UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Arbeitsgruppe Codes und Kryptographie

Title

Bachelor's Thesis

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science

by
Niklas Isermann

submitted to:
Prof. Dr. Johannes Blömer
and
???

Paderborn, September 9, 2022

Eidesstattliche Versicherung

Nachname: ______________________________ Vorname: ______________________________

Matrikelnr.: _____________ Studiengang: ____

☐ Bachelorarbeit ☐ Masterarbeit

Titel der Arbeit: Title

☐ Die elektronische Fassung ist der Abschlussarbeit beigefügt.

☐ Die elektronische Fassung sende ich an die/den erste/n Prüfenden bzw. habe ich an die/den erste/n Prüfenden gesendet.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit (Ausarbeitung inkl. Tabellen, Zeichnungen, etc.) selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Abschlussarbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die elektronische Fassung entspricht der gedruckten und gebundenen Fassung.

Belehrung

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist die Vizepräsidentin / der Vizepräsident für Wirtschafts- und Personalverwaltung der Universität Paderborn. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz NRW in der aktuellen Fassung).

Die Universität Paderborn wird ggf. eine elektronische Überprüfung der Abschlussarbeit durchführen, um eine Täuschung festzustellen.

Ich habe die oben genannten Belehrungen gelesen und verstanden und bestätige dieses mit meiner Unterschrift.

Ort: ___________________________________ Datum: _____________

Unterschrift: ____

Datenschutzhinweis

Die o.g. Daten werden aufgrund der geltenden Prüfungsordnung (Paragraph zur Abschlussarbeit) i.V.m. § 63 Abs. 5 Hochschulgesetz NRW erhoben. Auf Grundlage der übermittelten Daten (Name, Vorname, Matrikelnummer, Studiengang, Art und Thema der Abschlussarbeit) wird bei Plagiaten bzw. Täuschung der/die Prüfende und der Prüfungsausschuss Ihres Studienganges über Konsequenzen gemäß Prüfungsordnung i.V.m. Hochschulgesetz NRW entscheiden. Die Daten werden nach Abschluss des Prüfungsverfahrens gelöscht. Eine Weiterleitung der Daten kann an die/den Prüfende/n und den Prüfungsausschuss erfolgen. Falls der Prüfungsausschuss entscheidet, eine Geldbuße zu verhängen, werden die Daten an die Vizepräsidentin für Wirtschafts- und Personalverwaltung weitergeleitet. Verantwortlich für die Verarbeitung im regulären Verfahren ist der Prüfungsausschuss Ihres Studiengangs der Universität Paderborn, für die Verfolgung und Ahndung der Geldbuße ist die Vizepräsidentin für Wirtschafts- und Personalverwaltung.

Contents

1 Abstract

2 Introduction

2.1 MPC and Databases

A famous problem in the context of MPC is Yao's millionaire's problem. In Yao's millionaire's problem there are two millionaires Alice and Bob. We will call Alice's wealth x and Bob's wealth y. Alice and Bob want to know who of them is has more money. i.e. they want to compute the function F(x,y) := $\begin{cases} Alice\, is\, richer & y \leq x \\ Bob\, is\, richer & y > x \end{cases}$. Yet neither of them is willing to trust the other and tell him how much money he has. Yao's millionaire's problem can be generalised into the general MPC problem. Instead of Bob and Alice, we now consider n parties $p_0, \ldots, p_{n-1}$ and each party i holds an arbitrary input x_i for an arbitrary function $\mathrm{F}(x_0, \ldots, x_{n-1})$, that all parties have agreed upon. A MPC protocol π is protocol, that allows $p_0, \ldots, p_{n-1}$ to compute $\mathrm{F}(x_0, \ldots, x_{n-1})$ without revelling any information about $x_0, \ldots, x_{n-1}$.

Andrew Yao proposed a solution for Yao's millionaire's problem in 1982 []. It has also been shown that MPC is Turing-complete[]. This means that for any function f that can be computed with a Turing machine. There exists a MPC protocol π that can compute f. -Databases ...

2.2 related work

2.2.1 From Keys to Databases—Real-World Applications of Secure Multi-Party Computation

2.3 goals

In this section we describe the goals of our work.

2.4 structure

In this section we outline the structure this document.

3 Preliminary

3.1 Secure Multiparty Computation

In the an secure multiparty computation(short MPC) scenario there are n parties $p_0, \ldots, p_{n-1}$. They want to compute an agreed upon functionality $\mathrm{F}(x_0, \ldots, x_{n-1})$. A functionality is a function that is allowed to have internal randomness, so its not function in the strict mathematically sense of the word. Each party p_i holds an input value x_i. The parties hold their input private and do not want to reveal any information about it. The Goal of secure multiparty computation is to develop a protocol π that enables them to jointly compute $\mathrm{F}(x_0, \ldots, x_{n-1}.)$. The security goal of "not revelling the inputs" is often formalised through the Real-/Ideal-World Paradigm.

3.1.1 Real World and Ideal World

When modelling security of secure multiparty computation we compare the real world, to a perfect ideal world, where the problem can be solved in a perfect way.

Real World In the real world there exists a protocol π that enables the parties to compute F. All parties execute the protocol together. During the execution they exchange several rounds of communication. The attacker or adversary has the ability to corrupt one or more of the parties. His capability to influence the corrupted parties is an important parameter and may differ based on different security assumptions. These may range for example from a relative weak adversary that can only read massages to a very powerful adversary. We explain the adversary models that are of importance for our benchmarks in Section 3.2 in detail. The real world view of party A consist of the input of A, all massages A sends or receives during the execution of the protocol and his internal randomness. The protocol achieves the security goal of confidentially if the attacker is unable to derive new information from the views of the parties he did corrupt. ...

Ideal World In the Ideal World the parties to not need run a protocol. Instead they can rely on a trusted, incorruptible third party P that aids them. With the aid of P, the parties can evaluate F in two simple steps. In a first round of communication every party send P its input. P now holds all information it needs to compute F. Afterwards P can send each party the result in a second round of communication. Like in the real world, in the ideal world their also exists an adversary. Similar to his real world counterparty he is also able to corrupt one or more parties. Compared to his real world

Schaubild einfügen ?

counter part the ideal world adversary has otherwise very limited ability's. He can only see the input and output of the parties he corrupts. Since the computation with the aid of P produces no intermediate results that he can observe. Depending on the underlying security assumptions he also may be able to modify the input a corrupted party sends to P in the first round of communication. Because of these very limited ability's it is desired that for every adversary in the real world their exits a comparable powerful adversary in the ideal world. This is often formalised using simulation based proof.

Security Given an real-world adversary A , a secure multiparty protocol π , and a functionality F for π to be secure we require the existence of a so called simulator S. S is an ideal world adversary for F that mirror's the behaviour of A. This means that S and A corrupt the same parties and also that if a A changes the output of F then S does the same. After S has performed its attack S outputs a real world view. π is secure against A, if the view S outputs is indistinguishable from a view of A. This means that the real world attacker A cannot learn more then the ideal world attacker S. Despite the very limited ability's S has compared to A. Finally we say π is secure against if, for all A π is secure against A.

3.2 Adversarial Models

There are multiple models and categorizations of adversary's and their capability's. These distinctions have significant impact on feasibility and difficulty of secure multiparty computation. In the following we will outline the models and assumptions that are of importance for our benchmarks.

Passive Adversary vs Active Adversary A passive adversary can not force a corrupted party to deviate from the protocol in an any way. One could think of a passive adversary as a "read-only" adversary. As a passive adversary is only able to read the messages his corrupted parties receive or send. A active adversary can do everything a passive adversary can additionally he has the power to force a corrupted party to deviate from the protocol in an arbitrary way. So if for example the protocol would at some point require that each party choses an integer between 1 and n uniformly at random. Then a passive adversary would have no choice but to choose the integer between 1 and n uniformly at random. On the contrary an active adversary would be able to force a corrupted party to chose the value 42 or any other value that the adversary considers to be advantageous for him. In the ideal world a passive adversary is bound to forward the real input values. A active adversary can choose to ignore the real input and forward any value instead.

Monolithic Adversary In the following we will assume a monolithic adversary unless explicitly stated otherwise. This means that there is only a single adversary that controls all corrupt parties. For the honest parties a monolithic adversary is a worst-case scenario. A monolithic adversary is more powerful compared to multiple adversary's that control the same total amount of parties but to not corporate with each other. A protocol that is secure in the presence of a single adversary that corrupts n parties and is able to coordinate their efforts. Will be secure in the presence of up to n adversary's that corrupt n parties total and do not coordinate their efforts.

General Adversary vs Threshold Adversary In the threshold MPC setting the adversary can choose to corrupt any party. The threshold adversary is only limited in the way that can at most corrupt t parties where t is set to be 0 < t < n. A common setting for t is t = $\lfloor \frac{n}{2} \rfloor$, which called the honest majority. For example and for n=3 the presence of an honest majority means, that it is assumed that the threshold adversary can corrupt at most 1 party. Threshold MPC best fits scenarios that feature a very homogenous group of parties. A general adversary is limited in his choice which party he corrupts by an adversary structure $Z = \{Z_1, \ldots, Z_l\}$. Where Z_i can be any set of parties. The general adversary must corrupt a set of parties P such that there exists an $x \in Z$ that holds $P \subset x$. This allows for a flexible way to formalise assumptions. If for example in protocol there are two parties that hold a very vital role and one want to assume that no adversary can corrupt both of these parties. That can be formalised by using a general adversary an defining Z so that no element in Z contains both of these two parties.

Static vs Dynamic Corruptions Another important distinction is the distinction between static and adaptive adversary's. A static adversary is bound to chose which parties he wants to corrupt before the execution of π starts. An adaptive adversary can corrupt a party during the execution of the protocol. This makes the adaptive adversary much more powerful. As he can try to identify "weak links" based on the information he gets during the execution of the protocol and then choose corrupt those.

3.2.1 Additional Properties

binary secret sharing ???

garbled circuits ???

3.3 Databases

coming soon

4 framework description

4.1 Conclave

functionality Conclave allows to perform MPC analytics on "big data". Conclave aims provide a high-level interface that abstracts internal MPC details away from the user. Through this high abstraction level conclave aims to make MPC more accessible for those who are not experts in this field. Every operation done with conclave is composable, that means that the output of every query can be the input of another query. This mechanism makes it possible to construct very complex queries out of multiple relative simple queries. With conclave one can join tables using the equivalent of an equi-join or an union operator. Conclave also supports a range of aggregate functions these include sum, mean, standard deviation.

underlying MPC technology Conclave utilises existing MPC frameworks for its backend to perform its underlying MPC opertaions. Therefore Conclave inherits most security guarantees and assumptions from these frameworks. The concrete frameworks of use are Sharemind and Obliv-C. As both of these frameworks are designed to withstand passive adversary's and do not support more then 3 parties. Conclave also assumes a passive adversary and supports up to 3 parties. Since Obliv-C is based on garbled circuits and Sharemind on secret sharing, Conclave uses both. Conclave interacts with its backend through a generic interface. Therefore it is theoretically feasible to integrate another framework to add support for more then 3 parties. Conclave assumes a threshold adversary that corrupts statically and is bound by an honest majority.

4.1.1 Optimizations

MPC techniques are multiple orders of magnitude slower then cleartext processing. Its conclave key principle to archive better performance by avoiding the use of MPC techniques where possible. Instead of exclusively using MPC operations, conclave evaluates queries with a combination of local cleartext processing and MPC operations. When Conclave compiles a query it applies various optimizations to it, one such optimization is conclave's query rewriting

Query Rewriting - moving operations outside of MPC to maximise performance
- maintaining same end-to-end security as "pure" MPC
- contrary to conventional sql operations that aims to minimize the total amount of work e.g. filters before join

Trust Annotations Conclave features optional trust annotations that allow for trade-off between security and performance. With these trust annotations one party can annotate that it does trust another party to learn the values of a specific column. There exists a variety of use-cases that fit these mechanism. For example, the sensitivity of data may largely differ between columns. Therefore it may be desirable , for a party to reveal some less sensible data in order to speed up the computation. If a party decides to do so , Conclave uses these annotations to apply optimisations, that speed up query evaluation. One such optimization are conclaves hybrid operators.

Hybrid Operations When possible Conclave substitutes expansive MPC operations with cheaper hybrid operations. In a hybrid operation one party is "promoted" to a selectively-trusted party (short STP). Conclave reveals some input columns to the STP. Such leakage is only possible if the parties did explicitly allow it with the trust annotations. Otherwise it is not possible to apply hybrid operations. With the information the STP obtains, it can evaluate the operator using mainly local computation and only minor MPC based aid from the other parties. Besides the leakage of the input columns to the STP Conclave upholds it's normal security guarantees for every other column. For these special operations conclaves security assumptions differ from its normal security assumptions and can be modelled using a general adversary. Conclave's hybrid operations can withstand any adversary that can corrupt a set of parties that, does contain the STP but no other party, or does not contain the STP and could be withstand by a normal operation.

Sorts and Shuffles Many of conclaves high-level operators include "sub-protocols" like sorts and shuffles. These sorts and shuffles are MPC operations. As such they are highly expansive operations. Yet not all of these sorts and shuffles are always necessary. If for example a operator produces a sorted intermediate result like for example an order by operation would do, it is redundant to sort again as part of the next operator. Conclave is able to identify such redundant sorts and shuffles and eliminates them where possible. The ability to skip such expansive MPC operations provides significant performance gains.

- published in 2019,
- compares to "SMCQL most similar existing system"
- jiff dependency
- requires python 3.5
- ... - no secure channel setting

4.2 ABY3

ABY3 ...

4.2.1 functionality

ABY3 is a 3-party MPC framework that allows to compute queries on relational database tables. It focuses on computing various SQL-like join as efficiently as possible. Therefore it features a large range of different join operations these include but are not limited to left join, right join, set union, set minus, and also full joins. Besides joins it is also possible to query a single table with query's that have a comparable semantic to the SELECT, FROM,WHERE; statement in SQL. For example a selection like "select X1 from X where X2 > 42" can be done with relative ease using the implemented features of ABY3. One of ABY3 great strengths is its composability. Each operation done on one or more tables produces as output also a table, which is a valid input for another query. This allows to build larger complex applications out of many small ones, very similar like one would do with a pipes-and-filters architecture. Furthermore, ABY3 comes with a description of how aggregate functions like MAX, SUM, COUNT can be realized when utilizing ABY3. For example, the maximum operator can be evaluated with a recursive algorithm that computes the maximum of the first and second half of the rows. In theory, ABY3 is able to compute any polynomial time function of a table, in practice, the efficiency may differ between functions and may not always be sufficient. For executing its MPC operations ABY3 relies mainly on secret sharing.

Prototype Implementations ABY3 demonstrates its capability in two prototype applications. One of them could be used by the states of the United States to help ensure the validity of voter registration records. In the United States, each state maintains its own list of registered voters. Through the highly sensitive nature of these records coordination between states to ensure their faultlessness is not trivial. For that reason, one person moving from one state to another may often result in being registered in both states, which would allow them to illegitimacy cast a vote in both of these states. ABY3 demonstrates how it could be used by the states to detect such double registration while preserving the confidentially of the records.

4.2.2 underlying MPC technology

ABY3 works within a 3 party setting. This is a conscious decision as the two partiy and tree party setting each provide their own advantages and disadvantages. The third party allows to deploy more efficient algorithms that could not be deployed in a two-party setting. For example, oblivious permutations can be done in O(n) in a three-party setting instead of O(n log n) in a two-party setting. On the other hand, there are already established solutions for many problems in the two-party setting that are not easily extendable to a three-party setting. ABY3 guarantees security against a semi-honest threshold adversary that is bound by an honest majority. For executing its MPC operations ABY3 relies mainly on secret sharing. As secret sharing comes with the advantage that algorithms based on secret sharing can have their input present in secret shared form and their output also is secret shared. When considering composability this is a great advantage as having input and output in the same format, as it allows to directly feed the output of one operation as input into the next one. While other MPC techniques like oblivious transfer require either input or output to be in the clear and would need to expansively transform it after each operation. ABY3s key feature are its new protocols for joins based on a MPC based cuckoo hash table. With these new protocols it is possible to join n rows with only O(n) overhead.

Computing Joins One key task for computing any kind of join is identifying which rows have identical join keys. More precisely if for two tables X,Y and key columns X_1, Y_1 and any given i their exists j such that $X_1[i] = Y_1[y]$. Where X[i] denotes the i-th row of table X and $X_1[i]$ the i-th entry of the first column of table X.

ABY3 implements an algorithm that solves this problem using a secure cuckoo hash table T with two hash functions h_1 and h_2. In a first step each row of Y is inserted into the hash table, such that Y[i] is inserted into $T[h_0(Y_1[i])]$ or $T[h_1(Y_1[i])]$. If $X_1[i]$ has a matching join key, such that $X_1[i] = Y_1[y]$, the matching row can only be located in $T[h_0(X_1[i])]$ or $T[h_1(X_1[i])]$. Therefore in a second step a match can found by comparing $X_1[i]$ and $T[h_0(X[i])]$, $T[h_1(X[i])]$ in a secure way. The key challenge in this algorithm is the construction and usage of a secure cuckoo hash table that does not leak sensitive information. ABY3 implements such a hash table based on an oblivious switching network.

Oblivious Switching Network TODO

4.3 smcql

SMCQL is an MPC based framework for relational database operations that is based on an already existing MPC framework, namely ObliVM. With SMCQL one can specify a query and SMCQL automatically generates secure code for evaluating the query.

functionality SMCQL realizes a private data network. A private data network is a union of many mutually distrusting databases that can be queried like a single engine that holds all data of every party. From the user's perspective, a private data network functions exactly like one monolithic database. With SMCQL one can specify queries in a semantic very similar to SQL and SMCQL translates these queries into a sequence of MPC operations. Therefore SMCQL allows using MPC without having detailed knowledge of the underlying system. With this approach, SMCQL wants to increase the accessibility of MPC. SMCQL supports a variety of SQL operators, these include selection, projection, aggregation, equi-joins, theta joins, and cross products. With its SQL like Syntax SMCQL can evaluate every query consisting of a combination of these operators that would be a valid query in plain-standard SQL.

underlying MPC technology SMCQL currently works in a two-party setting and provides security against a semi-honest, threshold adversary that can corrupt at most one party. The two parties are aided by an honest broker a neutral third party that plans and orchestrates the execution of the protocol. The honest broker functions as an access point for the user and receives his query. Once the honest broker receives the query it parses the query into a directed acyclic graph of operators. Each node in the graph represents one operation and an edge between two notes annotates that the incoming node consumes the output data of the outgoing node. With the operator graph, the honest broker is able to analyze the flow of data through the query and decide which of Smcql's different optimizations are applicable to each node. A detailed description of these optimization can found in Section 4.3.4. Once all optimizations a planed the honest broker generates secure MPC based code and provides it to the parties. For its secure computations SMCQL uses the already existing ObliVM framework.

ObliVM TODO hier beschreibung von ObliVM und ORAM

Accesses Control SMCQL features an accesses control system that enables the data owners to adequately model the sensitivity of their data. The accesses control is column based and each column is either public, protected, or private. A public column may always be revealed to any party including the honest broker. A protected column may be revealed if the query is k-anonymous. A query is k-anonymous if for each queried tuple it holds that the projection onto its protected attributes is indistinguishable from at least k-1 other tuples. A private column is under no circumstances revealed to any party. With these accesses control mechanisms, SMCQL is able to speed up querie evaluation by applying various optimizations. If for example an operator only works with public columns it can be evaluated without using expansive MPC operations.

4.3.1 Optimizations

SMCQL implements various techniques that speed up query evaluation and help it scale.

Slicing One such optimization is slicing. When SMCQL identifies an operator as sliceable, it partitions the input data into smaller units of computation. The partitioning of the input tuples is done horizontally. Small units of computation are easier to evaluate compared to a large monolithic operator, they allow for less complex secure code and in some cases, the evaluation of the units can be parallelized. Projections and filters are particularly easy to slice, as they can be evaluated working one tuple at a time.

Split Operators Another optimization that helps SMCQL scale are its split operators. A split operator splits the evaluation of a high operator that requires MPC in two phases. First, a phase of local plaintext computation that is followed by a second phase of MPC computation. The intuition behind this is that the MPC computation in the second phase is cheaper than the evaluation of the entire operator with MPC would be. Most aggregate functions can be spilt, in the first phase each party local aggregates over its own columns, and in the second phase, MPC is used to compute the correct aggregate out of these intermediate aggregates. The evaluation of a count(*) operator, for example, can be split, in the first phase each party locally counts its own input data and in the second phase these intermediate results are added up with the help of MPC.

4.4 rejected frameworks

CipherCompute On candidate for our study was Cipher Compute. With the CypherCompute framework it is possible to solve a huge range of MPC problems using Rust. These include SQL operations like joins that are of interest for us. Furthermore CypherCompute provides a rich documentation, consisting of a full quickstart guide and several well documented example projects. CypherCompute utilises the SCALE-MAMBA framework for its underlying MPC operations. SCALE-MAMBA itself has evolved out of the well-known SPDZ protocol. Unfortunately the early access version of CypherCompute is not functioning by the time we conducting this study. Therefore we have decided to not include CypherCompute in our study.

Prio+ Prio+ [AGJ+21] is the next generation of the highly influential Prio [CGB17]. Prio+ strives to maintain the same use and security as Prio, while significantly increasing performance compared to its predecessor. Prio Plus allows an arbitrary number of parties to jointly compute aggregated statistics, like SUM, MAX, MIN operators. Prio+ utilises a client server model. In which the (potentially many) input parties use a small number of servers to compute the statistics. Prio+ guarantees confidentiality of the input values if at least one server stays honest. Unlike CipherCompute or conclave Prio+ is not a framework for developing MPC solutions. Its rather an already complete system. This means that the use of Prio+ can not be extended beyond the usecases that have been originally implemented by the authors of Prio+. This leaves Prio+ with a relatively small range of usecases compared to frameworks like aby3 or conclave. Therefore we have decided to not include Prio+ in our study.

VaultDB coming soon

5 implementation

6 evaluation

Bibliography

[Eti14] Y. Eti. On the Importance of Correct Stirring. *International Journal of Cookie Theory*, 13(1):1–247, 2014.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. *Fast Database Joins and PSI for Secret Shared Data*, page 1271–1287. Association for Computing Machinery, New York, NY, USA, 2020.