Developing a flexible and deployable MPC system used for statistical data analysis

Emiel Rous[a]

[a]*University of Twente, Enschede, Netherlands*

Abstract

As the significance and availability of data continue to grow, ensuring data privacy and security has become paramount. A possible solution to obtain value from data while maintaining privacy is Multi-Party Computation (MPC); a technology that allows calculations on private data. MPC is maturing to a point where it may see widespread use in the near future, so the question of technological acceptance is becoming more relevant. Systems that handle private data require a minimum level of trust of its users, which may play a role in its acceptance.

This research is motivated by the requirements of Fraunhofer FOKUS, which aims to align security standards across different partner companies through their Cyrille system. This thesis will focus on (1) developing a flexible MPC system used for data analysis and (2) investigating the question of trust in the context of MPC. The prototype developed in this thesis can be used to execute several statistical calculations on private data, includes data persistence, and is able to handle multiple parties asynchronously. Utilizing the Sharemind framework, it allows filtering data and contains multiple datasets making it flexible and usable in different contexts.

The second focus of this thesis is on the sociological aspects of trust in new technologies such as MPC. Design choices can be made to improve user trust. The interface design of the prototype is based on research into these choices to ensure that the final system elicits the most trust. This thesis demonstrates that MPC systems can be both practical and secure, paving the way for broader adoption in various domains where data privacy is crucial. It advances the technical implementation of MPC and provides insights into the increasing trust in emerging MPC technologies.

1. Introduction

As data is becoming more important and widely available, the question of privacy and security becomes increasingly important. A lot of value can be obtained from aggregating data, which is reflected by the size of some of the current big-tech companies dealing in data. Aggregating data will inherently expose some information about the data if not done correctly. The field of multi-party computation (MPC) tries to solve this crucial issue by creating protocols that maintain data privacy while still producing results. The main idea is to evaluate a function on private input data from different parties while not revealing any information about the data other than the final result. In other cryptographic problems, the goal is usually to secure data by encrypting it or to secure the data communication channels so that only the designated parties can access the data; the adversary is kept out of the system. In MPC the goal is to secure the protocol itself so that it is secure despite the adversary taking part in the protocol. There is no need for a third party because the entire calculation is done by exchanging encrypted messages between all parties.

The guarantees of MPC are promising, but currently the results have mainly been theoretical and the number of accessible and efficient tools is limited. This is changing rapidly as more tools are developed, which means that the technology may be maturing to a point at which it will be ready for real-world applications. This also means that the question of technological adoption will become important. Users must be willing to use the technology on their data, which introduces a question of trust. The user may only want to upload their data to an MPC system if they trust that system. Thus, adoption of MPC systems is not only a technological challenge, but also a sociological one. This thesis will (1) focus on the development of a flexible MPC system capable of handling different types of data and be easily accessible to users, as well as (2) investigate what it means to trust a system, what can be done to influence trust in a system, and incorporate some of these findings into the system.

The main stakeholder in this thesis is the Fraunhofer FOKUS research institute in Berlin. Fraunhofer is developing a system called Cyrille, which will help align security requirements across different partner companies by assessing the security level of those partners. Each partner company verifies their security by filling out security questionnaires that will determine their level of security according to a security standard such as ISO270011. The data from these questionnaires are numeric, categorical, and temporal (since the security questionnaire is a snapshot and subject to change over time), and the goal of the parties is to be able to generate aggregate statistics of the results of multiple parties. There may be different assessment requirements for each party, so users need to be able to filter data, and they need a number of statistical functions to select from.

Many MPC solutions focus on single calculations and require all parties to be online at the same time, but data persistence or asynchronous calculations are not common. The question-

naire data is time dependent, which introduces specific requirements in terms of data storage and asynchronous calculation, compared to commonly tackled use-cases in MPC. One of the requirements from Fraunhofer is that companies may want to update their answers to their security questionnaire, the questionnaires themselves may be updated, or results need to be recalculated. This means the data has timestamps and the system needs to securely keep track of updates, store or re-request secret values, and be able to recalculate statistics at the users request. It is also important that the system allows for more than two parties, because the goal of the system is to compare and choose parties to collaborate with, so a high number of parties should be able to upload their data. In summary, the requirements we will set for this system are as follows:

1. There needs to be support for statistical calculations, such as the mean and variance and possibly other descriptive statistics.
2. The system needs filters and flexibility, so each party has the freedom to create their own assessment of the other parties.
3. Input data must be persisted so that input parties are not required to stay online and so that data can be easily appended, updated, and so that changes can be tracked.
4. There need to be at least three parties, preferably more.

Currently, there are not many tools that meet all these requirements, and at the time of writing, there are only a few accessible systems (i.e., readily deployable). Although there is a lot of existing work on MPC and securely calculating descriptive statistics, there are few tools that also incorporate secure storage. A tool like EMNET (Hailemichael et al., 2015) meets all requirements, but is restricted to medical data. Tools such as SEPIA (Burkhart et al., 2010), Carbyne-stack (Becker et al., 2021), or FRESCO (Alexandra Institute, 2023), meet most requirements, but none of these meet all requirements. The only system found that meets all the requirements is Sharemind (Bogdanov et al., 2008), which will be used in this thesis. Additional support for this choice and a comparison with the other frameworks will be given in chapter 2.1.

The goal of this thesis is twofold: implement a system according to the requirements and investigate what influences trust in a new technology such as MPC. In this thesis, the following research questions (RQs) will be answered:

1. RQ1: What is a suitable design for an MPC system in line with the requirements from Fraunhofer?
2. RQ2: What does a suitable user interface look like for a flexible MPC system?
3. RQ3: Which challenges does MPC face in terms of gaining user in MPC and what can be done to influence and increase this trust?

This thesis will take two paths to answer these questions. The first is the technical implementation and background of MPC to answer RQ1 and RQ2. This includes related work chapters 2.1 and 2.2, background chapter 3.1, and result chapter 5.1. Chapters 2.3, 3.2, and 5.2 will answer RQ3 about trust in MPC. The discussion and conclusion will also be split to discuss each of these topics separately.

Requirements	**SEPIA**	**Sharemind**	**MPyC**	**FRESCO**	**Carbyne stack**
Security model	S.h. min.	S.h. min.	S.h. min.	D. maj.	D. maj.
Number of parties	∞	In: ∞, Nodes: 3-5	∞	∞	∞
Stat. calculations	No	Yes	Yes	Yes	No
Persist data	Yes	Yes	No	No	Yes

Table 1: Comparison of requirements across different MPC frameworks (S.m. min. = semi-honest minority, D.maj. = dishonest majority)

2. Related work

Each of the research questions has related work associated with them. For RQ1, which is about creating a back-end, several technologies will be looked at which are both available and deployable. By looking at the requirements, one can compare the available technologies and motivate the choice for Sharemind. For RQ2, about the front-end, MPC solutions that have a user interface will be looked at. Then lastly, to investigate trust in MPC (RQ3), the available work will be investigated and some related examples in a different field will be used, namely that of trust in AI.

2.1. Related work: back-end

For the implementation of an MPC back-end, there are several options. There is, for example, an MPC programming language like MPyC (Schoenmakers, 2018), which is a Python based language using *Shamir's* secret sharing scheme. Then there are protocol suites such as MP-SPDZ (Keller, 2020), which run under the hood of some frameworks, such as FRESCO (Alexandra Institute, 2023) and Carbyne-stack (Becker et al., 2021). These have a large set of implemented functions for statistical calculations as well as many other useful functions for data analysis, but all lack the ability to persist data. Another framework that was looked at was SEPIA (Burkhart et al., 2010), which is a secret shared system very similar to Sharemind, but it has no inherent support for statistical calculations nor has it the ability to persist data.

Although it is not a hard requirement, the security of each of the frameworks will be taken into account because it is a crucial part of any MPC protocol. In MPC the distinction is made between dishonest and semi-honest parties. Dishonest parties are parties that actively try to manipulate the calculation by spoofing data or not following protocol. Semi-honest parties are following protocol but are listening in and trying to gather as much information to reconstruct private input data. Then there is the distinction between a majority and a minority, which refers to

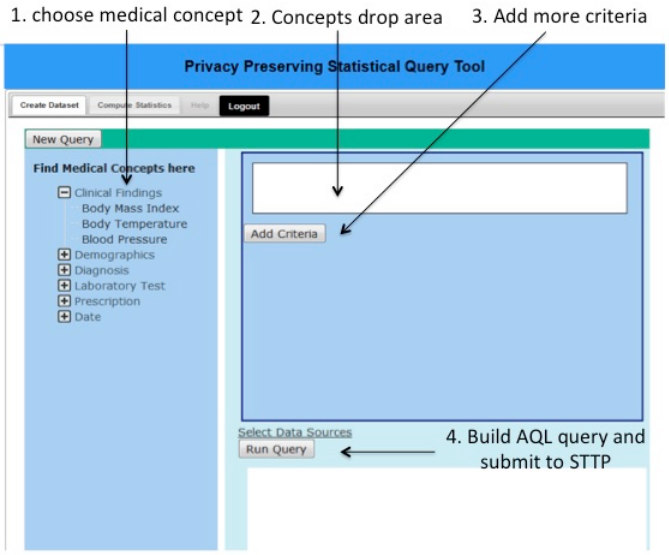

Figure 1: Build dataset using queries in Emnet (Hailemichael et al., 2015)

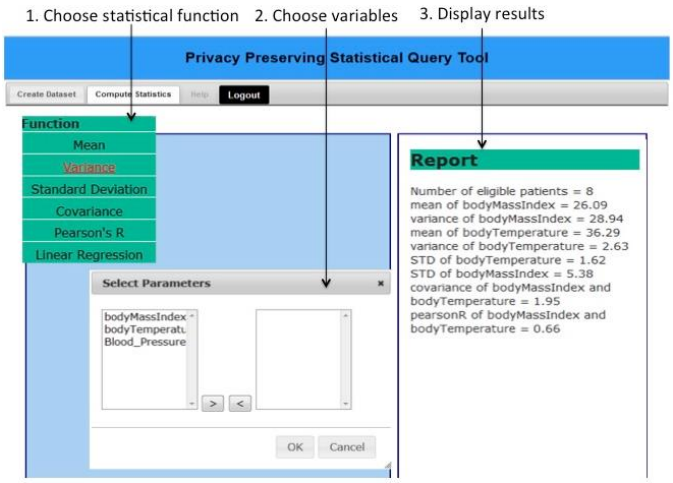

Figure 2: Select statistical function in Emnet (Hailemichael et al., 2015)

the number of semi-honest or dishonest parties that the protocol can tolerate while still remaining secure. For an overview of the MPC tools and their features, please examine table 1. Note that in chapter 1, Emnet (Hailemichael et al., 2015) was also mentioned, but because that system is strictly limited to medical data taken from a specialized medical database, it will be left out of this comparison.

2.2. *Related work: front-end*

There are some works that implemented a front-end. The one most closely related to what will be developed in this thesis is Emnet (Hailemichael et al., 2015). This framework is based on additive secret sharing and creates computation graphs of each function before solving it securely. Emnet has a front-end and can be used to perform statistical functions on medical data. The user can select a dataset or create one using a special query language (Figure 1) and subsequently can select parameters (*columns*) and apply functions to them (Figure 2)

Another work similar to this thesis is Bogdanov et al. (2012b), which is based on Sharemind and in which the authors also developed a web front-end. Companies can use it to provide sensitive financial data and compare themselves with other companies, without compromising their trade secrets (Figure 3).

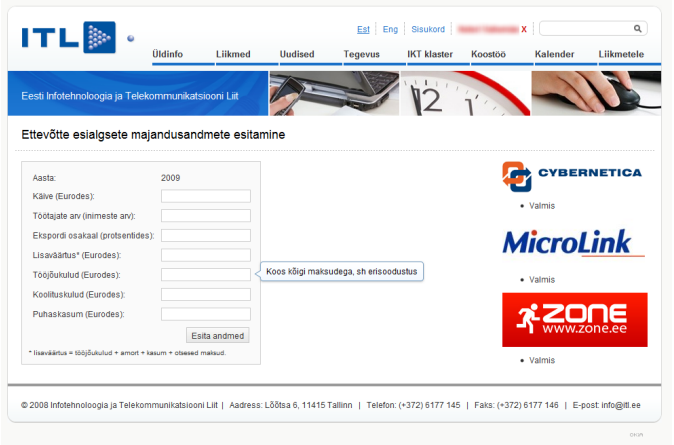

Figure 3: Comparing sensitive metrics with other companies using Sharemind (Bogdanov et al., 2012b)

Web-MPC (Lapets et al., 2018), uses a JavaScript library called JIFF for the implementation of MPC calculations. The architecture consists of a single server to which clients connect, and once they are connected, the calculation can start between them. JIFF uses additive secret sharing, and the implementation in JavaScript and an easily accessible library makes it straightforward to implement a web-application. An example taken from the main article can be seen in Figure 4.

2.3. *Related work: trust*

On the topic of MPC and trust, there appears to be no research as of the time of writing. Some work investigates the reception of MPC in end users (Bogdanov et al., 2013), but this focuses on the needs of the user and finding a fitting solution using MPC. Because of the limited available work, one can look at a different field, namely that of trust in AI, to gain some insight. There is much research on explainable AI, trust in AI and perception of AI. Like MPC it is very complicated to understand the inner workings for a non-expert, so often these technologies are referred to as being a black box. Because of the similarities, one can look at some findings in this field and compare it with MPC. However, there are also many differences, which will be discussed in the discussion chapter 6.

For the acceptance of a technology, it is important that users understand the technology (Saariluoma et al., 2019). It seems to be important for the acceptance of AI that there is trust (Choung et al., 2023) in addition to perceived usefulness, This could mean that to trust the technology, it needs to first be accepted and, to be accepted, it needs to be understood. Findings from other studies indicate that transparency and reliability are important antecedents in building trust in AI (Glikson and Woolley, 2020). Some studies also highlight that tangibility and appearance are important (Omrani et al., 2022).

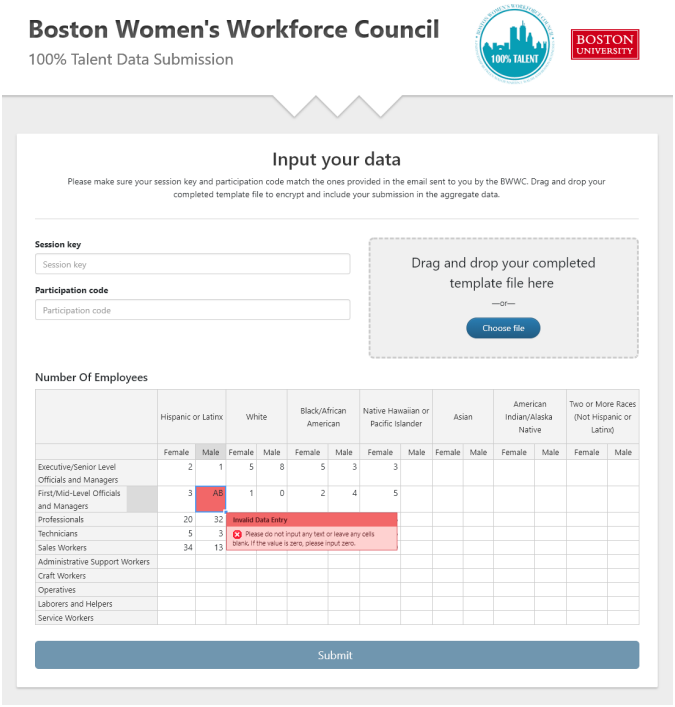

Figure 4: A JIFF based web application (Lapets et al., 2018)

3. Background

For this thesis, the framework that will be used is Sharemind (Bogdanov et al., 2008). It is very easily deployable, uses a familiar C-like programming language, and fulfills all the requirements from Fraunhofer. In addition to implementing an MPC system, it is also important to investigate whether users will trust the system. To do this, theoretical frameworks of trust will be employed to argue about the components of trust in an MPC and to support design choices that may improve the trust in the system.

This chapter will discuss the workings of Sharemind, including its architecture, security guarantees, and some of its protocols. The second part of this chapter will be dedicated to laying the theoretical foundations for understanding trust in technology and the factors that influence that trust.

3.1. Sharemind

The Sharemind framework (Bogdanov et al., 2008) is a secret shared computation platform. It consists of a (distributed) computation runtime environment, a programming language called SecreC, and several libraries such as a CSV importer for securely uploading data for calculation and a JavaScript library for interacting with the back-end. Sharemind is built up of layers of protocols from low-level (addition, multiplication, bit extraction) to high-level (outlier detection, clustering, statistical calculations).

SecreC is a programming language developed with the philosophy of making an accessible language that leverages MPC but can be easily learned by an average programmer. This is achieved by having a familiar C-like syntax and forcing explicit public and private types. An inherent downside of MPC

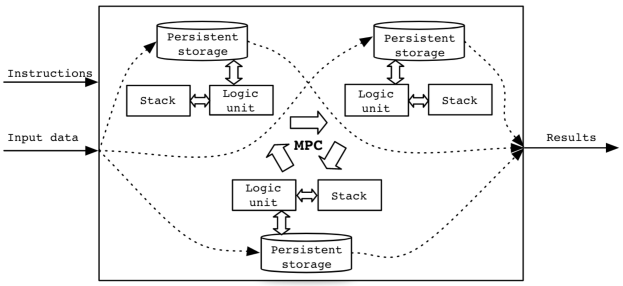

Figure 5: Overview of Sharemind infrastructure (Bogdanov et al., 2008)

programming languages is inefficiency because MPC computations may require several network communication rounds, meaning they can take several milliseconds compared to public computations, which can take nanoseconds. To combat this a solution is parallelism, because the computations can be done in parallel without needing more communication rounds; they require only more bandwidth. The result of this is that all operations on the matrices are performed in parallel and are comparatively efficient. Furthermore, SecreC has many modules for different types of calculations, such as statistical summaries, regression, and principal component analysis.

3.1.1. Infrastructure

The Sharemind infrastructure can be seen as two separate parts. The calculation run-time environment and the input parties. The calculation run-time environment consists of three to five nodes that do the calculation and manage the database. They are referred to computation or miner nodes. The input parties provide the data to the computation nodes and may also be interested in the result of the aggregated data. There can be any number of input parties since they do not take part in the computation protocol. The input parties create and deal the shares, which means that no one computation node ever sees the complete input data. Each miner has a local database that stores the incoming shares of the client. How the shares are dealt will be discussed in section 3.1.5.1 and how the shared are stored will be discussed in section 3.1.6. An overview of the infrastructure can be seen in figure 5.

An input party uses a public-key infrastructure to connect to each miner and deal the shares. Since clients know the public keys of each calculation node, data uploading could even be done offline through the use of a public database, where the encrypted shares are stored. Each miner can then retrieve the encrypted share and store the decrypted share in their local database. Note that what is decrypted by the miners is still a share and not the secret value itself.

3.1.2. Gateway, JavaScript client and CSV-importer

The Sharemind framework consists of several applications. There are the calculation modules themselves (the processes responsible for doing the calculations running on each of the computation nodes), there is a CSV-importer, a secret shared version of R called Rmind, a JavaScript client library, and a

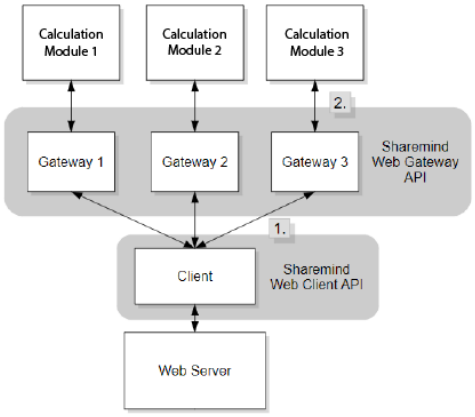

Figure 6: Schema of the client-server interaction

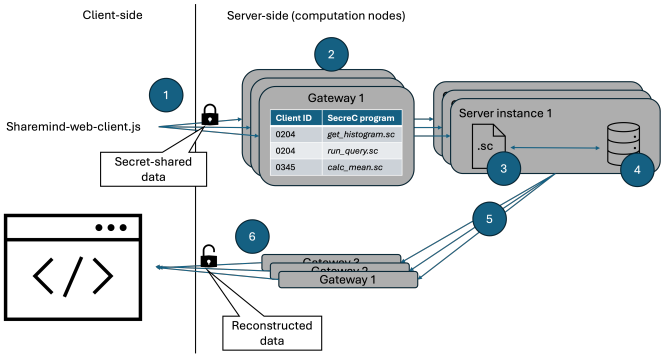

Figure 7: Overview of program execution flow (note that in a production environment each gateway and associated server instance would be hosted by different parties)

gateway application. The latter acts as an interface between the JavaScript client and the calculation modules.

The gateway runs in front of the calculation modules and handles requests coming from the client. Each calculation module has an associated gateway that does not have to be on the same machine. Each gateway has a key-pair and exchanges public keys with their respective calculation module instance. The gateway configuration is done using configuration files and a JavaScript file. The JavaScript file handles how clients are added and removed and maintains a list of available SecreC programs that each client can execute. Once a client has requested to execute a certain program, the gateway starts negotiation with their calculation module, and the program is executed.

The client interface is a *Node.js* module which is used to communicate with the gateway applications. It is responsible for setting up a connection to each of the gateways whenever a new program needs to be executed. The client library also takes care of secret sharing the values. An overview of the interaction between these applications can be seen in Figure 6.

The CSV-importer is used to upload data to the database. It supports automatic mapping of categorical variables to numerical values and can create a new database or append to an existing database. The configuration of how data must be imported, including data types, public-private distinction of columns, column names, and database name, is done using an *.xml* file. The CSV-importer uses the public keys of each of the calculation modules to upload the data which can be done from any machine.

3.1.3. Overview of program execution

To better illustrate how the whole system operates, including which library is responsible for what, a typical program execution flow will be analyzed. Figure 7 can be used as a reference. The numbered circles indicate actions of the program. The following actions are taken when a user requests to run a program:

1. The *Sharemind-web-client.js* library is used to secret share any secret variables that must be sent to the server and subsequently starts a TCP connection with each of the gateway instances on the servers. Note that the gateways as well as the computation modules can be on separate machines/locations).
2. The gateways check the *client-id* of the incoming client request and check if the client is authorized to execute the requested program
3. The gateways start negotiation with their respective computation module to execute the requested program. Any variables sent from the client are passed to the computation module.
4. Each computation module runs the SecreC program and interacts with their (secret-shared) database.
5. The result is declassified and sent back to the gateways
6. The client receives the declassified values and displays them in the front-end

3.1.4. Security

The common implementation of Sharemind uses three calculation nodes. These nodes are separated from each other and should be managed by (physically) different parties. This is because the security of the system is guaranteed in the semi-honest minority setting, meaning that in the case of three miners, security is ensured if no two parties collaborate. Even though there exist protocols which allow for a dishonest majority, these have the drawback of being very computationally intensive as well as often requiring the input parties to be online during computation. The authors of Sharemind argue that it is not hard to find three parties which do not want to collaborate, or which can be restricted from collaboration through inter-organization contracts and software-auditing. Examples are governments or hospitals, which have a strong incentive to guarantee privacy and not collaborate with other parties.

Each node in the system is also part of a public-key infrastructure. As mentioned before, the input parties are able to encrypt data for each miner node individually using their respective public key. Each miner node is connected to the other miner nodes using symmetric encryption and message authentication. The channels used for secure communication are over

UDP and provided by the RakNet[1] library. Message authentication is done with MAC keys using the HMAC-MD5 algorithm. The input parties follow an access structure to ensure that resources are protected. This includes write access to certain databases and access to results.

Because all high-level protocols are built from low-level protocols, proving the security of them can be done through the universal composability framework (Canetti, 2001). In short, if protocol A can be replaced by another protocol B, which is perfectly secure, and an outside observer cannot distinguish the protocols, then protocol A is also perfectly secure. If you prove that the low-level protocols are universally composable, then the security of the high-level protocols can be inferred because combining universally composable protocols will result in a universally composable protocol. The proof of the security of the low-level protocols can be found in Bogdanov et al. (2008) and Bogdanov et al. (2012a).

3.1.5. Secret sharing and low level protocols

This chapter delves into the foundational aspects of Sharemind's secure computation. It will discuss the secret sharing scheme, but also discuss some important low-level protocols which are crucial for constructing more complex high-level protocols.

3.1.5.1. Secret sharing. When a value u is shared, it is divided into different parts $u_1, u_2, ..., u_n$ where each part i is held by party P_i. The vector that describes the shares is denoted as $[u] = (u_1, u_2, ..., u_n)$. Sharemind uses a simple additive secret sharing scheme, meaning that $u = u_1 + u_2 + ... + u_n$. The scheme works as follows. A secret value x is shared by generating uniform random values $r_1, ..., r_n$ and adding them to x:

$$s = x + r_1 + ... + r_n \tag{1}$$

Each miner node $M_1, ...M_n$ receives a share of the secret s. In the case $n = 3$ each miner has the following shares:

$$M_1 : x_1 = s - r_1 \tag{2}$$
$$M_2 : x_2 = - r_2 \tag{3}$$
$$M_3 : x_3 = - r_3 \tag{4}$$

To reconstruct the secret value, the miners simply sum their shares:

$$x_1 + x_2 + x_3 \tag{5}$$
$$= s - r_1 - r_2 - r_3 \tag{6}$$
$$= x \tag{7}$$

3.1.5.2. Addition and multiplication by scalar. Using this secret sharing scheme there are several fundamental protocols which form the foundation of the higher-level protocols. The first two are rather straight forward: addition and multiplication by a public scalar. Both of these can be done locally. For addition of secret values u and v every party will simply add the shares of both together and summing all the shares will result in the sum of u and v.

$$u + v = (u_1 + v_1) + (u_2 + v_2) + (u_3 + v_3) \tag{8}$$

The same goes for multiplication by a public scalar: every party simply multiplies its share with the scalar.

$$uc = (u_1 c) + (u_2 c) + (u_3 c) \tag{9}$$

3.1.5.3. Resharing protocol. To ensure universal composability, it is important that the input is independent from the output (Canetti, 2001). The resharing protocol obfuscates a secret with a random value so that the output is independent of the input. By applying this protocol on the input or output shares, many of the protocols are made universally composable. The proof of this can be found in (Bogdanov et al., 2012a). The resharing of secret $[u]$ is done in the following way.

1. Each party (P_1, P_2, P_3) generates one random value from a uniform distribution (r_1, r_2, r_3 respectively)
2. Then each party sends their r to one other party in the following way: P_2 receives r_1, P_3 receives r_2 and P_1 receives r_3
3. Each party computes: $w_i = u_i - r_i - r_*$ (where r_* is the secret received)

Now the secret value v is independent of the output w

$$w = w_1 + w_2 + w_3 \tag{10}$$
$$= (u_1 + r_1 - r_3) + (u_2 + r_2 - r_1) + (u_3 + r_3 - r_2) \tag{11}$$
$$= u_1 + u_2 + u_3 = u \tag{12}$$

3.1.5.4. Multiplication protocol. The addition and multiplication by a scalar protocols are rather trivial, but multiplication of two secrets is more difficult. The multiplication algorithm from Bogdanov et al. (2012a) that is used in the Sharemind framework will be examined. Multiplying two secrets u and v, which are shared additively between three parties, results in the sum $uv = \sum_{i=0}^{3} \sum_{j=0}^{3} u_i v_j$. Each party can locally compute $u_i v_j$ for $(i = j)$ but when $(i \neq j)$ the parties need to exchange their share, which they do in a very similar way to sharing the random value in the resharing protocol. Each party only sends out one share of both u and v to one other party. Each party can then use their own shares in addition to the received shares to calculate $u_i v_j$ locally. Because having two secret shares may reveal too much about the secret value, each secret is reshared before calculation. To demonstrate this, the protocol is as follows:

1. The parties reshare $[u]$ and $[v]$:
   $Reshare([u]) = [u']$, $Reshare([v]) = [v']$
2. Each party sends out their share of $[u']$ and $[v']$ to one other party, similar to the resharing protocol: P_2 receives u'_1 and v'_1, P_3 receives u'_2 and v'_2 and P_1 receives u_3 and v_3
3. The parties calculate their respective shares: $w_i = u'_i v'_i + u'_i v'_* + u'_* v'_i$ (where u'_* and v'_* are the received values)

[1] http://www.jenkinssoftware.com/

This protocol is correct:

$$w = w_1 + w_2 + w_3 \tag{13}$$
$$= u_1{}'v'_1 + u'_1v'_3 + u'_3v'_1 + u'_2v'_2 + u'_2v'_1 + \tag{14}$$
$$u_1{}'v'_2 + u'_3v'_3 + u'_3v'_2 + u'_2v'_3 \tag{15}$$
$$= (u_1{}' + u'_2 + u'_3)(v'_1 + v'_2 + v'_3) \tag{16}$$
$$= (u_1 + u_2 + u_3)(v_1 + v_2 + v_3) = uv \tag{17}$$

These were only a few of the low-level algorithms. Another important algorithm to mention is the protocol for share conversion. This protocol is used to convert, for example, Boolean values to bit values of length 32. This serves two purposes. The first is that algorithms are more easily implemented on homogeneous data types. The second is that it creates a form of input validation because only valid inputs will have a bit-length which is set by the share conversion algorithm. Additionally, being able to set the bit length of variables means you can choose bit lengths which are easy for most hardware to handle such as 32 or 64 bit. Since its creation, Sharemind has been constantly improved and amended with new protocols. These include protocols for bit-wise operations such as conjunction and bit extraction, but also protocols for equality testing and protocols for public and private division (Bogdanov et al., 2012a).

3.1.6. Database

An important feature of the Sharemind framework is the database. It allows for asynchronous calculation where the input parties do not need to be online. It also allows the computing parties to use and reuse data when it is needed. There are two types of databases in Sharemind: relational databases and key-value databases. A relational database consists of tuples of equal length where each value represents an attribute. It is rather straightforward to convert a relational database into a secret database by secret sharing each value of each tuple. In practice, however, a database almost never consists of only secret values, and therefore Sharemind uses hybrid databases: the tuples consist of public and private values. For key-value databases, one can simply store the share instead of the value.

3.1.6.1. Inputting data. The input parties can write directly to the database of each miner. They will first create the shares before sending them to each computing party for storage. The access structure defines which data providers can upload to which database and can execute which functions. Figure 8 shows how a secret value is distributed to the computation nodes' databases.

3.1.6.2. Manipulating secret shared data. Running a query on a secret shared database is somewhat different than running a query on a normal database. Conventional filtering methods may reveal individual records by publishing their location in memory. Therefore, in Sharemind, filtering is done on individual columns or rows. Using the name or index, a column can be loaded into a vector. By using row indexes individual rows can be addressed, and by combining the column and row selection individual cells can be targeted. Note that the rows, columns,

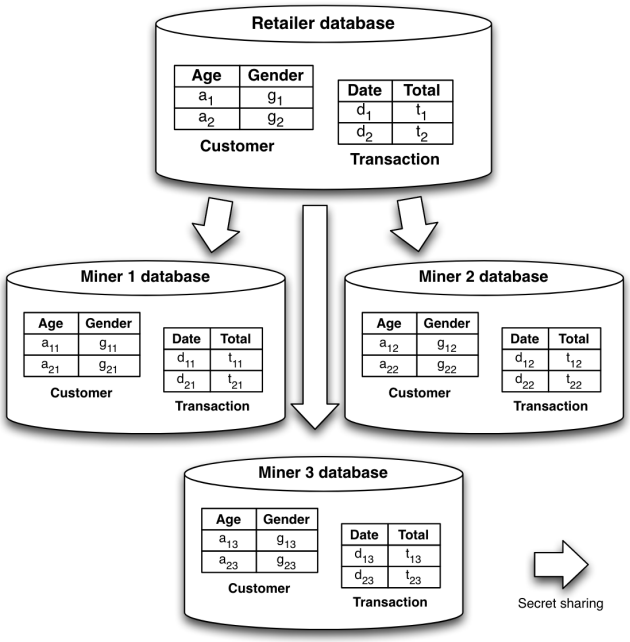

Figure 8: Example of distributing secret shares to the databases

and cells addressed are still secrets and stored in a vector, so they reveal no information about the actual value or about the location in the database. Furthermore, because the databases are hybrid, a column of the database can contain public values which can be used to identify the rows.

For simple calculations, load a column into a vector and apply a function, such as the mean, on that column. To filter the data first before doing calculations, first extract the column that will be used for filtering. Then we create a Boolean mask vector that contains a 1 for each row that satisfies the filter. When we multiply the column we are interested in by the mask vector elementwise, we get a vector which contains the filtered values, which we can use for calculations. Take an example where we want to take the average income of people with an age over 30. Assume that there is a database with a secret *age* and *income* column. The steps are the following:

1. Load the secret *age* column into a vector $\bar{a}$
2. Create a mask vector which identifies the relevant rows:
$$\bar{m}_i = \begin{cases} 1, & \text{if } a_i \geq 30 \\ 0, & \text{otherwise} \end{cases} \tag{18}$$
(note that the predicate is tested on a secret value using the inequality algorithm from (Bogdanov et al., 2012a))
3. Load the secret income column into a vector $\bar{b}$ and multiply elementwise by mask vector $\bar{m}$:
$$\bar{f} = \bar{a}\bar{m} = \begin{cases} b_i, & \text{if } m_i = 1 \\ 0, & \text{otherwise} \end{cases} \tag{19}$$
4. Now we can use the share multiplication and division algorithms to calculate the average income:
$$avg(income) = \frac{\sum_{i=0}^{n} f_i}{\sum_{i=0}^{n} m_i} \tag{20}$$

5. Publish the result of this query, revealing no information about the underlying data.

This is how nearly all calculations are done in Sharemind. First, the columns are extracted from the database, after which they are used for calculation. There are many functions that can be applied on a secret vector that will return a secret value, and only when the value (or vector) is explicitly 'declassified' (or published) will the actual value be shown.

3.2. Designing for trust

For the acceptance of MPC it is important to explore the perception of the technology by end-users. They will be the ones uploading the data, so they must put trust in a system that they are likely not familiar with. Compared to traditional data collaboration techniques, MPC reduces the need for trust in other parties, decreases perceived risk, and increases the feeling of control over the data (Agahari et al., 2022). However, this result assumes that users know and trust that the technology itself works. By assuming that the claims of MPC - like complete data privacy and mathematically provable security – are true, users may perceive less risk compared to other data collaboration techniques, but this is a relative assessment. What we are interested in is the trust in MPC for a system like Sharemind. We would like to identify the important factors in gaining trust of a new user when presented with this technology system.

When talking about trust between parties, there is a trustor, who is the one who will give their trust, and a trustee, who wants to be trusted by the trustor. The goal of the trustee is to get the trustor to trust them enough so that they will act. The goal of the trustor is to gauge the level of trustworthiness of the trustee by determining whether the trustee will uphold the promises they made and fulfill the request of the trustor. In the framework of Riegelsberger et al. (2005) the authors differentiate between contextual and intrinsic properties. For contextual properties, they identify a *temporal*, *institutional* and *social* context. *Temporal* context identifies whether the interaction between both parties may occur again. If so, it would be in the trustee's best interest to fulfill the request of the trustor (the user). *Social* context can be seen as reputation; if a trustee has a good reputation, they are more trustworthy, because they may incur a loss of reputation in the case of non-fulfillment. The last context is *institutional*, which looks at whether the trustee is part of a larger collective or needs to adhere to certain rules and regulations. For example, someone may trust a shop owner more than a street salesman, because they know that shops may be audited, must adhere to safety regulations, etc.

The second set of properties are intrinsic properties. The first is *ability*, which could be the professionalism a company projects and by that it signals that it is likely to fulfill a request based on merit. The second is *internalized norms* which are the inherent principles a company or person adheres to, which could, for example, be a mission statement on a company website. The last is *benevolence*, meaning the willingness off the trustee to act in the best interest of the trustor; parties may be more trustworthy if they show signs of selflessness. When we look at what determines trust in a technology itself, there are different approaches. Some authors argue that trust in technology is based on the individual assessment of the technology, which leads to expert-users being more inclined to trust a system (Kivijarvi et al., 2013). Then there is also evidence suggesting that trust in a system by non-expert users is not dependent on the system at all, but more on the organizational context or on the evaluation of near-peers (Rogers, 1986) (Misiolek et al., 2002).

There are many contextual factors that influence trust, but for this thesis, it is more important to loot at the factors that can be influenced. We will therefore look at the paper from Mcknight et al. (2011). In this work, the authors define the parallels between trust in people and trust in technology and they define *benevolence*, *integrity*, and *perceived competence* as important factors in developing trust in humans that in the context of technology are respectively: *helpfulness*, *reliability*, and *functionality*. They argue that the system needs to help the user reach their goal, for example, by having a help function, which is the *helpfulness* (*benevolence*) of the system. Then there is *reliability* (*integrity*), which is the measure to which the system fulfills the user's request reliably and without failures. Lastly, there is *functionality* (*perceived competence*) which is the extent to which the system can do what is expected of it.

In summary, we will use two frameworks to assess what influences trust in the system and what we can do to improve the trust of new users. We will use the framework of Riegelsberger et al. (2005) to argue about how the context of the MPC system will influence trust, for example, by looking at the different parties and their *temporal*, *institutional* or *social* context. The framework of Mcknight et al. (2011) will be used to analyze the system itself, looking at the *functionality*, *reliability*, and *helpfulness*.

4. Methodology

There are many methods for designing and implementing a software system. For example, an iterative approach like Agile (Beck et al., 2001) or a linear approach such as the waterfall approach (Royce, 1987). Most approaches separate the development into different stages that together form the software life cycle. Agile is a spiral approach, where - after the requirements are gathered - there is a cycle of: design, develop, test, deploy, and review. The waterfall approach also separates development into stages, but there is not necessarily a cycle; it is a waterfall. It is possible to revisit different stages in this approach when necessary, but the overall direction is linear. The actual implementation of this prototype will not require many iterations, since the complexity of the software itself is relatively low and the only requirements from Fraunhofer were a few system requirements, so the approach used in this thesis will be the waterfall approach.

The stages of the waterfall approach are:

1. Gathering system and software requirements
2. Analysis
3. Program design
4. Coding

5. Testing
6. Deployment and operations

In the first stage, the requirements for the system and the software are gathered. These requirements will be gathered through several meetings with some of the stakeholders at Fraunhofer. In addition to that, some ideas for the system will be developed, and in a short feedback cycle with the stakeholders, one will be selected. The next stage is analysis, which involves gathering the necessary information to then design the program. In the case of this thesis, analysis will involve learning SecreC (including writing small scripts to help with the understanding of the language) and developing a thorough understanding of the Sharemind architecture. Additionally, this phase is used to choose the other tools that will be used in the final system, such as the front-end framework or the tool to use for data uploading. The program design phase is used to create software diagrams, user interface designs, and architecture designs. The coding and testing stages will be used to implement both the front- and the back-end of the system and will consist of short cycles of coding and testing. The final step (deployment and operations) is not as relevant for this thesis, since the prototype serves only an exemplary purpose and will not be deployed.

5. Implementation and results

5.1. Results: Prototype

The implementation of this project depends on several of the applications mentioned in chapter 3.1.2. For data importing, the CSV-importer is used and for the front-end the *Sharemind.js* library is used. The bridge between the calculation module and the front-end module is the gateway application. In this project, all processes and applications run in the same virtual environment, but in real-world applications these would be managed by separate parties. These applications and other resources are available at an APT repository for major Debian and Ubuntu distributions. There is also a preinstalled virtual machine that runs Debian 12 and has all Sharemind applications installed and configured. For this thesis, a pre-installed virtual machine is used.

5.1.1. Datasets

The system is implemented in such a way that it can be easily extended to include more datasets. To illustrate this, the prototype comes with three datasets taken from Kaggle [2]. They were chosen based on their prospective usefulness in their field of origin and to demonstrate the flexibility of the system. The first dataset contains simple questionnaire data from an airline satisfaction survey. Data analysis of questionnaires is an often used problem addressed by MPC, and the idea of Sharemind was initially developed on this problem statement (Bogdanov, 2007). The airline satisfaction survey data contain personal traits, such as age and gender, as well as making the distinction between business and leisure travelers. The satisfaction metrics range from cleanliness to ease of online booking to gate location. These metrics are measured in a *Likert* scale from 1-5.

The second dataset is a supply-chain dataset with categorical variables and floating point values. There are, among others, columns with product type, number of products sold, and stock levels. This data is usually highly sensitive because it is directly related to the competitive advantage of a company, but collaboration in this field usually has many benefits. When suppliers and buyers collaborate on things like stock levels, it leads to better decision making, inventory management, and order fulfillment (Tai et al., 2022). Therefore, this dataset is used to illustrate the flexibility of the system.

The third dataset contains e-commerce transactions which can be used to detect fraud. The dataset has, among others, a column for the transaction date, columns for amount, type and quantity, and customer age. Finding fraudulent transactions is something well suited for MPC, since it may involve the collaboration of different financial institutions to identify fraud.

5.1.2. Back-end implementation

The system's back-end contains three important SecreC programs. These are: *get-database-name*, *get-column-names* and *run-query*. The first two are self-explanatory as they simply return the names of the columns and databases as a list. The latter contains all the functionality to make the program work. To highlight how the program is set up, take a look at the high-level pseudocode in algorithm 1. The actual implementation of some of the functions is left out, since they should be self-evident.

Lines 1-5 of the program are used to read the data coming in. The function argument reads data received from a client and in this case, these are all public values. The arguments $stat\_function$ and *column* indicate which *column* should be selected and which function should be used on that *column*. Then there are three arrays. The first variable ($filter\_columns$) contains the columns that are used for filtering, the second variable ($filter\_operators$) describes the operators for each column, and the last one contains the values ($filter\_values$). If there are multiple filters the arrays could look something like this:

$$\text{filter\_columns} = \begin{bmatrix} \text{age} \\ \text{income} \\ \text{gender} \end{bmatrix} \tag{21}$$

$$\text{filter\_operators} = \begin{bmatrix} > \\ \leq \\ = \end{bmatrix} \tag{22}$$

$$\text{filter\_values} = \begin{bmatrix} 30 \\ 41500 \\ \text{female} \end{bmatrix} \tag{23}$$

On line 8 a mask is created with only true values with a size equal to the total number of rows. For each filter, another mask is created (line:13) and the resulting mask is updated with the filter mask using an *AND* operation (line:15). This is exactly as how filtering is described in section 3.1.6.2. After all filters are combined, the column of interest is read (line:20). The $resulting\_mask$ is multiplied with this column (line:22) which

[2] www.kaggle.com

Algorithm 1 Pseudocode main program

```
1: int stat_function = argument("stat_function")
2: int column = argument("column")
3: int[] filter_cols = argument("f_columns")
4: int[] filter_oprs = argument("f_operators")
5: int[] filter_vals = argument("f_values")
6:
7: int nrOfRows = size(database.column)
8: private bool[] resulting_mask[nrOfRows]=true
9: for int i = 0 to size(filter_cols) do
10:    int f_col = filter_cols[i]
11:    int f_opr = filter_oprs[i]
12:    int f_val = filter_vals[i]
13:    private int[] mask =
14:       getMask(f_col, f_opr, f_val)
15:    resulting_mask = mask & resulting_mask
16:    if size(resulting_mask) < data_guard then
17:       return Error:not enough data
18:    end if
19: end for
20: private any[] column_to_read =
21:    database.readColumn(column)
22: private any[] filtered_data =
23:    resulting_mask * column_to_read
24: private any priv_result =
25:    applyFunction(stat_function,
26:    filtered_data)
27: public any pub_result =
28:   declassify(priv_result)
29: publish(pub_result)
```

leaves only the values of interest (note that SecreC is parallelized so multiplying matrices will be done elementwise). On the resulting filtered column the *stat_function* is applied, and the result is declassified and published (i.e. sent back to the gateway which sends it to the client application) (lines:24-29).

For security reasons, it is important to maintain a minimum number of records that can be extracted from the private column. Therefore, on lines 16-18, there is an *if* statement that checks if the mask (or rather the true values of the mask) is smaller than a specified *data_guard_number* and throws an error when this is not the case. The *data_guard_number* can be any integer, but higher integers will be safer by definition, as this results in applying the statistical function on more data. A data guard is also needed to prevent users from applying filters in such a way that they can single out records from the database.

The functions implemented so far are mean, minimum, maximum, five-number-summary, standard deviation and variance. The system is built with scalability in mind, and Sharemind offers a large set of functions in their auxiliary libraries, so appending new ones requires minimum effort. To implement a new function, one can add it to the *apply_function* method, which can be seen in algorithm 2. To append new functions, one can simply add the function to the *if-else* statement and return the result.

Algorithm 2 Pseudocode for apply_function

```
1: function APPLY_FUNCTION(int stat_function, private any[] filtered_column)
2:     if stat_function == 0 then
3:         return mean(filtered_column)
4:     else if stat_function == 1 then
5:         return minimum(filtered_column)
6:     else
7:                 ▷ Add other statistical functions as needed
8:     end if
9: end function
```

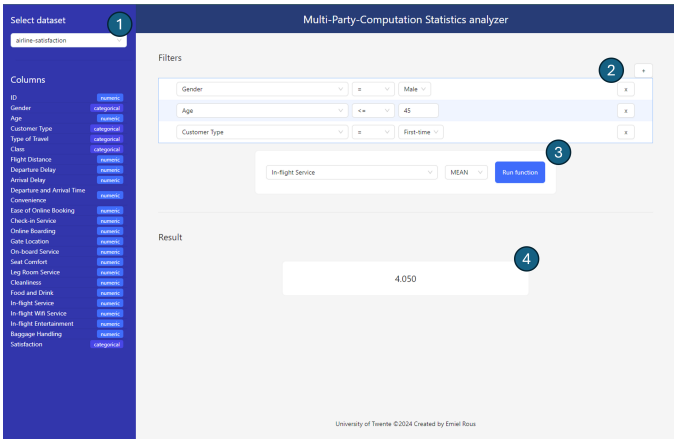

Figure 9: Web-page layout

5.1.3. *Front-end implementation*

Because the goal of this project is to create a modern, professional looking system, modern and state-of-the-art web development tools are used. The front-end runs on *Node.js* and uses *React* in combination with *ant-design* as a front-end framework. *React* is developed by Facebook and is the most popular UI framework for web development (Krotoff, 2023). *ant-design* is a popular *React* component library. An overview of the webpage can be seen in Figure 9.

The application consists of a single web page, which has the following functionalities (these correspond to the numbers in Figure 9):

1. Select a database to be analyzed
2. Create as many filters as needed on any column in the database
3. Select a function and a target column to execute the function on
4. Display the result

5.2. *Results: Trust*

Within the Sharemind deployment, one can identify three components that each add to the trust in the whole system. These are: trust in the technology itself, trust in the developer of the code, and trust in the miner-parties. Trust in the technology itself means trust in the cryptographic primitives, the protocols, the security assumptions, and the understanding of these concepts. Trust in the developer means trusting that the program

written by the developer will not reveal sensitive information, either on purpose or by accident. Trust in the miner-parties means trusting two parties will not collaborate to reveal sensitive information. These are three variables that all play a role in the amount of trust the user may put in the system.

Which parties will eventually host the calculation nodes is hard to determine without having the complete context of how the system will be deployed up-front. The parties are likely to play an enormous role in the assessment of trust in the whole system, since the context of a software system is potentially more important than the system itself (Davis, 1989)(Misiolek et al., 2002). This is an important thing to mention, but the question of which parties will host a production environment is not the focus of this thesis, so the focus will be on factors that can be influenced by the design. The factors that can be influenced are the trust in the developer and the trust in the technology, which will be discussed in more detail.

Using the framework of Riegelsberger et al. (2005) one can argue about what increases trust in the developer. Since this is a master thesis written by a student, one can argue that there is a strong *institutional* context that increases trust. The work that is done will be checked and graded, a student must adhere to ethical guidelines of the university, and the end goal is not profit or valuable data, but increasing the knowledge in this field (and a good grade). Although the *institutional* context is worth mentioning, it is something that is implied and cannot be influenced by the choices in the design of the system. The other two contextual properties (*social* and *temporal*) do not influence the assessment of trust in this case, because there is no direct incentive to create other systems in this setting, nor is there a reputation to uphold. However, the creator of Sharemind may have a reputation to maintain, so knowing that the tool used is from an established company that has reputable partners may help in increasing the trust through the *social* context. The intrinsic properties *benevolence* and *internalized norms* do not play a role for the student writing this thesis but may, in a similar line of reasoning as before, be relevant to the creator of Sharemind.

Ability is therefore the only property that can be influenced through the choices that will add to the trust assessment of the system. By designing a system that shows a high ability, it will positively influence the trust of the user. Research on designing for trust can help to establish which factors influence this. In Karimov and Brengman (2011) the authors identified two dimension of design: *graphics* and *structure*. An important factor for graphical design is good style (Everard and Galletta, 2005), where good style refers to the *visual design*, *layout*, *typography*, and overall *aesthetic appeal* of the website. Moreover, studies have shown that the perceived credibility of a website is significantly influenced by its aesthetic quality, confirming *good style* a key factor in interface design Fogg et al. (2003). *Color* could also be a factor, as a study between blue and green found that blue induces more trust (Lee and Rao, 2010). A study between blue and red found no correlation (Hawlitschek et al., 2016), so blue and red are equally trustworthy, or color does not have a significant influence compared to other factors such as context. The other dimension mentioned by Karimov and Brengman (2011) is the *structure* of the website. *Completeness of information* is an example of a website structure and refers to the ability of a user to learn the important information needed to complete a transaction (Everard and Galletta, 2005). *Navigation* which leads to a positive user experience makes an interface more trustworthy, as well as *clarity* and a clear *visual hierarchy* to guide users smoothly through the content (Nielsen and Loranger, 2006).

To improve trust in the technology it is important to create a system that is *reliable*, *functional*, and *helpful* Mcknight et al. (2011). This means that the system will need to be tested for reliability, needs the functionality to be able to execute the requests a user may have, and needs to have a help function or some other tool to inform the user about the functionality and help them troubleshoot. Looking at the two factors of website design that were discussed earlier (*structure* and *graphics*) one can see that there is some overlap with the *helpfulness* and *functionality* of the framework by Mcknight et al. (2011). The *graphics* dimension (*aesthetic appeal*, *color*, *typography* and *layout*) are all factors that play into the *functionality* of the system, which correlates to perceived competence in human terms, and the *structure* dimensions (*complete information*, *navigation*, *clarity* and *visual hierarchy*) can be seen as increasing the *helpfulness* for the user.

To conclude, to make the system itself as trustworthy as possible the system needs to be *reliable*, *functional*, and *helpful*. *Helpfulness* can be achieved by designing a good *structure* of the website to improve *completeness of information*, *navigation*, *clarity* and *visual hierarchy*. To improve the perceived *functionality* of the system, as well as the perceived *ability* of the developer, the website needs to have a *good style*, for which *layout*, *typography* and *aesthetic appeal* are important. *Reliability* can be improved by limiting the number of bugs in the system and performing many test runs, but since this is only a prototype, the true *reliability* can only be tested in a fully deployed system.

5.2.1. Implementation in prototype

In the previous section two main dimensions have been identified in website design: *graphics* and *structure*, which have overlap with*helpfulness* and *functionality* respectively. For *graphics*, *good style* and *aesthetic appeal* is important, which was considered in the design of the web interface, for example, by the choice of a modern component library. This thesis is not a design project, so *good style* was not tested by classical means, such as user interviews or surveys. Instead, by choosing modern frameworks, which have a high popularity, one can try to argue about the shared appeal they possess. A framework like ant-design follows certain design principles outlined by some scientific research in the field of human-computer interaction. These include visual consistency across all elements (Shneiderman and Plaisant, 2010) and having predictable and understandable interfaces [3] (Norman Donald, 2013). By using the out-of-the-box components one can somewhat assume that they could be aesthetically appealing. Typography is another

[3] https://ant.design/docs/spec/values

	Factor	Fulfillment
Style	Color	Blue
	Aesthetic appeal	Modern framework
	Typography	Commonly used fonts
	Layout	Commonly used layout (sidebar, top-bar)
Structure	Completeness of information	Graying components, loading symbol
	Navigation	Revealing components when needed
	Clarity	Error messages, color scheme
	Visual hierarchy	Top down, left to right

Table 2: Design factors and their fulfillment in the prototype

factor, and the default typography components of ant-design use modern popular fonts such as *Roboto* and *Segoe UI*, which are widely used throughout the web. For layout, the choice was to use a sidebar and a header, both components that are present in many websites and thus familiar to most users. The color blue was chosen as the main theme because it may influence trust. With these design choices, a web interface has been created which can be assumed to have *good style*.

The website *structure* is about the user *navigation*, providing *complete information*, *clarity* and *visual hierarchy*. In the design of the website special care has been put into displaying informational messages, graying out unused components, and visually guiding the user towards which steps need to be taken by ordering elements from left to right and from top to bottom. This ties together with *clarity* and *visual hierarchy* respectively, which are important to improve *helpfulness*. Each component and each message are colored according to severity or use (gray is used to display additional information about what the user should do and red is used for error messages), and there are features such as loading symbols and grayed-out loading components that help the user obtain *complete information*, leading to a more *positive user experience*. For an overview of the design choices in relation to the factors that influence trust, consult table 2.

Reliability was tested through many end-to-end tests once the system was finished. This helped reduce the number of bugs in the front and back-end code. However, all of the deployment and development have been done on one machine, and in real-world applications there are delays, unforeseen timeouts, and probably other unforeseen problems.

6. Discussion

The goals of this were to develop a prototype of a flexible MPC system and to tackle the question of trust in MPC. However, there are some shortcomings and recommendations that will be discussed in this chapter. The first part of this chapter is about the development of the system, and the second part addresses trust and some of the assumptions made in the arguments from previous chapters.

6.1. Prototype

The prototype was deployed on a single system, which means that the actual production deployment may look very different in terms of security and efficiency. For example, there is no access structure in place in the front-end. This means that anyone can access the website and potentially run any kind of query. In a real-world deployment, an access structure would be in place and it would be linked to the gateways, ensuring only authorized accounts are able to execute certain functions. The way the system is set up now is very open and allows anyone with access to run the *run-query* file, giving them unlimited access to all functions. Ideally, there would be account access for specific datasets, functions, and filter options. When looking at real-life deployment, other factors like securing the connections and secure user systems also need to be considered.

Another drawback, which results from the flexibility of the system, is that running statistical functions on subsets of the data, even with minimum data guard in place, may result in the reconstruction of some of the data or at least some of its patterns. To ensure flexibility does not come at the cost of privacy, it is recommended to mathematically prove the security of the statistical function so that they cannot be used to reconstruct data and to determine the best minimum data guard for each of them. Additionally, automatic filtering and automatic running of the queries may result in data collection at such a scale that data patterns of the data itself could potentially be reconstructed. To limit the potential for this, an option would be to limit the number of queries a user can make. The recommendation is to mathematically determine the number of queries a user can make in a given time-frame so that it becomes impossible to reconstruct the patterns.

The choice of dataset may also influence the security requirements of the system. Certain data may be more sensitive than others, so the security requirements may be higher. In addition to this, a dataset that has many labels for one of its categorical columns is able to segregate the data quite well and could, in combination with other filters, reveal some sensitive information even with the data guards in place. This would also require a mathematical approach to ensure that this cannot happen.

Lastly, the system requires three parties to host the environment, making this potentially a major point of failure in real-life deployment. As mentioned in the background chapter 3.1, the system is secure against a dishonest minority, so ensuring that no two parties will collaborate is vital, which could, for example, be done through contractual agreements.

6.2. Trust

Different factors and components related to trust have been identified. In this case, the only components that could be influenced were trust in the developer and trust in technology by creating an interface that follows the guidelines for a trustworthy system. However, in the context of the whole system, this trust is likely only a small part of the whole equation. The hosts

of the calculation nodes are likely the most important factor that influences trust. The recommendation is therefore to choose hosting parties carefully and to determine their trustworthiness based on their *temporal*-, *social*- or *institutional*-contexts.

The factors a developer can influence are only the way the system looks, which can affect someone without much technological knowledge. However, they may not understand the effect of the developer, and therefore do not even realize that the developer is a part of the chain that affects the security of the system. As mentioned in Section 5.1, ensuring that hosting parties do not collaborate is vital to maintain data privacy, but communicating the choices for these parties to the user may be important in gaining their trust. The user needs to be able to trust the hosting parties and trust that they do not collaborate.

Next, the arguments made for gaining trust in the developer of the interface hinges on the assumption that the interface follows design principles which are related to trust but which were not tested in a controlled environment. There is, for example, the assumption that the interface has some aesthetic appeal, which would increase trust in some systems, but perhaps the interface is not aesthetically appealing to everyone, or perhaps the users of MPC may not be influenced in the same manner as, for example, the average e-Commerce user. To verify the undoubted influence of design and structure on the trust people are willing to put into MPC, specific studies would have to be conducted with controllable variables. In addition to that, research on the different components of an MPC system and their relative weight in the trust equation needs to be investigated. If it turns out that effect of the interface design on trust is almost negligible compared to the knowledge and trust of the hosting parties, it does not make sense to spend significant effort on design and it would be more logical to look at how to choose the hosting parties.

There is not much work on trust in MPC, so a comparison with AI was made to give some indication of what might become important when investigating trust in MPC. However, this is merely what it is: an indication. There are some similarities between MPC and AI, but also many differences. For example, what are the components that a user needs to trust and what exactly does trust in AI mean? Does it mean that the user trusts that the AI will reach the right solution, or does it mean that the user trusts that no one can learn their private data? The research analyzed in this thesis did not always make this distinction clear. What component of the system is trusted is an important question because MPC is concerned with privacy and AI not necessarily so. There are situations when AI deals with private data and safeguards need to be in place to prevent reconstructing or obtaining private data, but in a lot of use-cases of AI privacy does not seem a problem.

Lastly, for trust, it is important that the system is reliable, but this was only tested in a sandbox environment. When deploying the system on a larger scale, there are many new factors which may influence the reliability, such as message delays or loss of connection. To ensure that the system can be trusted as a whole, the reliability of the deployed system needs to be tested.

7. Conclusion

7.1. RQ1 and RQ2: Prototype

The first goal of this thesis was to develop an MPC system that is flexible enough to be used on different datasets by users with different questions for those datasets. The first two research questions were about creating a back-end (RQ1) and creating a front-end (RQ2). Both of these had to fulfill the requirements posed by Fraunhofer. To reiterate, these were:

1. There needs to be support for statistical calculations, such as the mean and variance and possibly other descriptive statistics.
2. The system needs filters and flexibility, so each party has the freedom to create their own assessment of the other parties.
3. Input data needs to be persisted so that input parties are not required to stay online and so that data can easily be appended, updated so that changes can be tracked.
4. There need to be at least three parties, preferably more.

To prove that these requirements were met, the system was tested with three different datasets. These included different types of data, such as categorical, numeric, and temporal data. Each of these datasets could easily be uploaded, the filters are flexible and use the columns of the selected dataset and switching between the datasets is done with one drop down menu.

In summary, this thesis contributes to the field of MPC by showing that a framework like Sharemind can be used to develop modern and deployable software that leverages the security guarantees of the field in addition to being flexible enough to handle different data types and allow the user to develop their own queries. The system is implemented in such a way that it can serve as a starting point for a more complex system by easily allowing the addition of more statistical functions and data sets.

7.2. RQ3: trust

The second question of this thesis was about trust in a (new) technology like MPC. There are different factors which influence trust in humans, institutions, and systems. In the case of this MPC system, different components were identified, each of which have a different trust equation. These are: trust in the technology, trust in the developer, and trust in the hosting parties. The component which could not be influenced in this thesis are the hosts of the production environment.

Trust in the technology is increased by making a system that is *reliable*, *functional* and *helpful*. *Reliability* can be achieved by testing the system and reducing the number of bugs and downtime. *Functionality* can be improved by creating a system that can fulfill the request of users. *Helpfulness* is improved by guiding the user through the product and making it clear what the user has to do.

The trust in the last component, the designer, can be influenced through interface design. Following certain principles on *structure* and *graphics* the interface can communicate a strong *ability* of the designer, thus increasing the trust the user will

have in the system. For *structure* important factors are *completeness of information*, *navigation*, *clarity* and *visual hierarchy*. For *graphics* important factors are *layout*, *typography* and *aesthetic appeal*

By following proper design guidelines, the trust in the developer is increased, but there is also some overlap between these guidelines and the factors that influence the *functionality* and *helpfulness*. The *graphics* of an interface influence trust by increasing the *functionality* (i.e. *perceived competence*) of a system. The *structure* helps increase *helpfulness* of the system by guiding the user toward their goal.

In short, this thesis contributes by showing that there are several factors that influence trust in a software system. By highlighting the different components in a framework such as Sharemind, it was shown that each of these may be important in increasing trust. There are factors in interface design which can increase trust and developing a system following these guidelines will result in an overall more trustworthy system.

References

Agahari, W., Ofe, H., de Reuver, M., 2022. It is not (only) about privacy: How multi-party computation redefines control, trust, and risk in data sharing. Electronic markets 32, 1577–1602.

Alexandra Institute, 2023. FRESCO - A Framework for Efficient Secure Computation. `https://github.com/aicis/fresco`.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al., 2001. Manifesto for agile software development .

Becker, S., Duplys, P., Graf, J., Graffi, K., Grassi, A., Greven, D., Grewe, J., Jain, S., Klenk, T., Matyunin, N., Modica, H., Raskin, V., Scherer, P., Suschke, V., Trieflinger, S., Vlasakiev, V., Weinfurtner, J., 2021. Carbyne Stack. doi:10.5281/zenodo.8192460.

Bogdanov, D., 2007. How to securely perform computations on secret-shared data. Mater's Thesis Publisher: Citeseer.

Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., 2013. Secure multi-party data analysis: end user validation and practical experiments. Publication info: Preprint. MINOR revision.

Bogdanov, D., Laur, S., Willemson, J., 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations, in: Jajodia, S., Lopez, J. (Eds.), Computer Security - ESORICS 2008. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 5283, pp. 192–206. doi:10.1007/978-3-540-88313-5_13.

Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J., 2012a. High-performance secure multi-party computation for data mining applications. International Journal of Information Security 11, 403–418. doi:10.1007/s10207-012-0177-2.

Bogdanov, D., Talviste, R., Willemson, J., 2012b. Deploying Secure Multi-Party Computation for Financial Data Analysis: (Short Paper), in: Financial Cryptography and Data Security. Springer Berlin Heidelberg. volume 7397, pp. 57–64. doi:10.1007/978-3-642-32946-3_5.

Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X., 2010. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics, in: 19th USENIX Security Symposium (USENIX Security 10).

Canetti, R., 2001. Universally composable security: a new paradigm for cryptographic protocols, in: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145. doi:10.1109/SFCS.2001.959888.

Choung, H., David, P., Ross, A., 2023. Trust in AI and its role in the acceptance of AI technologies. International Journal of Human–Computer Interaction 39, 1727–1739. Publisher: Taylor & Francis.

Davis, F.D., 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly 13, 319–340. doi:10.2307/249008. publisher: Management Information Systems Research Center, University of Minnesota.

Everard, A., Galletta, D.F., 2005. How presentation flaws affect perceived site quality, trust, and intention to purchase from an online store. Journal of management information systems 22, 56–95. Publisher: Taylor & Francis.

Fogg, B.J., Soohoo, C., Danielson, D.R., Marable, L., Stanford, J., Tauber, E.R., 2003. How do users evaluate the credibility of web sites? a study with over 2,500 participants, in: Proceedings of the 2003 conference on Designing for user experiences, pp. 1–15.

Glikson, E., Woolley, A.W., 2020. Human Trust in Artificial Intelligence: Review of Empirical Research. Academy of Management Annals 14, 627–660. doi:10.5465/annals.2018.0057.

Hailemichael, M.A., Yigzaw, K.Y., Bellika, J.G., 2015. Emnet: a system for privacy-preserving statistical computing on distributed health data .

Hawlitschek, F., Jansen, L.E., Lux, E., Teubner, T., Weinhardt, C., 2016. Colors and Trust: The Influence of User Interface Design on Trust and Reciprocity, in: 2016 49th Hawaii International Conference on System Sciences (HICSS), IEEE, Koloa, HI, USA. pp. 590–599. doi:10.1109/HICSS.2016.80.

Karimov, F.P., Brengman, M., 2011. The effect of website design dimensions on initial trust: A synthesis of the empirical literature 12.

Keller, M., 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, ACM, Virtual Event USA. pp. 1575–1590. doi:10.1145/3372297.3417872.

Kivijarvi, H., Leppanen, A., Hallikainen, P., 2013. Technology Trust: From Antecedents to Perceived Performance Effects, in: 2013 46th Hawaii International Conference on System Sciences, IEEE, Wailea, HI, USA. pp. 4586–4595. doi:10.1109/HICSS.2013.510.

Krotoff, T., 2023. Front-end frameworks popularity (React, Vue, Angular and Svelte).

Lapets, A., Jansen, F., Albab, K.D., Issa, R., Qin, L., Varia, M., Bestavros, A., 2018. Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities, in: Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, ACM, Menlo Park and San Jose CA USA. pp. 1–5. doi:10.1145/3209811.3212701.

Lee, S., Rao, V.S., 2010. Color and store choice in electronic commerce: the explanatory role of trust. Journal of Electronic Commerce Research 11.

Mcknight, D.H., Carter, M., Thatcher, J.B., Clay, P.F., 2011. Trust in a specific technology: An investigation of its components and measures. ACM Transactions on Management Information Systems 2, 1–25. doi:10.1145/1985347.1985353.

Misiolek, N.I., Zakaria, N., Zhang, P., 2002. Trust in Organizational Acceptance of Information Technology- A Conceptual Model and Preliminary Evidence. rd Annual Meeting .

Nielsen, J., Loranger, H., 2006. Prioritizing web usability. Pearson Education.

Norman Donald, A., 2013. The design of everyday things. MIT Press.

Omrani, N., Rivieccio, G., Fiore, U., Schiavone, F., Agreda, S.G., 2022. To trust or not to trust? An assessment of trust in AI-based systems: Concerns, ethics and contexts. Technological Forecasting and Social Change 181, 121763. Publisher: Elsevier.

Riegelsberger, J., Sasse, M.A., McCarthy, J.D., 2005. The mechanics of trust: A framework for research and design. International Journal of Human-Computer Studies 62, 381–422. doi:10.1016/j.ijhcs.2005.01.001.

Rogers, E.M., 1986. Communication technology. Simon and Schuster.

Royce, W.W., 1987. Managing the development of large software systems: concepts and techniques, in: Proceedings of the 9th international conference on Software Engineering, pp. 328–338.

Saariluoma, P., Karvonen, H., Rousi, R., 2019. Techno-Trust and Rational Trust in Technology – A Conceptual Investigation, in: Barricelli, B.R., Roto, V., Clemmensen, T., Campos, P., Lopes, A., Gonçalves, F., Abdelnour-Nocera, J. (Eds.), Human Work Interaction Design. Designing Engaging Automation. Springer International Publishing, Cham. volume 544, pp. 283–293. doi:10.1007/978-3-030-05297-3_19.

Schoenmakers, B., 2018. MPyC—Python package for secure multiparty computation, in: Workshop on the Theory and Practice of MPC. https://github.com/lschoe/mpyc.

Shneiderman, B., Plaisant, C., 2010. Designing the user interface: strategies for effective human-computer interaction. Pearson Education India.

Tai, P.D., Anderson, M.R., Hien Duc, T.T., Thai, T.Q., Yuan, X.M., 2022. Strategic information sharing in supply chain with value-perceived consumers. Industrial management & data systems 122, 841–863. Publisher: Emerald Publishing Limited.