and $x_1 = \langle x \rangle_1^A$ as $\langle x_0 \rangle^Y = \mathsf{Shr}_0^Y(x_0)$ and $\langle x_1 \rangle^Y = \mathsf{Shr}_1^Y(x_1)$ and compute $\langle x \rangle^Y = \langle x_0 \rangle^Y + \langle x_1 \rangle^Y$.

4.5.2.4 Arithmetic to Boolean Sharing (A2B)

Converting an arithmetic share $\langle x \rangle^A$ to a Boolean share $\langle x \rangle^B$ can either be done using a Boolean addition circuit (similar to the A2Y conversion described in §4.5.2.3) or by using an arithmetic bit-extraction circuit [ST06, DFK+06, CH10, CS10]. As summarized in §4.2.1, a Boolean addition circuit can either be instantiated as size-optimized variant with $\mathcal{O}(\ell)$ size and depth, or as depth-optimized variant with $\mathcal{O}(\ell \log_2 \ell)$ size and $\mathcal{O}(\log_2 \ell)$ depth (cf. §4.2.1.2). Since the Y2B conversion is for free, we simply compute $\langle x \rangle^B = A2B(\langle x \rangle^A) = Y2B(A2Y(\langle x \rangle^A))$, as our evaluation in §4.6.2 shows that Yao sharing is often more efficient for smaller circuits. In case a higher number of rounds can be tolerated, a lower communication can be achieved using the size-optimized addition circuit (cf. §4.2.1.2) and N-MT (cf. §4.3.3).

4.5.2.5 Boolean to Arithmetic Sharing (B2A)

A simple solution to convert an ℓ-bit Boolean share $\langle x \rangle^B$ into an arithmetic share $\langle x \rangle^A$ is to evaluate a Boolean subtraction circuit where P_0 inputs $\langle x \rangle_0^B$ and a random $r \in_R \{0,1\}^\ell$ and sets $\langle x \rangle_0^A = r$ and P_1 inputs $\langle x \rangle_1^B$ and obtains $\langle x \rangle_1^A = x - r$. However, evaluating such a Boolean subtraction circuit would either have $\mathcal{O}(\ell)$ size and depth or $\mathcal{O}(\ell \log_2 \ell)$ size and $\mathcal{O}(\log_2 \ell)$ depth (cf. §4.2.1).

To improve the performance of the conversion, a technique similar to the arithmetic MT generation described in §4.4.3 can be used. The general idea is to perform an OT for each bit where we obliviously transfer two values that are additively correlated by a power of two. The receiver can obtain one of these values and, by summing them up, the parties obtain a valid arithmetic share.

More detailed, P_0 acts as sender and P_1 acts as receiver in the OT protocol. In the i-th OT, P_0 randomly chooses $r_i \in_R \{0,1\}^\ell$ and inputs (s_i^0, s_i^1) with $s_i^0 = \left(1 - \langle x \rangle_0^B[i]\right) \cdot 2^i - r_i$ and $s_i^1 = \langle x \rangle_0^B[i] \cdot 2^i - r_i$, whereas P_1 inputs $\langle x \rangle_1^B[i]$ as choice bit and receives $s_{\langle x \rangle_1^B[i]} = \left(\langle x \rangle_0^B[i] \oplus \langle x \rangle_1^B[i]\right) \cdot 2^i - r_i$ as output. Finally, P_0 computes $\langle x \rangle_0^A = \sum_{i=1}^{\ell} r_i$ and P_1 computes $\langle x \rangle_1^A = \sum_{i=1}^{\ell} s_{\langle x \rangle_1^B[i]} = \sum_{i=1}^{\ell} \left(\langle x \rangle_0^B[i] \oplus \langle x \rangle_1^B[i]\right) \cdot 2^i - \sum_{i=1}^{\ell} r_i = \sum_{i=1}^{\ell} x[i] \cdot 2^i - \sum_{i=1}^{\ell} r_i = x - \langle x \rangle_0^A$. Security and correctness are similar to the OT-based multiplication protocol in §4.4.3.

Efficiency. Observe that, since we transfer one random element and the other as correlation and only require the $\ell - i$ least significant bits in the i-th OT, we can use C-OT and the same trick outlined in §4.4.3, resulting in (on average) C-OT$^{\ell}_{(\ell+1)/2}$ and a constant number of rounds. In comparison, when evaluating a subtraction circuit using Boolean sharing, the parties would need to evaluate $\mathcal{O}(\ell \log_2 \ell)$ R-OTs for a circuit with depth $\mathcal{O}(\log_2 \ell)$ or 2ℓ R-OTs for a circuit with depth ℓ. Our conversion method is also cheaper than converting to Yao shares (which already requires C-OT$^{2\ell}_{\kappa}$) and doing the subtraction within a garbled circuit.

4.5.2.6 Yao to Arithmetic Sharing (Y2A)

A conversion from a Yao share $\langle x \rangle^Y$ to an arithmetic share $\langle x \rangle^A$ was described in [HKS+10, KSS13a, KSS14]: P_0 randomly chooses $r \in_R \mathbb{Z}_{2^\ell}$, performs $\mathsf{Shr}_0^Y(r)$, and both parties evaluate a Boolean subtraction circuit with $\langle d \rangle^Y = \langle x \rangle^Y - \langle r \rangle^Y$ to obtain their arithmetic shares as $\langle x \rangle_0^A = r$ and $\langle x \rangle_1^A = \mathsf{Rec}_1^Y\left(\langle d \rangle^Y\right)$.

In case the communication is the bottleneck, we compute $\langle x \rangle^A = Y2A\left(\langle x \rangle^Y\right) = B2A\left(Y2B\left(\langle x \rangle^Y\right)\right)$. In case the round complexity is the bottleneck, we use the existing solution of [HKS+10] based on a Boolean subtraction circuit.

4.6 Evaluation

In this section, we evaluate our improved secure computation protocols and circuit constructions. We first compare our depth-efficient circuit constructions from §4.2.1 with their size-optimized counterparts (§4.6.1). We then evaluate our 2-MT and N-MT precomputation methods for GMW from §4.3.2 and §4.3.3 and compare their efficiency with a state-of-the-art implementation of Yao's garbled circuits protocol (§4.6.2). Next, we evaluate our special-purpose protocols, outlined in §4.4, and compare them to an evaluation using GMW (§4.6.3). Finally, we evaluate the efficiency of the transformations between secure computation technique representations from §4.5 (§4.6.4).

We stress that our results provide a reasonably fair comparison even though all results are implementation dependent, since all protocols were implemented in the same programming language, use the same underlying libraries and primitives, and were implemented on a similar optimization level. The results of our experiments can be summarized as follows:

1. §4.6.1: For GMW, the depth-optimized circuit constructions perform better than the size-optimized constructions, even in low-latency networks. However, the size-optimized circuit constructions scale better when the number of parallel operations is increased.

2. §4.6.2: The comparison between 2-MT and N-MT translates to a trade-off between computational power and bandwidth: 2-MT generates MTs three times as fast as N-MT (4.1 million per second for 2-MT vs. 1.4 million per second for N-MT, cf. Table 4.8) but N-MT requires only half of the communication of 2-MT (128 bit per party for 2-MT vs. 69 bit per party for N-MT, cf. Table 4.7) and hence N-MT scales better to settings with high computational power and low bandwidth.

3. §4.6.2: Yao's garbled circuits has a more computation efficient setup phase than GMW with 2-MT but requires more bandwidth per AND gate, making it slower even in a Gigabit Ethernet network (cf. Table 4.7 and Figure 4.9). Yao's garbled circuits with inter party parallelization (IPP) requires the same bandwidth per AND gate in the setup phase as 2-MT and performs similar to 2-MT in the WAN setting but outperforms 2-MT by nearly factor 2× in the LAN setting. In the online phase, Yao's garbled circuits scales in the circuit size while GMW scales mostly in the circuit depth and hence GMW becomes more efficient if the same circuit is evaluated many times in parallel.

4. §4.6.3 and §4.6.4: Our special-purpose protocols can improve the overall run-time for certain operations by up to factor 30× in some settings (cf. Figure 4.11) and can be "mixed" with generic secure computation techniques to achieve up to factor 4× better overall run-time for a single amortized evaluation of standard operations (cf. §4.6.4).

Benchmark Settings. We implement all protocols in C++ and use our semi-honest OT extension implementation from §3 where we process the OTs in blocks of 2^{15}. For all protocols, we separate the run-times into the setup phase, where the parties know the function but the inputs are not available, and the online phase, where the inputs become available and the function is evaluated. For the GMW-style protocols, the setup phase gives the time for performing the OTs and the online time gives the time for evaluating the function. For Yao's garbled circuits, we use state-of-the-art optimizations (cf. §2.4.1) without pipelining and hence the setup phase consists of the time for garbling and sending the circuit while the online time consists of the time for evaluating the garbled circuit. We give separate timings when using the IPP optimization for Yao's garbled circuits, where each party garbles half of the circuit and evaluates the other half, which reduces the overall time for secure computation by nearly half (cf. §4.6.2). The base-OTs require 295 ms for the $\binom{2}{1}$ OT extension protocol and 584 ms for the $\binom{N}{1}$ OT extension protocol but are omitted from the

results, since they present a constant overhead and amortize with growing circuit size. All operations are benchmarked on 32-bit inputs, except for the AES S-Box, which operates on 8-bit inputs. Throughout this section, we assume that the inputs are pre-shared and the outputs remain in a secret-shared form in order to focus on the actual evaluation time of the operation. We instantiate the CRF in Yao's garbled circuits and GMW with 2-MT pre-computation using fixed-key AES (cf. §2.2.4) and in GMW with N-MT pre-computation using AES-256 with key schedule (cf. §3.2.4). All run-times are averaged over 10 executions and, except where explicitly stated, all benchmarks are performed in a single-threaded fashion by bounding the process to one CPU core using the Linux command `taskset`.

Benchmark Environment. For our evaluation, we assume two benchmark settings summarized in Table 4.5: A *LAN* setting and a *WAN* setting. The LAN setting consists of two Desktop PCs (Intel Haswell i7-4770K CPU with 4 cores and 16 GB RAM) that are connected by Gigabit Ethernet. The WAN setting corresponds to a conservative setting where the parties are run on an Amazon EC2 c3.xlarge instance (2.5 GHz Intel Xeon E5-2680v2 CPU, 4 virtual CPUs and 7.5 GB RAM) located in Frankfurt and a Google Compute n1-standard-4 instance (Intel Xeon E5v1-v4, 4 virtual CPUs and 15 GB RAM) located in central US with 28 MBit bandwidth and a 112 ms ping latency. We argue that the WAN setting presents a practical MPC setting, since the machines are controlled by two different cloud providers and located on two different continents.

Name	**Server** P_0	**Client** P_1	**Bandwidth**	**Latency**
LAN	Haswell i7-4770K CPU (4 cores)		876 MBit/s	0.2 ms
WAN	Amazon EC2 c3.xlarge (4 vCPUs)	Google Compute n1-standard-4 (4 vCPUs)	28 MBit/s	112 ms

Table 4.5: Summary of our benchmark environments, bandwidth was measured using `iperf` and latency was measured using `ping`.

4.6.1 Circuit Evaluation

In the following, we compare our depth-efficient circuit constructions to existing size-efficient constructions when evaluated using GMW with our 2-MT pre-computation method (cf. §4.3.2). We omit a comparison using N-MT pre-computation (cf. §4.3.3), since only the setup times would vary compared to 2-MT while the overall the results would be similar. We evaluate the addition (ADD), multiplication (MUL), greater-than (CMP) and S-Box (S-Box) from §4.2.1. We benchmark the circuits in two settings: A *sequential execution* setting, where the operation is evaluated 1 000 times sequentially, and a *parallel execution* setting, where the operation is evaluated 1 000 times

in parallel. An overview of the size and depth of the circuits is given in Table 4.6 and the benchmark results are given in Figure 4.8.

From the results, we can observe that the depth-efficient circuits (D) have a lower total run-time than the size-efficient circuits (S) in both sequential settings and the parallel WAN setting, even though they have up to factor 8× more AND gates. The reason for the better overall run-time of the depth-efficient circuits is that the online time dominates the setup time, due to the latency. In the parallel LAN setting, on the other hand, the setup time is much higher than the online time and hence the size-efficient circuits have a better overall run-time. Clearly, when further increasing the number of parallel operations, the size-efficient circuits also would perform better than the depth-efficient circuits in the WAN setting.

Circuit	**ADD**		**MUL**		**CMP**		**S-Box**	
Version	**S**	**D**	**S**	**D**	**S**	**D**	**S**	**D**
Size	31	272	1 489	1 730	32	89	32	34
Depth	31	11	32	12	32	6	6	4

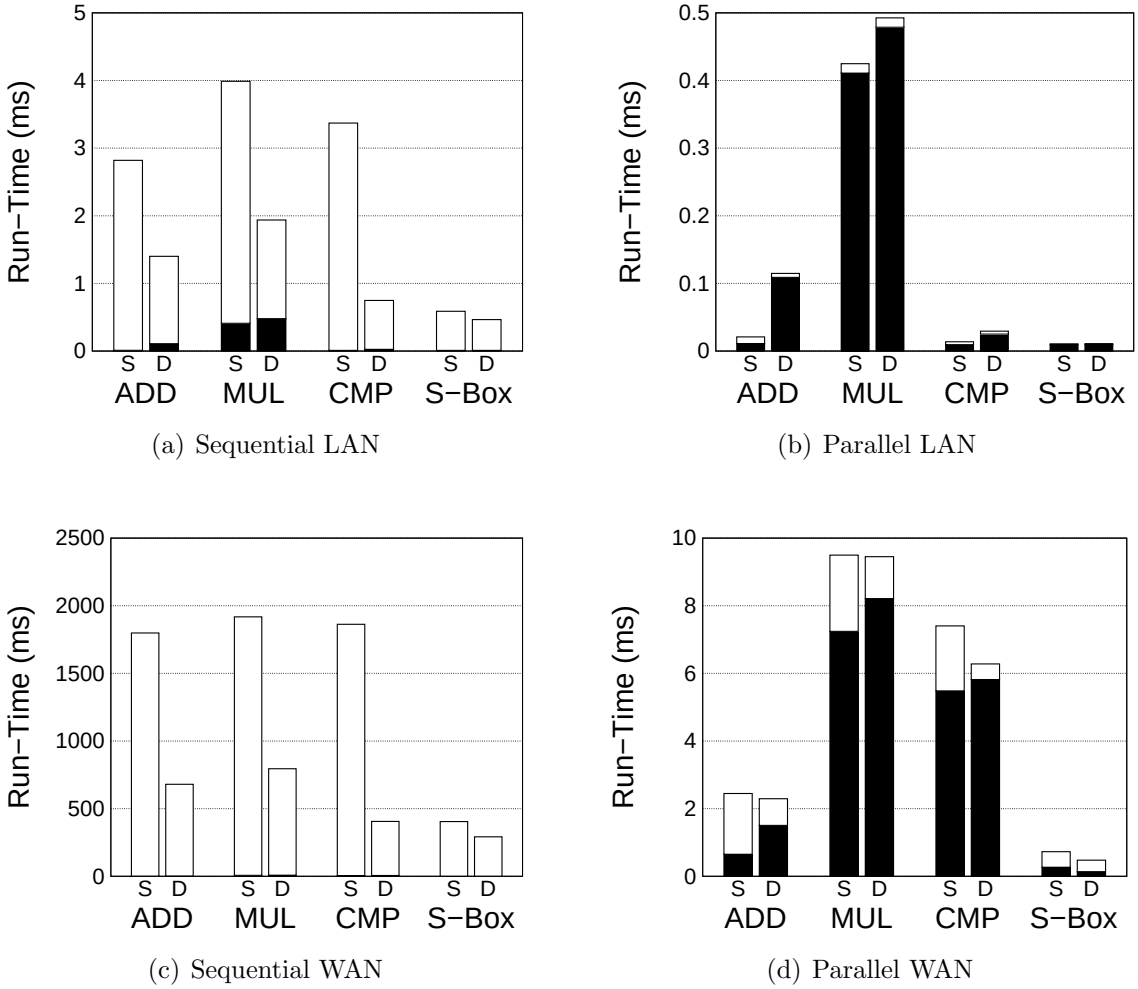
Table 4.6: Circuit size and depth for the size- (S) and depth-optimized (D) circuit constructions outlined in §4.2.1 on 32-bit inputs (8-bit for the AES S-Box).

4.6.2 Evaluation of GMW vs. Yao

In the following, we compare the efficiency of Yao's garbled circuits protocol without and with IPP to the GMW protocol using the 2-MT and N-MT pre-computation protocols. We first conceptually compare the protocols by separating the complexities into computation, communication, and interaction rounds (§4.6.2.1). We then analyze the potential performance of the protocols by running them on one machine with a single thread per party, which allows us to estimate the run-time of the protocols in different environments (§4.6.2.2). Finally, we evaluate the performance of the protocols in our benchmark settings and compare our estimations with the actual run-times (§4.6.2.3). The conceptual comparison can be found in Table 4.7, the potential performance analysis is given in Table 4.8, and the empirical evaluations in our benchmark settings are given in Figure 4.9 and Table 4.9.

4.6.2.1 Conceptual Comparison

We conceptually compare Yao's garbled circuits without and with IPP and GMW using 2-MT and N-MT pre-computation in Table 4.7 and graphically arrange them based on their local computation and communication complexities when evaluating the AES circuit in Figure 4.1 on page 63. From the asymptotic complexities, we can observe that, on average, Yao's garbled circuits requires half of the fixed-key AES-128



(a) Sequential LAN

(b) Parallel LAN

(c) Sequential WAN

(d) Parallel WAN

Figure 4.8: Setup time (black) and online time (white) per operation of size (S) and depth (D) efficient circuit constructions (§4.2) using the GMW protocol with 2-MT generation (§4.3.2) in the LAN (a,b) and WAN setting (c,d) amortized over 1 000 sequential operations (a,c) and over 1 000 parallel operations (b,d).

evaluations of GMW with 2-MT pre-computation. GMW with 2-MT pre-computation is again much faster than GMW with N-MT pre-computation, which requires a similar number of more expensive evaluations of AES-256 with key schedule (cf. §3.2.4.2). In terms of communication, Yao's garbled circuits without and with IPP has the same total communication as GMW with 2-MT pre-computation and both have roughly twice the communication of GMW with N-MT pre-computation. The number of interaction rounds for Yao's garbled circuits is constant, independently of the function, while the

GMW protocol requires $\mathbf{D}(\mathrm{C})$ communication rounds in the online phase, where $\mathbf{D}(\mathrm{C})$ is the multiplicative depth of the circuit C. Compared to the previous state-of-the-art GMW implementation of [CHK+12] (cf. §2.4 on page 22), our 2-MT and N-MT pre-computation methods increase the computation by at least one additional AES evaluation but reduce the communication by factor $2\times$ and $4\times$, respectively.

Complexity	**Yao**		**Yao-IPP**	**2-MT**	**N-MT**
	P_0	P_1	P_0 & P_1	P_0 & P_1	P_0 & P_1
Computation	4 AES	2 AES	3 AES	6 AES	0.75 AES + 4.25 AES256
Data Sent [Bytes]	258	0	129	129	69
Rounds	$\mathcal{O}(1)$		$\mathcal{O}(1)$	$\mathbf{D}(\mathrm{C})$	$\mathbf{D}(\mathrm{C})$

Table 4.7: Asymptotic complexities per AND gate for Yao's garbled circuits without and with IPP (cf. §2.4.1.2) and the GMW protocol using 2-MT (cf. §4.3.2) and N-MT pre-computation (cf. §4.3.3) given separately for both parties. AES refers to an evaluation of fixed-key AES-128 and AES256 refers to AES-256 with key-schedule (cf. §3.2.4.2). Note that we count the communication for Yao's garbled circuits including the permutation bit of the point-and-permute technique (cf. §2.4.1.2) which is added to each garbled table entry.

4.6.2.2 Potential Performance

To obtain more accurate performance estimations of the protocols, we run the protocols on one machine in our LAN setting with a single thread, which resembles an idealized network with 72.3 GBit/s bandwidth and 0.01 ms ping latency. We then use these performance numbers to estimate the run-times in different benchmark scenarios. We evaluate 8 192 parallel AES circuits with size-optimized S-Box circuits (cf. §4.2.1.12), which have a total of 42 million AND gates and a depth of 60. Note that we evaluate functionalities multiple times in parallel instead of sequential to keep the circuit depth and hence the communication rounds for GMW constant. A sequential instead of parallel circuit evaluation would increase the online time of GMW linear in the depth overhead (cf. §4.6.1) but would not change the setup run-time, which is independent of the circuit structure. We give the resulting run-times, the number of AND gates per second for the setup and online phase, and the generated communication overhead in the setup phase in Table 4.8.

Protocol Run-Times. Our first observation from the local run-times, is that the total run-time for 2-MT is lower than that for Yao's garbled circuits without IPP, even though each party has more computational workload in 2-MT. This can be explained by our load balancing optimization (cf. §4.3.1), which allows the parties to compute fully in parallel to each other while for Yao's garbled circuits, the evaluator has to wait

Complexity	**Yao**		**Yao-IPP**	**2-MT**	***N*-MT**
	P_0	P_1	P_0 & P_1	P_0 & P_1	P_0 & P_1
Setup Time [s]	9.04	0	4.66	10.16	30.14
Online Time [s]	0	3.88	1.87	0.04	
Total Time [s]	12.92		6.53	10.20	30.18
Setup [ANDs / s]	$4.6 \cdot 10^6$	0	$9.0 \cdot 10^6$	$4.1 \cdot 10^6$	$1.4 \cdot 10^6$
Setup Data Sent [MBit / s]	1 133	0	1 099	504	89
Online [ANDs / s]	0	$10.8 \cdot 10^6$	$22.4 \cdot 10^6$	$1\,049 \cdot 10^6$	

Table 4.8: Run-time for a local evaluation of 8 192 parallel AES circuits with 42 million AND gates for Yao's garbled circuits without and with IPP (cf. §2.4.1.2) and the GMW protocol using 2-MT (cf. §4.3.2) and N-MT pre-computation (cf. §4.3.3) with a single thread per party.

for the garbler to finish the circuit garbling.[3] This is improved on by Yao with IPP, where both parties evenly distribute and perform the workload in parallel, reducing the total run-time by nearly factor 2× compared to Yao without IPP. Yao with IPP overall performs better than 2-MT, due to its efficient setup phase, which is more than factor 2× better than the setup phase of 2-MT. N-MT is the slowest protocol, with a factor 3× more total run-time than 2-MT and nearly factor 5× more run-time than Yao with IPP. The high run-time of N-MT can be explained by the expensive CRF instantiation using AES-256 with key schedule (cf. §3.2.4.2). However, both 2-MT and N-MT have a very fast online run-time, compared to Yao's garbled circuits, since the online phase consists only of efficient one-time pad operations on single bits and the latency is low due to the execution on the same machine. In contrast, in the online phase of Yao's garbled circuits, the evaluator has to process symmetric keys using more expensive symmetric cryptographic operations.

Performance Estimations. Given the run-times on the same machine, we can estimate the performance of each protocol in different settings. We assume that the computation time on all machines in our benchmark settings is similar and that a multi-threaded evaluation achieves perfect parallelization.
As a first observation, note that Yao's garbled circuits without and with IPP generates more than one GBit per second of data that needs to be sent over the network. Hence, we estimate that both protocols achieve lower performance in all of our benchmark environments, where the highest bandwidth is 876 MBit/s for the LAN setting. Using a multi-threaded evaluation of Yao's garbled circuits would decrease the online time

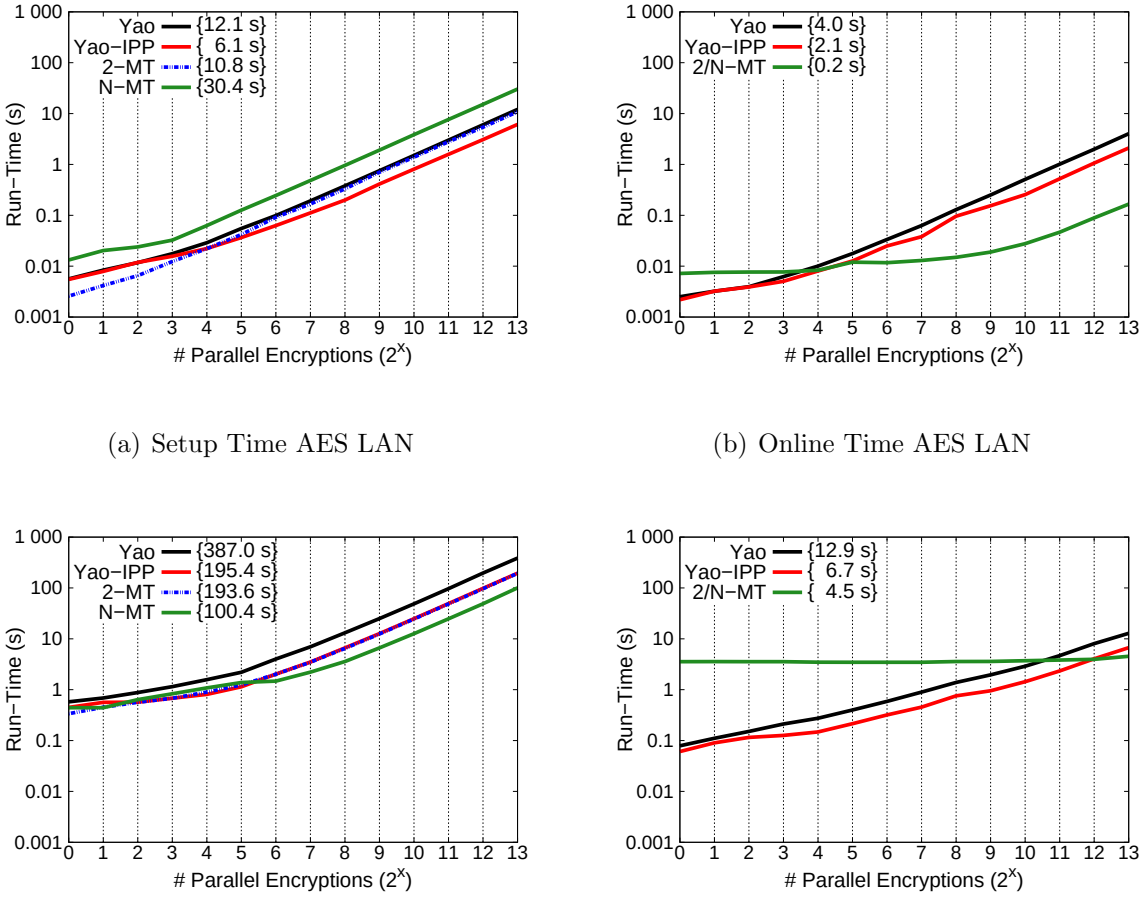
[3] Note that the total run-time for Yao's garbled circuits could be improved using a pipelined garbling and evaluation approach [HEKM11, HS13], which we omitted due to our use of the pre-computation model.

but not the setup time since a single thread is sufficient to fully utilize the bandwidth in all settings.
2-MT generates data at a rate of 504 MBit/s in the setup phase and a two-threaded evaluation of the setup phase would hence improve performance in the LAN setting but not in the WAN setting. However, 2-MT can at most achieve the same setup time as Yao's garbled circuits with IPP, since both protocols have the same communication overhead.
N-MT achieves the lowest performance in the setup phase but has the highest potential for parallelization. Assuming perfect parallelization, N-MT would require 10 threads to fully utilize the bandwidth in the LAN setting, and would then achieve nearly twice the performance of Yao with IPP in the setup phase, corresponding to its communication advantage. In the WAN setting, a single thread for N-MT would suffice to fully utilize the bandwidth. Furthermore, note that if the available bandwidth in the execution environment is low, which is the case for our WAN setting, our N-MT protocol would perform best among all protocols even when using only a single thread. Based on the performance numbers, we estimate that N-MT would perform best if the bandwidth is lower than 170 MBit/s, at which both Yao's garbled circuits and 2-MT would process the same number of AND gates per second in the setup phase as N-MT.

4.6.2.3 Performance Evaluation

We verify our performance estimations of the protocols by evaluating the AES circuit using a single-threaded evaluation in our LAN and WAN settings and a multi-threaded evaluation in our LAN setting. We omitted the multi-threaded evaluation in the WAN setting, since the run-times did not change compared to a single-threaded evaluation. We simulated a multi-threaded evaluation of Yao's garbled circuits by running multiple program executions on the respective fraction of circuits. The single-threaded results in the LAN and WAN setting for 2^0 (= 1) to 2^{13} (= 8 192) parallel AES evaluations are given in Figure 4.9 while the multi-threaded run-times for 2^{13} parallel AES circuits are given in Table 4.9.

Single-Threaded Run-Times (Figure 4.9). The single-threaded evaluations confirm our estimations: The setup time for both Yao's garbled circuits without and with IPP increases in the LAN setting by factor $\sim 1.3\times$, corresponding to the decrease in available bandwidth, while the online time for both remains similar. On the other hand, the setup time for 2-MT and N-MT increases only by factor $\sim 1.06\times$ but both their online times increase by factor $5\times$, which is due to the added latency. These trends are even more visible in the WAN setting, where the setup time of both Yao's garbled circuits protocols is increased by factor $30\times$ and their online times are increased by factor $3\times$ while the setup times of 2-MT and N-MT is increased by factor $20\times$ and $3\times$, respectively, but their online times are increased by factor $22\times$ due to the higher latency.
Regarding the online time, we can observe that in both, the LAN and WAN settings,



(a) Setup Time AES LAN

(b) Online Time AES LAN

(c) Setup Time AES WAN

(d) Online Time AES WAN

Figure 4.9: Single-threaded setup and online time when evaluating 2^0 to 2^{13} parallel AES circuits (5 120 AND gates per AES circuit) using Yao's garbled circuits protocol without and with IPP (§2.4.1.2) and the GMW protocol with 2-MT (§4.3.2) and N-MT generation (§4.3.3) in the LAN (a,b) and WAN (c,d) setting for increased number of parallel encryptions. The time in {} gives the run-time for performing 2^{13} parallel encryptions.

GMW outperforms Yao's garbled circuits with increasing number of parallel encryptions. This can be explained by the high impact of the latency on GMW, which presents a constant overhead but amortizes with increasing number of parallel circuits while Yao's garbled circuits needs to perform work linear in the number of gates. In the WAN setting, the break-even point in the online phase happens much later than in the LAN setting, due to the higher latency.

Protocol	1 Thread	2 Threads	3 Threads	4 Threads	Imp. $4 \mapsto 1$
Setup Time					
Yao	12.07	11.73	11.72	11.72	1.03×
Yao-IPP	6.13	5.96	5.94	5.94	1.03×
2-MT	10.87	5.95	5.77	5.77	1.88×
***N*-MT**	30.40	15.23	10.15	7.79	3.90×
Online Time					
Yao	4.03	2.11	1.39	1.05	3.84×
Yao-IPP	2.10	1.12	0.80	0.67	3.13×
2-MT & *N*-MT	0.16	0.16	0.16	0.16	1.00×

Table 4.9: Setup and online time for an increasing number of threads and time improvement of 4 threads over 1 thread when evaluating the AES circuit 2^{13} (= 8 192) times in parallel in the LAN setting.

Multi-Threaded Run-Times (Table 4.9). We increase the computational power in the LAN setting by increasing the number of threads from one to four and give the setup and online time for evaluating the AES circuit 2^{13} times in parallel. The results are also in line with our expectations: The setup time for Yao's garbled circuits hardly decreases with multiple threads, 2-MT achieves a speed-up of nearly factor 2×, and N-MT achieves nearly linear speed-up in the number of threads. For the online time, both Yao's garbled circuits with and without IPP achieve a speed-up of more than factor 3×, since the online phase only involves local computation. The online phase of the GMW protocol does not improve when using multiple threads, since the implementation is single-threaded as the main bottleneck in the GMW online phase is the network latency.

4.6.3 Evaluation of Special-Purpose Protocols

We introduced three special-purpose protocols that can be used to improve the performance for certain operations: Vector MTs (cf. §4.4.1), SP-LUT (cf. §4.4.2.3), and our OT-based multiplication protocol (cf. §4.4.3). In this section, we empirically evaluate the performance gains of these protocols on two example functionalities: The AES circuit and the integer multiplication circuit. We give the resulting run-times in Figure 4.10 and Figure 4.11.

Vector MTs (§4.4.1). Vector MTs can be used together with GMW and reduce the number of AND gates whenever the same wire is used in multiple AND gates. We apply our vector MT optimization to several circuits described in §4.2.1 and give the resulting efficiency gains in Table 4.10. The best improvement can be seen for the multiplexer circuit, where for ℓ-bit inputs, the number of AND gates is reduced by factor ℓ×. Other than the multiplexer, the vector MT optimization can be applied to the size-

Circuit	Size Regular	Size Vector MT	Improvement
Ladner-Fischer ADD	272	192	1.42×
Ripple-Carry Network MUL	1 489	993	1.50×
Carry Save Network MUL	1 730	1 234	1.40×
MUX	32	1	32.00×
Depth-Optimized AES S-Box	34	23	1.48×

Table 4.10: Regular and vector MT optimized circuit sizes for circuits from §4.2.1 with 32-bit inputs (8-bit for the AES S-Box).

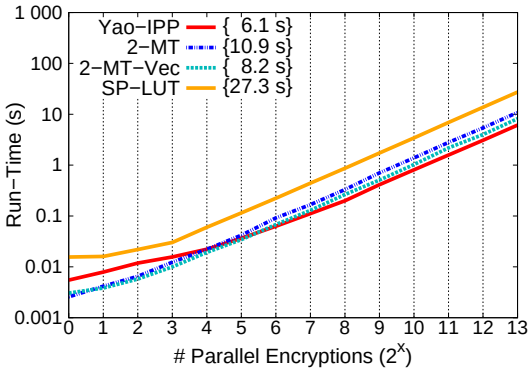
and depth-optimized multipliers, the Ladner-Fischer adder, and the AES S-Box, where it reduces the number of AND gates by factor $\sim 1.4\times$. We empirically evaluate and show the performance gains for GMW with 2-MT using the S-Box circuit within the AES application in Figure 4.10 and for the multiplication circuit in Figure 4.11.

From the empirical results we can observe that the setup run-time for 2-MT with vector MTs improves roughly linear with the reduced number of AND gates. The online time in the LAN setting increases when using vector MTs, since our vector MT implementation processes bits individually, whereas the standard 2-MT generation processes multiple bits in parallel. Also note that the vector MT optimization improves the online time of the AES circuit in the WAN setting, since we optimize and use the depth-efficient S-Box with depth 4, whereas the 2-MT routine uses the size-efficient S-Box with depth 6 (cf. §4.2.1.12), which scaled better for larger number of parallel encryptions.

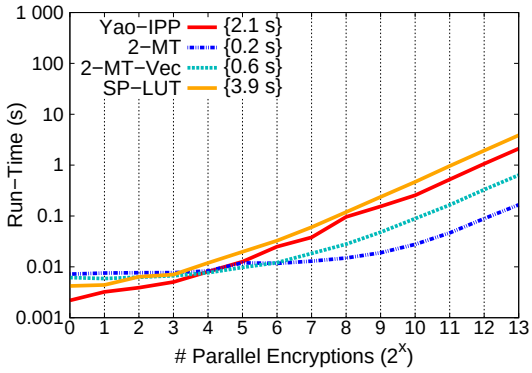
SP-LUT (§4.4.2.3). In contrast to the previous experiments, our SP-LUT protocol represents the functionality as a network of multi-input lookup tables (LUTs) that are evaluated securely. A functionality that can be expressed easily as such a network of LUTs is the AES S-Box, which has 8 input and output bits. We represent the AES circuit as a LUT network by building a 8-input to 8-output LUT for the AES S-Box, which we evaluate securely using our SP-LUT protocol. The remaining operations in AES can be done via local XOR operations or remapping wires and hence can be done locally by each party. Hence, for each S-Box, our SP-LUT protocol needs to perform 256 CRF evaluations, instantiated by AES-256 with key-schedule (cf. §3.2.4.2), send 255 bits in the setup phase and 2 048 bits in the online phase, and perform 11 communication rounds in the online phase (cf. Table 4.3 on page 84). We compare our SP-LUT protocol to an evaluation using Yao's garbled circuits with IPP and GMW with 2-MT with and without vector MTs in the LAN and WAN setting. The results can be found in Figure 4.10.

From our results we can observe that our SP-LUT protocol performs worse than all other protocols in the LAN setting, which is due to its high computation overhead. In the WAN setting, however, our SP-LUT protocol is the overall fastest protocol due to its low communication. In contrast to the other protocols, our SP-LUT protocol has a

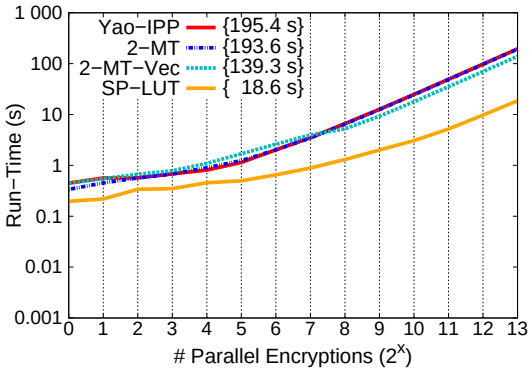
fast setup phase but an inefficient online phase, which is due to its high communication overhead (per S-Box 255 bits in the setup phase and 2 048 bit in the online phase).



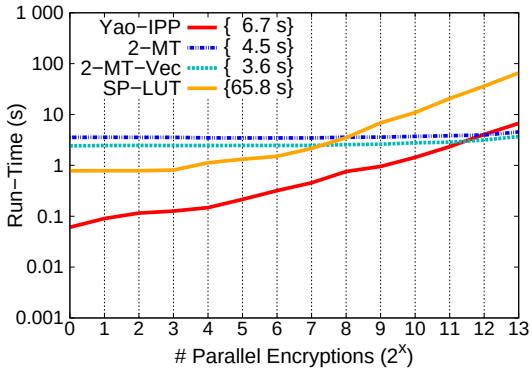
(a) Setup Time AES LAN



(b) Online Time AES LAN



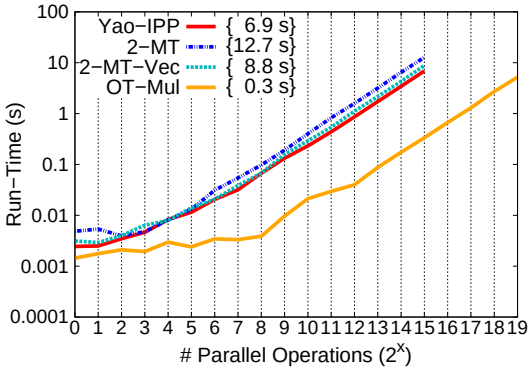
(c) Setup Time AES WAN



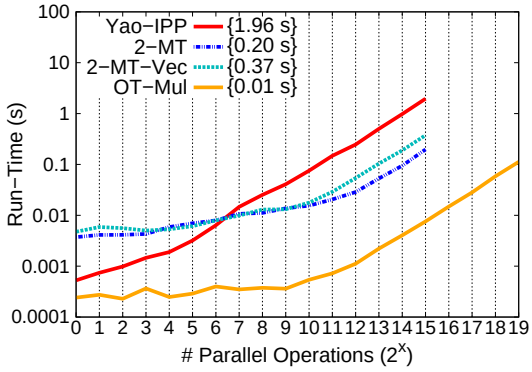
(d) Online Time AES WAN

Figure 4.10: Setup and online run-time for secure evaluation of the AES circuit using Yao's garbled circuits with IPP (§2.4.1.2), the GMW protocol with 2-MT (§4.3.2) with and without our vector MT optimization (§4.4.1) and our special-purpose SP-LUT protocol (§4.4.2.3) in the LAN (a,b) and WAN (c,d) setting. The time in {} gives the run-time for performing 2^{13} parallel encryptions.

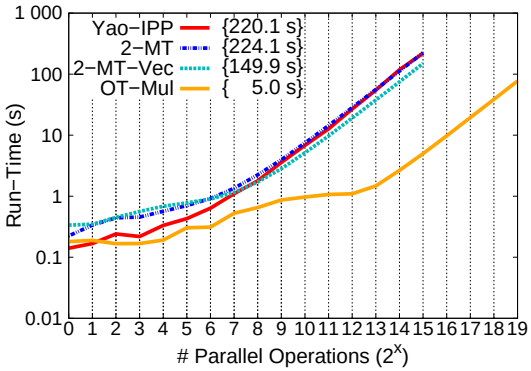
OT-Based Multiplication (§4.4.3). We evaluate and compare the efficiency of the Boolean circuit-based integer multiplication with our special-purpose OT-based integer multiplication protocol. We benchmark Yao's garbled circuits and GMW with 2-MT pre-computation with and without vector MTs using the size-optimized Boolean circuit



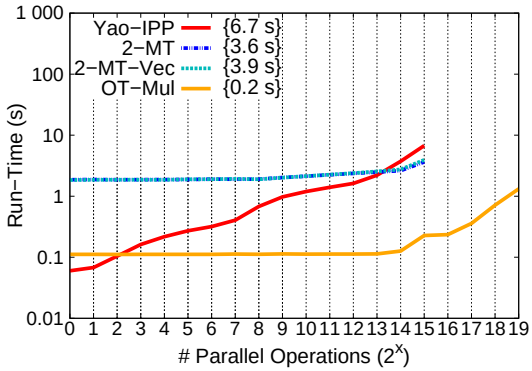
(a) Setup Time Multiplication LAN



(b) Online Time Multiplication LAN



(c) Setup Time Multiplication WAN



(d) Online Time Multiplication WAN

Figure 4.11: Run-time for secure evaluation of the integer multiplication circuit using Yao's garbled circuits with IPP (§2.4.1.2), the GMW protocol with 2-MT (§4.3.2) with and without our vector MT optimization (§4.4.1) and our special-purpose protocols OT-based multiplication protocol (§4.4.3) in the LAN (a,b) and WAN (c,d) setting. The time in {} gives the run-time for performing 2^{15} parallel multiplications.

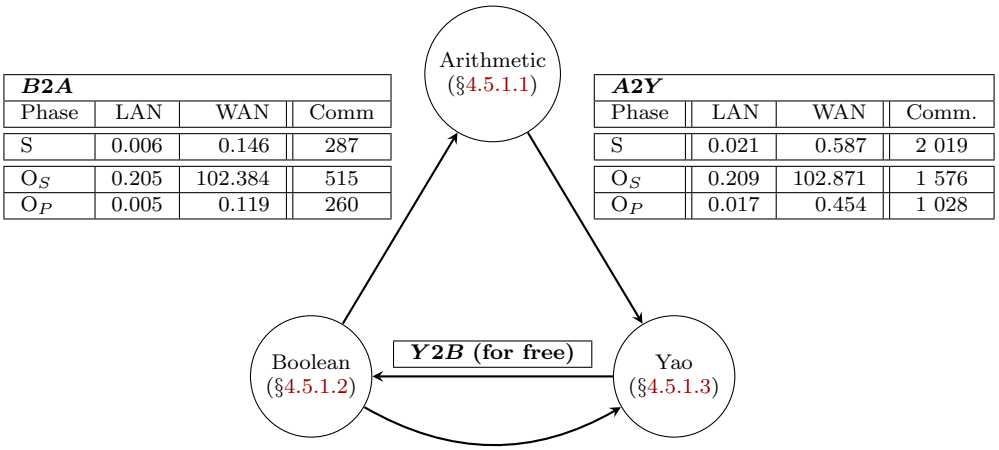
for integer multiplication with 1 489 AND gates and a depth of 32 (cf. §4.2.1.5) and perform up to 2^{15} (= 32 768) parallel multiplications. For our OT-based multiplication protocol, we perform up to 2^{19} (= 524 288) parallel multiplications to explicitly show the amortization. The results can be found in Figure 4.11.

From the results we can observe that our OT-based multiplication protocol outperforms all other Boolean circuit-based protocols except for the online time of Yao's

garbled circuits in the WAN setting when evaluating only few parallel multiplications. In this case, Yao's garbled circuits protocol performs better than our OT-based multiplication protocol, since it has one communication round less. For a larger number of multiplications, the OT-based multiplication protocol outperforms the other protocols by factor 20× to 30×.

4.6.4 Evaluation of Mixed Protocols

An important observation of the previous evaluations is that no protocol consistently performed best for all operations and benchmark settings. To combine the benefits of multiple protocols and achieve consistently good performance over multiple benchmark settings, we categorized our protocols into different representations (cf. §4.5.1): Arithmetic (A), Boolean (B), and Yao's garbled circuits-based (Y), and outlined protocols for transforming between them (cf. §4.5). In this section, we benchmark three of the six protocols for transforming between the different representations: Arithmetic to Yao (A2Y), Boolean to Arithmetic (B2A), and Boolean to Yao (B2Y). The Yao to Boolean (Y2B) transformation only involves local operations and hence is for free while the remaining two protocols can be obtained by concatenating two transformations.



B2A

Phase	LAN	WAN	Comm
S	0.006	0.146	287
O_S	0.205	102.384	515
O_P	0.005	0.119	260

A2Y

Phase	LAN	WAN	Comm.
S	0.021	0.587	2 019
O_S	0.209	102.871	1 576
O_P	0.017	0.454	1 028

B2Y

Phase	LAN	WAN	Comm
S	0.007	0.224	515
O_S	0.204	103.248	1 055
O_P	0.011	0.344	1 028

Figure 4.12: Setup (S) and online (O_S, O_P) time (in ms) and communication (in Bytes) for transformations from one share representation to another in the LAN and WAN settings amortized over 1 000 sequential (O_S) and 1 000 parallel operations (O_P).

From the results, we can observe that the transformations all have similar secure evaluation costs. Furthermore, some operations can be performed more efficiently by trans-

forming to another representation, performing the operation in this representation, and transforming back than by performing the operation in the current representation. For instance, a single amortized sequential multiplication in Yao's garbled circuits with IPP requires 1.443 ms in the setup phase and 0.241 ms in the online phase in the LAN setting. On the other hand, a transformation from Yao's garbled circuits to the arithmetic representation, a multiplication in the arithmetic representation, and the transformation back to Yao's garbled circuits requires $0.006 + 0.020 + 0.021 = 0.047$ ms in the setup (factor 30× improvement) and $0.205 + 0.117 + 0.209 = 0.531$ ms in the online phase (factor 2× decrease). Overall, this reduces the total run-time by factor 3× from 1.684 ms for Yao's garbled circuits to 0.578 ms for the full round of transformations.

4.7 Applications

We evaluate the performance of our improved protocols and compare them to state-of-the-art Yao's garbled circuits on two typical applications in secure computation: private set intersection (§4.7.1) and biometric identification (§4.7.2).

4.7.1 Private Set Intersection (PSI)

PSI allows two parties to identify the intersection of their sets X and Y without revealing any element that is not in the intersection. An efficient Boolean circuit that computes the PSI functionality is the sort-compare-shuffle (SCS) circuit that was introduced in [HEK12] (cf. §5.3.1 for more details on the SCS circuit). The circuit computes the intersection by fist *sorting* the elements, then *comparing* adjacent elements, and finally *shuffling* the outputs to hide information that could be obtained from the position of an intersecting element. We build two versions of the SCS circuit: A size-optimized version for Yao's garbled circuits that uses the sequential comparison circuit and has $\sim 3\sigma n \log_2(2n)$ AND gates and depth $\sim \sigma \log_2 n$ and a depth-optimized version for GMW which uses the divide-and-conquer comparison circuit and vector MTs and has approximately the same size but depth $\sim \log_2(\sigma)\log_2(n)$ (cf. §4.2.1.7 for the comparison circuits). We give the concrete circuit sizes and depths in Table 4.11. We benchmark and compare state-of-the-art Yao's garbled circuits with IPP (cf. §2.4.1.2) to GMW with our 2-MT (§4.3.2) and N-MT (§4.3.3) pre-computation methods using a single thread. We fix the bit-length of elements to $\sigma = 32$ and increase the set sizes in steps of powers of two from 2^1 (= 1) to 2^{15} (= 32 768). For Yao's garbled circuits, we apply IPP by switching the roles after the sort and before the compare and shuffle sub-circuits. By switching the roles, we add two additional communication rounds for Yao's garbled circuits but achieve nearly factor 2× better run-times due to the load balancing. We give the resulting setup and online run-times in the LAN and WAN setting in Figure 4.13.

The results for the PSI circuit are similar to the results for the AES circuit in §4.6.2.3:

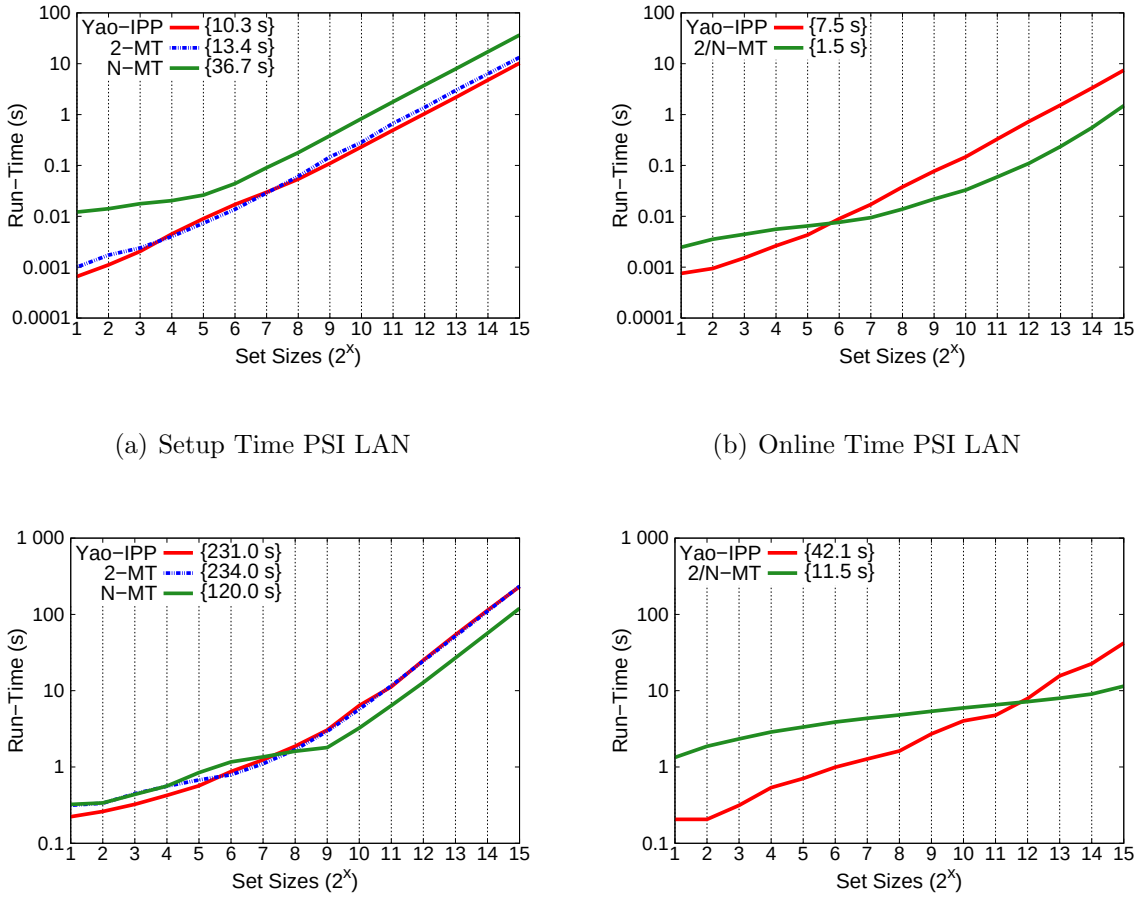
Set Size	2^1	2^3	2^5	2^7	2^9	2^{11}	2^{13}	2^{15}
Size Yao	446	3 320	19 424	102 272	507 392	2 422 784	11 264 000	51 347 456
Size GMW	519	3 625	20 417	105 057	513 505	2 426 849	11 198 433	50 757 601
Depth GMW	24	42	60	78	96	114	132	150

Table 4.11: Size in ANDs and depth of the SCS PSI circuit of [HEK12] in a size-optimized version for Yao's garbled circuits and a depth-optimized version for GMW.

While Yao's garbled circuits performs best overall in both settings for smaller sets, GMW achieves better overall performance for larger sets.
In the LAN setting, the online phase of GMW scales better with increasing set sizes, since the circuit depth of $\log_2(\sigma)\log_2(n)$ is comparably high for small values but scales only with $\log_2(n)$ while the online phase of Yao's garbled circuits scales directly in the circuit size, i.e., with $\mathcal{O}(n \log_2 n)$. For larger set sizes, we can observe that the online phase of GMW also scales similar to Yao's garbled circuits, due to the amortization of the circuit depth by the local computation.
In the WAN setting, the setup and online phase of N-MT scale better with increasing circuit sizes than Yao's garbled circuits, due to its smaller communication overhead in the setup phase and latency dependence in the online phase. Interestingly, the setup phase of N-MT in the WAN setting first scales rather poorly up to set sizes of 2^6, then scales fairly well until the set sizes reach 2^9, and from then scales similar to the other protocols. This can be explained by the block-wise evaluation of our OT extension implementation: Until set sizes of 2^6, all OTs are evaluated in a single block. From set sizes 2^6 to 2^9, multiple blocks are evaluated in parallel, achieving better throughput and bandwidth utilization. After set sizes of 2^9, the bandwidth is saturated, resulting again in a scaling with $\mathcal{O}(n \log_2 n)$.

4.7.2 Biometric Matching

In the privacy-preserving biometric matching application, one party wants to determine whether its biometric sample matches one of several biometric samples that are stored in a database held by another party. Several protocols for privacy-preserving biometric matching have been proposed, e.g., for face-recognition [EFG+09, HKS+10] or fingerprint-matching [BG11, HMEK11]. A fundamental building block of these protocols is to compute the squared Euclidean distance between the query and all biometric samples in the database and afterwards determine the minimum value among these distances. For our experiments we use similar parameters as [KSS14]: Each sample has $d = 4$ dimensions and each element is 32-bit long, but we increase the database size in powers of two from $n = 2^0$ $(= 1)$ to $n = 2^{13}$ $(= 8\ 192)$ entries. More specifically, we securely compute $min\left(\sum_{i=1}^{d}(S_{i,1} - C_i)^2, \cdots, \sum_{i=1}^{d}(S_{i,n} - C_i)^2\right)$ where P_0 inputs the database $S_{i,j}$ and P_1 inputs the query C_i (cf. [KSS14]). We use two sets



(a) Setup Time PSI LAN

(b) Online Time PSI LAN

(c) Setup Time PSI WAN

(d) Online Time PSI WAN

Figure 4.13: Setup and online time for evaluating the PSI circuit of [HEK12] using Yao's garbled circuits protocol with IPP (cf. §2.4.1.2) and the GMW protocol with 2-MT (cf. §4.3.2) and N-MT generation (cf. §4.3.3) in the LAN (a,b) and WAN (c,d) setting for increasing set sizes. The time in {} gives the run-time for processing 2^{15} elements.

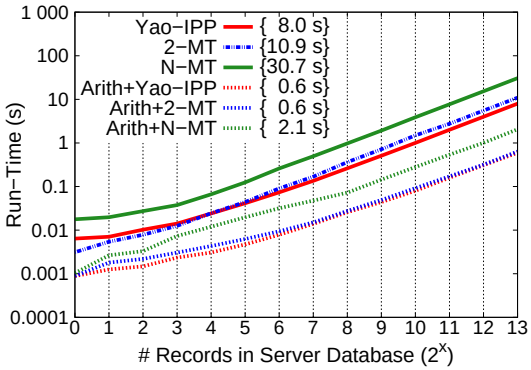
of protocols to evaluate the biometric matching circuit. In the first set, we compute the squared Euclidean distance and minimum using Yao's garbled circuits protocol with IPP and the GMW protocol with 2-MT and N-MT pre-computation. In the second set, we compute the squared Euclidean distance using our OT-based multiplication protocol (cf. §4.4.3) and then use our share transformation protocols (cf. §4.5.2) to evaluate the minimum using Yao's garbled circuits or GMW with 2-MT and N-MT. For Yao's garbled circuits, we use IPP to evaluate and compute the minimum of two

halves of the server's database and then switch roles to identify the smaller of both minimums, which adds two communication rounds. We give the resulting circuit sizes of the biometric matching circuit in Table 4.12 and the setup and online run-times for the LAN and WAN setting in Figure 4.14.

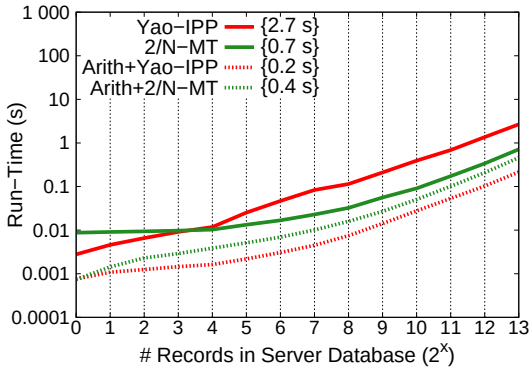
Database Size	2^1	2^3	2^5	2^7	2^9	2^{11}	2^{13}
Pure Instantiation							
Size Yao	13 674	54 888	219 744	879 168	3 516 864	14 067 648	56 270 784
Size GMW	10 188	41 022	164 358	657 702	2 631 078	10 524 582	42 098 598
Depth GMW	47	61	75	89	103	117	131
Mixed-Protocol							
Size Yao	126	696	2 976	12 096	48 576	194 496	778 176
Size GMW	90	630	2 790	11 430	45 990	184 230	737 190
Depth GMW	15	29	43	57	71	85	99

Table 4.12: Size and depth of the biometric matching circuit in a size-optimized version for Yao's garbled circuits and a depth-optimized version for GMW as pure instantiation and mixed with our OT-based multiplication (cf. §4.4.3).

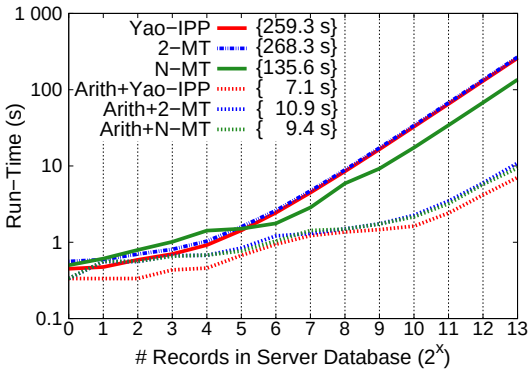
The results for the biometric matching application show that mixed-protocols always outperform the pure protocols, both in the setup phase and for most of the protocols also in the online phase. The only exception is the WAN setting, where the online phase of Yao's garbled circuits is more efficient for less than 2^6 elements in the database, which is due to the fewer communication rounds (5 rounds for pure Yao IPP and 8 rounds for Yao IPP with OT-based multiplication). The run-times for the pure protocols are similar to the run-times for the AES (cf. §4.6.2.3) and PSI (cf. §4.7.1) examples: Yao's garbled circuits performs better than GMW for small sizes while GMW performs better for larger sizes. The results for the mixed-protocols, on the other hand, differ from this pattern, since mixed-protocols that use Yao's garbled circuits always perform better than mixed-protocols that use GMW. This can be explained by the small Boolean circuit sizes in the mixed-protocols, which only go as high as 800 000 AND gates while the other circuit sizes go as high as 42 to 56 million AND gates (cf. Table 4.12). If the server's database would grow larger, we would see the same pattern also for the mixed-protocols.



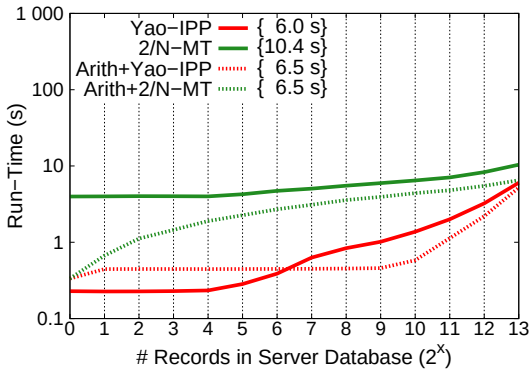
(a) Setup Time Biometric Matching LAN



(b) Online Time Biometric Matching LAN



(c) Setup Time Biometric Matching WAN



(d) Online Time Biometric Matching WAN

Figure 4.14: Setup and online time for evaluating the biometric matching circuit using Yao's garbled circuits protocol with IPP (cf. §2.4.1.2) and the GMW protocol with 2-MT (cf. §4.3.2) and N-MT generation (cf. §4.3.3) in a pure instantiation or as a mixed-protocol using our arithmetic OT-based multiplication protocol (§4.4.3) in the LAN (a,b) and WAN (c,d) setting for increasing database size. The time in {} gives the run-time for processing 2^{13} records using the {pure/mixed} protocols.

5 Faster Private Set Intersection

Private set intersection (PSI) is a prominent application in secure computation, for which several protocols based on different techniques have been proposed. However, previous protocols are two to three orders of magnitude slower than insecure solutions that are currently used in practice. In addition, existing protocols were often evaluated and compared in biased settings, making it hard to identify which protocol to use in a particular setting. In this chapter, we review, categorize, optimize, and compare existing PSI protocols. Furthermore, we propose a novel PSI protocol that is based on our OT extension improvements from §3 and reduces the overhead over insecure practical solutions to less than a single order of magnitude.

Remark. Parts of this chapter have been published in [PSZ14, PSSZ15] and have been combined into a journal paper that is currently in submission [PSZ16]. The novel protocol is the result of extensive research collaborations of the authors from TU Darmstadt with Benny Pinkas (Bar-Ilan University, Israel) and Gil Segev (The Hebrew University of Jerusalem, Israel). The implementations are available online at `http://github.com/encryptogroup/PSI` and `http://github.com/encryptogroup/ABY`.

5.1 Motivation

Private set intersection (PSI) allows two parties P_0 and P_1 holding sets X and Y with sizes $|X| = n_0$ and $|Y| = n_1$, respectively, to identify the intersection $X \cap Y$ without revealing any information about elements that are not in the intersection. The basic PSI functionality can be used in applications where two parties want to perform JOIN operations over database tables that they must keep private, e.g., private lists of preferences, properties, or personal records of clients or patients. PSI was used in several research projects for privacy-preserving computation of functionalities such as relationship path discovery in social networks [MPGP09], botnet detection [NMH+10], testing of fully-sequenced human genomes [BBC+11], proximity testing [NTL+11], or cheater detection in online games [BHLB11]. Moreover, the PSI functionality has several practical applications, of which we list three in the following:

Measuring Ad Conversion Rates. Online advertising, which is a huge business, typically measures the success of ad campaigns by measuring the success of converting viewers into customers. A popular method of measuring ad performance is to compare the list of people who have seen an ad with those who have completed a transaction.

These lists are held by the advertiser (say, Google or Facebook), and by merchants, respectively. It is often possible to identify users on both ends, using identifiers such as credit card numbers, email addresses, etc. A simple solution, which ignores privacy, is for one side to disclose its list of customers to the other side, which then computes the necessary statistics. Another option is to run a PSI protocol between the two parties. (The protocol should probably be a variant of PSI, e.g. compute total revenues from customers who have seen an ad. Such protocols can be derived from basic PSI protocols.) In fact, Facebook is running a service of this type with Datalogix, Epsilon and Acxiom, companies which have transaction records for a large part of loyalty card holders in the US. According to reports[1], the computation is done using a variant of the *insecure* naive hashing PSI protocol, where the parties hash their elements and compare the hashes. The ad conversion rate application typically needs to perform PSI on large sets.

Security Incident Information Sharing. Security incident handlers can benefit from information sharing since it provides them with a global view during incidents. However, incident data is often sensitive and potentially embarrassing. The shared information might reveal information about the business of the company that provided it, or of its customers. Therefore, information is typically shared rather sparsely and protected using legal agreements. Automated large scale sharing will improve security, and there is in fact work to that end, such as the IETF Managed Incident Lightweight Exchange (MILE) effort. Many computations that are applied to the shared data compute the intersection and its variants. Applying PSI to perform these computations can simplify the legal issues of information sharing. Efficient PSI protocols will enable it to be run often and at large scale.

Private Contact Discovery. When a new user registers to a service it is often essential to identify current registered users who are also contacts of the new user. This operation can be done by simply revealing the user's contact list to the service, but can also be done in a privacy preserving manner by running a PSI protocol between the user's contact list and the registered users of the service. This latter approach is used by the TextSecure and Secret applications, but for performance reasons they use the insecure naive hashing PSI protocol.[2] In these cases each user has a small number of records n_1, e.g., $n_1 = 256$, whereas the service has millions of registered users (in our experiments we use $n_0 = 16\ 777\ 216$). It therefore holds that $n_0 \gg n_1$. In our best PSI protocol, the client needs only $\mathcal{O}(n_1 \log n_0)$ memory, $\mathcal{O}(n_1)$ symmetric cryptographic operations and $\mathcal{O}(n_0)$ cheap hash table lookups, and the communication is $\mathcal{O}(n_0 \log n_0)$. (The communication overhead is indeed high as it depends on n_0, but this seems inevitable if brute force searches are to be prevented.)

[1] See, e.g., https://www.eff.org/deeplinks/2012/09/deep-dive-facebook-and-datalogix-whats-actually-getting-shared-and-how-you-can-opt.

[2] See https://whispersystems.org/blog/contact-discovery/ and https://medium.com/@davidbyttow/demystifying-secret-12ab82fda29f, respectively.

PSI has been a very active research field, and there have been many suggestions for PSI protocols. The large number of proposed protocols makes it non-trivial to perform comprehensive cross-evaluations. This is further complicated by the fact that many protocol designs have not been implemented and evaluated, were analyzed under different assumptions and observations, and were often optimized w.r.t. overall runtime while neglecting other relevant factors such as communication. Furthermore, even though the rich literature on secure PSI protocols, practical applications, including the ones described above, often compute the intersection of privacy-sensitive lists using insecure solutions. The reason for the poor acceptance of secure solutions is, among others, the poor efficiency of existing schemes, which have more than two orders of magnitude more overhead than the insecure naive hashing solution.

5.1.1 Our Contributions

We survey existing PSI protocols that are secure against semi-honest adversaries. We first categorize these protocols based on their underlying techniques into: *Semi-trusted third party*-based protocols, *public-key cryptography*-based protocols, *generic secure two-party computation techniques-based (circuit-based)* protocols, and *OT*-based protocols. We then perform a concrete parameter analysis for hashing to bins techniques (§5.2), improve the performance of existing protocols (§5.3 and §5.4.1) and introduce a new OT-based PSI protocol (§5.4.2). Finally, we perform an experimental comparison of the most promising PSI protocols (§5.5). We give our contributions in more detail next.

Concrete Parameter Estimation for Hashing (§5.2). In [FNP04] the use of hashing-to-bins was suggested in the context of PSI to reduce the overhead for pairwise comparisons. However, their analysis of the involved parameters was only asymptotic. We empirically analyze the hashing-to-bins techniques that were suggested in [FNP04] and identify concrete parameters (§5.2.1 and §5.2.2). We then utilize the permutation-based hashing techniques of [ANS10] to reduce the bit-length of the representations in the bins (§5.2.3). This improves the performance of PSI protocols that require an overhead linear in the bit-length of elements, e.g., the protocols in §5.3.2 and §5.4.2.

Optimizations of Existing Protocols (§5.3 and §5.4.1). We improve existing PSI protocols using our optimizations for OT extension from §3 and generic secure computation from §4. For generic secure computation protocols, we utilize our hashing analysis from §5.2 to show that a pairwise comparison (PWC) circuit achieves better performance than the previously best PSI circuit of [HEK12] (§5.3.2) and revisit and efficiently instantiate the oblivious pseudo random function (OPRF) protocol of [PSSW09] (§5.3.3). For OT-based protocols, we utilize the sender random OT functionality of §3.4.2 to improve the performance of the Bloom-filter-based protocol of [DCW13] (§5.4.1.3).

PSI Protocol	**Naive**	**Server-Aided** [KMRS14]	**Public Key** [Mea86]	**Circuit** PWC / OPRF	**OT+Hashing** (§5.2+§5.4.2)
Equal set sizes $n_0 = n_1 = 2^{20}$					
Run-time (s)	0.7	1.3	818.3	124.7	5.8
Comm. (MB)	10	20	74	14 014	111
Unequal set sizes $2^{24} = n_0 \gg n_1 = 2^{12}$					
Run-time (s)	6.1	7.6	12 712.3	7.3	42.6
Comm. (MB)	160	160	593	947	480

Table 5.1: Run-time and transferred data for PSI protocols on sets of equal size ($n_0 = n_1 = 2^{20}$) and unequal size ($n_0 = 2^{24}, n_1 = 2^{12}$) with elements of $\sigma = 32$-bit length and 128-bit security with a single thread over Gigabit LAN. Assuming that for circuit-based PSI the PWC circuit (cf. §5.3.2) is used for $n_0 = n_1$ and the OPRF circuit (cf. §5.3.3) is used when $n_0 \gg n_1$.

A Novel OT-Based PSI Protocol (§5.4.2). We present a new PSI protocol that is based on OT and directly benefits from our OT extension improvements in §3. Our PSI protocol is based on an efficient OPRF that is instantiated using our optimized $\binom{N}{1}$ OT extension protocol of [KK13] from §3.2.4 and uses the hashing techniques from §5.2 to reduce the communication overhead from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. The resulting protocol has very low computation complexity since it mostly requires symmetric key operations and has even less communication than some public-key-based PSI protocols, which had the lowest communication before.

A Detailed Comparison of PSI Protocols (§5.5). We conceptually discuss the most promising candidate PSI protocols (§5.5.1), implement them using state-of-the-art cryptographic techniques, and empirically compare their performance on one platform (§5.5.2). As far as we know, this is the first time that such a wide comparison has been made, since previous comparisons were either theoretical, compared implementations on different platforms or programming languages, or used implementations without state-of-the-art optimizations. We give a partial summary of our results in Table 5.1 and describe our most prominent findings next.

- The public-key-based protocol of [Mea86], which was the first PSI protocol, is actually the most efficient w.r.t. communication (when implemented using elliptic-curve cryptography). Therefore it is suitable for settings with distant parties which have strong computation capabilities but limited connectivity.

- Circuit-based protocols achieve an order of magnitude faster run-time than public-key-based PSI protocols and achieve a similar run-time as the special-purpose OT-based Bloom filter protocol of [DCW13] for larger security parameters and given sufficient bandwidth.

- Compared to the insecure naive hashing solution, previous two-party PSI protocols are at least two orders of magnitude less efficient in run-time or communication for equal set sizes. Our OT-based PSI protocol reduces this overhead to less than one order of magnitude in both run-time and communication.
- When intersecting sets with unequal sizes ($n_0 \gg n_1$), the run-times scale similar to the run-times for equal set sizes ($n_0 = n_1$). The only exception is the circuit-based OPRF protocol (cf. §5.3.3), which achieves similar performance as the insecure naive hashing and server-aided solutions.

5.1.2 Previous Works

Securely intersecting two sets without leaking any information but the result of the intersection is a prominent problem in secure computation. Several techniques have been proposed that realize the PSI functionality, such as the efficient but insecure *naive hashing* solution, protocols that require a *semi-trusted third party*, or *two-party PSI* protocols. The earliest proposed two-party PSI protocols were special-purpose solutions based on *public-key* cryptography. Later, solutions were proposed using *circuit*-based generic techniques for secure computation, that are mostly based on symmetric cryptography. The most recent development are PSI protocols that are based on *oblivious transfer (OT)* alone, and combine the efficiency of symmetric cryptographic primitives with special-purpose optimizations.

A Naive Solution. When confronted with the PSI problem, most novices come up with a solution where both parties apply a cryptographic hash function to their inputs and then compare the resulting hashes. Although this protocol is very efficient, it is insecure if the input domain is small or does not have high entropy, since one party could easily run a brute force attack that applies the hash function to all items that are likely to be in the input set and compare the results to the received hashes. (When inputs to PSI have high entropy, a protocol that compares hashes of the inputs can be used [NCD+13].)

Third Party-Based PSI. Several PSI protocols have been proposed that utilize additional parties, e.g., [BG85]. This approach was extended to multiple untrusted hardware tokens in [FPS+11]. Several efficient server-aided protocols for PSI were presented and benchmarked in [KMRS14].

Public-Key-Based PSI. A PSI protocol based on the Diffie-Hellmann (DH) key agreement scheme was presented in [Mea86] (related ideas were presented in [Sha80, HFH99]). Their protocol is based on the commutative properties of the DH function and was used for private preference matching, which allows two parties to verify if their preferences match to some degree.

Freedman et al. [FNP04] introduced PSI protocols secure against semi-honest and malicious adversaries in the standard model (rather than in the random oracle model

assumed in the DH-based protocol). This protocol was based on polynomial interpolation, and was extended in [FHNP16], which presents protocols with simulation-based security against malicious adversaries, and evaluates the practical efficiency of the proposed hashing schemes. We discuss the proposed hashing schemes in §5.2. A protocol that uses polynomial interpolation and differentiation for finding intersections between multi-sets was presented in [KS05].

Another PSI protocol that uses public-key cryptography (more specifically, blind-RSA operations) and scales linearly in the number of elements was presented in [CT10] and efficiently implemented and benchmarked in [CT12]. In [DD15], a family of Bloom filter-based PSI protocols was introduced that realize PSI, PSI cardinality and authenticated PSI functionalities. These protocols also use public-key operations, linear in the number of elements.

Circuit-Based PSI. Generic secure computation protocols have been subject to substantial efficiency improvements in the last decade (cf. §4). They allow the secure evaluation of arbitrary functions, expressed as arithmetic or Boolean circuits. Several Boolean circuits for PSI were proposed in [HEK12] and evaluated using the Yao's garbled circuits framework of [HEKM11]. The authors showed that their Java implementation scales very well with increasing security parameter and outperforms the blind-RSA protocol of [CT10] for larger security parameter.[3] We reflect on and present new optimizations for circuit-based PSI in §5.3.

OT-Based PSI. A recent PSI protocol of [DCW13] uses Bloom filters [Blo70] and the OT extension protocol of [IKNP03] to obtain very efficient PSI protocols with security against semi-honest and malicious adversaries. Recently in [Lam16], it was shown that the Bloom filter-based protocol is insecure with respect to malicious adversaries. We review and optimize the passive secure Bloom filter-based protocol of [DCW13] in §5.4.1 and propose a novel OT-based PSI protocol in §5.4.2.

PSI from an OPRF. An oblivious pseudo-random function (OPRF) [FIPR05] $F : (\{0,1\}^\kappa, \{0,1\}^\sigma) \mapsto (\perp, \{0,1\}^\ell)$ is a function which, given a κ bit key k from P_0 and a σ bit input element e from P_1, computes and outputs an ℓ bit string $F_k(e)$ to P_1. P_0 obtains no output and learns no information about e while P_1 learns no information about k. The size of the output ℓ must be chosen to ensure that no collision happens except with negligible probability (cf. §2.5). OPRFs can be used for PSI by first evaluating the OPRF protocol on the set of P_1 and then having P_0, who knows the secret key k, evaluate the OPRF locally on its own set, and send the OPRF output to P_1, who computes a plaintext intersection. There exist several instantiations for OPRFs, described in [FIPR05]: Based on generic secure computation techniques (using the AES circuit [PSSW09]), based on the Diffie-Hellman assumption, or based

[3] Subsequent work of [CT12] claimed that the blind-RSA protocol of [CT10] runs faster than the circuit-based protocol of [HEK12] even for larger security parameter. Their implementation is in C++ instead of Java.

on OT. Furthermore, a trusted-third party-based OPRF protocol was given in [HL08], where a trusted hardware token is used to evaluate an OPRF. In §5.3.3 we analyze the efficiency of generic secure computation-based OPRF instantiations and in §5.4.2 we give a more efficient OT-based instantiation.

5.1.3 Follow-Up Works

In [RR16], our OT-based PSI protocol of [PSZ14] was extended to security against weakly malicious adversaries and used as a building block in a batch dual-execution Yao's garbled circuits protocol. In [Lam16], it was shown how to achieve security against a malicious P_1 and a semi-honest P_0 for our OT-based PSI protocol in [PSSZ15].

An improved version of our OT-based PSI protocol in [PSSZ15] is given in [KKRT16], which presents an efficient construction of an OPRF using the OT extension protocol of [KK13]. The main observation of the authors is that the [KK13] OT does not require an error correcting code but can instead use a pseudo-random code, which can be generated from a pseudo-random generator. The authors then apply their efficient OPRF construction to our [PSSZ15] protocol and thereby achieve performance independent of the bit-length of elements σ. The OPRF construction of [KKRT16] is similar to our new OPRF construction described in §5.4.2. The idea of both works is to instantiate the OPRF that is implicitly used in the [PSSZ15] OT-based PSI protocol using larger codes. However, while [KKRT16] replace the error correcting code with a pseudo-random code, we keep the error correcting code. Thereby, our OPRF construction achieves better communication for smaller values of $\sigma - \log_2 n_1$ when using custom-tailored error correcting codes (e.g., 271 bits for $13 = \sigma - \log_2 n_1$ using the codes of [SS06] instead of $\geq$ 424 bits in [KKRT16]) but does not achieve performance independent of σ.

The works of [OOS17, PSS17] describe how to more efficiently instantiate the one-sided malicious secure version of our [PSSZ15] PSI protocol, outlined in [Lam16]. Both works showed how to achieve active security for the $\binom{2}{1}$ OT extension protocol of [KK13] and applied their protocols to the PSI protocol of [Lam16]. Furthermore, independently and concurrently to our work, [OOS17] also suggested that using a linear BCH code instead of a Walsh-Hadamard code in the $\binom{N}{1}$ OT extension protocol of [KK13] can greatly improve its efficiency for longer input elements (cf. §3.2.4.1 and §5.4.2).

5.2 Hashing Schemes and PSI

Computing the plaintext intersection between two sets is often done using *hashing techniques*. The parties agree on a publicly known random hash function to map elements to a *hash table*, which consists of multiple *bins*. If an input element is in the intersection, both parties map it to the same bin. Hence, the parties only need

to compare the elements that are in the same bin to identify intersecting elements. Thereby, the average number of comparisons between elements can be reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$ for pairwise comparisons.

In a similar fashion, PSI protocols that privately compute the equality between values can use hashing techniques in order to reduce the number of comparisons [FNP04, FHNP16]. Examples for such private equality test protocols are [FNP04, HEK12, CADT14], the circuit-based protocol in §5.3.2 or our OT-based protocol in §5.4.2. When naively using hashing techniques, if n items are mapped to n bins then the average number of items in a bin is $\mathcal{O}(1)$, checking for an intersection in a bin takes $\mathcal{O}(1)$ work, and hence the total number of comparisons is $\mathcal{O}(n)$. However, privacy requires that the parties hide from each other how many of their inputs were mapped to each bin.[4] As a result, we must calculate in advance the number of items that will be mapped to the *most populated* bin (w.h.p.), and then set all bins to be of that size. (This can be done by storing dummy items in bins which are not fully occupied.) This change hides the bin sizes but also increases the overhead of the protocol, since the number of comparisons per bin now depends on the size of the most populated bin (worst case) rather than on the actual number of items in the bin (average case).

In fact, this worst case analysis is key to balancing security and efficiency. On the one hand, if the estimation is too optimistic, the probability of a party failing to perform the mapping becomes intolerable. As a result, the output might be inaccurate (since not all items can be mapped to bins), or one party needs to request a new hash function (a request that leaks information about the input set of that party). On the other hand, the number of comparisons and hence the protocol overhead can become prohibitive if the analysis is too pessimistic. The work of [FNP04, FHNP16] gave asymptotic values for this analysis and of the resulting overhead. They left the task of setting appropriate parameters for the hashing schemes to future work.

In this section, we revisit the simple hashing (§5.2.1) and Cuckoo hashing (§5.2.2) schemes, used in [FNP04, FHNP16]. We describe how to use both hashing schemes in the context of PSI and give a concrete parameter analysis that balances security and efficiency. Finally, we show how the bit-length of the representations that are stored in the bins can be reduced using permutation-based mapping, which improves the performance of some PSI protocols (§5.2.3).

Note that, for our hashing failure analysis, we use a dedicated hashing failure parameter ϕ, which is different from the statistical security parameter λ. We use a dedicated parameter since our analysis requires running empirical experiments for determining concrete numbers. Running the analysis on 2^{40} iterations in the Amazon EC2 cloud would have cost several hundred thousand USD. Hence, we perform the experiments and give concrete numbers for $\phi = 30$ and interpolate from these results to $\phi = 40$.

[4] Otherwise, and since the hash function is public, some information is leaked about the input. For example, if no items of P_0 were mapped to the first bin by the hash function h, then P_1 learns that P_0 has no inputs in the set $h^{-1}(1)$, which covers about $1/n$ of the input range.

5.2.1 Simple Hashing

In the simplest hashing scheme, the hash table consists of b bins $B_1 ... B_b$. Hashing is done by mapping each input element e to a bin $B_{h(e)}$ using a hash function $h : \{0,1\}^\sigma \mapsto [1, b]$ that was chosen uniformly at random and independently of the input elements. An element is always added to the bin to which it is mapped, regardless of whether other elements are already stored in that bin.

5.2.1.1 Simple Hashing for PSI

To apply simple hashing in the context of PSI, both parties map their elements to b bins. The intersection is then computed by having both parties separately compare the items mapped to bin $i \in [1, ..., b]$. In order to hide the number of elements that were mapped to a bin, the parties need to pad their bins using dummy elements to contain max_b elements. This maximum bin size must ensure that except with probability $< 2^{-\phi}$, no bin will contain more than max_b real elements.

5.2.1.2 Simple Hashing Parameter Analysis

Estimating max_b has been subject to extensive research [Gon81, RS98, Mit01]. When hashing n elements to $b = n$ bins, [Gon81] showed that $max_b = \frac{\ln n}{\ln \ln n}(1 + o(1))$ w.h.p. In this case, there is a difference between the expected and the maximum number of elements mapped to a bin, which are 1 and $\mathcal{O}(\frac{\ln n}{\ln \ln n})$, respectively. Let us revisit the analysis of [MR95] that examines the probability of the following event, "n balls are mapped at random to b bins, and there exists a bin with at least max_b balls":

$$\begin{aligned} P(\text{"}\exists \text{ bin with } \geq max_b \text{ elements"}) &\leq \sum_{i=1}^{b} P(\text{"bin } i \text{ has } \geq max_b \text{ elements"}) \\ &= b \cdot [P(\text{"bin 1 has } \geq max_b \text{ elements"})] \\ &= b \cdot \left[\left(\sum_{i=max_b}^{n} \binom{n}{i} \left(\frac{1}{b}\right)^i \left(1 - \frac{1}{b}\right)^{n-i} \right] . \end{aligned} \quad (5.1)$$

Case $n = b$. We calculate max_b when mapping $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}\}$ elements to $b = n$ bins using Equation 5.1, choose the minimal value of k that reduces the failure probability to below 2^{-30} and 2^{-40} and depict the results in Table 5.2.

Hash Failure Parameter ϕ	**30**					**40**				
Set Size n	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$
max_b (Equation 5.1)	13	15	16	17	18	16	17	18	19	20

Table 5.2: Bin sizes max_b required to ensure that an overflow occurs except with probability $\leq 2^{-\phi}$ when mapping n items to $b = n$ bins, according to Equation 5.1.

Case $n \gg b$. In certain settings, the server P_0 has a much larger set than the client P_1. For simple hashing, this translates to the number of elements n being much larger than the number of bins b. Later in the thesis, we perform experiments for this setting (cf. §5.5.2.3), where P_1 has a set of size $n_1 \in \{2^8, 2^{12}\}$, while P_0 has a set of size $n_0 \in \{2^{16}, 2^{20}, 2^{24}\}$ and both map $n = 2n_0$ elements into $b = 2.4n_1$ bins. To determine max_b in this setting, we evaluate Equation 5.1 with these set sizes and depict max_b for hashing failure probabilities 2^{-30} and 2^{-40} in Table 5.3. From the results, we can observe that as the fraction n_0/n_1 grows, the maximum number of bins grows closer to the expected number of bins.

Set Size n_1	2^8			2^{12}		
Set Size n_0	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
Hash Failure Parameter $\phi = 30$						
max_b (Equation 5.1)	323	3 825	56 196	48	329	3 852
Hash Failure Parameter $\phi = 40$						
max_b (Equation 5.1)	338	3 881	56 412	53	344	3 905

Table 5.3: Bin sizes max_b required to ensure that an overflow occurs except with probability $\leq 2^{-\phi}$ when mapping $n = 2n_0$ items to $b = 2.4n_1$ bins for $n_0 \gg n_1$, according to Equation 5.1.

5.2.2 Cuckoo Hashing

Cuckoo hashing [PR01] uses k hash functions $h_1, ..., h_k : \{0,1\}^\sigma \mapsto [1, b]$ to map m elements to $b = \epsilon n$ bins. The scheme avoids collisions by relocating elements when a collision is found using the following procedure: An element e is inserted into a bin $B_{h_1(e)}$. Any prior contents o of $B_{h_1(e)}$ are evicted to a new bin $B_{h_i(o)}$, using h_i to determine the new bin location, where $h_i(o) \neq h_1(e)$ for $i \in [1...k]$. The procedure is repeated until no more evictions are necessary, or until a threshold of 1 000 relocations has been performed. In the latter case, the last element is put in a special stash. A lookup in this scheme is very efficient as it only compares e to the k items in $B_{h_1(e)}, ..., B_{h_k(e)}$ and to the s items in the stash. The size of the hash table depends on the number of hash functions k as well as on the stash size s. The higher k is chosen, the more likely it is that the insertion process succeeds and hence the smaller the number of bins b becomes. On the other hand, the higher s is chosen, the more insertion failures can be tolerated.

5.2.2.1 Cuckoo Hashing for PSI

A major problem occurs when using Cuckoo hashing for PSI: Every item can be mapped to one of k bins, and therefore it is unclear with which of P_0's bins should P_1 compare its own input elements. Furthermore, the protocol must hide from each party

the choice of bins made by the other party to store an item, since that choice depends on other input elements and might reveal information about them. The solution to this is that P_1 uses Cuckoo hashing whereas P_0 maps each of its elements using simple hashing with each of the k hash functions. In addition, for Cuckoo hashing, we must ensure that the hashing succeeds except with probability $< 2^{-\phi}$, since a hashing error on the side of P_1 reveals information about its set or results in an incorrect result. As in PSI with simple hashing (cf. §5.2.1), P_0 will need to pad its bins to size max_b using dummy elements.

5.2.2.2 Cuckoo Hashing Parameter Analysis

Cuckoo hashing has three parameters that affect the hashing failure probability: The stash size s, the number of hash functions k, and the number of bins $b = \epsilon n$ [KMW09]. It was shown in [KMW09] that Cuckoo hashing of n elements into $(1+\zeta)n$ bins with $\zeta \in (0,1)$ for any $k \geq 2(1+\zeta)\ln(\frac{1}{\zeta})$ and $s \geq 0$ fails with probability $O(n^{1-c(s+1)})$, for a constant $c > 0$ and $n \mapsto \infty$. The constants in the big "$\mathcal{O}$" notation are unclear, which makes it hard to compute a concrete failure probability given a set of parameters.

In the following, we empirically determine the failure probability given the stash size s, the number of hash functions k, and the number of bins b. We analyze the effect of all three parameters separately. We first fix the number of bins $b = 2.4n$ and hash functions $k = 2$ (as was done in [KMW09]) and determine the necessary stash sizes s. In order to improve performance, we increase the number of hash functions k and determine the number of bins b for which no stash is required (i.e., $s = 0$). While both approaches achieve good overhead when $n_0 = n_1$, they perform poorly when the parties have unequal set sizes $n_0 \gg n_1$. Hence, in the last step, we show how to obtain low values for the stash size s and a low number of hash functions k by increasing the number of bins $b = \epsilon n$, which results in a collection of trade-offs for unequal set size applications.

Adjusting the Stash Size s. In the following, we identify the exact stash size s that ensures that the hashing failure probability is smaller than a given $2^{-\phi}$. To obtain concrete numbers, we ran 2^{30} repetitions of Cuckoo hashing, where we mapped n items to $b = \epsilon n = 2.4n$ bins, for $n \in \{2^{11}, 2^{12}, 2^{13}, 2^{14}\}$, using $k = 2$ hash functions and recorded the stash size s that was needed for Cuckoo hashing to be successful. We fix $\epsilon = 2.4$ as was done in the original Cuckoo hashing with a stash paper [KMW09]. The solid lines in Figure 5.1 depict the probability that a stash of size s prevented a hashing failure.

From the results we can observe that, to achieve 2^{-30} failure probability of Cuckoo hashing, we require a stash of size $s = 6$ for $n = 2^{11}$, $s = 5$ for $n = 2^{12}$, and $s = 4$ for both $n = 2^{13}$ and $n = 2^{14}$ elements. However, in our experiments we need the stash sizes for smaller as well as larger values of n to achieve a Cuckoo hashing failure probability of 2^{-30}. To obtain the failure probabilities for these values of n, we extrapolate the results using linear regression and illustrate the results as dotted

lines in Figure 5.1. We give the extrapolated stash sizes for achieving a hashing failure probability of 2^{-30} and 2^{-40} for $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}\}$ in Table 5.4. We observe that the stash size for achieving a failure probability of 2^{-30} is drastically reduced for higher values of n: For $n = 2^{16}$ we need a stash of size $s = 4$, for $n = 2^{20}$ we need $s = 3$, and for $n = 2^{24}$ we need $s = 2$. This observation is in line with the asymptotic failure probability of $\mathcal{O}(n^{-s})$.

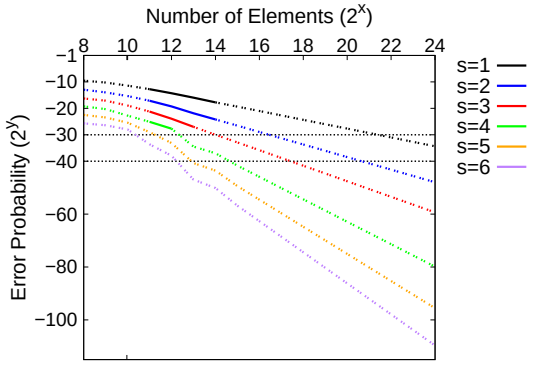


Figure 5.1: Error probability when mapping n elements to $2.4n$ bins using Cuckoo hashing with $k = 2$ hash functions for stash sizes $1 \leq s \leq 6$. The solid lines correspond to actual measurements, the dashed lines were extrapolated using linear regression. Both axes are in logarithmic scale.

# Elements n	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$
Stash s ($\phi = 30$)	8	5	3	2	1
Stash s ($\phi = 40$)	12	6	4	3	2

Table 5.4: Required stash sizes s to achieve $2^{-\phi}$ failure probability when mapping n elements into $2.4n$ bins.

Adjusting the Number of Hash Functions k. The original Cuckoo hashing procedure [PR04] fixed the number of hash functions to $k = 2$. It was later shown in [DGM+10] that increasing the number of hash functions to $k > 2$ achieves much better utilization of bins in the hash table. I.e., while the average utilization for $k = 2$ hash functions is around 50%, the utilization increases to 91.8% for $k = 3$, 97.7% for $k = 4$, and 99.2% for $k = 5$. Hence, higher values of k allow us to drastically decrease the number of bins. However, similar to the previous stash allocation, the analysis in [DGM+10] was only asymptotic and does not allow to compute the concrete hashing failure probability.

In order to determine the concrete failure probability, we again perform 2^{30} iterations of Cuckoo hashing on $n = 1\,024$ elements using $k \in \{3, 4, 5\}$ hash functions. Our

goal in this analysis is to determine the minimum number of bins $b_{min} = \epsilon_{min} n$ for which the hashing procedure succeeds without a stash except with probability 2^{-30}. In order to determine the value of b_{min}, we run Cuckoo hashing on an initialization value $\epsilon_{min} = 1.0$ and increase ϵ_{min} by 0.01 each time more than one hashing failure has occurred. An interesting observation that we made during the experiments with multiple hash functions was that after a certain threshold value, the hashing failure probability decreased drastically. E.g., only increasing ϵ by as little as 0.1 when using $k = 5$ hash functions could reduce the required stash size from $s = 2$ to $s = 0$. Overall, we determined the following bin sizes that resulted in a hashing failure probability of $< 2^{-30}$: $\epsilon_{min} = 1.20$ for $k = 3$, $\epsilon_{min} = 1.07$ for $k = 4$, and $\epsilon_{min} = 1.04$ for $k = 5$. We extrapolate the values of ϵ_{min} for $\phi = 2^{-40}$ by adding an additional security margin of factor 4/3 to the ϵ_{min} values for $\phi = 2^{-30}$ which results in: $\epsilon_{min} = 1.27$ for $k = 3$, $\epsilon_{min} = 1.09$ for $k = 4$, and $\epsilon_{min} = 1.05$ for $k = 5$.

A consequence of increasing the number of hash functions is that the party P_0, who uses simple hashing, needs to increase the maximum bin size max_b. This is due to two factors: On the one hand, the hash table of P_0 contains more elements since P_0 needs to map each element k times to its hash table. On the other hand, the parties decrease the number of bins due to the reduced ϵ. We re-compute the maximum bin size of P_0 for the set sizes used in our PSI experiments in §5.5 given the increased number of hash functions using Equation 5.1 and give the results in Table 5.5. Given these results, we can compute the total number of comparisons by multiplying the number of bins b with max_b. From these results, we observe that $k = 3$ achieves the best performance. In fact, using $k = 3$ hash functions performs better than using $k = 2$ hash functions, even without considering the stash size for $k = 2$.

Hash Failure Parameter ϕ	**30**					**40**				
Set Sizes $n_0 = n_1$	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$
$\mathbf{max_b}$ **for** $\boldsymbol{k=2}$ ($n = 2n_0, b = \{2.4/2.4\}n_1$)	13	14	15	16	17	15	16	17	18	19
$\mathbf{max_b}$ **for** $\boldsymbol{k=3}$ ($n = 3n_0, b = \{1.20/1.27\}n_1$)	19	21	22	23	25	22	23	25	26	27
$\mathbf{max_b}$ **for** $\boldsymbol{k=4}$ ($n = 4n_0, b = \{1.07/1.09\}n_1$)	23	25	26	28	29	27	28	29	31	32
$\mathbf{max_b}$ **for** $\boldsymbol{k=5}$ ($n = 5n_0, b = \{1.04/1.05\}n_1$)	26	28	29	31	32	30	31	33	34	36

Table 5.5: Bin sizes max_b required to ensure that an overflow occurs except with probability $\leq 2^{\phi}$ when mapping n items to b bins using k hash functions, according to Equation 5.1.

Adjusting the Number of Bins b. The required stash sizes for $b = 2.4n$ bins and $k = 2$ hash functions are relatively large for small set sizes (e.g., $s = 8$ for $n = 256$). In case of equal set sizes $n_0 = n_1$, this does not impact the performance of the protocols much. In the case of unequal set sizes $n_0 \gg n_1$, however, large stash sizes will greatly decrease the performance, since each element in the stash needs to be compared with each item in the large set with possibly millions of elements. Furthermore, even when increasing the number of hash functions $k > 2$ to remove the stash, P_0 would need to

map each of its million elements k times into its hash table, which increases max_b and hence incurs a great overhead.

To improve the performance for unequal set sizes, we fix $s \in \{0, 1, 2, 3, 4\}$ and $k = 2$ and try to identify $b = \epsilon n$ such that the hashing failure probability is less than $2^{-\phi}$. Similarly to the previous experiments, we ran 2^{30} repetitions of Cuckoo hashing, mapping n items to $b = \epsilon n$ bins, for $n = 256$ and $\epsilon = \{2.4, 3, 4, 5, 6, 7, 8, 9, 10, 20, 100, 200\}$, and recorded the stash size s that was needed for Cuckoo hashing to be successful. We chose $n = 256$ since it is a good approximation of the number of contacts in a user's address book and it is used in our experiments in §5.5.2.3.

The results of our experiments are depicted as solid lines in Figure 5.2. From the results, we can observe that the probability of requiring a stash size of s decreases logarithmically with growing ϵ: While for small ϵ the probabilities decrease quickly, they decrease slower for large ϵ. E.g., when increasing ϵ from 2.4 to 4, the hashing failure probability for a stash of size $s = 0$ decreases from 2^{-6} to 2^{-12}. If, on the other hand, ϵ is increased from 20 to 100, the hashing failure probability for $s = 0$ only decreases from 2^{-21} to 2^{-28}. Since we are interested in identifying ϵ such that the probability of requiring a stash of size s decreases below $2^{-\phi}$, we use regression via a logarithmic function to extrapolate the probabilities. These estimated probabilities are depicted as dotted lines in Figure 5.2 and the smallest ϵ for which the hashing failure probability decreases below 2^{-30} and 2^{-40} is given in Table 5.6.

The estimations indicate that, in order to reduce the stash size to $s = 0$, we need to set $\epsilon = 166$ to guarantee 2^{-30} hashing failure probability and to $\epsilon = 2\,500$ to guarantee 2^{-40} hashing failure probability. When allowing a bigger stash size $s = 1$, ϵ decreases drastically, allowing us to set $\epsilon = 7.8$ for 2^{-30} hashing failure probability and $\epsilon = 16$ for 2^{-40} hashing failure probability. In our experiments, the exact choice of ϵ and s depends on the difference between the set sizes n_0 and n_1 as well as the protocol that is used (cf. §5.5.2.3). I.e., if n_1 is only a few hundred while n_0 is several million, it can be more efficient to choose $\epsilon = 166$ to achieve stash size $s = 0$.

5.2.3 Permutation-Based Hashing

The overhead of our circuit-based PSI protocols in §5.3.2 and of the OT-based PSI protocol in §5.4.2 depends on the bit-length σ of the items that the parties map to bins. The bit-length of the stored items can be reduced using a permutation-based hashing technique that was suggested in [ANS10] for reducing the memory usage of Cuckoo hashing. That construction was presented in an algorithmic setting to improve memory usage. As far as we know this is the first time that it is used in secure computation or in a cryptographic context.

The construction uses a Feistel-like structure. Let $x = x_L|x_R$ be the bit representation of an input item, where $|x_L| = \log b$, i.e. is equal to the bit-length of an index of an entry in the hash table. (We assume here that the number of bins b in the hash table is a power of 2. It was shown in [ANS10] how to handle the general case.) Let $f()$ be

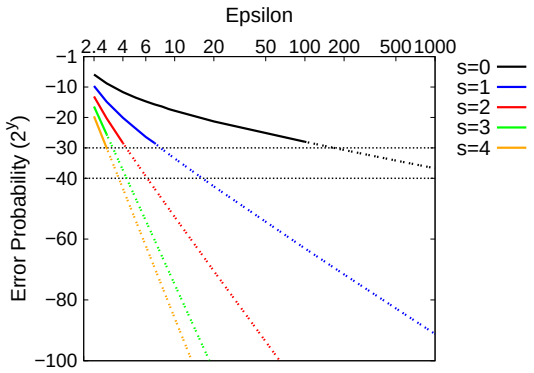


Figure 5.2: Error probability when mapping 256 elements to $b = 256\epsilon$ bins using Cuckoo hashing with $k = 2$ hash functions for stash sizes $0 \leq s \leq 4$. The solid lines correspond to actual measurements, the dashed lines were extrapolated using logarithmic regression. Both axes are in logarithmic scale.

Stash Size s	**0**	**1**	**2**	**3**	**4**
ϵ ($\phi = 30$)	166	7.8	4.2	3.4	3
ϵ ($\phi = 40$)	2 500	16	6.2	4.4	3.8

Table 5.6: Required number of bins $b = 256\epsilon$ to achieve $< 2^{-\phi}$ hashing failure probability given a fixed stash size s.

a random function whose range is $[0, b-1]$. Then item x is mapped to bin $x_L \oplus f(x_R)$. The value that is stored in the bin is x_R, which has a length that is shorter by $\log b$ bits than the length of the original item. This is a great improvement, since the length of the stored data is significantly reduced, especially if $|x|$ is not much greater than $\log b$. As for the security, it can be shown based on the results in [ANS10] that if the function f is k-wise independent, where $k = \text{polylog}\, n$, then the maximum load of a bin is $\log n$ with high probability.

The structure of the mapping function ensures that if two items x, x' store the same value in the same bin then it must hold that $x = x'$: if the two items are mapped to the same bin, then $x_L \oplus f(x_R) = x'_L \oplus f(x'_R)$. Since the stored values satisfy $x_R = x'_R$ it must also hold that $x_L = x'_L$, and therefore $x = x'$.

As a concrete example, assume that $|x| = 32$ and that the table has $b = 2^{20}$ bins. Then the values that are stored in each bin are only 12 bits long, instead of 32 bits in the original scheme. Note also that the computation of the bin location requires a single instantiation of f, which can be implemented with a medium-size lookup table. Note that, when mapping an element into a bin using multiple hash functions, e.g., when using Cuckoo hashing, the index of the hash function needs to be added to the

representation in the bin to preserve uniqueness. This observation was pointed out in [Lam16].

5.3 Circuit-Based PSI

Unlike special-purpose PSI protocols, the protocols that we describe in this section are based on generic secure computation techniques that can be used for computing arbitrary functionalities. We outline the sort-compare-shuffle (SCS) circuit of [HEK12], which is a Boolean circuit of size $\mathcal{O}(n \log n)$ for computing the PSI functionality (§5.3.1). We then show how to use the hashing methods described in §5.2 to achieve better complexity than the SCS circuit using a pairwise comparison (PWC) circuit (§5.3.2). Finally, we revisit the method of [PSSW09] where generic secure computation techniques are used to instantiate an OPRF (cf. §5.1.2), which is used to process the input elements of one party (§5.3.3).

The usage of generic protocols has the advantage that the functionality of the protocol can easily be extended, without having to change the protocol or the security of the resulting protocol. For example, it is straightforward to change the SCS or PWC circuits to compute the size of the intersection, or a function that outputs if the intersection is greater than some threshold, or compute a summation of values (e.g., revenues) associated with the items that are in the intersection. Computing these variants using other PSI protocols is non-trivial.

5.3.1 Sort-Compare-Shuffle Circuit for PSI

A Boolean circuit for PSI that has $\mathcal{O}(n \log n)$ size is the *sort-compare-shuffle (SCS)* circuit described in [HEK12]. (We refer here to the SCS circuit that uses the Waksman permutation for shuffling). The SCS circuit computes the intersection between two sets by first *sorting* both sets into a single sorted list, then *comparing* all neighboring elements for equality, and finally *shuffling* the intersecting elements to hide any information that could be obtained from the resulting order.

Sort. To sort their sets with $n = n_0 = n_1$ elements into a single sorted list, both parties locally pre-sort their sets and merge them using a bitonic merging circuit [Bat68]. In contrast to a sorting network, a bitonic merging circuit takes advantage of the fact that the inputs are already sorted and allows the parties to obtain a globally sorted list of $2n$ input elements using $n \log_2(2n)$ sorter circuits. A sorter circuit takes as input two elements x and y, swapping them if $x > y$ and preserving the order if $x \leq y$. Each sort gate consists of a comparison and a conditional swap sub-circuit.

Compare. All elements in the sorted list are then compared to their neighbors to determine if a duplicate exists. Since each party's input consists of different values, duplicates only occur for items in the intersection of the two inputs. A duplicate item

is passed on, whereas if no duplicate is found then the item is replaced by a special dummy element.

Shuffle. Finally, all elements are randomly shuffled using a Waksman permutation network [Wak68]. An n input Waksman circuit consists of $n\log_2(n) - n + 1$ conditional swap gates, which either forward their two input elements or swap their order depending on the required randomly chosen output permutation. A special-purpose optimization which allows the workload for the gates to be reduced by a factor $2\times$ was outlined in [HEK12] but is omitted in this thesis.

Efficiency. The overall size of the SCS circuit for inputs of length σ bit is $\sigma(3n\log_2 n + 4n) - n$ AND gates, which is the sum of $2\sigma n\log_2(2n)$ AND gates for the sort circuit, $\sigma(3n-1) - n$ AND gates for the compare circuit, and $\sigma(n\log_2 n - n + 1)$ for the shuffle circuit. It is important to note that approximately 2/3 of the AND gates in the circuit are due to multiplexers. These multiplexer gates can be efficiently evaluated in GMW using vector MTs (cf. §4.4.1), reducing the pre-computation cost in GMW from σ AND gates to the equivalent of 1 AND gate for a σ-bit multiplexer.

Instantiation. For our experiments in §5.5, we use GMW to evaluate a depth-optimized variant of the SCS circuit, where the comparison gates have $3\sigma - \log_2(\sigma) - 2$ AND gates instead of σ but have a depth of $\log_2(\sigma)$ instead of σ for σ-bit values (cf. §4.2.1.7). Consequently, the size of the SCS circuit is increased from approximately $3n\sigma\log_2 n$ to $5n\sigma\log_2 n$, but its depth is decreased from $\sigma\log_2(n)$ to $\log_2(\sigma)\log_2(n)$. Using the vector MTs optimization from §4.4.1, the size of the depth-optimized SCS circuit is again decreased back to approximately $3\sigma n\log_2(n)$.

5.3.2 Pairwise Comparison (PWC) and Hashing

A simpler circuit for performing the PSI functionality is a pairwise comparison (PWC) circuit, where each element in the set of P_0 is compared to each element in the set of P_1. However, this circuit would scale with $\mathcal{O}(n_0 n_1)$, making it impractical for larger sets. Using the hashing methods from §5.2, we can drastically reduce the number of comparisons. In particular, we let P_0 use simple hashing and P_1 use Cuckoo hashing and then evaluate an equality circuit that compares the elements of both parties in the same bin. We outline the PWC protocol in more detail Protocol 15.

Efficiency. Let $b \cdot max_b$ be the number of element comparisons that are performed for the hash tables, i.e., for each of the b bins, the parties perform max_b comparisons per bin. Each element is of length μ bits, which is the reduced length of the elements after being mapped to bins using permutation-based hashing, i.e. $\mu = \sigma - \log_2 b + \log_2 k$ (cf. §5.2.3). A comparison of two μ-bit elements is done by computing the bitwise XOR of the elements and then a tree of μ OR gates, with depth $\lceil\log_2 \mu\rceil$. The topmost gate of this tree is a NOR gate. Afterwards, the circuit computes the XOR of the results of all comparisons involving each item of P_1. (Note that at most one of

the comparisons results in a match, therefore the circuit can compute the XOR, rather than the OR, of the results of the comparisons.) In addition, the parties evaluate sn_0 comparison circuits for the σ-bit elements on the stash, which have depth $\lceil \log_2 \sigma \rceil$. Overall, the circuit consists of about $b \cdot max_b \cdot (\mu - 1) + s \cdot n_0 \cdot (\sigma - 1)$ non-linear gates and has an AND depth of $\lceil \log_2 \sigma \rceil$.

PROTOCOL 15 (Our PWC Protocol)**.**

- **Input of P_0:** $X = \{x_1, ..., x_{n_0}\}$.
- **Input of P_1:** $Y = \{y_1, ..., y_{n_1}\}$.
- **Common Input:** Bit-length of elements σ; Number of bins $b = \epsilon n_1$ (cf. §5.2.2.2); k random hash functions $\{h_1, ..., h_k\} : \{0,1\}^\sigma \mapsto [1...b]$; Reduced bit-length of items in the hash table $\mu = \sigma - \log_2 b + \log_2 k$; Symmetric security parameter κ; Dummy elements d_0 and d_1 with $d_0 \neq d_1$; Stash size s; Maximum number of elements in a bin max_b (cf. §5.2.1.2).

1. **Hashing:**
   a) P_0 maps the elements in its set X into a two-dimensional hash table $T_0[\,][\,]$ using simple hashing and k hash functions $\{h_1, ..., h_k\}$. The first dimension has size b and addresses the bin in the table while the second dimension addresses the elements in the bins. P_0 then pads all bins $T_0[i]$ with less than max_b elements to max_b using d_0, for $1 \leq i \leq b$.
   b) P_1 maps the elements in its set Y into a one-dimensional hash table $T_1[\,]$ and stash $S[\,]$ using Cuckoo hashing and k hash functions $\{h_1, ..., h_k\}$. The hash table has size b and the stash has size s. P_1 then fills all empty entries in T_1 and S with d_1.

   Let $|T_0[i]|$ be the number of elements that are stored in the i-th bin of the hash table T_0 and μ be the bit-length of these elements for $1 \leq i \leq b$.

2. **Private Equality Testing (via generic secure computation protocol):** For each bin $1 \leq i \leq b$, the parties perform the following steps:
   a) Let $v_j = T_0[i][j]$ and $w = T_1[i]$ for $1 \leq j \leq max_b$.
   b) The parties evaluate a Boolean circuit using a generic secure two-party computation protocol that computes and outputs $\bigoplus_{j=1}^{max_b} \text{EQ}^\mu(v_j, w)$ to P_1 (cf. §4.2.1.6). P_1 will add the element in bin i to the intersection Z, if the circuit evaluates to 1.

   Stash: For each element in the stash S, the parties repeat the same steps where, for the i-th stash position P_0 and P_1 compute the private equality test between $S[i]$ and all elements in X, which have bit-length σ.

- **Output:** P_0 has no output; P_1 outputs $Z = X \cap Y$.

Advantages. In the following we describe the advantages of the PWC circuit over the SCS circuit from §5.3.1:

- Compared to the number of AND gates in the SCS circuit, which is $3n\sigma \log n$, and recalling that $\sigma' < \sigma$, and that max_b was shown in our experiments to be no greater than $2 \log n$ (and not greater than $\log n$ asymptotically), the number of non-linear gates in the PWC circuit is smaller by more than a factor 1.5× compared to the number of non-linear gates in the SCS circuit (even though both circuits asymptotically scale the same).

- The main advantage of the PWC circuit is the low AND depth of $\log_2 \sigma$, which is also independent of the number of elements n. This affects the overhead of the GMW protocol that requires a round of interaction for every level in the circuit.

- Another advantage of the PWC circuit is its simple structure: The same small comparison circuit is evaluated for each bin. This property allows for a SIMD (Single Instruction Multiple Data) evaluation with a very low memory footprint and easy parallelization (cf. §4.2.2).

- Finally, the efficiency of the SCS circuit is tailored to equal set sizes. For unequal set sizes, the circuit size does not scale well. The PWC circuit, on the other hand, scales much better for unequal set sizes.

5.3.3 Secure Evaluation of an OPRF

Another method for circuit-based PSI was outlined in [FIPR05, PSSW09] and uses an OPRF (cf. §5.1.2). In this protocol, the parties use secure computation to evaluate a pseudo-random function $F_k(y) = z$, which takes as input a random key k from P_0 and an element y from P_1 and returns the output z to P_1. The use of secure computation guarantees the obliviousness, i.e., that P_0 learns no information about y or z while P_1 learns no information about k. The PSI functionality can then be achieved by evaluating the OPRF on each element in the set of P_1 and having P_0 locally evaluate and send $F_k(x_i)$ for all elements $x_i \in X$. P_1 can then identify the intersection by computing the plaintext intersection between its output of the OPRF with the elements sent by P_0.

Efficiency. The efficiency of the circuit-based OPRF construction depends mainly on the instantiation of the pseudo-random function F. While it is possible to instantiate F with a cipher that is optimized for use in secure computation such as [ARS+15], we consider an AES-based instantiation in our efficiency analysis, since the security of AES is better established. The number of AND gates in the AES circuit is 5 120 and its multiplicative depth is 60 [BP12] (cf. §4.2.1.12, we use the size-optimized S-Box circuit, since the circuit needs to be evaluated n_1 times in parallel). In total, we have to perform n_1 parallel oblivious AES evaluations, resulting in a total of 5 $120n_1$ AND gates and a depth of 60. P_0, on the other hand, can perform a plaintext AES evaluation on its elements and only needs to send n_0 collision-resistant strings of length

Optimization	Party	# Bits Sent	# Calls to CRF
Original GBF PSI [DCW13]	P_0	$2m\lambda$	$2m$
	P_1	$m\kappa$	m
Original GBF PSI with OT of §3.2	P_0	$m\lambda$	m
	P_1	$m\kappa$	$m/2$
Optimized Random GBF PSI (§5.4.1.3)	P_0	$n_0\ell$	$1.44n_1\kappa/2$
	P_1	$1.44n_1\kappa^2$	$1.44n_1\kappa/2$

Table 5.7: Communication and computation complexities for Garbled Bloom filter-based PSI without and with correlated OT extension, and our optimized random Garbled Bloom filter protocol. (λ: statistical security parameter, κ: symmetric security parameter, $m \approx 1.44\kappa \max(n_0, n_1)$, $\ell = \lambda + \log_2 n_0 + \log_2 n_1$, correlation-robust function CRF).

$\ell = \lambda + \log(n_0) + \log(n_1)$. Hence, due to the large constants, the OPRF-based approach is less efficient in concrete terms than the SCS or PWC circuits, even though it scales with $\mathcal{O}(n)$ while both other circuits scale with $\mathcal{O}(n \log n)$. However, if the set sizes of the parties greatly differ, i.e., for the mobile messenger application where $n_0 \gg n_1$, the OPRF-based approach can be more efficient than other circuit constructions and in fact even more efficient than all other PSI protocols, since the elements in the much larger set of P_0 can be processed at very low cost (cf. §5.5.2.3).

5.4 OT-Based PSI

In this section, we give special-purpose PSI protocols based on OT: The Garbled Bloom filter protocol of [DCW13] (§5.4.1) and our novel OT-based PSI protocol (§5.4.2).

5.4.1 Bloom Filter-Based PSI

The PSI protocol of [DCW13] uses *Bloom Filters (BF)* and OT to compute the intersection. We summarize Bloom filters in §5.4.1.1 and the PSI protocol of [DCW13] in §5.4.1.2. We then present a redesigned optimized version of the protocol in §5.4.1.3. Our optimized protocol reduces the computation and communication complexity by factor 1.3× to 2× (cf. Table 5.7 and Table 5.11).

5.4.1.1 The Bloom Filter

A BF that represents a set of n elements consists of an m-bit string F and k independent uniform hash functions $h_1, ..., h_k$ with $h_i : \{0,1\}^\sigma \mapsto [1, m]$, for $1 \leq i \leq k$. Initially, all bits in F are set to zero. An element x is inserted into the BF by setting $F[h_i(x)] = 1$ for all i. To query if the BF contains an item y, one checks all

bits $F[h_i(y)]$. If there is at least one j such that $BF[h_j(y)] = 0$, then y is not in the BF. If, on the other hand, all bits $BF[h_i(y)]$ are set to one, then y is in the BF except for a false positive probability ϵ. An upper bound on ϵ can be computed as $\epsilon = p^k(1+\mathcal{O}(\frac{k}{p}\sqrt{\frac{\ln m - k\ln p}{m}}))$, where $p = 1-(1-\frac{1}{m})^{kn}$. In the PSI protocol of [DCW13] the number of hash functions is set to $k = 1/\epsilon$ and the size of the BF is set to $m = kn/\ln 2 \approx 1.44kn$. Setting $\epsilon = 2^{-\kappa}$ then results in $k = \kappa$ and a filter of size $m \approx 1.44\kappa n$.[5]

The intersection between two BFs F_X and F_Y, representing sets X and Y, respectively, can be computed as $F_{(X\wedge Y)} = F_X \wedge F_Y$. However, as described in [DCW13], $F_{(X\wedge Y)}$ has more bits set to one than a BF $F_{(X\cap Y)}$ that was generated from the intersection $X \cap Y$. For example, assume that there are two sets $X = \{x\}$ and $Y = \{y\}$ with $x \neq y$. If there exist i, j such that $h_i(x) = h_j(y)$, we have $F_{(X\wedge Y)}[h_i(x)] = 1$. However, the intersection $X \cap Y = \varnothing$, results in $F_{(X\cap Y)}[h_i(x)] = 0$. Thus, learning $F_{(X\wedge Y)}$ reveals more information about the set of the other party than is revealed by only obtaining the result, so a different approach is needed, as described next.

5.4.1.2 Garbled Bloom Filter-Based PSI

To avoid unintentional information leakage when using Bloom filters for PSI, the authors of [DCW13] introduced a variant of the BF, called Garbled Bloom Filter (GBF). Like a BF, a GBF G uses k hash functions $h_1, ..., h_k$, but instead of single bits, it holds shares of length ℓ at each position $G[i]$, for $1 \leq i \leq m$. These shares are chosen uniformly at random, subject to the constraint that for every element x contained in the filter G it holds that $\bigoplus_{j=1}^{\kappa} G[h_j(x)] = x$.

To represent a set X using a GBF G, all positions of G are initially marked as unoccupied. Each element $x \in X$ is then inserted as follows. First, the insertion algorithm tries to find a hash function $t \in [1...\kappa]$ such that $G[h_t(x)]$ is unoccupied (the probability of not finding such a function is equal to the probability of a false positive in the BF, which is negligible due to the choice of parameters). All other unoccupied positions $G[h_j(x)]$ are set to random ℓ-bit shares for $j \neq t$. Finally, $G[h_t(x)]$ is set to $G[h_t(x)] = x \oplus \left(\bigoplus_{j=1, j\neq t}^{k} G[h_j(x)]\right)$ to obtain a valid sharing of x. We emphasize that because existing shares need to be re-used, the generation of the GBF cannot be fully parallelized. We describe below in §5.4.1.3 how the protocol can be modified to enable a parallel execution. The GBF can then be used to perform PSI as outlined in Protocol 16.

Optimization. Since one input of the sender in the OT is fixed to zero, the OT extension protocol can be optimized such that only one value needs to be transferred

[5] In the GBF-based PSI protocol, ϵ needs to be set to the *computational* security parameter κ, since one of the parties might mount a brute force attack where it attempts to find items that are mapped to "1" locations in the Bloom filter. The parameters must ensure that the success probability of this attack is negligible.

PROTOCOL 16 (Garbled Bloom Filter PSI Protocol of [DCW13]).

- **Input of P_0:** $X = \{x_1, ..., x_{n_0}\}$.
- **Input of P_1:** $Y = \{y_1, ..., y_{n_1}\}$.
- **Common Input:** Symmetric security parameter κ; Statistical security parameter λ; $m = \kappa \cdot max(n_0, n_1)/\ln 2$; GBF entry length $\ell = \lambda$; $k = \kappa$ random hash functions $\{h_1, ..., h_k\} : \{0,1\}^\sigma \mapsto [1...m]$.
- **Oracles and cryptographic primitives:** Both parties have access to a $\binom{2}{1}$ OT_ℓ^m functionality.

1. Using the hash functions $\{h_1, ..., h_k\}$, P_0 generates a m-bit GBF G_X with ℓ-bit entries from X and P_1 generates a m-bit BF F_Y from Y.
2. The parties invoke the $\binom{2}{1}$ OT_ℓ^m functionality where, in the i-th OT, P_0 plays the sender and inputs $(0^\ell, G_X[i])$ and P_1 plays the receiver and inputs $F_Y[i]$. P_1 receives as output the intersection GBF $G_{(X \wedge Y)}$ for which $G_{(X \wedge Y)}[i] = G_X[i]$ if $F_Y[i] = 1$ and $G_{(X \wedge Y)}[i] = 0^\ell$ else.
3. P_1 computes $Z = \{y_j | y_j = \bigoplus_{i=1}^k G_{(X \wedge Y)}[H_i(y_j)]\}$ for every $1 \leq j \leq n_1$.

- **Output:** P_0 has no output; P_1 outputs $Z = Y \cap Y$.

from sender to receiver by using the correlated OT flavor of §3.4.1, which reduces the communication complexity. Additionally, P_0 only needs to evaluate the CRF in the OT for one value, since the other value is ignored, while P_1 only needs to evaluate the CRF in the i-th OT if $F_Y[i] = 1$ since $GBF_{(X \wedge Y)}[i]$ is set to 0^ℓ if $F_Y[i] = 0$, for $1 \leq i \leq m$. To estimate the number of CRF evaluations of P_1, we need to estimate the number of entries in the BF F_Y that are set to 1. Based on our choice of parameters, we can approximate the probability that a single bit in the BF is set to 1 as $1 - (1 - (\frac{1}{m}))^{kn} \approx \frac{1}{2}$. Overall, this means that for a BF of size m with $k = \kappa$ hash functions, P_1 has to perform $m/2$ CRF evaluations.

Efficiency. The highest cost for the original GBF-based PSI protocol of [DCW13] comes from the $\binom{2}{1}$ OT_ℓ^m. Using our optimized $\binom{2}{1}$ OT extension protocol from §3, P_0 has to perform $2m$ CRF evaluations and send $2m\ell$ bits while P_1 has to perform m CRF evaluations and send $m\kappa$ bits. Using the optimization described above, the computation is reduced by factor $2\times$ and the communication is reduced by factor $1.3\times$: P_0 has to perform m CRF evaluations and send $m\ell$ bits while P_1 has to perform $m/2$ CRF evaluations and send $m\kappa$ bits.

5.4.1.3 Random GBF-Based PSI

We introduce a further optimization of the GBF-based PSI protocol of [DCW13], which we call the *random GBF protocol*. The core idea is to have parties collaboratively

generate a random GBF. This is in contrast to the original protocol where the GBF had to be of a specific structure (i.e., have the XOR of the entries of $x \in X$ be x). The modified protocol can be based on sender random OT extension (SR-OT, cf. §3.4.2) and can be further optimized taking into account the constant output, similar to the original GBF protocol. P_0 learns all positions of this random GBF and then sends to P_1 the XOR of the GBF values corresponding to each of its inputs, and P_1 compares these values to the XOR of the GBF values of its own inputs. We describe the random GBF protocol in more detail in Protocol 17.

PROTOCOL 17 (Our Random Garbled Bloom Filter PSI Protocol)**.**

- **Input of P_0:** $X = \{x_1, ..., x_{n_0}\}$.
- **Input of P_1:** $Y = \{y_1, ..., y_{n_1}\}$.
- **Common Input:** Symmetric security parameter κ; Statistical security parameter λ; GBF entry length $\ell = \lambda + \log_2(n_0) + \log_2(n_1)$; $m = \kappa \cdot max(n_0, n_1)/\ln 2$; $k = \kappa$ random hash functions $\{h_1, ..., h_k\} : \{0,1\}^\sigma \mapsto [1...m]$.
- **Oracles and cryptographic primitives:** Both parties have access to a $\binom{2}{1}$ SR-OT$_\ell^m$ functionality.

1. P_1 generates a m-bit BF F_Y from Y using the hash functions $\{h_1, ..., h_k\}$.
2. The parties invoke the $\binom{2}{1}$ SR-OT$_\ell^m$ functionality where P_0 plays the sender and has no input while P_1 plays the receiver and inputs F_Y as choice bits. From the i-th OT, P_0 obtains two random strings $(m_0^i, m_1^i) \in \{0,1\}^{2\ell}$ while P_1 obtains $m^i_{F_Y[i]}$, for $1 \leq i \leq m$.
3. P_0 creates a GBF G_X s.t. $G_X[i] = m_1^i$ while P_1 creates a GBF G_Y s.t. $G_Y[i] = m^i_{F_Y[i]}$ if $F_Y[i] = 1$ and 0^ℓ else.
4. P_0 computes $M_i = \bigoplus_{l=1}^k G_X[h_l(x_i)]$ and sends the M_i's in randomly permuted order to P_1, for every $1 \leq i \leq n_0$.
5. P_1 computes $Z = \{y_j | \exists i \text{ s.t. } \bigoplus_{l=1}^k G_Y[h_l(y_j)] = M_i\}$.

- **Output:** P_0 has no output; P_1 outputs $Z = Y \cap Y$.

Correctness. For each item in the intersection, P_1 gets from P_0 the same XOR value that it computed from its own GBF, and therefore identifies that the item is in the intersection. For any item which is not in the intersection, it holds with overwhelming probability that the XOR value computed by P_1 is independent of the n_0 values received from P_0. The probability of a false positive identification for that value is therefore $n_0 \cdot 2^{-\ell}$. The probability of a false positive identification for any of the values is $n_0 n_1 \cdot 2^{-\ell}$. To achieve correctness with probability 1-$2^{-\lambda}$, we therefore set $\ell = \lambda + \log_2 n_0 + \log_2 n_1$.

Security. The security of each party can be easily proved using a simulation argument. P_1's security is obvious, since the only information that P_0 learns are the random

outputs of the SR-OT protocol, which are independent of P_1's input and can be easily simulated by P_0. P_0's security is apparent from observing that the information that P_1 receives from P_0 is composed of:

- The XOR values that P_1 computed for each item in the intersection.
- The XOR values that P_0 computed for its $n_0 - |X \cap Y|$ items that are not in the intersection. These values are independent of P_1's BF unless one of these items is a false-positive identification in the filter, which happens with negligible probability ϵ.

Therefore, the information received from P_0 can be easily simulated by P_1 given its legitimate output, i.e., $X \cap Y$.

Optimization. We can apply the same optimizations to our random GBF protocol as to the GBF protocol. In addition, note that if P_0 builds a Bloom filter F_X based on its set X, it can ignore the CRF evaluation in the i-th OT if $F_X[i] = 0$, for $1 \leq i \leq m$. Analogue to P_1 in the GBF protocol, this reduces the number of CRF evaluations for P_0 from m to $m/2$.

Efficiency. As shown in Table 5.7 on page 134, our resulting random GBF-based PSI protocol has less computation and communication complexity than the original GBF protocol in [DCW13] (even with the optimizations described in §5.4.1.2). In terms of communication, in our random GBF protocol, P_0 has to send the $n_0\ell$-bit vector m_{P_0} and P_1 has to send $m\kappa$ bits in the SR-OT. (This is compared to $2m\lambda$ bits and $m\kappa$ bits sent in the original protocol. Later in our experiments in §5.5.2 we show that the communication is reduced by a factor between 1.3× to 1.5×, cf. Table 5.11).

A main advantage of our protocol is that it allows to parallelize *all* operations: BFs can be generated in parallel (bits in the BF are changed only from 0 to 1) and, most importantly, the random GBF can also be constructed in parallel, in contrast to the original GBF-based protocol.

5.4.2 PSI via OT-Based OPRF

In this section, we describe our new OT-based PSI protocol, of which an earlier version appeared in [PSZ14, PSSZ15]. In contrast to the conference versions, we improve our protocol such that its complexity is now independent of the bit length σ for realistic set sizes. The core of our OT-based PSI protocol is an efficient OPRF (cf. §5.1.2) instantiation using our OT extension improvements, in particular the sender random OT functionality (cf. §3.4.2) and the $\binom{N}{1}$ OT extension protocol (cf. §3.2.4). Our protocol operates in three steps: The parties *hash* their elements into hash tables, mask their elements using the *OPRF*, and compute the *plaintext intersection* of these masks to identify the intersecting elements. In the hashing step we use the methods

from §5.2 for creating the hash tables. We give a full description of the protocol in Protocol 18.

PROTOCOL 18 (Our OT-based PSI Protocol)**.**

- **Input of P_0:** $X = \{x_1, ..., x_{n_0}\}$.
- **Input of P_1:** $Y = \{y_1, ..., y_{n_1}\}$.
- **Common Input:** Bit-length of elements σ; Number of bins $b = \epsilon n_1$ (cf. §5.2.2.2); k random hash functions $\{h_1, ..., h_k\} : \{0,1\}^\sigma \mapsto [1...b]$; Reduced bit-length of items in the hash table $\mu = \sigma - \log_2 b + \log_2 k$; Symmetric security parameter κ; Statistical security parameter λ; Mask-length $\ell = \lambda + \log_2(kn_0) + \log_2(n_1)$; $N = 2^\mu$; Dummy element d_1; Stash size s.
- **Oracles:** Both parties have access to a $\binom{N}{1}$ SR-OT$^1_\ell$ functionality.

1. **Hashing:**
   a) P_0 maps the elements in its set X into a two-dimensional hash table $T_0[\,][\,]$ using simple hashing and k hash functions $\{h_1, ..., h_k\}$. The first dimension has size b and addresses the bin in the table while the second dimension addresses the elements in the bins.
   b) P_1 maps the elements in its set Y into a one-dimensional hash table $T_1[\,]$ and stash $S[\,]$ using Cuckoo hashing and k hash functions $\{h_1, ..., h_k\}$. The hash table has size b and the stash has size s. P_1 then fills all empty entries in T_2 and S with d_1.

   Let $|T_0[i]|$ be the number of elements that are stored in the i-th bin of the hash table T_0 and μ be the bit-length of these elements for $1 \leq i \leq b$.

2. **OPRF evaluation (via OT):**
   For each bin $1 \leq i \leq b$, the parties perform the following steps:
   a) Let $v_j = T_0[i][j]$ and $w = T_1[i]$ for $1 \leq j \leq |T_0[i]|$.
   b) The parties evaluate an OPRF using the $\binom{N}{1}$ SR-OT$^1_\ell$ functionality, where P_0 has no inputs and obtains a random N-entry lookup table L and P_1 inputs w as choice bits and obtains a random mask $L[w]$.
   c) P_0 computes $M_0[i][j] = L[v_j]$ and P_1 computes $M_1[i] = L[w]$.

   Stash: For each element in the stash S, the parties repeat the same steps where, for the i-th stash position, P_0 evaluates the OPRF on its whole input set X and obtains n_0 masks $M_{S_0}[i]$ while P_1 evaluates the OPRF on $S[i]$ and obtains one mask $M_{S_1}[i]$.

3. **Plaintext Intersection:**
   a) Let $V = \bigcup_{1 \leq i \leq b, 1 \leq j \leq |T_1[i]|} M_0[i][j]$. P_0 randomly permutes and sends V to P_1.
   b) P_1 computes the intersection $Z = \{T_1[i] | M_1[i] \in V\}$.

   Stash: P_0 permutes and sends $M_{S_0}[i]$ to P_1, who adds $S[i]$ to Z if $M_{S_1}[i] \in M_{S_0}[i]$.

- **Output:** P_0 has no output; P_1 outputs $Z = X \cap Y$.

In the first step of our OT-based PSI protocol, the parties map their elements

into hash tables T_0 and T_1 where the elements in the tables have bit-length $\mu = \sigma - \log_2 b + \log_2 k$ due to permutation-based mapping (cf. §5.2.3). P_0 uses simple hashing and hence its hash table T_0 has two dimensions, where the first dimension addresses the bins and the second dimension addresses the elements in the bins. P_1 uses Cuckoo hashing and hence its hash table T_1 has only one dimension, which addresses the bins. Our OT-based PSI protocol then evaluates an OPRF F (cf. §5.1.2) where, for each bin, P_0 samples a random key and P_1 inputs the μ-bit element in bin $T_1[i]$ and obtains the resulting mask $M_1[i] = F_{k_i}(T_1[i])$, for $1 \leq i \leq b$. The OPRF must ensure that P_0 learns no information on the input of P_1 and that P_1 learns no information except the outputs that correspond to its elements.

The main observation is that we can instantiate an OPRF for μ-bit inputs using a $\binom{2^\mu}{1}$ SR-OT$^1_\ell$, where P_0 plays the sender and obtains a lookup table $L : \{0,1\}^\mu \mapsto \{0,1\}^\ell$ while P_1 plays the receiver who inputs $T_1[i]$ and obtains $L[T_1[i]]$. P_0 can then evaluate the OPRF on the elements in its bin $T_0[i]$ locally by computing $M_0[i][j] = L[T[i][j]]$, for $1 \leq i \leq b$ and $1 \leq j \leq |T_0[i]|$. After P_0 has evaluated the OPRF for all bins i, it collects the OPRF outputs $M_0[i]$ for all $|T_0[i]|$ elements in a bin to a set V and randomly permutes and sends V. P_1 identifies whether $T_1[i]$ is in the intersection by checking whether $M_1[i]$ matches any element in V. If the element $T_1[i]$ matches any element in $T_0[i]$, their OPRF outputs will be equal. If $T_1[i]$ matches no element in $T_0[i]$, their OPRF outputs will differ except with probability $|T_0[i]| \cdot 2^{-\ell}$.

The elements in the stash of P_1 are processed independently in a similar fashion: Both parties evaluate the OPRF, P_1 obtains the output for the elements in its stash, and P_0 evaluates the OPRF locally on each element of its set and sends the permuted outputs to P_1, who identifies the intersection.

Efficiency. The main computation and communication overhead comes from the OPRF evaluation. The efficiency of the OPRF depends greatly on the underlying instantiation. We instantiate the OPRF that maps μ-bit inputs to ℓ-bit outputs using our improved $\binom{2^\mu}{1}$ SR-OT$^1_\ell$ extension protocol from §3.2.4 with the linear BCH code $[2^{77}, 512, 129]$, generated by [MZ06]. This $[2^{77}, 512, 129]$ code encodes elements with up to length $\mu = 77$ bits using $\rho = 512$ bit codewords with relative Hamming distance κ. Keeping in mind the hashing to shorter representation techniques in §2.5, this code allows our protocol to process sets with up to 100 billion (2^{37}) elements, independently of their bit-length σ. Overall, the parties perform $s + b$ OPRF evaluations, which correspond to $\binom{2^\mu}{1}$ SR-OT$^{s+b}_\ell$, where the stash size s and the number of bins $b = \epsilon n_1$ are chosen to achieve negligible Cuckoo hashing error probability (cf. §5.2.2.2). Regarding the communication, P_1 sends $512(s+b)$ bits for the $\binom{2^\mu}{1}$ SR-OT$^1_\ell$, while P_0 sends $k\ell n_0$ bits for the permuted OPRF output, where k is the number of hash functions used for Cuckoo hashing (cf. §5.2.2.2) and $\ell = \lambda + \log_2(kn_0) + \log_2(n_1)$. Regarding the computation, note that in a naive $\binom{2^\mu}{1}$ SR-OT$^1_\ell$ evaluation the sender P_0 would need to perform 2^μ CRF evaluations, instantiated using SHA-256 (cf. §3.2.4), one for each message. However, since P_0 only needs to obtain the output for actual elements in its

bins, it only needs to perform $(k+s)n_0$ CRF evaluations.

Correctness. In the following, we analyze the correctness of the scheme. We assume that in Step 1 in Protocol 18, P_0 has used simple hashing to map each element k times into the hash table T_0 while P_1 has used Cuckoo hashing to map each element once into the hash table T_1.

If $x = y$ then P_0 and P_1 will have the same item in a bin in their hash tables (P_1 has mapped the item to one of k bins while P_0 has mapped the item to all k bins). For this bin, P_1 obtains $M_x = L[x]$ as output of the OPRF and P_0 can locally compute $M_y = L[y]$ with $M_x = M_y$, and P_1 successfully identifies equality.

If $x \neq y$ then the probability that $M_x = M_y$ is $2^{-\ell}$. However, we require that *all* OPRF outputs M_1 for elements in the hash table T_1 of P_1 are distinct from *all* outputs M_0 for elements in the hash table T_0 of P_0, which happens with probability $kn_0n_12^{-\ell}$. Thus, to achieve correctness with probability $1\text{-}2^{-\lambda}$, we must increase the bit-length of the OT outputs to $\ell = \lambda + \log_2(kn_0) + \log_2(n_1)$.

Security. P_1's security is obvious, since the only information that P_0 learns are the random values chosen in the random OT, which are independent of P_1's input.

As for P_0's security, note that P_1's view in the protocol consists of its outputs M_1 of the $\binom{N}{1}$ SR-OT$^1_\ell$ protocols, and of the values M_0 sent by P_0. If there are two elements $x \in X$ and $y \in Y$ with $x = y$, then there are outputs $M_x = M_y$. Otherwise, for $x \neq y$, these values are uniformly distributed and P_1 can gain no information about M_x, which is guaranteed by the properties of the $\binom{N}{1}$ SR-OT$^1_\ell$ protocol. In both cases, the view of P_1 can be easily simulated given the output of the protocol (i.e., knowledge whether $x = y$). The protocol is therefore secure according to the common security definitions of secure computation [Gol04].

5.5 Evaluation

In the following, we experimentally evaluate the most promising PSI protocols of each category: The naive hashing protocol, the server-aided protocol [KMSB13], the DH-based protocol [Mea86], the blind-RSA protocol [CT10], the sort-compare-shuffle circuit (SCS) [HEK12], the pairwise comparison circuit (PWC, cf. §5.3.2), the OPRF circuit (cf. §5.3.3), the Garbled Bloom filter protocol (GBF) [DCW13], the random Garbled Bloom filter protocol (random GBF, cf. §5.4.1.3), and our OT-based PSI protocol (cf. §5.4.2). We first discuss their implementation features and compare the protocols theoretically (§5.5.1). We then give an empirical performance comparison between the protocols for different settings (§5.5.2). Throughout the evaluation, we divide the PSI protocols into four categories, depending on whether the protocol is based on *public-key* operations, *circuits*, *OT*, or provides *limited security* and mark the best result of each category in bold.

5.5.1 Theoretical Evaluation

Before evaluating the empirical performance of the PSI protocols, we discuss implementation features of the protocols such as their suitability for large-scale PSI on sets with several million elements (§5.5.1.1) or the ability of the schemes for parallelization (§5.5.1.2), and give their asymptotic computation and communication complexities (§5.5.1.3).

5.5.1.1 Suitability for Large-Scale PSI

Although hardly discussed, memory consumption poses a very big problem when implementing cryptographic schemes that operate on large amounts of data. As such, many of the implemented PSI protocols quickly exceeded the main memory, requiring more engineering effort and a more careful implementation to allow for PSI on larger sets. In fact, even computing the plaintext intersection for sets of billions of elements becomes a tedious problem, since at least one set needs to be fully stored at one point during the execution. In this case, one can store the data on disk, which decreases performance greatly when arbitrary look-ups are performed.

Limited Security & Public-Key-Based PSI. The naive-hashing, server-aided, and public-key-based PSI schemes are very memory efficient, since they operate only on single elements and can be easily pipelined, allowing PSI on millions of elements even on standard PCs.

Circuit-Based PSI. The circuit-based PSI schemes have a very high memory consumption. In our implementations, we evaluate and delete gates if they are not used anymore to decrease the memory consumption. Yao's garbled circuits requires more memory for pre-computation than GMW, since κ-bit keys have to be stored for each wire instead of single bits (cf. §2.4.3). A pipelined circuit generation and evaluation, as is done in VMCRYPT [Mal11], FastGC [HEKM11, HS13], or PCF [KMSB13] would allow us to perform PSI on larger sets. The main memory limitation of our GMW implementation comes from the circuit having to be fully built and stored in memory. To decrease the memory footprint of the circuit, we build circuits that are evaluated many times in parallel in a SIMD fashion (cf. §4.2.2), which evaluates the circuit on multiple values in parallel. This SIMD evaluation especially benefits the PWC and OPRF circuits, since the same circuit is evaluated on all elements in parallel.

OT-Based PSI. The garbled Bloom filter and random garbled Bloom filter PSI protocols have to store the full Bloom filter in memory to identify the intersecting elements. The garbled Bloom filter holds $1.44n\kappa$ entries of at least λ-bit shares, resulting in at least 875 MB for sets of one million elements. In addition, the parties have to perform arbitrary element look-ups, which greatly decrease the performance if the Bloom filter is outsourced to the hard disk.

The main memory limitation of our OT-based PSI protocol are the hash tables, in particular the Cuckoo hash table. While the hash table for simple hashing can

be easily stored on disk, the Cuckoo hash table needs to perform arbitrary look-ups when evicting elements. The Cuckoo hash table holds $1.2n$ elements of at most $\ell = \lambda + \log(n_0) + \log(n_1)$-bit length, resulting in 12 MB for sets of one million elements and hence scales much better than the Bloom filter-based protocols.

5.5.1.2 Parallelizability

The experiments we perform in the empirical evaluation only consider execution using a single thread. However, if more computational resources are available, the schemes can be run using multiple threads in order to improve their performance. Note, however, that the bottleneck for many protocols (i.e., all except the public-key-based protocols) quickly shifts from computation to communication, since symmetric cryptographic operations can be evaluated very efficiently using AES-NI. In the following, we discuss the ability of the schemes to be parallelized.

Limited Security & Public-Key-Based PSI. The naive-hashing, server-aided, and public-key-based PSI schemes can easily be parallelized since the elements are processed independently of each other. The main bottleneck for parallelization in all these schemes is the plaintext intersection of hash values that is done at the end of each protocol.

Circuit-Based PSI. For a discussion on the parallelizability of Yao's garbled circuits and the GMW protocol, see §2.4.3. In summary, the setup phase of the GMW protocol can be parallelized easily while the parallelizability of Yao's garbled circuits depends on the evaluated circuit. The PWC and OPRF circuits can be parallelized well, since they consist of many independent sub-circuits. The SCS circuit requires more involved circuit-dependent parallelization methods, since the sub-circuits have many interconnections. For the SCS circuit, an automatically parallelizing compiler could be used [BK15].

OT-based PSI. For all OT-based PSI protocols it holds that the underlying OT extension protocol can be parallelized well. The main differences in parallelizability are due to the hashing scheme that is used to map the elements into the corresponding structure. In the GBF-based PSI protocol of [DCW13], P_0 has to generate the garbled Bloom filter in advance, and this step does not parallelize well. This is improved on by our random GBF protocol from §5.4.1.3, where the GBF is generated as an output of OT extension and can hence be fully parallelized. In our OT-PSI protocol, the main bottleneck for parallelization is the Cuckoo-hashing procedure. However, Cuckoo hashing can be pre-processed since no input of the other party is required.

5.5.1.3 Asymptotic Performance Comparison

We depict the asymptotic computation complexity for the party with the majority of the workload and total communication complexity of the PSI protocols in Table 5.8.

Type	Protocol	Computation [#Ops sym/pk]	Communication [bit]
Limited Security	Naive Hashing	w sym	$n_0\ell$
	Server-aided [KMRS14]	w sym	$t + \lvert X \cap Y \rvert$
Public-Key	DH FFC [Mea86]	$2t$ pk	$t\psi + n_0\ell$
	DH ECC [Mea86]	$2t$ pk	$t\xi + n_0\ell$
	RSA [CT10]	$2t$ pk	$t\psi + n_0\ell$
Circuit	Yao SCS [HEK12]	$12w\sigma \log w + 3w\sigma$ sym	$6w\kappa\sigma \log w + 2w\kappa\sigma$
	GMW SCS [HEK12]	$18w\sigma \log w$ sym	$6w(\kappa+2)\sigma \log w$
	Yao PWC (§5.3.2)	$\sigma(4\epsilon n_1 max_b + 4sn_0 + 3\epsilon n_1)$ sym	$2\epsilon n_1 \kappa max_b \sigma + 3sn_0\kappa\sigma + 2\epsilon n_1\sigma$
	GMW PWC (§5.3.2)	$6\sigma(\epsilon n_1 max_b + sn_0)$ sym	$2(2+\kappa)\sigma(\epsilon n_1 max_b + sn_0)$
	Yao OPRF (§5.3.3)	$21\ 760n_1 + 3\sigma n_1$ sym	$10\ 880n_1\kappa + 2n_1\kappa\sigma + n_0\ell$
	GMW OPRF (§5.3.3)	$32\ 640n_1$ sym	$10\ 880n_1(\kappa+2) + n_0\ell$
OT	GBF [DCW13]	$3.6w\kappa$ sym	$1.44w\kappa(\kappa+\lambda)$
	Random GBF (§5.4.1.3)	$3.6n_1\kappa$ sym	$1.44n_1\kappa^2 + n_0\ell$
	OT + Hashing (§5.4.2)	$3\epsilon n_1 + (k+s)n_0$ sym	$512\epsilon n_1 + (k+s)n_0\ell$

Table 5.8: Asymptotic complexities for PSI protocols (σ: bit length of set elements; $t = n_0 + n_1$; $w = max(n_0, n_1)$; pk: public-key operations; sym: symmetric cryptographic operations; $\ell = \lambda + \log n_0 + \log n_1$; κ, ψ, ξ, λ: security parameters as defined in §2.1.3; ϵ, k, s, max_b: Hashing parameters as defined in §5.2.1 and §5.2.2). Computation gives the number of operations that need to be performed in sequence.

The computation complexity is expressed as the number of symmetric cryptographic primitive evaluations (sym) and the number of asymmetric cryptographic primitive operations (pk). We assume 3 sym per OT (2.5 sym for the Bloom filter-based protocols), 4 sym per AND gate in Yao's protocol, and 6 sym per AND gate in the GMW protocol. For GMW, we use 2-MT pre-computation (cf. §4.3.2) and omit N-MT pre-computation (cf. §4.3.3). For a comparison between 2-MT and N-MT refer to §4.6.

The most crucial observation we make from the asymptotic complexities is that, asymptotically, the performance amongst the schemes with the same type is very similar. The naive hashing and server-aided protocol both require 1 sym operation per element, the public-key-based protocols all require 2 pk operations per element and need to send two ciphertexts and a hash value, the circuit-based protocols all have to perform work linear in the number of AND gates in the circuit, and the Bloom filter-based protocols both have to perform work linear in the size of the Bloom filter. The main discrepancy can be seen among the OT-based protocols, where the communication of the Bloom filter-based protocols scales quadratically with the symmetric security parameter κ while our OT-based PSI protocol scales only linear in the security parameter κ (we need 512-bit codewords to achieve relative Hamming distance κ, cf. §3.2.4.1).

5.5.2 Empirical Evaluation

We empirically evaluate and compare the performance of the presented semi-honest PSI protocols. We first describe our benchmarking environment and outline our implementations (§5.5.2.1). We then benchmark the protocols in a LAN and a WAN setting and give their concrete communication for sets of equal (§5.5.2.2) and unequal size (§5.5.2.3). Finally, we evaluate the performance on the parallelizability of PSI schemes (§5.5.2.4) as well as for large-scale PSI (§5.5.2.5).

5.5.2.1 Benchmarking Environment

We ran our experiments in a *LAN* and a *WAN* setting. The LAN setting consists of two PCs (Intel Haswell i7-4770K CPU with 3.5 GHz and AES-NI support and 16 GB RAM) that are connected via a Gigabit Ethernet. The WAN setting consists of two Amazon EC2 m3.medium instances (Intel Xeon E5-2670 CPU with 2.6 GHz and AES-NI support and 3.75 GB RAM) that are located in North Virginia (US east coast) and Frankfurt (Europe) with an average bandwidth of 98 MBit/s and an average round-trip time of 94 ms.

We evaluate the performance of the PSI protocols in two scenarios. In the first scenario, P_0 and P_1 hold the same number of input elements $n_0, n_1 \in \{2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}\}$. In the second scenario, P_0 has a larger set than P_1 and we set $n_0 \in \{2^{16}, 2^{20}, 2^{24}\}$ and $n_1 \in \{2^8, 2^{12}\}$. We present the total run-times of the protocols (setup time and online time). For the SCS and PWC circuit-based protocols whose complexity depends on the bit-length of elements σ, we fix $\sigma = 32$ (e.g., for PSI on IPv4 addresses). We use the long-term security parameters as described in §2.1.3. We benchmarked the server-aided PSI protocol of [KMRS14] by executing the trusted server on one machine and the two clients that wish to compute the intersection on the second machine.

Implementations. The implementation of the blind-RSA-based [CT10] and garbled Bloom filter [DCW13] protocols were taken from the authors, but we used a hash-table to compute the last step in the blind-RSA protocol that finds the intersection (the original implementation used pairwise comparisons with quadratic run-time overhead) and our $\binom{2}{1}$ OT extension implementation from §3 for the Bloom filter protocol. We use our state-of-the-art Yao's garbled circuits and GMW protocol implementations with 2-MT pre-computation (cf. §4.3.2) in the C++ ABY framework from §4. Note, however, that we use Yao's garbled circuits without inter party parallelization (cf. §2.4.1.2) to analyze the performance, independent of the parallelizability of the circuit. For Yao's garbled circuits protocol, we evaluated a size-optimized version of the SCS circuit (comparison circuits of size and depth σ) while for GMW we evaluated a depth-optimized version (comparison circuits of size 3σ and depth $\log_2 \sigma$) for σ-bit input values (cf. §4.2.1.7). We instantiate the pseudo-random permutation of the server-aided PSI protocol in [KMRS14] and the CRF in the $\binom{2}{1}$ OT extension with AES, and the RO and the CRF in the $\binom{N}{1}$ OT extension with SHA-256. We imple-

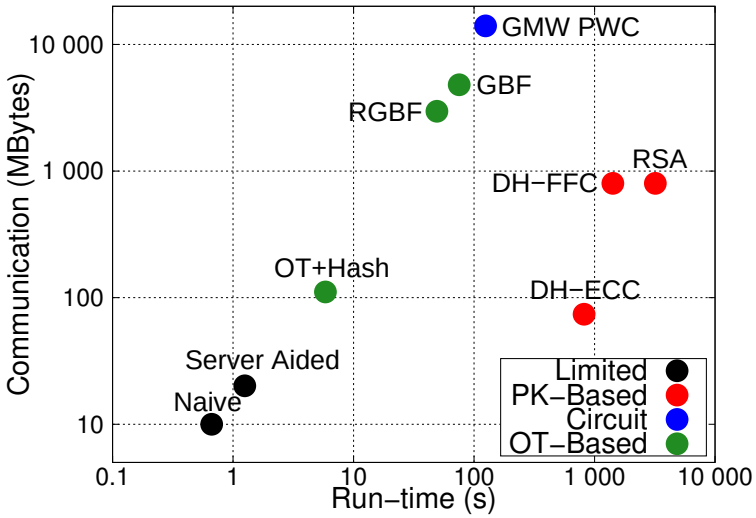


Figure 5.3: LAN run-time in s and communication in MBytes of PSI protocols for $n_0 = n_1 = 2^{20}$ elements and $\kappa = 128$-bit security. Detailed results are given in Table 5.9 and Table 5.11.

mented FFC (finite field cryptography) using the GMP library (v. 5.1.2), ECC using the Miracl library (v. 5.6.1), and symmetric cryptographic primitives using OpenSSL (v. 1.0.1e). We perform all operations in FFC in a subgroup of order q, where $|q| = 2\kappa$-bits. We argue that we provide a fair comparison, since all protocols are implemented in the same programming language (C/C++), run on the same hardware, and use the same underlying libraries for cryptographic operations.

For each protocol we measured the time from starting the program until the client outputs the intersecting elements. All run-times are averaged over 10 executions.

5.5.2.2 PSI with Equal Set Sizes

We evaluate the empirical performance of the PSI protocols in the LAN setting and give the concrete communication of the protocols. While the LAN setting does not necessarily represent a real-world setting for PSI, it allows us to benchmark the protocols in an almost ideal network setting and hence focus on the computation complexity of the protocols. We give a classification for $n = 2^{20}$ element sets in Figure 5.3 and depict the detailed run-time for the LAN setting in Table 5.9 and for the WAN setting in Table 5.10. The communication is given in Table 5.11. We now compare the performance of the different types of PSI protocols and then compare the PSI protocols of the same type.

Type	Set Size n	2^8	2^{12}	2^{16}	2^{20}	2^{24}
Limited Security	Naive Hashing	**1**	**3**	**38**	**665**	**12 368**
	Server-aided [KMRS14]	**1**	5	78	1 250	20 053
Public-Key	DH FFC [Mea86]	386	5 846	88 790	1 418 772	22 681 907
	DH ECC [Mea86]	**231**	**3 238**	**51 380**	**818 318**	**13 065 904**
	RSA [CT10]	779	12 546	203 036	3 193 920	50 713 668
Circuit	Yao SCS [HEK12] ($\sigma = 32$)	**320**	3 593	74 548	-	-
	GMW SCS [HEK12] ($\sigma = 32$)	361	1 954	40 872	-	-
	Yao PWC (§5.3.2, $\sigma = 32$)	428	2 294	28 491	-	-
	GMW PWC (§5.3.2, $\sigma = 32$)	460	**1 324**	**10 656**	**124 732**	-
	Yao OPRF (§5.3.3)	968	12 518	-	-	-
	GMW OPRF (§5.3.3)	690	6 672	101 231	-	-
OT	GBF [DCW13]	105	448	4 179	65 218	-
	Random GBF (§5.4.1.3)	**95**	**346**	2 991	49 171	-
	OT + Hashing (§5.4.2)	311	362	**702**	**5 847**	**86 278**

Table 5.9: Run-times in ms for PSI protocols with one thread in the LAN setting. σ: bit-length of elements. "-" indicates that the execution ran out of memory.

Type	Set Size n	2^8	2^{12}	2^{16}	2^{20}
Limited Security	Naive Hashing	**51**	**119**	**886**	**7 277**
	Server-aided [KMRS14]	124	248	1 987	15 578
Public-Key	DH FFC [Mea86]	3 577	56 786	880 075	11 557 061
	DH ECC [Mea86]	**1 949**	**28 686**	**466 606**	**5 007 681**
	RSA [CT10]	10 508	166 453	1 356 757	21 094 586
Circuit	Yao SCS [HEK12] ($\sigma = 32$)	**2 763**	20 826	518 136	-
	GMW SCS [HEK12] ($\sigma = 32$)	5 929	14 415	187 750	-
	Yao PWC (§5.3.2, $\sigma = 32$)	4 248	17 897	178 522	-
	GMW PWC (§5.3.2, $\sigma = 32$)	2 872	**7 644**	**59 572**	**472 687**
	Yao OPRF (§5.3.3)	6 001	65 156	-	-
	GMW OPRF (§5.3.3)	6 939	27 660	386 243	-
OT	GBF [DCW13]	1 248	5 424	31 581	345 484
	Random GBF (§5.4.1.3)	**968**	3 863	22 031	220 570
	OT + Hashing (§5.4.2)	2 278	**2 915**	**8 215**	**58 418**

Table 5.10: Run-times in ms for PSI protocols with one thread in the WAN setting. σ: bit-length of elements. "-" indicates that the execution ran out of memory.

Comparison between Types. From Figure 5.3, we can observe that PSI protocols of the same type have a similar run-time and communication with the exception of the OT-based PSI protocols. The insecure naive hashing protocol and server-aided PSI protocol outperform the other PSI protocols by at least an order of magnitude in computation and communication. The public-key-based PSI protocols require only little communication (especially the DH-ECC protocol), but have the highest run-time. The circuit-based PWC protocol is the only circuit-based protocol to successfully process sets of 2^{20} elements and has a faster run-time than the public-key-based protocols but requires two orders of magnitude more communication. Finally, the OT-based PSI

Type	**Set Size** n	$\mathbf{2^8}$	$\mathbf{2^{12}}$	$\mathbf{2^{16}}$	$\mathbf{2^{20}}$	$\mathbf{2^{24}}$
Limited Security	Naive Hashing	**0.002**	**0.031**	**0.600**	**10.000**	**176.000**
	Server-aided [KMRS14]	0.003	0.063	1.133	20.125	354.000
Public-Key	DH-based FFC [Mea86]	0.195	3.125	50.000	800.000	12 800.000
	DH-based ECC [Mea86]	**0.020**	**0.280**	**4.560**	**74.000**	**1 200.000**
	RSA-based [CT10]	0.195	3.125	50.000	800.000	12 800.000
Circuit	Yao SCS [HEK12] ($\sigma = 32$)	7.522	168.590	3 484.751	-	-
	GMW SCS [HEK12] ($\sigma = 32$)	7.319	162.851	3 348.011	-	-
	Yao PWC (§5.3.2, $\sigma = 32$)	8.833	124.098	1 751.780	-	-
	GMW PWC (§5.3.2, $\sigma = 32$)	**5.587**	**78.229**	**1 101.383**	**14 014.427**	-
	Yao OPRF (§5.3.3)	44.033	704.210	-	-	-
	GMW OPRF (§5.3.3)	43.193	690.890	11 054.050	-	-
OT	GBF [DCW13]	1.037	17.314	288.560	4 801.639	-
	Random GBF (§5.4.1.3)	0.723	11.574	185.241	2 964.855	-
	OT + Hashing (§5.4.2)	**0.055**	**0.456**	**6.799**	**111.299**	**1 828.528**

Table 5.11: Concrete communication in MB for PSI protocols. σ: bit-length of elements. "-" indicates that the execution ran out of memory.

protocols differ in performance: The GBF protocol of [DCW13] has a similar run-time and communication as the circuit-based PWC protocol and our OT-based PSI protocol has a faster run-time than the public-key and circuit-based protocols and requires at least an order of magnitude less communication compared to the circuit-based protocols. Among all PSI protocols, our novel OT-based PSI protocol is the fastest and requires similar communication as the DH-ECC protocol of [Mea86].

Limited Security-Based PSI. The naive hashing protocol outperforms the server-aided protocol by factor 2× in run-time and communication. However, these protocols have weaker security guarantees than the other protocols that we describe.

Public-Key-Based PSI. For the public-key-based PSI protocols, we observe that the DH-based protocol of [Mea86] outperforms the RSA-based protocol of [CT10] when using finite field cryptography (FFC). The elliptic curve cryptography (ECC) instantiation of the DH-based protocol becomes even more efficient and outperforms the FFC instantiation by factor 2×. The advantage of the ECC-based protocol is its communication complexity, which is lowest among all PSI protocols (cf. Table 5.11). We note that a major advantage of these protocols is their simplicity, which makes them relatively easy to implement.

Circuit-Based PSI. We compare the sort-compare-shuffle (SCS) circuit of [HEK12], our PWC circuit (§5.3.2), and the OPRF circuit (§5.3.3), evaluated using Yao's garbled circuits and GMW. We can observe that the PWC circuit scales better than the SCS and OPRF circuits with increasing set sizes and is at least 3 times more efficient for sets of 2^{16} elements. Due to its simple functionality, the PWC circuit can scale up to much larger set sizes and can even process two sets of 2^{20} elements sets using GMW. The SCS circuit outperforms the OPRF circuit and scales better to larger set sizes due

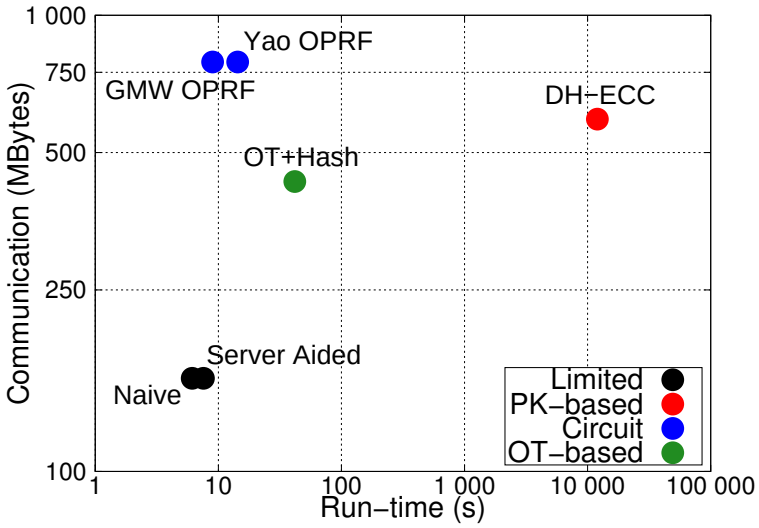


Figure 5.4: LAN run-time in s and communication in MBytes of PSI protocols for unequal sets of $2^{24} = n_0 \gg n_1 = 2^{12}$ elements and $\kappa = 128$-bit security. Detailed results are given in Table 5.13 and Table 5.15.

to the large constants in the OPRF circuit.

OT-Based PSI. The random GBF protocol in §5.4.1.3 improves the optimized GBF variant in §5.4.1.2 by around factor 1.5× in run-time and communication.

Our OT-based PSI protocol has a higher run-time than both of the Bloom filter-based protocols for small set sizes since the number of base-OTs (and hence public-key operations) that are required for the $\binom{N}{1}$ OT extension is four times higher. However, this workload is linear in the security parameter and amortizes with increasing set sizes. For larger set sizes of $n \geq 2^{12}$, our OT-based PSI protocol is up to 9× more efficient in terms of run-time than the random GBF protocol and has between factor 12× and 25× less communication.

5.5.2.3 PSI with Unequal Set Sizes

In many applications of PSI, the set sizes of the parties are not equal. In fact, often a client with a small set of only a few hundred elements wants to perform PSI with a server, which holds a database of millions of records. We perform PSI with unequal set sizes $n_0 \in \{2^{16}, 2^{20}, 2^{24}\}$ and $n_1 \in \{2^8, 2^{16}\}$ using the previously best performing protocols of each category: Naive hashing, the server-aided protocol of [KMRS14], the DH-ECC protocol of [Mea86], the PWC (cf. §5.3.2) and OPRF circuits (cf. §5.3.3), and our OT-based PSI protocol (cf. §5.4.2). We evaluate their performance in the

LAN and WAN setting and graphically compare the protocols based on their run-time and communication in Figure 5.4, give the resulting run-times in Table 5.13 and Table 5.14, and give the concrete communication in Table 5.15. For the circuit PWC protocol and our OT-based PSI protocol, which both use hashing techniques, we used the parameters given in Table 5.12.

Server Set Size n_0	2^{16}				2^{20}				2^{24}			
Client set size $n_1 = 2^8$												
Parameter	k	ϵ	s	$\max_b$	k	ϵ	s	$\max_b$	k	ϵ	s	$\max_b$
Circuit PWC (§5.3.2, $\sigma = 32$)	3	1.27	0	804	3	1.27	0	10 426	2	16	1	8 938
OT + Hashing (§5.4.2)	3	1.27	0	0	3	1.27	0	0	2	2500	0	0
Client set size $n_1 = 2^{12}$												
Parameter	k	ϵ	s	$\max_b$	k	ϵ	s	$\max_b$	k	ϵ	s	$\max_b$
Circuit PWC (§5.3.2, $\sigma = 32$)	3	1.27	0	98	3	1.27	0	816	3	1.27	0	10 488
OT + Hashing (§5.4.2)	3	1.27	0	0	3	1.27	0	0	3	1.27	0	0

Table 5.12: Parameters for circuit PWC and our OT-based protocol used for the unequal set size experiments.

Type	**Client Set Size n_1**	2^8			2^{12}		
	Server Set Size n_0	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
Limited Security	Naive Hashing	**30**	**362**	**5 965**	**31**	**362**	**6 126**
	Server-aided [KMRS14]	63	515	7 267	65	524	7 571
Public-Key	DH ECC [Mea86]	**52 073**	**814 839**	$\mathbf{12 \cdot 10^6}$	**52 057**	**815 715**	$\mathbf{12 \cdot 10^6}$
Circuit	Yao PWC (§5.3.2, $\sigma = 32$)	7 468	-	-	9 351	-	-
	GMW PWC (§5.3.2, $\sigma = 32$)	2 879	32 879	-	**4 915**	31 897	-
	Yao OPRF (§5.3.3)	996	1 194	3 882	11 414	11 764	14 347
	GMW OPRF (§5.3.3)	**692**	**821**	**3 425**	**6 283**	**6 394**	**8 975**
OT	OT + Hashing (§5.4.2)	**624**	**2 738**	**41 815**	**641**	**3 197**	**42 597**

Table 5.13: Run-times in ms for PSI protocols with unequal set sizes $n_0 \gg n_1$ in the LAN setting. σ: bit length of elements. "-" indicates that the execution ran out of memory.

The results are similar to the equal set size experiments with two notable exceptions. Firstly, the OPRF circuit performs extremely well and achieves a similar run-time as the server-aided protocol and even outperforms naive hashing for $n_0 = 2^{24}$ and $n_1 = 2^8$. This good performance of the OPRF circuit can be explained by the asymmetric costs for processing the sets of the client and server. While each element in the set of the client is encrypted by securely evaluating an AES circuit using generic secure computation techniques, the server only needs to encrypt each element in its set using AES with a fixed key and send $\ell = \lambda + \log_2(n_0) + \log_2(n_1)$ bits from the resulting ciphertext to the client. Since the set size of the client is small, the overhead for the generic secure computation techniques does not impact the overall run-time significantly.

Type	Client Set Size n_1	2^8		2^{12}	
	Server Set Size n_0	2^{16}	2^{20}	2^{16}	2^{20}
Limited Security	Naive Hashing	**59**	**1 066**	**179**	**1 139**
	Server-aided [KMRS14]	170	1 871	267	1 989
Public-Key	DH ECC [Mea86]	**156 068**	**2 451 092**	**158 159**	**2 486 141**
Circuit	Yao PWC (§5.3.2, $\sigma = 32$)	39 815	-	57 581	-
	GMW PWC (§5.3.2, $\sigma = 32$)	13 879	103 251	**16 534**	128 972
	Yao OPRF (§5.3.3)	6 636	7 947	64 418	67 284
	GMW OPRF (§5.3.3)	**5 730**	**7 545**	**31 653**	**33 593**
OT	OT + Hashing (§5.4.2)	**2 278**	**8 721**	**2 538**	**11 576**

Table 5.14: Run-times in ms for PSI protocols with unequal set sizes $n_0 \gg n_1$ in the WAN setting. σ: bit length of elements. "-" indicates that the execution ran out of memory.

Type	Client Set Size n_1	2^8			2^{12}		
	Server Set Size n_0	2^{16}	2^{20}	2^{24}	2^{16}	2^{20}	2^{24}
Limited Security	Naive Hashing	**0.5**	**9.0**	**144.0**	**0.6**	**9.0**	**160.0**
	Server-aided [KMRS14]	0.5	9.0	144.0	0.6	9.0	160.0
Public-Key	DH ECC [Mea86]	**2.3**	**30.0**	**592.0**	**2.6**	**37.5**	**592.3**
Circuit	Yao PWC (§5.3.2, $\sigma = 32$)	336.3	-	-	535.1	-	-
	GMW PWC (§5.3.2, $\sigma = 32$)	204.2	2 643.4	-	**333.5**	2 778.3	-
	Yao OPRF (§5.3.3)	**41.0**	**49.4**	**184.4**	646.7	**655.1**	**790.1**
	GMW OPRF (§5.3.3)	41.5	49.9	187.4	654.6	663.0	798.0
OT	OT + Hashing (§5.4.2)	**1.5**	**27.1**	**432.0**	**2.0**	**27.3**	**480.3**

Table 5.15: Concrete communication in MB for PSI with unequal set sizes $n_0 \gg n_1$. σ: bit-length of elements. "-" indicates that the execution ran out of memory.

The second exception is our OT-based PSI protocol, which achieves better communication than even the DH-based ECC protocol of [Mea86]. The low communication of our OT-based protocol is because the number of OTs, which dominate the communication for equal set sizes, is very low for unequal set sizes since it scales with the smaller set size n_1. The majority of the communication for the unequal set size experiments comes from the $\ell = \lambda + \log_2(kn_0) + \log_2(n_1) \approx 80$ bit masks that P_0 sends to P_1, which are smaller than the elliptic curve field size of 283 bit.

5.5.2.4 Multi-Threaded PSI

We evaluate the parallelizability of the best performing PSI protocol in each category by running up to four threads in parallel and depict the results in Table 5.16. We benchmark the FFC instantiation of the DH-based protocol instead of the ECC instantiation since the Miracl library does not allow for easy parallelization. Of special interest is the last column, which shows the ratio between the run-times with four threads and a single thread, for an input of 2^{20} elements.

The DH-based protocol, which is very simple and is easily parallelizable, achieves the best speedup of 2.8 as computation is the performance bottleneck. The GMW protocol achieves a speedup of about 1.9 at 2 threads already and, for 3 and 4 threads, does not decrease much due to the communication bottleneck. Finally, the naive hashing protocol and our OT-based PSI protocol both achieve a moderate speedup of 1.7 and 1.4, respectively, also due to the communication bottleneck.

Threads	**1**	**2**	**3**	**4**	**Speedup** $4 \mapsto 1$
Naive Hashing	0.665	0.494	0.398	0.385	1.73
DH-based FFC [Mea86]	1 418.772	961.123	659.895	509.990	**2.78**
GMW PWC (§5.3.2, $\sigma = 32$)	124.732	66.144	64.814	64.188	1.94
OT + Hashing (§5.4.2)	5.847	4.626	4.193	4.084	1.44

Table 5.16: Run-times in seconds for PSI on sets with sizes $n_0 = n_1 = 2^{20}$ using multiple threads.

5.5.2.5 PSI on Billion Element Sets

Finally, we demonstrate the scalability of our OT-based PSI protocol by evaluating it on sets of a billion $\sigma = 128$-bit elements each. For these sizes, the input elements require 15 GB of storage, which exceeds the main memory of our local servers. Instead, the servers store the elements and intermediate values on their respective solid state drive (SSD). We also benchmark the naive hashing protocol as a baseline for performance. We refrained from adding more main memory to process these sets, even though it is the most simple solution, since we are interested in the performance of the protocols if data needs to be stored on the SSD.

To compute the intersection between two sets of a billion elements, naive hashing requires 74 min, of which 19 min (26%) are spent on hashing and transferring data and 55 min (74%) are spent on computing the plaintext intersection. Our OT-based PSI protocol requires 34.2 hours in total, of which 30.0 hours (88%) are spent on simple hashing (Cuckoo hashing runs in parallel and requires 16.3 hours), 3.0 hours (9%) are spent on computing the OT routine, and 1.2 hours (4%) are spent on computing the plaintext intersection.

6 Conclusion

We conclude this thesis by summarizing our results (§6.1) and outlining directions for future work (§6.2).

6.1 Summary

This thesis addresses the research question *"Can OT extension enable more efficient secure two-party computation?"*. We answer this question in the positive by improving the efficiency of OT extension protocols and showing that OT extension can be used to enable more communication-efficient generic secure two-party computation and faster private set intersection. More detailed, our results can be summarized as follows:

Faster OT Extension (§3). We improve the efficiency of OT extension in the semi-honest and malicious model. We first optimize the 1-out-of-2 OT extension protocol of [IKNP03] and show that our optimized protocol achieves factor 2× faster run-time and fully utilizes a Gigabit network using only a single thread. We then optimize the 1-out-of-N OT extension protocol of [KK13] and show that it requires roughly factor 3× more computation complexity than our optimized 1-out-of-2 OT extension protocol but saves factor 2× communication for 1-out-of-2 OT on short strings.

Communication-Efficient Generic Secure Two-Party Computation (§4). We show that the communication often is the main bottleneck for generic secure computation protocols. We demonstrate that Yao's garbled circuits performs better than the GMW protocol for small sequential functionalities while the GMW protocol performs better for functionalities that are evaluated many times in parallel or if the bandwidth is low. We outline OT-based special-purpose protocols that improve the performance for various functions and combine them with generic secure protocols using our mixed-protocol framework called ABY, which results in overall faster protocols.

Faster Private Set Intersection (§5). Our evaluation shows that the first PSI protocol of [Mea86] outperforms many of the more recent solutions. Our OT-based PSI protocol improves the run-time compared to existing protocol by around two orders of magnitude on average and reduces the overhead compared to the insecure naive hashing protocol that is used in practice from factor 100× for the previous best protocol to factor 8×. In case the set sizes of the parties greatly differ, protocols based on generic secure computation techniques even achieve nearly the same performance as the insecure naive hashing protocol.

6.2 Future Work

The field of practical secure computation has advanced considerably in recent years due to many works that have steadily improved the performance. In fact, local computation has been improved by such an extent, that the communication has become the main bottleneck for many secure computation protocols. In the following, we give directions for future work on improving the practicality and real-world adoption of secure computation: Further *improving the communication and computation complexity* of protocols (§6.2.1), *reducing the overhead of a Boolean circuit* representation (§6.2.2), and extending our results to the *malicious adversary model* (§6.2.3).

6.2.1 Reducing Communication / Computation Complexity

Improving the communication and computation complexity of secure computation protocol is an important step that helps pushing secure computation closer to adoption in practice. An obvious way of improving the computation would be to optimize the implementations. For instance, matrix transposition, required in OT extension protocols (cf. §3.2.2), has been further improved using Intel AVX instructions [OOS17], while the JustGarble framework [BHKR13] uses a cache-efficient and highly specialized architecture that is tailored to Yao's garbled circuits and achieves nearly twice the local garbling speed of our Yao's garbled circuit implementation within our more generic mixed-protocol framework (8 million AND/s in the setup phase for JustGarble [BK15] vs. 4.6 million ANDs/s for our implementation, cf. §4.6.2).

Other than optimizing the implementation, the run-time of protocols can be improved using more efficient instantiations of cryptographic primitives. For instance, our optimized $\binom{N}{1}$ OT extension protocol from §3.2.4 has a factor 3× higher computation overhead than our optimized $\binom{2}{1}$ OT extension protocol from §3.2.3, which is due to the instantiation of the correlation-robust function (CRF) (cf. §2.2.4). More detailed, while the $\binom{2}{1}$ OT extension protocol allows to instantiate the CRF using fixed-key AES, the $\binom{N}{1}$ OT extension protocol needs to process values that are larger than the AES block-length and hence requires more expensive CRF instantiations (cf. §3.2.4.2). The Simpira family of cryptographic permutations [GM16] allows to process inputs of length $128b$, where $b \geq 1$ using only the AES round function and seems like a perfect fit to improve the computation complexity for $\binom{N}{1}$ OT extension.

In §4.4.2 we show that for some functionalities, the communication complexity can be decreased using a representation as lookup table. We introduced the SP-LUT protocol, which for lookup tables with ℓ input and o output bits, pre-computes a $\binom{2^\ell}{1}$ OT in the setup phase and sends $o2^\ell$ correction bits in the online phase. An interesting question for future work is whether it is possible to extend the multiplication triples (cf. §2.4.2) to arbitrary polynomials with multiple input bits that can be pre-computed using our improved $\binom{N}{1}$ OT extension protocol to achieve better communication in the online phase.

6.2.2 Boolean Circuit Representation

The Boolean circuit representation imposes a limitation on the local computation and scalability of state-of-the-art secure computation techniques. The limitation on the local computation is because the parties have to perform memory lookups and computation linear in the size of the Boolean circuit, i.e., for each gate separately. The scalability limitation of the Boolean circuit representation is particularly relevant for multi-round protocol such as GMW, where the parties have to store the full Boolean circuit in memory during one point of the computation to schedule the gates such that they are evaluated when the inputs become available [KSS13a, ZSB13].

In the following, we discuss two directions for future research that aim to cope with these problems: Reducing local computation time *using data locality optimizations* and improving scalability using a *modular circuit representation*. Note that, even though our SIMD optimization (cf. §4.2.2) can be used to achieve both goals, it only works for specific functions and does not generalize well to arbitrary functions.

Optimizing Data Locality. Since the cost for evaluating symmetric cryptographic operations for an AND gate has been drastically reduced [BHKR13, GLNP15], the memory lookup time for accessing individual gates is increasingly dominating the local computation time. Indeed, when representing a function as Boolean circuit, each gate requires a memory lookup, which seems like an inevitable overhead.

In order to reduce the cost for a memory lookup, one could decrease the cache misses using data locality optimized scheduling techniques (e.g., [WL91]). These techniques could allow to re-use gates that are held in cache by scheduling the gate evaluation such that connected gates are evaluated in direct sequence. Related ideas to this approach were presented but not evaluated in [SHS+15], which introduced a compact circuit format that reduced the number of cache misses as a side-effect.

Modular Circuit Representation. To achieve communication rounds linear in the multiplicative depth, the GMW protocol needs to schedule the gates in the circuit layer-wise. To perform this scheduling, the complete circuit needs to be stored in memory at one point during the computation [KSS13a, ZSB13]. This approach, however, does not scale well to large circuits with billions of gates.

To solve this scheduling problem, one could adopt a multi-stage circuit compilation process together with an in-time circuit substitution. The multi-stage circuit compilation process would allow high level operations, such as additions or multiplications, to be kept in high-level form that could be scheduled using existing automatic parallelization compiler techniques [BK15]. When the circuit is evaluated, the high-level operations could then be substituted by optimized circuit building blocks during runtime. This would allow to perform compilation and scheduling on a smaller and more high-level description of the circuit, without requiring the large memory footprint of a Boolean circuit representation.

6.2.3 Extension to Malicious Adversaries

The focus in this thesis has been on protocols that provide security against semi-honest adversaries. While this model has practical relevance (cf. §2.1.4), its main goal is to serve as a stepping stone for designing efficient protocols in the malicious adversary model. Although some of our optimizations directly translate to improvements for protocols in the malicious adversary model, e.g., our algorithmic improvements for OT extension in §3.2 and our circuit-based optimizations in §4.2, most of our protocols are only secure in the semi-honest model and are non-straight forward to extend. Recent work extended partial results of this thesis to the malicious model: [KOS16] extended our OT-based integer multiplication from §4.4.3 to malicious adversaries, [OOS17, PSS17] extended the $\binom{N}{1}$ OT extension protocol of [KK13] to malicious adversaries, and [Lam16] extended our OT-based PSI protocol from §5.4.2 to the setting of malicious client and semi-honest server.

We see it as open problem for future work to extend our N-MT protocol from §4.3.3, our SP-LUT protocol from §4.4.2.3, and our transformations for realizing mixed-protocols from §4.5.2 to the malicious model and to achieve full malicious security for our OT-based PSI protocol from §5.4.2.

Bibliography

[ABL+04] M. J. Atallah, M. Bykova, J. Li, K. B. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *Workshop on Privacy in the Electronic Society (WPES'04)*, pages 103–114. ACM, 2004.

[AFL+16] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM Computer and Communications Security (CCS'16)*, pages 805–817. ACM, 2016.

[AHI11] B. Applebaum, D. Harnik, and Y. Ishai. Semantic security under related-key attacks and applications. In *Innovations in Computer Science (ICS'10)*, pages 45–60. Tsinghua University Press, 2011.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Computer and Communications Security (CCS'13)*, pages 535–548. ACM, 2013. Code: http://encrypto.de/code/OTExtension.

[ALSZ15] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015. Full version: http://eprint.iacr.org/2015/061.

[ALSZ16] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions. In *Journal of Cryptology*. Springer, 2016. To appear. Online at http://eprint.iacr.org/2016/602.

[ANS10] Y. Arbitman, M. Naor, and G. Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *Foundations of Computer Science (FOCS'10)*, pages 787–796. IEEE, 2010.

[ARS+15] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[Bat68] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314. Thomson Book Company, 1968.

[BB94] M. L. Bonet and S. R. Buss. Size-depth tradeoffs for Boolean fomulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BB16] M. Blanton and F. Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *Proceedings on Privacy Enhancing Technologies (PoPETs'16)*, 2016(4):144–164, 2016.

[BBC+10] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti, and A. Piva. Privacy-preserving fingercode authentication. In *Multimedia and Security (MM&SEC'10)*, pages 231–240. ACM, 2010.

[BBC+11] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *ACM Computer and Communications Security (CCS'11)*, pages 691–702. ACM, 2011.

[BBG+16] J. A. Buchmann, N. Büscher, F. Göpfert, S. Katzenbeisser, J. Krämer, D. Micciancio, S. Siim, C. van Vredendaal, and M. Walter. Creating cryptographic challenges using multi-party computation: The LWE challenge. In *ASIA Public Key Cryptography (AsiaPKC@AsiaCCS'16)*, pages 11–20. ACM, 2016.

[BCP13] J. Bringer, H. Chabanne, and A. Patey. SHADE: secure hamming distance computation from oblivious transfer. In *Financial Cryptography and Data Security (FC'13)*, volume 7862 of *LNCS*, pages 164–176. Springer, 2013.

[BCP+14] J. Bringer, H. Chabanne, A. Patey, M. Favre, T. Schneider, and M. Zohner. GSHADE: Faster privacy-preserving distance computation and biometric identification. In *Workshop on Information Hiding and Multimedia Security (IH&MMSec'14)*, pages 187–198. ACM, 2014.

[Bea91] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

[Bea95] D. Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Symposium on the Theory of Computing (STOC'96)*, pages 479–488. ACM, 1996.

[BG85] R. W. Baldwin and W. C. Gramlich. Cryptographic protocol for trustable matchmaking. In *IEEE Symposium on Security and Privacy (S&P'85)*, pages 92–100. IEEE, 1985.

[BG11] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS'11*, volume 6879 of *LNCS*, pages 190–209. Springer, 2011.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC'88*, pages 1–10. ACM, 1988.

[BHKR13] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.

[BHLB11] E. Bursztein, M. Hamburg, J. Lagarenne, and D. Boneh. OpenConflict: Preventing real time map hacks in online games. In *IEEE Symposium on Security and Privacy (S&P'11)*, pages 506–520. IEEE, 2011.

[BHR12] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM Computer and Communications Security (CCS'12)*, pages 784–796. ACM, 2012.

[BHWK16] N. Büscher, A. Holzer, A. Weber, and S. Katzenbeisser. Compiling low depth circuits for practical secure computation. In *European Symposium on Research in Computer Security (ESORICS'08), Part II*, volume 9879 of *LNCS*, pages 80–98. Springer, 2016.

[BJH+11] I. Bilogrevic, M. Jadliwala, J.-P. Hubaux, I. Aad, and V. Niemi. Privacy-preserving activity scheduling on mobile devices. In *ACM Data and Application Security and Privacy (CODASPY'11)*, pages 261–272. ACM, 2011.

[BJL12] D. Bogdanov, R. Jagomägis, and S. Laur. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Pacific Asia Workshop on Intelligence and Security Informatics (PAISI'12)*, volume 7299 of *LNCS*, pages 112–126. Springer, 2012.

[BK15] N. Büscher and S. Katzenbeisser. Faster secure computation through automatic parallelization. In *USENIX Security Symposium 2015*, pages 531–546. USENIX, 2015.

[BKK+16] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste. Students and Taxes: a Privacy-Preserving Study Using Secure Computation. *PoPETs*, 2016(3):117–135, 2016.

[Bla06] J. Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *Fast Software Encryption (FSE'06)*, volume 4047 of *LNCS*, pages 328–340. Springer, 2006.

[BLN+15] S. S. Burra, E. Larraia, J. B. Nielsen, P. S. Nordholt, C. Orlandi, E. Orsini, P. Scholl, and N. P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. IACR Cryptology ePrint Archive, Report 2015/472, 2015.

[Blo70] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BLO16] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM Computer and Communications Security (CCS'16)*, pages 578–590. ACM, 2016.

[BLR13] D. Bogdanov, P. Laud, and J. Randmets. Domain-polymorphic language for privacy-preserving applications. In *PETShop@ACM CCS'13*, pages 23–26. ACM, 2013.

[BLW08] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security (ESORICS'08)*, volume 5283 of *LNCS*, pages 192–206. Springer, 2008.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Symposium on the Theory of Computing (STOC'90)*, pages 503–513. ACM, 1990.

[BNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Computer and Communications Security (CCS'08)*, pages 257–266. ACM, 2008.

[BoSV15] D. Bogdanov, M. J oemets, S. Siim, and 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)*, volume 8975 of *LNCS*, pages 227–234. Springer, 2015.

[BP06] J. Boyar and R. Peralta. Concrete multiplicative complexity of symmetric functions. In *Mathematical Foundations of Computer Science (MFCS'06)*, volume 4162 of *LNCS*, pages 179–189. Springer, 2006.

[BP10] J. Boyar and R. Peralta. A new combinational logic minimization technique with applications to cryptology. In *Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.

[BP12] J. Boyar and R. Peralta. A small depth-16 circuit for the AES S-Box. In *Information Security and Privacy Conference (SEC'12)*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Computer and Communications Security (CCS'93)*, pages 62–73. ACM, 1993.

[BSMD10] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium 2010*, pages 223–240. USENIX, 2010.

[CADT14] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. For your phone only: Custom protocols for efficient secure function evaluation on mobile devices. *Journal of Security and Communication Networks*, 7(7):1165–1176, 2014.

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

[CH10] O. Catrina and S. Hoogh. Improved primitives for secure multiparty integer computation. In *Security and Cryptography for Networks (SCN'10)*, volume 6280 of *LNCS*, pages 182–199. Springer, 2010.

[CHK+12] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of Boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers' Track at the RSA Conference (CT-RSA'12)*, volume 7178 of *LNCS*, pages 416–432. Springer, 2012.

[CKKZ12] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the "free-xor" technique. In *Theory of Cryptography Conference (TCC'12)*, volume 7194 of *LNCS*, pages 39–53. Springer, 2012.

[CMP11] E. De Cristofaro, M. Manulis, and B. Poettering. Private discovery of common social contacts. In *Applied Cryptography and Network Security (ACNS'11)*, volume 6715 of *LNCS*, pages 147–165. Springer, 2011.

[CO15] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology – LATINCRYPT'15*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[CS10] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 35–50. Springer, 2010.

[CT10] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC'10)*, volume 6052 of *LNCS*, pages 143–159. Springer, 2010.

[CT12] E. De Cristofaro and G. Tsudik. Experimenting with fast private set intersection. In *Trust and Trustworthy Computing (TRUST'12)*, volume 7344 of *LNCS*, pages 55–73. Springer, 2012.

[DCW13] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *ACM Computer and Communications Security (CCS'13)*, pages 789–800. ACM, 2013.

[DD15] S. K. Debnath and R. Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *Information Security Conference (ISC'15)*, volume 9290 of *LNCS*, pages 209–226. Springer, 2015.

[DDK+15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni. Automated synthesis of optimized circuits for secure computation. In *ACM Computer and Communications Security (CCS'15)*, pages 1504–1517. ACM, 2015.

[DFK+06] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference (TCC'06)*, volume 3876 of *LNCS*, pages 285–304. Springer, 2006.

[DGK08] I. Damgård, M. Geisler, and M. Krøigaard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.

[DGK09] I. Damgård, M. Geisler, and M. Krøigaard. A correction to 'Efficient and secure comparison for on-line auctions'. *International Journal of Applied Cryptography*, 1(4):323–324, 2009.

[DGKN09] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography (PKC'09)*, volume 5443 of *LNCS*, pages 160–179. Springer, 2009.

[DGM+10] M. Dietzfelbinger, A. Goerdt, M. Mitzenmacher, A. Montanari, R. Pagh, and M. Rink. Tight thresholds for cuckoo hashing via XORSAT. In *International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 6198 of *LNCS*, pages 213–225. Springer, 2010.

[DJ01] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography (PKC'01)*, volume 1992 of *LNCS*, pages 119–136. Springer, 2001.

[DJN10] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[DKL+13] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security (ESORICS'13)*, volume 8134 of *LNCS*, pages 1–18. Springer, 2013.

[DKS+17] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner. Pushing the communication barrier in secure computation using lookup tables. In *Network and Distributed System Security (NDSS'17)*. The Internet Society, 2017.

[DLT14] I. Damgård, R. Lauritsen, and T. Toft. An empirical study and some improvements of the MiniMac protocol for secure computation. In *Security and Cryptography for Networks (SCN'14)*, volume 8642 of *LNCS*, pages 398–415. Springer, 2014.

[DPSZ12] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security (NDSS'15)*. The Internet Society, 2015.

[DZ13] I. Damgård and S. Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In *Theory of Cryptography Conference (TCC'13)*, volume 7785 of *LNCS*, pages 621–641. Springer, 2013.

[DZ16] I. Damgård and R. W. Zakarias. Fast oblivious AES: A dedicated application of the MiniMac protocol. In *Progress in Cryptology – AFRICACRYPT'16*, volume 9646 of *LNCS*, pages 245–264. Springer, 2016.

[EFG+09] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies Symposium (PETS'09)*, volume 5672 of *LNCS*, pages 235–253. Springer, 2009.

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

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[Ekl72] J. O. Eklundh. A fast computer method for matrix transposing. In *IEEE Transactions on Computers*, volume C-21(7), pages 801–803. IEEE, 1972.

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

[FIPR05] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC'05)*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[FJN+13] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. Sebastian Nordholt, and C. Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In *Advances in Cryptology – EUROCRYPT'13*, volume 7881 of *LNCS*, pages 537–556. Springer, 2013.

[FJN14] T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen. Faster maliciously secure two-party computation using the GPU. In *Security and Cryptography for Networks (SCN'14)*, volume 8642 of *LNCS*, pages 358–379. Springer, 2014.

[FJNT15] T. K. Frederiksen, T. P. Jakobsen, J. Buus Nielsen, and R. Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. IACR Cryptology ePrint Archive, Report 2015/309, 2015.

[FJNT16] T. Kasper Frederiksen, T. P. Jakobsen, J. B. Nielsen, and R. Trifiletti. On the complexity of additively homomorphic UC commitments. In *Theory of Cryptography Conference (TCC'16)*, volume 9562 of *LNCS*, pages 542–565. Springer, 2016.

[FKOS15] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to MPC with preprocessing using OT. In *Advances in Cryptology – ASIACRYPT'15*, volume 9452 of *LNCS*, pages 711–735. Springer, 2015.

[FN13] T. K. Frederiksen and J. B. Nielsen. Fast and maliciously secure two-party computation using the GPU. In *Applied Cryptography and Network Security (ACNS'13)*, volume 7954 of *LNCS*, pages 339–356. Springer, 2013.

[FNP04] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT'04*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004.

[FPS+11] M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, and I. Visconti. Secure set intersection with untrusted hardware tokens. In *Cryptographers' Track at the RSA Conference (CT-RSA'11)*, volume 6558 of *LNCS*, pages 1–16. Springer, 2011.

[Gil99] N. Gilboa. Two party RSA key generation. In *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, 1999.

[GKK+12] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Computer and Communications Security (CCS'12)*, pages 513–524. ACM, 2012.

[GLMY16] A. Groce, A. Ledger, A. J. Malozemoff, and A. Yerukhimovich. CompGC: Efficient offline/online semi-honest two-party computation. IACR Cryptology ePrint Archive, Report 2016/458, 2016.

[GLNP15] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In *ACM Computer and Communications Security (CCS'15)*, pages 567–578. ACM, 2015.

[GM16] S. Gueron and N. Mouha. Simpira v2: A family of efficient permutations using the AES round function. In *Advances in Cryptology – ASIACRYPT'16*, volume 10031 of *LNCS*, pages 95–125. Springer, 2016.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[Gol01] O. Goldreich. Cryptography and cryptographic protocols, 2001. Online: http://www.wisdom.weizmann.ac.il/~oded/foc-sur01.html.

[Gol04] O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

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

[GSV07] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography (PKC'07)*, volume 4450 of *LNCS*, pages 330–342. Springer, 2007.

[Gue12] S. Gueron. Intel advanced encryption standard (AES) instructions set, Rev 3.01. Technical report, Intel, 2012.

[HCE11] Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *Hot topics in security (HotSec'11)*. USENIX, 2011.

[HEK12] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security (NDSS'12)*. The Internet Society, 2012.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium 2011*, pages 539–554. USENIX, 2011.

[HFH99] B. A. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce (EC'99)*, pages 78–86. ACM, 1999.

[HFKV12] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *ACM Computer and Communications Security (CCS'12)*, pages 772–783. ACM, 2012.

[HIKN08] D. Harnik, Y. Ishai, E. Kushilevitz, and J. Buus Nielsen. OT-combiners via secure computation. In *Theory of Cryptography Conference (TCC'08)*, volume 4948 of *LNCS*, pages 393–411. Springer, 2008.

[HKE12] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *Symposium on Security and Privacy (S&P'12)*, pages 272–284. IEEE, 2012.

[HKK+14] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. J. Malozemoff. Amortizing garbled circuits. In *Advances in Cryptology – CRYPTO'14*, volume 8617 of *LNCS*, pages 458–475. Springer, 2014.

[HKS+10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *ACM Computer and Communications Security (CCS'10)*, pages 451–462. ACM, 2010.

[HL08] C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM Computer and Communications Security (CCS'08)*, pages 491–500. ACM, 2008.

[HL10] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer, 1st edition, 2010.

[HMEK11] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed Security Symposium (NDSS'11)*. The Internet Society, 2011.

[HS13] W. Henecka and T. Schneider. Faster secure two-party computation with less memory. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, pages 437–446. ACM, 2013.

[Hua12] Y. Huang. Practical secure two-party computation. Ph.D. Thesis, 2012. Online: https://yhuangpress.files.wordpress.com/2014/02/dissertation.pdf.

[IKM+13] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Theory of Cryptography Conference (TCC'13)*, volume 7785 of *LNCS*, pages 600–620. Springer, 2013.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[IKOS08] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *ACM Symposium on Theory of Computing (STOC'08)*, pages 433–442. ACM, 2008.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.

[IR88] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Advances in Cryptology – (CRYPTO'88)*, volume 403 of *LNCS*, pages 8–26. Springer, 1988.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *ACM Symposium on Theory of Computing (STOC'89)*, pages 44–61. ACM, 1989.

[Ker11] F. Kerschbaum. Automatically optimizing secure computation. In *ACM Computer and Communications Security (CCS'11)*, pages 703–714. ACM, 2011.

[KK13] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology – CRYPTO'13*, volume 8043 of *LNCS*, pages 54–70. Springer, 2013.

[KKRT16] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM Computer and Communications Security (CCS'16)*, pages 818–829. ACM, 2016.

[KMR14] V. Kolesnikov, P. Mohassel, and M. Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In *Advances in Cryptology – CRYPTO'14, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, 2014.

[KMRR15] V. Kolesnikov, P. Mohassel, B. Riva, and M. Rosulek. Richer efficiency/security trade-offs in 2pc. In *Theory of Cryptography Conference (TCC'15)*, volume 9014 of *LNCS*, pages 229–259. Springer, 2015.

[KMRS14] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *Financial Cryptography and Data Security (FC'14)*, volume 8437 of *LNCS*, pages 195–215. Springer, 2014.

[KMSB13] B. Kreuter, B. Mood, A. Shelat, and K. Butler. PCF: a portable circuit format for scalable two-party secure computation. In *USENIX Security Symposium 2013*, pages 321–336. USENIX, 2013.

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

[KO62] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[KOS15] M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO'15*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[KOS16] M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Computer and Communications Security (CCS'16)*, pages 830–842. ACM, 2016.

[KS05] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO'05*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[KS16] Á. Kiss and T. Schneider. Valiant's universal circuit is practical. In *Advances in Cryptology – EUROCRYPT'16, Part I*, volume 9665 of *LNCS*, pages 699–728. Springer, 2016.

[KSS09] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology And Network Security (CANS'09)*, volume 5888 of *LNCS*, pages 1–20. Springer, 2009.

[KSS12] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium 2012*, pages 285–300. USENIX, 2012.

[KSS13a] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *ACM Computer and Communications Security (CCS'13)*, pages 549–560. ACM, 2013.

[KSS13b] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. *Journal of Computer Security*, 21(2):283–315, 2013.

[KSS14] F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic protocol selection in secure two-party computations. In *Applied Cryptography and Network Security (ACNS'14)*, volume 8479 of *LNCS*, pages 566–584. Springer, 2014. Extended abstract published in NDSS'13.

[Lam16] M. Lambæk. Breaking and fixing private set intersection protocols. IACR Cryptology ePrint Archive, Report 2016/665, 2016.

[Lar14] E. Larraia. Extending oblivious transfer efficiently, or - how to get active security with constant cryptographic overhead. In *Progress in Cryptology – LATINCRYPT'14*, volume 8895 of *LNCS*, pages 368–386. Springer, 2014.

[LF80] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, 1980.

[Lin13] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology – CRYPTO'12, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, 2013.

[LLXX05] B. Li, H. Li, G. Xu, and H. Xu. Efficient reduction of 1 out of n oblivious transfers in random oracle model. IACR Cryptology ePrint Archive, Report 2005/279, 2005.

[LOS14] E. Larraia, E. Orsini, and N. P. Smart. Dishonest majority multi-party computation for binary circuits. In *Advances in Cryptology – CRYPTO'14*, volume 8617 of *LNCS*, pages 495–512. Springer, 2014.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986. Also published as Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam.

[LP04] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity (ECCC'04)*, 063, 2004.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[LP09] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LPS08] Y. Lindell, B. Pinkas, and N. P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks (SCN'08)*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[LPSY15] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology – CRYPTO'12, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, 2015.

[LR14] Y. Lindell and B. Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In *Advances in Cryptology – CRYPTO'14*, volume 8617 of *LNCS*, pages 476–494. Springer, 2014.

[LR15a] P. Laud and J. Randmets. A domain-specific language for low-level secure multiparty computation protocols. In *ACM Computer and Communications Security (CCS'15)*, pages 1492–1503. ACM, 2015.

[LR15b] Y. Lindell and B. Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *ACM Computer and Communications Security (CCS'15)*, pages 579–590. ACM, 2015.

[LWN+15] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. Oblivm: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy (S&P'15)*, pages 359–376. IEEE, 2015.

[LZ13] Y. Lindell and H. Zarosim. On the feasibility of extending oblivious transfer. In *Theory of Cryptography Conference (TCC'13)*, volume 7785 of *LNCS*, pages 519–538. Springer, 2013.

[Mal11] L. Malka. VMCrypt - modular software architecture for scalable secure computation. In *ACM Computer and Communications Security (CCS'11)*, pages 715–724. ACM, 2011.

[MBF+09] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia. Privacy-aware proximity based services. In *Mobile Data Management (MDM'09)*, pages 31–40. IEEE, 2009.

[Mea86] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE Symposium on Security and Privacy (S&P'86)*, pages 134–137. IEEE, 1986.

[MF06] P. Mohassel and M. K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography (PKC'07)*, volume 3958 of *LNCS*, pages 458–473. Springer, 2006.

[MGC+16] B. Mood, D. Gupta, H. Carter, K. R. B. Butler, and 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)*, pages 112–127. IEEE, 2016.

[Mit01] M. D. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[MLB12] B. Mood, L. Letaw, and K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *Financial Cryptography and Data Security (FC'12)*, volume 7397 of *LNCS*, pages 254–268. Springer, 2012.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security Symposium 2004*, pages 287–302. USENIX, 2004.

[MPGP09] G. Mezzour, A. Perrig, V. D. Gligor, and P. Papadimitratos. Privacy-preserving relationship path discovery in social networks. In *Cryptology and Network Security (CANS'09)*, volume 5888 of *LNCS*, pages 189–208. Springer, 2009.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[MS13] P. Mohassel and S. S. Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology – EUROCRYPT'13*, volume 7881 of *LNCS*, pages 557–574. Springer, 2013.

[MZ06] Robert H Morelos-Zaragoza. *The art of error correcting coding*. John Wiley & Sons, 2006. Code generation tools online at http://eccpage.com.

[NCD+13] M. Nagy, E. De Cristofaro, A. Dmitrienko, N. Asokan, and A.-R. Sadeghi. Do I know you? – efficient and privacy-preserving common friend-finder protocols and applications. In *Annual Computer Security Applications Conference (ACSAC'13)*, pages 159–168. ACM, 2013.

[Nie07] J. B. Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. IACR Cryptology ePrint Archive, Report 2007/215, 2007.

[NIS15] NIST. Nist special publication 800 -90a rev. 1. Technical report, NIST, 2015. Online: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

[NMH+10] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: Finding P2P bots with structured graph analysis. In *USENIX Security Symposium 2010*, pages 95–110. USENIX, 2010.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[NO09] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *Theory of Cryptography Conference (TCC'09)*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Symposium on Discrete Algorithms (SODA'01)*, pages 448–457. ACM/SIAM, 2001.

[NP05] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, pages 129–139. ACM, 1999.

[NST17] J. B. Nielsen, T. Schneider, and R. Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *Network and Distributed System Security (NDSS'17)*. The Internet Society, 2017.

[NTL+11] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *Network and Distributed Security Symposium (NDSS'11)*. The Internet Society, 2011.

[OOS17] M. Orrù, E. Orsini, and P. Scholl. Actively secure 1-out-of-n OT extension with application to private set intersection. In *Cryptographers' Track at the RSA Conference (CT-RSA'17)*, volume 10159 of *LNCS*, pages 381–396, 2017.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[PBS12] P. Pullonen, D. Bogdanov, and T. Schneider. The design and implementation of a two-party protocol suite for SHAREMIND 3. Technical report, CYBERNETICA Institute of Information Security, 2012. T-4-17.

[PKUM16] E. Pattuk, M. Kantarcioglu, H. Ulusoy, and B. Malin. CheapSMC: A framework to minimize secure multiparty computation cost in the cloud. In *Data and Applications Security and Privacy (DBSec'16)*, volume 9766 of *LNCS*, pages 285–294. Springer, 2016.

[PR01] R. Pagh and F. F. Rodler. Cuckoo hashing. In *European Symposium on Algorithms (ESA'01)*, volume 2161 of *LNCS*, pages 121–133. Springer, 2001.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[PSS17] A. Patra, P. Sarkar, and A. Suresh. Fast actively secure OT extension for short secrets. In *Network and Distributed System Security (NDSS'17)*. The Internet Society, 2017.

[PSSW09] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT'09*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[PSSZ15] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium 2015*, pages 515–530. USENIX, 2015. Full version: http://eprint.iacr.org/2015/634.

[PSZ14] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security Symposium 2014*, pages 797–812. USENIX, 2014.

[PSZ16] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension, 2016. In Submission. Online at https://eprint.iacr.org/2016/930.

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.

[Rab81] M. O. Rabin. *How to exchange secrets with oblivious transfer*, TR-81 edition, 1981. Aiken Computation Lab, Harvard University.

[RHH14] A. Rastogi, M. A. Hammer, and M. Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *IEEE Symposium on Security and Privacy (S&P'14)*, pages 655–670. IEEE, 2014.

[RR16] P. Rindal and M. Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security Symposium 2016*. USENIX, 2016.

[RS98] M. Raab and A. Steger. "balls into bins" - a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science (RANDOM'98)*, volume 1518 of *LNCS*, pages 159–170. Springer, 1998.

[Sav97] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Pub, Boston, MA, USA, 1st edition, 1997.

[Sha80] A. Shamir. On the power of commutativity in cryptography. In *International Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *LNCS*, pages 582–595. Springer, 1980.

[SHS+15] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *IEEE Symposium on Security and Privacy (S&P'15)*, pages 411–428. IEEE, 2015.

[SK11] A. Schröpfer and F. Kerschbaum. Demo: secure computation in JavaScript. In *ACM Computer and Communications Security (CCS'11)*, pages 849–852. ACM, 2011.

[Skl60] J. Sklansky. An evaluation of several two-summand binary adders. *IRE Transactions on Electronic Computers*, EC-9(2):213–226, 1960.

[SKM11] A. Schröpfer, F. Kerschbaum, and G. Müller. L1 - an intermediate language for mixed-protocol secure computation. In *IEEE Computer Software and Applications Conference (COMPSAC'11)*, pages 298–307. IEEE, 2011.

[SLPR15] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 213–226. ACM, 2015.

[SS06] R. Schürer and W. Schmid. *Monte Carlo and Quasi-Monte Carlo Methods 2004*, chapter MinT: A Database for Optimal Net Parameters, pages 457–469. Springer, 2006. Online: http://mint.sbg.ac.at.

[SS13] A. Shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In *ACM Computer and Communications Security (CCS'13)*, pages 523–534. ACM, 2013.

[ST06] B. Schoenmakers and P. Tuyls. Efficient binary conversion for Paillier encrypted values. In *Advances in Cryptology – EUROCRYPT'06*, volume 4004 of *LNCS*, pages 522–537. Springer, 2006.

[SZ13] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

[TP14] M. S. Turan and R. Peralta. The multiplicative complexity of Boolean functions on four and five variables. In *Lightweight Cryptography for Security and Privacy (LightSec'14)*, volume 8898 of *LNCS*, pages 21–33. Springer, 2014.

[Wak68] A. Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, 1968.

[WFNL16] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies (PoPETs'16)*, 2016(4):335–355, 2016.

[WL91] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *ACM Programming Language Design and Implementation (PLDI'91)*, pages 30–44. ACM, 1991.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

[YSG95] J.-T. Yoo, K. F. Smith, and G. Gopalakrishnan. A fast parallel squarer based on divide-and-conquer. *IEEE Journal of Solid-State Circuits*, 32:909–912, 1995.

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

[ZSB13] Y. Zhang, A. Steele, and M. Blanton. PICCO: a general-purpose compiler for private distributed computation. In *ACM Computer and Communications Security (CCS'13)*, pages 813–826. ACM, 2013.

List of Figures

List of Tables

List of Protocols

List of Publications

Peer Reviewed Conference Publications

[1] Gilad Asharov, Daniel Demmler, Michael Schapira, Thomas Schneider, Gil Segev, Scott Shenker, and Michael Zohner. Privacy-Preserving Interdomain Routing at Internet Scale. To appear in *Proceedings on Privacy Enhancing Technologies*, Volume 2017.

[2] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *Network and Distributed System Security Symposium (NDSS'17)*. The Internet Society, 2017. **Part of this thesis**.

[3] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security Symposium 2015*, pages 515-530. USENIX, 2015. **Part of this thesis**.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *Advances in Cryptology – EUROCRYPT 2015 Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 673-701. Springer, 2015. **Part of this thesis**.

[5] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT 2015 Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430-454. Springer, 2015.

[6] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015. **Part of this thesis**.

[7] Daniel Demmler, Thomas Schneider, and Michael Zohner, Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens. In *USENIX Security Symposium 2014*, pages 893-908. USENIX, 2014.

[8] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster Private Set Intersection Based on OT Extension. In *USENIX Security Symposium 2014*, pages 797-812. USENIX, 2014. **Part of this thesis**.

[9] Julien Bringer, Hervé Chabanne, Mélanie Favre, Alain Patey, Thomas Schneider, and Michael Zohner. GSHADE: Faster Privacy-Preserving Distance Computation and Biometric Identification. In *Information Hiding and Multimedia Security Workshop (IH&MMSec'14)*, pages 187-198. ACM, 2014.

[10] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *ACM Conference on Computer and Communications Security (CCS'13)*, pages 535-548. ACM, 2013. **Part of this thesis**.

[11] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *Lecture Notes in Computer Science*, pages 275-292. Springer, 2013. **Part of this thesis**.

[12] Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, Michael Walter, and Johannes A. Buchmann. Improved Algebraic Side-Channel Attack on AES. In *Symposium on Hardware-Oriented Security and Trust (HOST'12)*, pages 146-151. IEEE, 2012.

[13] Michael Zohner, Marc Stöttinger, Sorin A. Huss, and Oliver Stein. An Adaptable, Modular, and Autonomous Side-Channel Vulnerability Evaluator. In *Symposium on Hardware-Oriented Security and Trust (HOST'12)*, pages 43-48. IEEE, 2012.

[14] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In *Constructive Side-Channel Analysis and Secure Design (COSADE'12)*, volume 7275 of *Lecture Notes in Computer Science*, pages 249-264. Springer, 2012.

[15] Michael Zohner, Michael Kasper, and Marc Stöttinger. Butterfly-Attack on Skein's Modular Addition. In *Constructive Side-Channel Analysis and Secure Design (COSADE'12)*, volume 7275 of *Lecture Notes in Computer Science*, pages 215-230. Springer, 2012.

[16] Michael Zohner, Michael Kasper, Marc Stöttinger, and Sorin A. Huss. Side Channel Analysis of the SHA-3 Finalists. In *Design Automation & Test in Europe (DATE'12)*, pages 1012-1017. IEEE, 2012.

Peer Reviewed Journal Publications

[1] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions. To be published in *Journal of Cryptology*. **Part of this thesis**.

[2] Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, Michael Walter, and Johannes A. Buchmann. Improved Algebraic Side-Channel Attack on AES. In *Journal of Cryptographic Engineering* volume 3 number 3 of *Lecture Notes in Computer Science*, pages 139-156. Springer, 2013.

In Submission

[1] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable Private Set Intersection Based on OT Extension. Journal submission. **Part of this thesis**.

Other Publications

[1] Josep Balasch, Oscar Reparaz, Ingrid Verbauwhede, Sorin A. Huss, Kai Rhode, Marc Stöttinger, and Michael Zohner. Chapter 8: Side-Channel Attacks. In *Circuits and Systems for Security and Privacy*, pages 287-327. CRC Press, 2016.

[2] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. AMASIVE: An Adaptable and Modular Autonomous Side-Channel Vulnerability Evaluation Framework. In *Number Theory and Cryptography 2013*, volume 8260 of *Lecture Notes in Computer Science*, pages 151-165. Springer, 2013.

[3] Michael Zohner, Michael Kasper, and Marc Stöttinger. Side Channel Analysis of the SHA-3 Finalists. In *3rd SHA-3 Candidate Conference 2012*. NIST, 2012.