UNIVERSITY
OF OSLO

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment

Master's thesis

Private Database Search

Benjamin Hansen Mortensen

Information Security
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo

Cybersecurity and Cyberoperations
Strategic Analyses and Joint Systems division
Norwegian Defence Research Establishment

19th June 2024

Benjamin Hansen Mortensen

Private Database Search

Supervisors:
Martin Strand
Håkon Jacobsen

Abstract

An apparent deadlock arises when a client wants to perform a search containing classified information on a database of classified records, stored on a server, as in order to perform the search, one of the parties has to give up classified information to the other unwillingly. From the literature, we know that this situation is resolvable with the use of (Secure) Multi-Party Computation (MPC), also known as just secure computation, as it allows the client and server to privately supply their inputs to a computation, in this case a database search, without revealing it to the other party. Together, we explore the case of the Passenger Name Record (PNR) registry and consider whether using MPC for this purpose is feasible in a two-party setting with communication over Local Area Network (LAN) and with security against semi-honest adversaries.

In this work, we introduce the new idea of Private Database Search (PDS), which extends the better-studied problems of Symmetric Private Information Retrieval (SPIR) and Oblivious Transfer (OT) to achieve a search functionality of the database for the client.

We propose a new private keyword search protocol that utilizes an inverted index matrix for enhanced efficiency. We also show how semantic search functionality can be achieved in MPC using a Large Language Model (LLM), but efficiency remains a challenge. Additionally, we introduce an OT protocol that combines oblivious sorting with the new concept of Private Swap (PS) to achieve oblivious retrieval of records. The integration of the search and retrieval protocols culminates in a PDS protocol that we use to address the question of feasibility.

We successfully implemented the proposed PDS protocol and demonstrated the practicality and effectiveness of using MPC for private database searches. With our implementation, we show that our protocol can support a database size in the order of megabytes, and we project that this capacity can be further increased to gigabytes with a different PS protocol.

This work marks the first step towards a suitable solution in the case of the PNR registry, and we observe that the involvement of jurists and others is necessary to define the proper requirements so that we can increase the capabilities of and the cooperation between the security services.

Acknowledgements

I want to thank my supervisors, Martin Strand and Håkon Jacobsen, for their continued support and assistance, and the Norwegian Defence Research Establishment (FFI) for enabling this work. I am incredibly grateful for my main supervisor, Martin, who facilitated this work and gave me trust and freedom to explore. He is a cornerstone in the Norwegian cryptology community who wants nothing but what is best for the community and those around him. I am lucky to have been able to work with you. My co-supervisor, Håkon, has been immensely helpful with giving feedback and suggestions for my work, and it would not have been the same without you. Thank you. I am also grateful to the developers and maintainers of MP-SPDZ, as this thesis would not have been possible without it. Thanks also to FFI, whom I wrote this thesis in partnership with, for providing me with the resources that I required and a platform to engage with subject experts. Special thanks to the cybersecurity and cyberoperations division at FFI and everyone I engaged with during my time there. Lastly, thanks to the University of Oslo (UiO) and the Department of Informatics (IFI) for all their efforts towards making the Information Security master's program such an excellent program.

Contents

Chapter 1

Introduction

By Norwegian law, the different Norwegian security services are only allowed to investigate and collect types of intelligence within the scope of their jurisdiction. However, intelligence gathered by the other security services could be valuable depending on the investigation. The problem arises when the intelligence collected in an investigation is classified, and the potentially critical intelligence gathered by another security service is also classified, leading to an apparent deadlock where intelligence is either not shared or unlawfully shared. Together, we examine the case of the Passenger Name Record (PNR) registry and show that such an apparent deadlock is avoidable. We then give a thesis statement, share the contributions of this work, and introduce the experimental setup.

1.1 The Passenger Name Record Registry

Chapter 60 of Politiregisterforskriften introduced the PNR registry with regulation changes FOR-2022-04-29-646. Its purpose, described in section 60-1 of the regulation, is to contribute to preventing, discovering, investigating, and legislating acts of terror and severe crimes by storing PNRs collected from airlines in a registry. A PNR is a record containing information about passengers for a given itinerary. section 60-5 outlines the information a PNR can contain, including a passenger's name, phone number, address, or luggage information. The interesting subsection to us is subsection 60-6 (3) as records are allowed to be shared with other competent authorities, a term which includes most of the security services, given sufficient justification. We can imagine a case where this justification contains classified information, thus preventing it from being shared.

Each security service is subject to auditing to ensure they adhere to regulations. By the nature of auditing, we can infer that they behave honestly, but honest behavior does not prevent curiosity. Thus, appointing a trusted person to review the justifications containing classified information could be too risky for some authorities, leading them not to take advantage of the PNR registry. We refer to this as the problem throughout our work, and it implies one of three options: 1) The purpose of the PNR registry, section 60-1, cannot sufficiently be fulfilled. Alternatively, 2) there are contradictions between the regulations that some security services are subject to and Politiregisterloven. Or, 3) the current solution used to process requests hinders cooperation and limits the usefulness of the PNR registry. We assume option 3) and want to identify what an improved solution should provide.

We describe the PNR register as a database stored on a server. The client's user is a security service that performs a search on the database and retrieves the resulting record(s), where:

1. The client does not learn anything about the records on the database except for the records from the search.

2. The server does not learn anything about the search and the retrieved records.

For the solution's functionality to be most helpful it should allow for integration with various types of searches that are present in modern search engines, we want two types:

1. Keyword search

2. Semantic search

This work focus on the feasibility of solutions, which does not limit us to finding the best solution but allows us to explore possible practical solutions. We, therefore, implicitly factor in additional aspects like ease of implementation, multipurpose, user-friendliness, simplicity, security (with an emphasis on post-quantum security), functionality, and expandability. These crucial aspects must be integrated into the design to lay the groundwork for future research.

We summarize everything into the following statement: It is feasible to use secure computation to enhance the privacy and capability of queries on the PNR registry compared to today's disclose all approach, with respect to Politiregisterforskriften chapter 60, under the constraints of a two-party setting with communication over Local Area Network (LAN) and with security against semi-honest adversaries.

1.2 Contributions

Motivation for Oblivious Transfer (OT) and Symmetric Private Information Retrieval (SPIR) often use example use-cases similar to ours [32]. However, to the author's best knowledge, no publication has yet been published about implementations for said use-cases. Therefore, the most valuable contribution of this work is that we give an in-depth dissertation of a real-world use case, the PNR registry, where we use OT as a practical solution.

In addition, contributions in this work include:

- We present Private Database Search (PDS), which extends OT, SPIR, and Private Information Retrieval (PIR) (given that the underlying PIR protocol does not reveal information other than the retrieved records) to facilitate a search functionality of the database for the client.
- We propose a private keyword search protocol that has an efficient initiation, and in the online phase has a constant round, time and communication complexity.
- We introduce how a Large Language Model (LLM) can be combined with (Secure) Multi-Party Computation (MPC) to achieve a private semantic search protocol.
- We present two Private Swap (PS) protocols, one protocol utilizing symmetric encryption and generic MPC, and one OT-based protocol, that allows for the private swap of two elements.
- We propose an OT protocol, that combines oblivious sorting and PS, to achieve a quasilinear time complexity without parallelization, but with perfect parallelization the time complexity can be reduced to sublinear, and in the online phase has a constant round, time and communication complexity.

1.3 Experimental Setup

An essential part of verifying the feasibility of protocols is implementing them as programs to use in experiments. These experiments aim to gather metrics that describe the various aspects of the solutions. The metrics are necessary to expose the overall practicability and to see the implications of design choices. They also reveal potentially missed or new challenges as we remove layers of abstractions when implementing a protocol as a program.

To not present misleading experimental results, we choose to use consumer-grade hardware. We perform the experiments on a laptop with a four-core Central Processing Unit (CPU) with eight threads and a Random Access Memory (RAM) capacity of eight gigebytes. In software, we implement algorithms using Python 3.10 and MP-SPDZ [26] 0.3.8. In each experiment there are two parties, a client and a server, that both are run locally on a single laptop to eliminate most network instabilities and bottlenecks. We use the default value for the security parameters $\lambda = 40$ (bits) in MP-SDPZ.

Chapter 2

Theory

Ideally, we hope to find in the literature a perfect candidate solution that optimally solves our problem, which is easier said than done, so we should be methodical about our search. The area of research where such a solution would be present is in the domain of MPC, an area in cryptology that focuses on secure computations. MPC is very broad and thus consists of many subfields focusing on specific computations. We explore the subfields relevant to our problem, narrowing the scope as we go, starting with exploring the subfields of Oblivious Pseudorandom Function (OPRF) and Private Set Intersection (PSI), as combining the two gives us a structure to assess different parts of potential solutions. Given developments in other subfields, we suspect that swapping out the latter part of the structure with other methods of retrieving records could yield a more functional solution, therefore, we also examine the subfields of Oblivious Random Access Machine (ORAM) and PIR. ORAM would allow us to read and write data, which is neat as the database could continuously be updated with new records even after the initiation. On the other hand PIR intrinsically respects the imbalance between a client and a server, which is a useful aspect as only the server could require excessive hardware costs while weaker clients could easily be added. Inherently, the PSI and ORAM/PIR structure falls under a different subfield OT, so we naturally explore that as well. The goal is to understand the nuances between the subfields and identify attributes our solution should exert. Therefore, the discussion outlining what acceptable solutions to our problem is will be integrated with the theory discussion and is had when seemed fit.

2.1 Multi-Party Computation

MPC is cited to the works of Andrew Yao [41, 21] on his idea of garbled circuits. At a high level of abstraction, it allows a set of parties to privately compute a function based on their inputs without revealing the function computed or the other parties' inputs. The function's output can be revealed to selected parties. Along with Yao's work, the work of Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson [6] is cited as fundamental to MPC as it gives theorems on the thresholds of parties that can be corrupted while an arithmetic function still can be computed with perfect security. In the same paper, they present the BGW protocol that utilizes linear secret sharing [30], which is notable as it differs from Yao's approach. Since their work, there have been many optimizations and new protocols, such as the SPDZ protocol [12] that uses Somewhat Homomorphic Encryption (SHE) in the preprocessing for a faster online phase. However, fundamentally, most protocols use binary and arithmetic circuits to represent the function that is computed [17, 27, 23, 28]. It is essential to highlight that these protocols are referred to as generic MPC protocols due to their ability to compute any function, so specific MPC protocols might not utilize a circuit approach due to their limitations.

Because MPC facilitates the secure computation of a function between a set of parties, it is also essential to protect against different adversarial behaviors that one or more parties might exhibit. There are three main adversarial models: semi-honest, covert, and malicious, but the literature mainly focuses on semi-honest and malicious. Semi-honest adversaries supply the correct inputs to the protocol and do not deviate from the protocol description in any way. They try to break the protocol by passively using the information they learn throughout the execution to learn more about the computation and secret values. Protection against this behavior becomes tricky in a setting with multiple semi-honest adversaries as the adversaries could cooperate and combine secret values to try to break the protocol.

All in all, the general security idea behind a protocol that has protection against semi-honest adversaries is that it leaks no information. On the other hand, malicious adversaries are actively trying any strategy to break the protocol. They could give false inputs, manipulate values, or halt the computation; anything goes. Protection against malicious adversaries is often more costly than protection against semi-honest adversaries. In addition to the type of adversaries we are dealing with in MPC, we must also consider the number of corrupt parties. Generally, we use the terms honest-majority and dishonest-majority to indicate if the corrupted parties are a minority or a majority in the computation. Notably, the majority implies a strictly greater than relation.

Preliminarily, MPC assumes that the involved parties already are authenticated to one another and that they have established a secure channel for communication. Contrary to secure communication, which deals with adversaries listening in on their communication, in MPC, the adversaries reside as an active party. The MPC paradigm is a set of protocols, which, in their ideal sense, is like having a mutually trusted third party that would collect all the inputs and compute the function for us. MPC is, therefore, useful in cases where no such mutually trusted third party exists, which makes it great for validating and mitigating certain types of behavior of the involved parties.

To ensure we understand generic MPC, let us examine Yao's garbled circuits. We consider a two-party setting with semi-honest adversaries. A garbled circuit is a binary circuit representation of the function we want to compute where we garble each gate. It suffices to understand how one gate works to understand how it all fits together, so to understand what we mean by a garbled gate, let us consider the logic gate g_i; for simplicity, imagine that g_i is an Exclusive OR (XOR) gate. Table 2.1 shows the truth table for the gate, with wires w_0 and w_1 being the inputs to the gate and w_2 being the output wire.

input w_0	input w_1	output w_2
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: Truth table of gate g_i

The goal is to hide the values in the truth table by assigning keys k to represent each bit; we refer to these keys as labels. These labels are randomly chosen and only used once. The intuition is that someone who does not know the labeling cannot tell if a random value represents a zero or one. We present the truth table and its labeling in an ordered manner, but we shuffle the rows in practice. To garble the gates, we use an encryption scheme E to encrypt the output wire bit under the corresponding input wire labels. Remark that different labels representing the same output bit value are encrypted to different values as the keys used for encryption are not the same. This is important as if each garbled value did not look different an observer could infer which type of logic gate it is by looking at the ratio of similar values. For example, an AND gate would have three similar values and one different one, while an XOR gate would have two and two. This property is intrinsic to Yao's as it hides the computed underlying function. Table 2.2 shows the labeling of the gate where superscript denotes the bit value associated with the key, and subscript denotes the associated wire.

So far, we have explained one party's role in the process: the circuit garbler. Once it has garbled the circuit, it only sends the garbled values to the other party, named the circuit evaluator. It also creates labels for its input, which can be used with the circuit, and sends that as well.

input w_0	input w_1	output w_2	garbled value
$\mathscr{k}_0^0$	$\mathscr{k}_1^0$	$\mathscr{k}_2^0$	$E_{\mathscr{k}_0^0,\mathscr{k}_1^0}(\mathscr{k}_2^0)$
$\mathscr{k}_0^0$	$\mathscr{k}_1^1$	$\mathscr{k}_2^1$	$E_{\mathscr{k}_0^0,\mathscr{k}_1^1}(\mathscr{k}_2^1)$
$\mathscr{k}_0^1$	$\mathscr{k}_1^0$	$\mathscr{k}_2^1$	$E_{\mathscr{k}_0^1,\mathscr{k}_1^0}(\mathscr{k}_2^1)$
$\mathscr{k}_0^1$	$\mathscr{k}_1^1$	$\mathscr{k}_2^0$	$E_{\mathscr{k}_0^1,\mathscr{k}_1^1}(\mathscr{k}_2^0)$

Table 2.2: Garbling of gate g_i

The circuit evaluator cannot use its raw input with the garbled circuit; therefore, we translate it into labels the garbler picks. We use 1-out-of-2 OT, denoted OT_1^2, to perform the translations as it allows the garbler to supply two labels and the evaluator to choose which one it learns without revealing it to the garbler. The evaluator then evaluates the circuit as it has both inputs. However, it can only decrypt the row where it knows both labels, thus guaranteeing that it can only decrypt one row per gate successfully. As described so far, the encryption scheme E would have to be an Indistinguishable under Chosen Ciphertext Attack (IND-CCA) secure scheme such that the evaluator knows which decryption was successful, but using signal bits [34] we can make it so that E only has to be an Indistinguishable under Chosen Plaintext Attack (IND-CPA) secure scheme.

After the evaluator is done evaluating the circuit, depending on the choice of the garbler, it could be left with the output in plain binary or with a set of labels. These labels can only be translated by knowing the correct labeling, which allows the parties to decide who should learn the computation's output.

We consider computing the Advanced Encryption Standard (AES) function to give an example of how this is useful. Let us say that the first party P_1 constructs AES as a binary circuit $\mathcal{C}$. P_1 proceeds to garble the circuit and ensure the output wires on the output gates spit out labels; we denote the garbled circuit with $\mathcal{G}$. Before it sends the garbled circuit $\mathcal{G}$, it keeps the labeling for each gate for itself and only sends the garbled values for each gate. With $\mathcal{G}$, it sends a garbling of the plaintext it wants to be encrypted. P_2 receives $\mathcal{G}$ and P_1's input, and then picks an encryption key. P_1 and P_2 together perform several OT_1^2 to translate the encryption key such that it can be used with the garbled circuit. P_2 proceeds to evaluate the circuit, effectively encrypting the plaintext with its key. After the evaluation, the output labels are sent back to P_1, where they can be translated to reveal the ciphertext. What P_1 and P_2 have done is to obliviously encrypt P_1's plaintext under P_2's key without P_1 learning the encryption key (following the properties of AES) and P_2 not learning the plaintext or ciphertext. The following example describes an OPRF functionality that is achieved through the use of generic MPC. Understanding this example is useful as it is one of the basic parts of the protocol presented in Section 5.4.

Any function can be computed using binary circuits, but some drawbacks exist. Modern computers use CPU instructions to perform computations; this allows a process to choose which operations to compute selectively. An example would be that if the function computed contains an if-else statement, then depending on the conditions, only one block of the if-else is computed, allowing the process to branch. On the other hand, circuit representations are static descriptions of the function, which means that both blocks of the if-else must be a part of the circuit description. That is not to say that in regular computing, we cannot only execute one block of it depending on the inputs, but in secure computations, the main idea is to hide the contents of the computation; thus, only computing one block would reveal information about the computation, which transitively reveals information about the inputs. The challenge then becomes how we can compute sublinear functions without the possibility of branching.

Suppose that P_1 inputs a database as an array of values and P_2 a search query into a secure computation. The function we want to compute is a keyword search, and for simplicity, the search query is an index on the database. P_2 is the circuit garbler and P_1 is the circuit evaluator. In the secure computation, P_2 could not directly use the index, even if it does not learn it, to fetch the array item in constant time as P_2 could tell which item was accessed as it only performs operations on that cell. The problem arises as P_1 is the one who supplies the array and, therefore, knows its ordering. We could solve this problem by having P_2 shuffle the array without P_1 learning the order. This idea is the basis for ORAM, which utilizes a model of computation different from circuits, namely the RAM model. We therefore infer that to find sublinear solutions, we have to explore subfields of MPC that use the RAM model.

2.1.1 MP-SPDZ

In Chapter 4 and Chapter 5, we implement our proposed solutions. Critical parts of those solutions rely on generic MPC, therefore, we use MP-SPDZ [26] by Marcel Keller to implement them. This framework allows us to implement high-level code stored in .mpc files and compile them into low-level code that we execute as a secure computation. Integrated into MP-SPDZ are different variable types, like secret integers (sint), secret bit(s) (sbit(s)), and container types like arrays and tensors. One great feature is that it also supports Bristol Fashion format binary circuits [2], which allows us to directly describe a function as a binary circuit and use it for secure computation. The different parties involved are emulated in their process, and an SSH connection is established between them for communication, allowing parties to be run locally on the same computer or stand-alone on separate computers. This gives the unique ability to test in LAN and Wireless Local Area Network (WLAN) conditions. Overall, MP-SPDZ is a well-optimized platform for various experiments on secure computations, with loads of generic MPC protocols, called programs, from which to choose.

2.2 Private Set Intersection

OPRF was defined in the works of Freedman et al. [16], and in the same paper, they proposed a protocol to access records on a database by keywords contained in the records. The paper considers a similar problem to ours, and from it, we take away the idea of using PSI to perform a keyword search.

PSI was introduced by Michael J. Freedman, Kobbi Nissim, and Benny Pinkas [15]. It allows parties to input their elements into the protocol and learn the intersecting elements between the sets. In addition, a function could be computed at the intersection before revealing the output, a valuable functionality. An example of the use of PSI is that it allows online databases of leaked passwords to be hosted where users can submit their password to check if it is in the database without the user revealing their password.

To ensure that we understand how it works, let us explore a well-known PSI protocol introduced in the works of Catherine A. Meadows [35]. Let us consider the two-party semi-honest setting. P_1 is in possession of set X and P_2 is in the possession of set Y. We compute the intersection function $f(X, Y) = X \cap Y$, private keys e_1 and e_2 are elements in a group where the Decisional Diffie-Hellman (DDH) assumption holds, and $\mathsf{Hash}(x)$ as a hash function, modeled as a random oracle. The protocol goes as follows:

1. P_1 picks its private key e_1.
2. P_2 picks its private key e_2.
3. P_1 computes $\{\mathsf{Hash}(x)^{e_1} | x \in X\} = X^{e_1}$.
4. P_2 computes $\{\mathsf{Hash}(y)^{e_2} | y \in Y\} = Y^{e_2}$.
5. P_1 sends X^{e_1} to P_1.
6. P_2 computes $\{(x^{e_1})^{e_2} | x^{e_1} \in X^{e_1}\} = \{\mathsf{Hash}(x)^{e_1 \cdot e_2} | x \in X\} = X^{e_1 \cdot e_2}$.
7. P_2 send $X^{e_1 \cdot e_2}$ to P_1.
8. P_1 computes $\{(x^{e_1 \cdot e_2})^{\frac{1}{e_1}} | x^{e_1 \cdot e_2} \in X^{e_1 \cdot e_2}\} = \{\mathsf{Hash}(x)^{e_1 \cdot e_2 \cdot \frac{1}{e_1}} | x \in X\} = \{\mathsf{Hash}(x)^{e_2} | x \in X\} = X^{e_2}$.
9. P_2 sends Y^{e_2} to P_1.
10. P_1 computes $O = \{x^{e_2} \in X_2^e | x^{e_2} = y^{e_2}, y^{e_2} \in Y^{e_2}\} = \{\mathsf{Hash}(x)^{e_2} \in X^{e_2} | \mathsf{Hash}(x)^{e_2} = H(y)^{e_2}, \mathsf{Hash}(y)^{e_2} \in Y^{e_2}\}$.

The set O contains all the masked elements at the intersection. Thus, P_1 learns $f(X, Y) = X \cap Y$, while P_2 learns nothing.

In the protocol, we can envision that P_1 is a client in possession of a keyword x and that P_2 is a server with a database Y. We construct Y such that all the keywords of the records y_i are paired with the index i of the record it originates from. We follow the same approach as the protocol above, the client P_1 would find the intersection of x and Y, which enables it to learn the index of the record it is interested in. Thus, the client successfully performed a keyword search on the database without learning anything about the records other than whether or not its keyword existed, and the server learned nothing. This broad idea is the basis of the protocol presented in Section 5.4, just with a different PSI protocol.

We have now determined that we can perform the search functionality with a PSI protocol. However, this only solves one part of the puzzle, as we still need to transfer the records from one party to the other oblivious. We, therefore, explore ORAM, PIR, and OT as potential candidate subfields to give us a solution.

2.3 Oblivious Random Access Machine

Rafail Ostrovsky and Oded Goldreich introduced ORAM [20, 37, 22], and its primary functionality is that it hides the access pattern of a CPU on the RAM. At first glance, this might not seem relatable to our problem, but instead of thinking about a CPU accessing RAM in the modern day, we think of a client accessing encrypted data in the cloud. The functionality of hiding the access pattern is trivially solvable by having the client access all items, giving a linear overhead. However, to be considered an ORAM protocol, it has to have a non-trivial solution, thus sublinear overhead.

We look at the adversarial modeling in ORAM to understand what ORAM guarantees. “We consider an "adversary" who can monitor communication between CPU$_k$ and RAM$_k$ (but can not monitor conversations between either CPU$_k$ and "program" or between CPU$_k$ and random oracle.) The goal of the "adversary" is to learn as much information about the request sequence as possible.” [37] Effectively, the adversary is an outside observer of the communication, where there is no guarantee for the privacy of the CPU with respect to the RAM and vice versa. Distinctly, this implies that ORAM is only concerned with hiding how data is accessed, not the data itself; therefore, in many cases, it is implicit that the data stored is encrypted, allowing us to shift the adversary from an outside observer to be residing with the stored data.

If we consider this more modern view, we can rephrase the goal of ORAM to suit our problem better. ORAM hides a client’s access pattern on encrypted data from the server where the data is stored. We take a step back, and note that this only implies privacy for the client, and in our case, we need it both ways. The protocol presented in Section 5.2 and Section 5.3 is inspired by the classical ORAM protocol [22], referred to as square-root ORAM, by its use of oblivious sorting and dummy items. In it, we provide two-way privacy, effectively transforming it into an OT scheme, but one caveat is that we cannot provide sublinear data writing. In ORAM, the client can both read and write data to the server in sublinear complexity; this is due to the data being generated by the client, and the server merely acts as a storage device. On the contrary, in our case, the server generates the data. To see why this is so different, let us imagine that we have initiated the ORAM by obliviously shuffling and encrypting the data on the server under the client's conditions. If the server wants to add some new record to that structure, using some sublinear number of accesses by the client, it would tell the server that the new record exists among these few records that it accessed. Given n number of records, we want the server to have a $\frac{1}{n}$ probability of guessing right. Thus, inserting a new record requires at least a linear number of accesses, thereby effectively making the insertion problem equivalent in magnitude to initiating the structure in the first place.

2.4 Private Information Retrieval

In the information-theoretic setting, PIR was introduced in the works of Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan [11]. It lets clients retrieve records from a public database privately without the servers learning which records were retrieved. A trivial solution that achieves the same functionality is for the server to send the whole database to the client; therefore, PIR has non-trivial requirements, which say that the communication cost should be sublinear in the database size. This is a neat property as it explicitly respects that the client's storage is less than the server's, and implicitly, protocol designers focus on having the client do less computational work than the servers. In PIR's introduction, replicating the database across multiple servers was used. The client would then secret share its query to retrieve individual shares of the data from the servers; thus, each server never learns the data retrieved by the client, assuming that the servers do not collaborate. In the computational setting, Rafail Ostrovsky and Eyal Kushilevitz showed that PIR could be achieved with a single server [31]. PIR protocols can, therefore, be realized with one or more servers, but implicitly, they should support multiple clients (see Section 5.6.1).

For our problem, recall that we are in the two-party setting; therefore, assuming the presence of an adversary, an honest party would never be in the majority. Implying that if the database were to be replicated across several servers, effectively introducing more parties, the assumption that these servers do not collaborate to break the privacy in the protocol is equivalent to assuming a third mutually trusted party. To see why let us imagine that the database, in possession of party P_2, is split into server ζ_1 and ζ_2. We now have three parties: the client P_1 and servers ζ_1 and ζ_2. From the client's perspective, we want to guarantee protection against semi-honest adversaries. To give a proof by contradiction, we assume an honest majority, implying that either ζ_1 or ζ_2 are honest, but as they are derived by P_2, the only way for this to be possible is for either of the servers to reside with a third independent party P_3. This party has to be trusted by P_2; if not, P_1 and P_3 could collaborate to break its privacy and contradict our assumption of an honest majority. Hence, P_3 has to be an independent mutually trusted third party, which would make the use of MPC redundant as the core of the MPC paradigm is to facilitate this functionality in the lack of a third independent mutually trusted party. This shows that, indeed, we cannot have an honest majority, and therefore, it prevents us from replicating the database across more than one server.

Presume we add PIR to our structure, we can then use PSI to have the client perform private searches on the database, and then use PIR to retrieve the records from the server. This only works if the underlying PIR protocol does not reveal any information about the database other than precisely the records the client is after.

2.5 Oblivious Transfer

OT protocols come in many variants; the first variant of OT, referred to as Rabin OT, was introduced by Michael Oser Rabin [38]. It functions differently from what we usually think of today as OT, as it gives the receiver a 50/50 chance of receiving some message from a sender without the sender learning if the message was received. Today, the common intuition is that the receiver can pick some elements from a more extensive set in possession of the sender. The sender then transfers those elements to the receiver without learning which elements were transferred. In its most basic form, OT_1^2 [14], a receiver can learn one of two messages without the sender learning which message the receiver chose. From OT_1^2, one can build other variants of OT, like OT_k^n [25], which allows the receiver to choose k-out-of-n elements. It was shown that OT requires asymmetric assumptions for a black-box construction in the standard model [19], which is why OT-extension [4] is essential for extending OT_1^2 to OT_1^n and OT_k^n. OT-extension works by utilizing a few slow base OTs from asymmetric assumptions to achieve many efficient symmetric OTs. This is similar to the standard approach to secure communication, where slow asymmetric encryption is initially used to derive a standard shared key, followed by fast symmetric encryption for the remainder of the encryption of messages.

An introduction to OT would not be complete without mentioning that

OT is complete for secure computation [29], meaning that generic MPC can be realized from OT_1^2, making OT one of the most fundamental research topics in secure computation. Considering the functionality of OT, it most definitely would be used in our structure to retrieve records, as PSI would facilitate the search obliviously.

Regarding our problem, we want the functionality to be intuitive and natural to the end user. By that, we want the functionality to resemble modern search engines. In terms of efficiency, we want to perform successive efficient searches based on what we learned from the previous search. An OT protocol with such a property is called an adaptive OT protocol [36]. Further, depending on the identity of the user, the server could want to control the data that is accessible to that specific user. OT protocols with this property are said to have access control [9]. From the literature, we identify the protocol by Libert et al. [32] as a potential candidate solution. However, they do not show experimental results, and there are questions raised about efficiency [10].

Recall that PIR assumes a public database, and subsequently the client is assumed to know which records it is after. There is a variant called SPIR [18] that additionally provides privacy for the server, and therefore SPIR is a special type of OT protocol. The key element differentiating SPIR as a class of protocols is the insistence on the non-trivial property. SPIR is, therefore, a good candidate subfield to find solutions to our problem; however, as argued by Freedman et al., "Given that KS allows clients to input an arbitrary search word, as opposed to selecting p_i by an input i, keyword search is strictly stronger than the better-studied problems of oblivious transfer (OT) and symmetrically private information retrieval (SPIR)." [16] To the author's knowledge, a suitable post-quantum secure, adaptive keyword search protocol with access control is still not yet present in the literature.

Private Database Query (PDQ) [7] is a generalization of SPIR. It allows a client to submit complex queries to a database to retrieve records obliviously. An example of a complex query could be to ask for all records where the total number of people without a listed phone number is over a threshold. Mainly, PDQ aims to extend SPIR to allow for SQL-like query functionality. Relating to our problem, this functionality is most likely desirable compared to regular search functionality. However, the problem arises when considering that such a solution requires the end user to know, for the sake of argument, SQL in order to perform searches. We cannot assume this to be the case, and given the use case, such a functionality would be overkill. Thus, to the best of the author's knowledge, there is no PDQ protocol present in the literature that, on its own, offers a sufficiently simplistic query interface for non-programmer users.

Chapter 3

Private Database Search

It is not trivial to perform private searches on a private database, therefore we take a step back to outline the general idea behind the type of protocols we seek. For now, we are not concerned about the feasibility of such protocols; instead, we want to identify and describe them. We build on the work of Ryan Henry in defining PIR [24], we extend the notion of PIR by combining it with PSI to introduce the notion of PDS, a set of protocols that fall under the OT category. We do so by outlining what a database search protocol is and describe the properties to which a PDS protocol should satisfy. The goal is not to give formal definitions but to relate the descriptions of the properties in a natural way to the desired functionality, for then to later discuss the implications of those descriptions at a lower level of abstraction, Section 5.6.

3.1 What is a Database Search Protocol?

A database search protocol consists of six main Probabilistic Polynomial-Time (PPT) algorithms:

Setup takes the single database D, of length $n \in \mathbb{Z}^+$, with $m \in \mathbb{Z}^+$ number of records, as an input, and outputs a set of system parameters Ξ. The system parameters Ξ is a tuple which can contain variables such as a certificate, IP-address, domain name, database length and so on.

Filter take the database D and client's identity $\mathcal{U}$ as inputs, and outputs some auxiliary information $A \in \{0,1\}^\rho$, of length ρ, about the database.

Search takes the search query $q \in \{0,1\}^\chi$, of length χ, and A as inputs, and outputs a set of indices $I \subseteq \{1, 2, ..., n\}$, of length $\nu \in \mathbb{Z}^+$.

Encode takes the indices I as an input, and outputs the encoded indices $I' \subseteq \{1, 2, ..., n\}$ of length $\sigma \in \mathbb{Z}^+$.

Retrieve takes D and I' as inputs, and outputs the set F' of encoded records.

Decode takes the encoded records F' as an input, and outputs the retrieved records $F \subseteq D$.

The server starts with running $\mathsf{Setup}(D)$ and $\mathsf{Filter}(\mathcal{U}, D)$, then it sends Ξ and A to the client. From here on out, it performs $k \in \mathbb{Z}^+$ instances of transfers. The client picks a value for q_i then runs $\mathsf{Search}(A, q_i)$ to get the indices I_i. Next, the client constructs the encoded indices I'_i, by running $\mathsf{Encode}(I_i)$, and sends it to the server. The server responds by sending back the encoded records F'_i by running, $\mathsf{Retrieve}(D, I'_i)$. Finally, the client retrieves the records F_i, by running $\mathsf{Decode}(F'_i)$, it was searching for. The client can halt or continue to do another search by performing a new instance of a transfer.

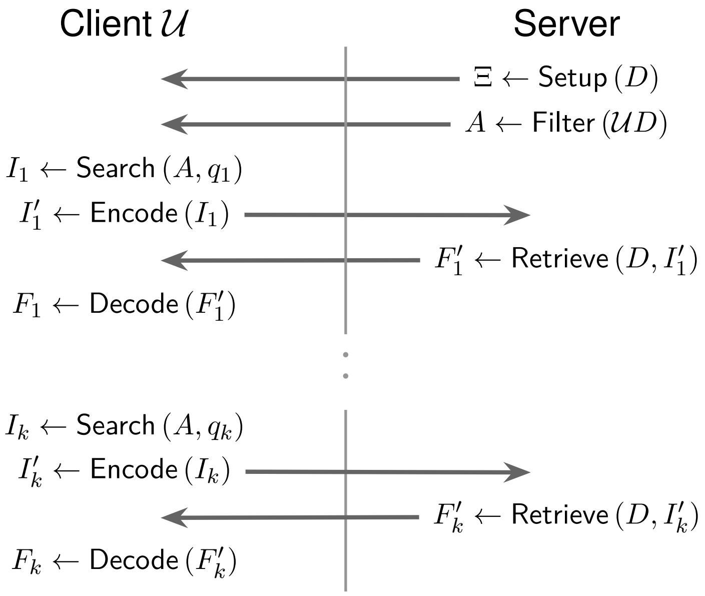

Client $\mathcal{U}$
Server
$\Xi \leftarrow \mathsf{Setup}(D)$
$A \leftarrow \mathsf{Filter}(\mathcal{U}D)$
$I_1 \leftarrow \mathsf{Search}(A, q_1)$
$I'_1 \leftarrow \mathsf{Encode}(I_1)$
$F'_1 \leftarrow \mathsf{Retrieve}(D, I'_1)$
$F_1 \leftarrow \mathsf{Decode}(F'_1)$
$I_k \leftarrow \mathsf{Search}(A, q_k)$
$I'_k \leftarrow \mathsf{Encode}(I_k)$
$F'_k \leftarrow \mathsf{Retrieve}(D, I'_k)$
$F_k \leftarrow \mathsf{Decode}(F'_k)$

Figure 3.1: The communication flow between the client and the server in a database search protocol.

We are not going into greater detail about the Setup algorithm. On the other hand, we discuss the remaining algorithms in great detail, as with these and some additional ones, is where we will transform the protocol.

3.2 Properties of a Private Database Search Protocol

The main challenge we face is that in OT, the client inherently knows the records it wants to retrieve. We solve this by introducing a searching step where the client can search the database and obtain the indices of the records it is interested in. We identified that a PSI protocol can be used to achieve this functionality, which we implement with the Search algorithm. To retrieve the records the Encode, Retrieve and Decode algorithms implement an OT protocol that retrieves the corresponding records for the indices.

To be precise about the functionality we desire we need to give descriptions of the properties of PDS protocols. Therefore, we take inspiration from the definitions of privacy and correctness [24] and introduce new descriptions for the adaptive, timely, access control, and storage imbalanced properties.

3.2.1 Correctness

A protocol is correct if the client gets the files it searched for. To be more precise, we expect a protocol to succeed with a probability equal to or greater than some negligible function,

$$\Pr\left[F \leftarrow \mathsf{Decode}\left(F'\right) \;\middle|\; \begin{array}{l} \Xi \leftarrow \mathsf{Setup}\left(D\right) \wedge \\ A \leftarrow \mathsf{Filter}\left(\mathcal{U}, D\right) \wedge \\ I \leftarrow \mathsf{Search}\left(A, q\right) \wedge \\ I' \leftarrow \mathsf{Encode}\left(I\right) \wedge \\ F' \leftarrow \mathsf{Retrieve}\left(D, I'\right) \end{array} \right] \geq 1 - \mathcal{E}\left(\lambda\right)$$

3.2.2 Privacy

The intuition of our privacy descriptions is that the client learns nothing more than the resulting records from the server and that the server learns nothing about the searches. We want the server, given a single transfer of a record, to have a probability of guessing which record was transferred equal to or less than $\frac{1}{n} + \mathcal{E}\left(\lambda\right)$, where n is the database length and $\mathcal{E}\left(\lambda\right)$ is a negligible function $\mathcal{E} : \mathbb{N} \mapsto \mathbb{R}^{+}$ describing the error probability.

We base the idea of our privacy description on the fact that both the search and retrieval of records are deterministic. Synonymous with semantic security is the IND-CPA game, but given that base assumption we modify the game by taking inspiration from the random oracle model. The resulting idea is that we have an adaptive game based approach, in the computational setting, where the adversary can query an encoding oracle which gives responses by running some algorithm or fetching them for previously seen queries. The core functionality of the game then simulates queries on some deterministic algorithm.

To describe privacy, we identify from Figure 3.1 that there are three exchanges, excluding Setup, between the client and the server for a single transfer. Therefore, we need three games to describe privacy. For simplicity, we introduce a new algorithm Garble that supplements the functionality of

Filter, which takes A as input, before it is sent to the client, and outputs the encoded auxiliary information A', which is sent to the client instead.

Algorithm 1: Server Privacy - The Search

```
Input: The database $D$
Output: Prediction $(b = b')$

$\Xi \leftarrow \mathsf{Setup}\,(D)$
$\mathsf{Oracle}\,(\Xi)$
$b' \leftarrow \mathcal{A}^{\mathsf{Oracle.query}(A_i)}$
return $(b = b')$

$\mathsf{Oracle}\,(\Xi)$:
    $H \leftarrow [\ \ ]$
    $b \xleftarrow{\$} \{0,1\}$
    function $\mathsf{query}\,(A_i)$:
        if $A_i$ in $H$ then
            $A_i' \leftarrow H.\mathrm{get}(A_i)$
        else
            if $(b = 0)$ then
                $A_i' \leftarrow \mathsf{Garble}\,(A_i)$
            else
                while $\$$ in $H$ do
                    $\$ \xleftarrow{\$} \{0,1\}^\rho$
                end
                $A_i' \leftarrow \mathsf{Garble}\,(\$)$
            end
            $H.\mathrm{add}(A_i, A_i')$
        end
    return $A_i'$
end
```

Server privacy in the search. Algorithm 1 describes the server's privacy in the search of a PDS protocol. In it, a PPT algorithm $\mathcal{A}$ represents an adversary whose goal is to demonstrate an advantage in distinguishing results from an oracle Oracle, a PPT algorithm. The oracle has a query functionality, which returns responses A_i' to a given auxiliary information A_i. Responses are given based on the value of a bit b that the oracle picks upon initiation. If the bit is zero, the oracle returns a proper encoding of A_i by running Garble. If the bit is one, the oracle picks a random substitute \$ for the auxiliary information to run Garble with instead. The oracle stores responses, which it gives in H. If the oracle is queried for a previously seen query, it fetches the stored response from H. Ultimately, the adversary gives its prediction b' to what value it believes the oracle's bit b has.

The intuition is that if the adversary, in this case, the client, cannot derive any correlation from A_i to A'_i, then the challenger, in this case, the server, perfectly hides A_i with Garble. Thereby enforcing the privacy of the database D in the search as A_i is derived from D. It might seem weird to make this claim as we did not run the Search algorithm, but we imagine that if Search broke the server's privacy, then it also would be a valid strategy for the adversary, thus the privacy of Search is also captured by this description.

We say that the server's computational privacy holds in the search if the adversary $\mathcal{A}$ has an advantage smaller than or equal to some negligible function.

$$\mathsf{Adv}^{\mathsf{Search}}(\mathcal{A}) = \left|2 \cdot \Pr\left[b = b'\right] - 1\right| \leq \mathcal{E}(\lambda)$$

Server privacy in the retrieval. Algorithm 2 describes the server's privacy in the retrieval of a PDS protocol. We follow the same setup as Algorithm 1, with an Oracle that has a query functionality. In this case the adversary $\mathcal{A}$ chooses sets of indices I'_i, of length σ, that it queries the oracle with. The oracle then considers each index by itself, but with a constant value for b, to either use it or a random substitute \$ in the Retrieve algorithm. Retrieve takes in the index ι', or the substitute, together with the database D and outputs an encoded record f'_i. This encoded record is a legitimate record in the database and is added to the set of encoded records F'_i, where F'_i is the output the oracle gives to the adversary. Therefore, the oracle tests whether the adversary can distinguish if Retrieve used the index or a random substitute to fetch a record on the database. Compared to the previous game the oracle uses an additional loop to consider each index separately which crucial to ensure that previously seen indices are fetched from the history of previous responses H.

The intuition is that if the adversary, in this case, the client, cannot derive any correlation from I'_i to F'_i, then the challenger, in this case, the server, perfectly hides I'_i with Retrieve. Thereby enforcing the privacy of the database D in the retrieval as D is an input to Retrieve. Again, this might seem weird as the client can use Decode to later decode the encoded records F'_i, but if the client can infer if a record was retrieved using ι' or \$ it can win with an advantage, showing that this description also captures the privacy of the retrieved records F_i.

We say that the server's computational privacy holds in the retrieval if the adversary $\mathcal{A}$ has an advantage smaller than or equal to some negligible function.

$$\mathsf{Adv}^{\mathsf{Search}}(\mathcal{A}) = \left|2 \cdot \Pr\left[b = b'\right] - 1\right| \leq \mathcal{E}(\lambda)$$

Algorithm 2: Server Privacy - The Retrieval

```
Input: The database $D$
Output: Prediction $(b = b')$

$\Xi \leftarrow \mathsf{Setup}\,(D)$
$\mathsf{Oracle}\,(\Xi, D)$
$b' \leftarrow \mathcal{A}^{\mathsf{Oracle.query}(I'_i)}$
return $(b = b')$

$\mathsf{Oracle}\,(\Xi, D)$:
    $H \leftarrow [\ \ ]$
    $b \xleftarrow{\$} \{0,1\}$
    function $\mathsf{query}\,(I'_i)$:
        $F'_i \leftarrow \{\ \ \}$
        for $\iota'$ in $I'_i$ do
            if $\iota'$ in $H$ then
                $f'_i \leftarrow H.\mathrm{get}(\iota')$
            else
                if $(b = 0)$ then
                    $f'_i \leftarrow \mathsf{Retrieve}\,(D, i')$
                else
                    while $\$$ in $H$ do
                        $\$ \xleftarrow{\$} \{1, 2, ..., n\}$
                    end
                    $f'_i \leftarrow \mathsf{Retrieve}\,(D, \$)$
                end
            end
            $H.\mathrm{add}(\iota', f'_i)$
            $F'_i.\mathrm{add}(f'_i)$
        end
    return $F'_i$
end
```

Client privacy in the protocol. Algorithm 3 describes the client's privacy in a PDS protocol. We use a similar setup as Algorithm 2, with an Oracle that has a query functionality query that the adversary can submit sets of indices I_i to get responses for, but this time the oracle uses Encode to derive a set of encoded indices I'_i for I_i or a random substitute \$. The adversary's goal is to try to distinguish if the response I'_i from the oracle was produced with I_i or a random substitute \$.

The intuition is that if the adversary, in this case, the server, cannot derive any correlation from I_i to I'_i, then the challenger, in this case, the client, perfectly hides I'_i with Encode. Thereby enforcing the privacy of its search query q_i in the protocol. Similar to the previous cases, this game captures the privacy of the Retrieve algorithm even if it is not explicitly executed it is still a possible strategy for the adversary.

Algorithm 3: Client Privacy - The Protocol

```
Input: The database $D$
Output: Prediction $(b = b')$

$\Xi \leftarrow \mathsf{Setup}\,(D)$
$\mathsf{Oracle}\,(\Xi, D)$
$b' \leftarrow \mathcal{A}^{\mathsf{Oracle.query}(I_i)}$
return $(b = b')$

$\mathsf{Oracle}\,(\Xi, D)$:
    $H \leftarrow [\ \ ]$
    $b \xleftarrow{\$} \{0, 1\}$
    function $\mathsf{query}\,(I_i)$:
        $I'_i \leftarrow \{\ \ \}$
        for $\iota$ in $I_i$ do
            if $\iota$ in $H$ then
                $\iota' \leftarrow H.\mathrm{get}(\iota)$
            else
                if $(b = 0)$ then
                    $\iota' \leftarrow \mathsf{Encode}\,(\iota)$
                else
                    while \$ in $H$ do
                        $\$ \xleftarrow{\$} \{1, 2, ..., n\}$
                    end
                    $\iota' \leftarrow \mathsf{Encode}\,(\$)$
                end
            end
            $H.\mathrm{add}(\iota, \iota')$
            $I'_i.\mathrm{add}(\iota')$
        end
    return $I'_i$
end
```

We say that the client's computational privacy holds in the protocol if the adversary $\mathcal{A}$ has an advantage smaller than or equal to some negligible function.

$$\mathsf{Adv}^{\mathsf{Search}}\,(\mathcal{A}) = \left|2 \cdot \Pr\left[b = b'\right] - 1\right| \leq \mathcal{E}\,(\lambda)$$

3.2.3 Storage Imbalance

The storage capacity of the client likely not be that of the server. Therefore, a protocol is storage imbalanced if the client's κ_C maximum required storage capacity at any given point in the protocol is a fraction of the server's storage capacity κ_S. Giving us the storage imbalance bound $\kappa_C \leq \frac{\kappa_S}{w}$, for some constant $w \in \mathbb{Z}^+$, where we describe protocols as c-imbalanced to emphasize that the client requires c times as little storage as the server.

3.2.4 Timely

For intelligence to be valuable, it has to be timely. We say that a protocol is timely if the initiation does not exceed some time budget τ and each transfer exerts sublinear computation and communication complexity in the size of the database.

3.2.5 Adaptive

Given a single execution of Setup, the client can adaptively choose its search q_i for k number of executions of Search, Encode, $\mathsf{Retrieve}$ and Decode, where $k \times l$ is the total number of records transferred. This implies that the number of records transferred in each round can vary $n \geq |F| \geq 0$ and that the set of PDS protocols is a subset of $\mathrm{OT}^n_{k\times l}$ protocols.

3.2.6 Access Control with Non-Repudiation

Recall that we assume that the involved parties are authenticated to each other. We say that a protocol has access control if, for the identity $\mathcal{U}$, there exists a PPT algorithm Filter that takes $\mathcal{U}$ and D as inputs, and outputs some auxiliary information A about the database such that the access control policy is enforceable. [1]

[1] This is a change to the common interpretation of access control in oblivious transfer protocols as we do allow the server to learn the identity of a client to enforce access control, but as a benefit, we now have non-repudiation of searches for the different identities which is a necessity for auditing the correctness of honest/semi-honest parties.

Chapter 4

Proof of Concept from Generic Multi-Party Computation

A natural place to begin is to build a proof of concept protocol out of generic MPC. In it, we only focus on the question of feasibility and, therefore, only look at the privacy properties as we want to establish a baseline to compare against later protocols. The protocol treats generic MPC as an ideal black box for computation by simultaneously giving it all the inputs, thereby giving us valuable information about the intricacies and limitations of such an approach. We do not consider the Setup algorithm, and we merge the remaining algorithms into two main steps:

1. Match the client's search query with the relevant records, consisting of algorithms Filter and Search.
2. Retrieve those relevant records from the server, consisting of algorithms Encode, Retrieve, and Decode.

Instead of describing our inputs at a high level of abstraction, we work directly with a realized database and searches on that database; therefore, we have to introduce additional steps along the way for it all to fit together. In doing so, we bridge the gap between a protocol's description and its realization, allowing us to present precise metrics for the efficiency of the protocol. In the end, an assessment of the security and feasibility of the protocol is conducted so that we can highlight potential improvements.

4.1 Mock Passenger Name Records

We have to fill database D with records. These records could be files with various structures, but let us use a concrete example. It is important to emphasize that the protocols presented in this work are not tied to the specific contents of the records. In a general sense, the only requirement we impose is that a record R_ι is representable in two ways: 1) The searchable information in a record, R_i^{values}, which is an array of all the values or attributes specific to a record. E.g: $R_\iota^{\text{values}} \leftarrow$ ["Oslo", 1970-01-01, 1]. 2) The bytes that make up the records, R_ι^{chars}, an array with all the individual bytes encoded as characters that make up a record. E.g: $R_\iota^{\text{values}} \leftarrow$ [", O, s, l, o, ", „ , 1, 9, 7, 0, -, 0, 1, -, 0, 1, „ , 1].

Recall that section 60-5 in Politiregisterforskriften outlines the registered information in a PNR. We use these bullet points to create synthetic mock records. Semantically, we can think of a mock record as information related to a passenger's booking of a trip for one or more people. A record contains information including payment information, the travel plan, and general information about the passengers. We represent each mock record with a well-defined structure of key-value pairs. Some keys, like the *PNR Number*, have a unique value to each record, while others, like *Name*, can have the same value across multiple records. The structure of the records is as follows, where keys are in bold and an indentation indicates sub-keys of a key (A sample record can be found in Appendix A):

PNR Number: Cf. 60-5. 1. A unique number is given to each record.

Payment Information: Cf. 60-5 6. Information related to the payment.

> **Ticket Number:** Cf. 60-5. 13. A unique nine-digit number given to a ticket.
>
> **Date:** Cf. 60-5. 2. & 3. & 13. The date and time of the payment.
>
> **Name:** Cf. 60-5. 4. & 17. The full name of a person.
>
> **Address:** Cf. 60-5. 5. The billing address.
>
> **Phone Number:** Cf. 60-5. 5. Phone number of a person.
>
> **Email:** Cf. 60-5. 5. Email address of a person.
>
> **Vendor:** Cf. 60-5. 6. The vendor of the credit or debit card used.
>
> **Type:** Cf. 60-5. 6. Credit or debit.
>
> **Bonus Program:** Cf. 60-5. 8. Associated bonus program.

Airline: Cf. 60-5. 9. The airline providing the flight(s).

Travel Agency: Cf. 60-5. 9. The travel-agency facilitating the trip.

Travel Plan: Cf. 60-5. 7. The travel plan for the passengers.

> **IATA Code:** The IATA codes of the airports.
>
> **Airport Name:** The name of the airports.
>
> **City:** The city location of the airports.
>
> **Time:** Time of arrival/departure of the flights.

Passengers: Associated passengers with the record.

- **Name:** Cf. 60-5. 4. & 17. The full name of the person.
- **Statuses:** Cf. 60-5. 10. The status of the passenger for a given flight.
- **Seat:** Cf. 60-5. 14. The seat each passenger has on a flight.
- **Luggage:** Cf. 60-5. 16. The luggage and its weight registered to a passenger.
  - **Cabin:** Luggage brought into the plane.
  - **Checked:** Luggage transported in the plane's hold.
  - **Special:** Special category luggage.

4.2 Proof of Concept Protocol

Now that we can generate records to fill our database, we move to the protocol. In the first step, referred to as the search, the Search algorithm derives a set of indices by taking a search query q from the client and some auxiliary information A from the server as inputs. A is derived from the database D using Filter, and for simplicity, let us assume that there is an open access control policy. In the second step, called the retrieval, we run the Retrieve algorithm, which takes the set of encoded indices I' and the database D as inputs and outputs the set of encoded records F' to the client. We use generic MPC for the computation of Search and Retrieve therefore the Garble, Encode and Decode algorithms are done for us for free

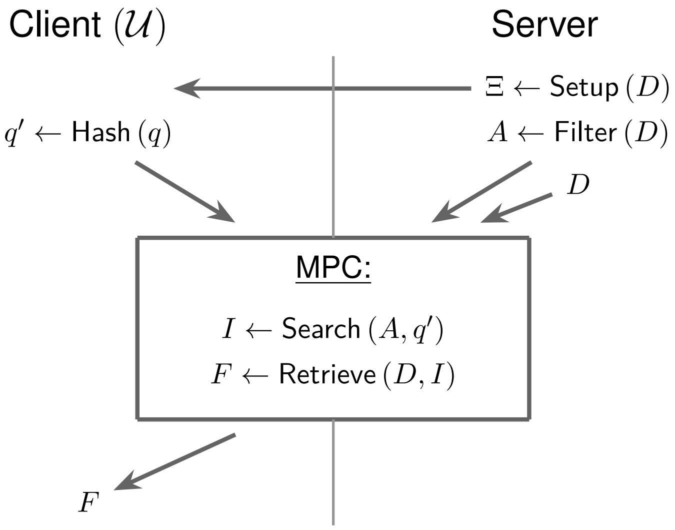

Client ($\mathcal{U}$)
Server
$\Xi \leftarrow$ Setup (D)
$q' \leftarrow$ Hash (q)
$A \leftarrow$ Filter (D)
D
MPC:
$I \leftarrow$ Search (A, q')
$F \leftarrow$ Retrieve (D, I)
F

Figure 4.1: The proof of concept protocol's communication flow between the client and the server.

4.2.1 Search

We must derive some auxiliary information A about the database in order to search it; therefore, A takes the form of an indexing. R_ι^{values} denote an array of the values in a specific record R_ι. Currently, the values in each record can be whatever, so we want a way to describe them in a standard format. A simple way to fingerprint a value is to produce its hash digest, so that is what we do, and we store the digests in A.

Algorithm 4: Proof of Concept - Filter

```
Input: Database $D = \left[R_1^{\text{values}}, R_2^{\text{values}}, ..., R_m^{\text{values}}\right]$
Output: Indexing of the database
        $A = \left[R_1^{\text{digests}}, R_2^{\text{digests}}, ..., R_m^{\text{digests}}\right]$

$A \leftarrow [\ \ ]$
for $R_\iota^{\text{values}}$ in $D$ do
    $R_\iota^{\text{digests}} \leftarrow [\ \ ]$
    for $v_j$ in $R_\iota^{\text{values}}$ do
        $R_\iota^{\text{digests}}$.append($\mathsf{Hash}(v_j)$)
    end
    $A$.append$\left(R_\iota^{\text{digests}}\right)$
end

return $A$
```

The only part remaining in the setup of the inputs is to encode the client's search query q. Similarly to encoding the values, we create a digest of the search query, allowing us to match it with the digests in A.

Algorithm 5: Proof of concept - Encoding q

```
Input: Search query $q$
Output: Encoded search query $q'$

$q' \leftarrow \mathsf{Hash}(q)$

return $q'$
```

We compute Search using generic MPC, meaning the client provides q' and the server A without revealing their input to the other party. Search outputs a set of indices I which is revealed to the client. The type of search we want to perform is a keyword search. To do this, we check if q' matches any digests in A. If it does so, we store the index in I.

Algorithm 6: Proof of Concept - Search

```
Input: Encoded search query $q'$
Input: Encoded indexing
       $A \leftarrow \left[R_1^{\text{digests}}, R_2^{\text{digests}}, ..., R_m^{\text{digests}}\right]$
Output: Indices $I$

$I \leftarrow [\ \ ]$
for $R_\iota^{\text{digests}}$ in $A$ do
    for $digest$ in $R_\iota^{\text{digests}}$ do
        if $(q' = digest)$ and $(digest$ not in $I)$ then
            $I$.append($i$)
        end
    end
end

return $I$
```

4.2.2 Retrieval

The second step is to retrieve the records for the indices I that the client learned in the search. We consider the database D to have the records as an array of characters, $D = \left[R_1^{\text{chars}}, R_2^{\text{chars}}, ..., R_m^{\text{chars}}\right]$. Implicitly, the order of the records in the database is the same as in the search. To retrieve the records, we only take the records with an index in I and add them to the retrieved records at F.

Algorithm 7: Proof of concept - Retrieve

```
Input: Indices $I$
Input: Database $D \leftarrow \left[R_1^{\text{chars}}, R_2^{\text{chars}}, ..., R_m^{\text{chars}}\right]$
Output: Retrieved records $F$

$F \leftarrow [\ \ ]$
for $i \in \{1, 2, ..., m\}$ do
    if $i$ in $I$ then
        $F$.append$\left(R_\iota^{\text{chars}}\right)$
    end
end

return $F$
```

4.3 Results

We now have the completed protocol, so let us implement it and perform some experiments. The experiment aims to determine which of MP-SPDZ's programs, i.e., generic MPC protocols, yield the most efficient execution. We only test the ones that match our modeling, dishonest majority with semi-honest adversaries in the two-party setting. Given the data we get from the experiments, we discuss a way of better describing the efficiency of this protocol. Lastly, we discuss important considerations that the extracted metric from the experiments do not directly reflect.

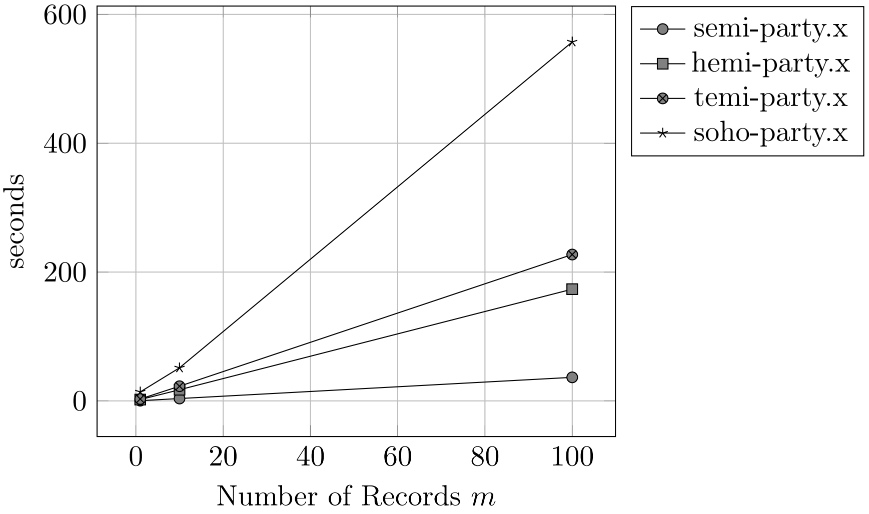
The realization of the program deviates from the pseudocode's mid-level abstraction as we cannot branch, use dynamic data structures or index on hidden values. An algorithm for the solution implemented in MP-SPDZ is located in Appendix B. Among the more significant differences is that the indexing A is padded such that all the arrays of digests are the same length σ. Secondly, the database D, consisting of the records as characters, was padded such that all the arrays were of length $\omega = 6016$. We chose ω as the maximum length any PNR could have, which introduces significant redundancy, making the average record two to three times larger. The computational complexity for Search ended up being $\mathsf{O}(m \cdot \sigma)$, and for $\mathsf{Retrieve}$ it is $\mathcal{O}(m \cdot \omega)$. This amortization alone does not answer the question of feasibility, so let us discuss the data from the experiments.

We combine Algorithm 6 (Search) and Algorithm 7 ($\mathsf{Retrieve}$) into one implementation as the results I from Search can be directly passed to $\mathsf{Retrieve}$. This means that we produce and supply A, q', and D into MP-SPDZ simultaneously. Respectively, the server runs Algorithm 4 (Filter), and the client runs Algorithm 5 (search query encoding). Together they execute the combination of Algorithm 6 (Search) and Algorithm 7 ($\mathsf{Retrieve}$). We instantiate Hash as SHA256.

For this program, we are only interested in the runtime of the algorithms implemented using generic MPC. MP-SPDZ provides us with a few different programs to test with, and of course, we want to figure out which of these provides us with the fastest execution. We take the benchmark metrics directly from MP-SPDZ as our results, but only consider programs that offer security for a dishonest majority with semi-honest adversaries. With the results, we are only concerned with the magnitude of the runtime; hence, a single execution suffices.

The results, Figure 4.2 and Table 4.1, show that semi-party.x yields the fastest runtime. It can perform a search query on one hundred records in 36.5 seconds, which might seem promising initially, but the number of records alone is an imprecise metric. A better metric for discussing the feasibility is to talk about the total byte size of the database. That is the number of records multiplied by the size of the records. For simplicity and to better communicate the actual throughput of the program, we use the size of the

Figure 4.2: Proof of Concept - Runtimes



padded records. Hence, $m * \omega$ is how we calculate the total size. The use of ω abstracts away the redundant data added by padding the records, which separates the problem of how to efficiently pad the records into a separate discussion. Performing the calculation $m * \omega = 100 \cdot 6016$ (one character is one byte) gives us a database size of 601.6 kilobytes. Scaling this up, we find that we can perform a single search query in time budget τ = eight hours on a database of size 300 megabytes. An interesting observation is that MP-SPDZ provides the total communication costs for the execution. We find that the difference between the inputs we provide to MP-SPDZ and the total communication costs is more than a factor of a hundred. This high communication cost comes from the additional communication required to compute, in our case, many multiplication operations.

Apart from the runtime, there are valuable considerations that the data above does not reflect: 1) The MP-SPDZ part of the program has to be recompiled before each execution as the number of values in the records, σ, varies depending on the records in the database. A solution would be to choose σ as the maximum possible number of values a record could have, but this would introduce even more redundant data. 2) We observe that the inability to have dynamic storage structures, indexing on secret values, and branching forces us to develop a solution that does more work than ideal. For example, we have to structure our data to skip a loop in Line 25 (Search & Retrieve) since we cannot index on secret values. Furthermore, we also have to loop over the whole array of I of length m instead of dynamically being able to append non-zero indices to it. [1]

[1] A set-like data structure and a dynamic array can be achieved using ORAM in MP-SPDZ, but from testing, these solutions yield much worse results in terms of runtime.

Table 4.1: Proof of Concept - Benchmarking

Program	Number of Records (m)	Runtime (s)	Total comm. (MB)
semi-party.x	1	0.336	0.67
	10	3.646	7.29
	100	36.538	73.08
hemi-party.x	1	2.036	4.07
	10	17.3	34.6
	100	173.462	346.92
temi-party.x	1	2.941	5.88
	10	22.787	45.57
	100	227.216	454.43
soho-party.x	1	13.914	27.38
	10	51.346	102.69
	100	557.26	1114.52

s denotes seconds, MB denoted megabytes, comm. denotes communcation.

To summarize, the proof of concept indicates that supporting a reasonably large database is possible. However, the constraints introduced by structuring the whole protocol in generic MPC limit its efficiency.

4.4 Security

Recall that we implemented Search and Retrieve as a program in MP-SPDZ. Therefore, security boils down to the security of the underlying generic MPC protocol, its implementation in MP-SPDZ, and the overall security of MP-SPDZ. These things are subject to change and are, therefore, out of the scope of this work. Instead, let us talk about the deviations from the desired functionality and additional risks not captured by it. We relate the program to our notion of privacy, discuss the information leakage in our program and finish with a discussion on an important practical implication, namely false positives, and propose mitigations to it.

For security, we want to protect against semi-honest adversaries. We are only concerned with a client and a server, so only two parties are involved, which means that if either party is dishonest, any protocol that assumes an honest majority is not sufficient. To model our parties, we instantiate them as an independent process whose only communication is through the interface MP-SPDZ. The only values that the client sees are q and F, and the server only sees A and D. We assume that the underlying generic MPC protocol provides semantic security, our privacy notion holds as the server does learn anything about the query and the result, while the client only learns the result.

4.4.1 Information Leakage

Regarding the program's output, it is not enough to consider the actual values revealed as the only information revealed to the client, as context and structure could also leak unwanted information. Here, we exclude all information in the records from the assessment and side channel leaks.

There is only one output of the program, which is revealed to the client. That output F has the same structure as the database input from the server, so it leaks the following information: 1) How many records, m, there are since the number of rows it has is directly correlated to m. A simple solution to avoid this is to pad the database to a fixed length. However, this would be costly in terms of adding redundant computations while being ineffective, as the magnitude of the number of records is still revealed. 2) The indices of the retrieved records. These can be used to deduce further information. For example, combining results from multiple searches can be used to infer deletion and creation of new records. A solution to avoid this is by shuffling the order in which the records are read. 3) The value of ω, which is intrinsic to the program's description.

4.4.2 False Positives

In the computation, we define a matching of the search query and a record if the digest of the search query is equal to the digest of one of the values in the record. It follows from using a cryptographically secure hash function that we do not get false positives in this matching, so that is not a concern, but what is a concern is the input-space to the hash function, as many of the values in the record have a small range. For example, the set of phone numbers, excluding prefixes, is small. This small input space could lead to false positives. An even better example is that some people have the same names. A solution to mitigate this weakness is for the server to filter keys and values when creating the indexing of the records, so the client has to provide sufficient evidence to satisfy the search criteria. A potential solution is to extend the search to allow the client to provide a set of inputs for the search query, which is combined to determine if the client has provided sufficient evidence. Like a scoring system, where a positive match to some values in the records is worth more than others, and in the end, the amount of information the client receive depends on the accumulated score.

Chapter 5

Private Database Search Protocol

PDS is realized by combining PSI and OT. Therefore, to create a scheme, we propose two PSI protocols where one provides a keyword search functionality and the other a semantic search functionality, and an OT protocol that allows the client to retrieve records from the database. These protocols interoperate, so to give a clear picture, we start by giving an overview of the communication flow between the client and server. Then, we introduce the OT protocol in two sections: one that describes the initiation phase and one for the transfer phase. We also separate the PSI protocols into an initiation and a transfer phase. To answer the question of feasibility, we perform experiments that implement the proposed protocols. Lastly, we discuss the results, in the context of the descriptions we gave for the properties of PDS, the practicability and post-quantum security of the proposed protocol.

5.1 Protocol Overview

The general idea is to present solutions to the five algorithms Filter, Search, Encode, Retrieve and Decode such that they satisfy our description of privacy, correctness, adaptive, storage imbalance, access control and timely. Figure 3.1 outlines, but does not limit, the communication flow between the client and server in the five algorithms, therefore we introduce additional algorithms to realize the protocol.

We do not focus on the Setup algorithm, but we highlight that Setup shuffles the order of the records on the database. Recall that PDS consists of a PSI protocol, to facilitate the search, combined with an OT protocol, to facilitate the retrieval. We propose two PSI protocols that provide the functionality of a keyword search and semantic search on the database, and one OT protocol that allows retrieval of records from the database. However, the two types of searches are independently realized so a PDS protocol might only include one, therefore, we present both as their own stand-alone protocol. Overall, the protocol is split into an initiation step of the subprotocols and a transfer step where the protocol is executed. To realize the protocol we extend the work in Chapter 3 and introduce eight new algorithms and re-introduce Garble:

PreProcess takes in the database D, and outputs the set of dummy item indices η, a permutation $\pi \subseteq \{1, 2, ..., n\}$ of length n, encryption keys K, and the encoded database D' with the files encrypted and shuffled.

Encrypt takes in a plaintext and encryption key of length β, and outputs a ciphertext of length β.

Garble takes in some auxiliary information A, and outputs the encrypted auxiliary information A'.

Permute takes in an array, and outputs a permutation π of that array.

Sort takes in a permutation π, and outputs a series of indices i, j and a bit b that indicates whether the indices should be swapped or not.

Hash takes in some binary string and outputs a digest of length β.

Decrypt takes in a ciphertext and encryption key of length β, and outputs a plaintext of length β.

Distance takes in two vectors and calculates the distance between them.

KeyGen takes in no input but outputs an encryption key and other variables like IVs, nonces and counters.

For the sake of brevity but without the lack of generality, we abstract some detailed functionalities into a general MPC description to best communicate the overall idea behind the algorithms.

5.1.1 Initiation

The initiation intends to preprocess as much as possible before the transfers. In it, a couple of things happen, showcased by Figure 5.1. The server fills the database D with dummy records such that it is of length n and pads all its records to a fixed length. Once that is done, the PreProcess algorithm encrypts the database under the client's keys K and sort it according to some random permutation π that the client gets by running Permute. PreProcess achieves this by internally using Sort and Encrypt. The idea is that the sorting is done obliviously to the server and that the encryption of the files is oblivious to the client, without the server learning K and the client learning the contents of D. At the end, the server is left with the encoded database D' where it cannot correlate the ciphertexts in D' to the records in D.

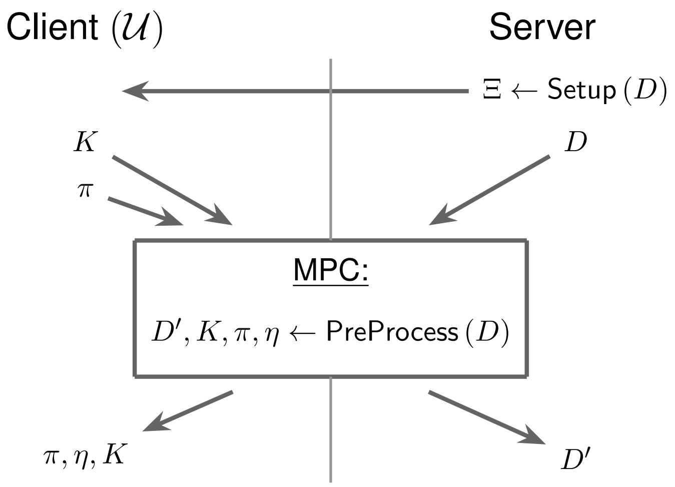

Client ($\mathcal{U}$)
Server
$\Xi \leftarrow \mathsf{Setup}(D)$
K
π
D
MPC:
$D', K, \pi, \eta \leftarrow \mathsf{PreProcess}(D)$
π, η, K
D'

Figure 5.1: The communication flow between the client and the server in the database initiation (for clarity this overview is simplified).

Next we do more preprocessing to set up the two types of searches. In both types, the Filter algorithm will be used to derive some auxiliary information A about the database, but the implementation of Filter will differ.

For the first type, keyword search, Figure 5.2, the server will create an indexing that is encrypted, using Garble, and then send it to the client. Garble internally uses some implementation of Encrypt and Hash to encrypt elements. This encryption will happen with two keys, e_1 and e_2. To enforce the access control the server will filter values when creating the indexing, based upon the client's identity $\mathcal{U}$.

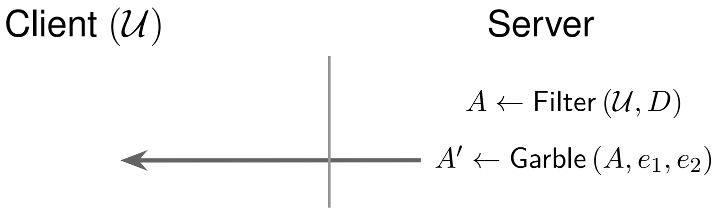

Client ($\mathcal{U}$)
Server
$A \leftarrow \mathsf{Filter}(\mathcal{U}, D)$
$A' \leftarrow \mathsf{Garble}(A, e_1, e_2)$

Figure 5.2: The communication flow between the client and the server in the keyword search initiation.

In the second type, semantic search, we use a LLM, denoted Γ, to create vector embeddings of each record. These embeddings are stored with the corresponding index of the records. The access control is enforced by the server filtering information from the records before supplying it to Γ, thereby only capturing the intended semantics in the vector embeddings.

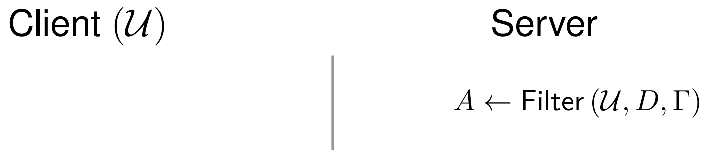


Figure 5.3: The communication flow between the client and the server in semantic search initiation.

5.1.2 Transfer

The transfer step can be executed k number of times. We call an execution a transfer, as the server obliviously transfers files to the client. A transfer has the client and server perform the Encode, Retrieve, and Decode algorithms in both the keyword search and semantic search variants, with the implementation of Search differing between the two.

In the first type, keyword search, the client has the encrypted auxiliary information A' that we want it to compare with its search query q_i. The way the client does this is by getting the hash q_i' of its search query, using the Hash algorithm, encrypted with algorithm Encrypt under the server's encryption keys e_1 and e_2 using generic MPC. Once it has the two encrypted variants q_{i,e_1} and q_{i,e_2}, it will use them with Search, which internally uses the Decrypt algorithm, to derive a set of indices I_i.

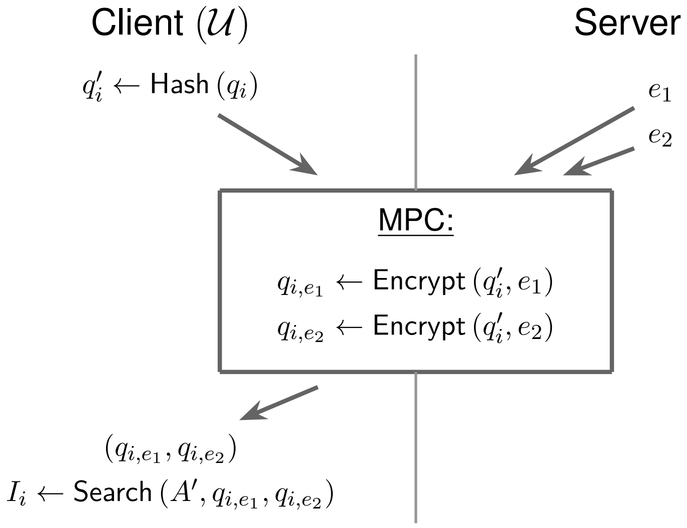
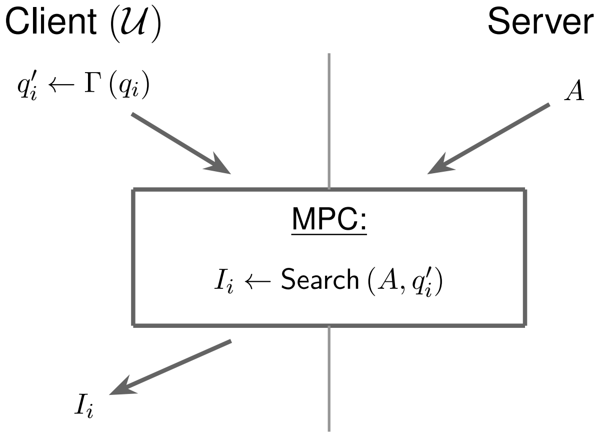

Client ($\mathcal{U}$)
Server
$q_i' \leftarrow \mathsf{Hash}(q_i)$
e_1
e_2
MPC:
$q_{i,e_1} \leftarrow \mathsf{Encrypt}(q_i', e_1)$
$q_{i,e_2} \leftarrow \mathsf{Encrypt}(q_i', e_2)$
(q_{i,e_1}, q_{i,e_2})
$I_i \leftarrow \mathsf{Search}(A', q_{i,e_1}, q_{i,e_2})$

Figure 5.4: The communication flow between the client and the server in the keyword search transfer.

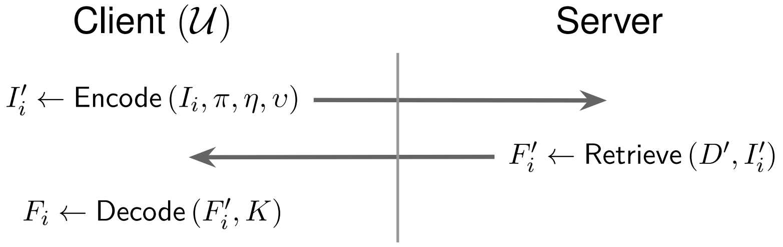
In the second type, semantic search, the client will use Γ to construct a vector embedding q_i' of its search query q_i. Search will be performed with the client inputting q'_i and the server inputting A to a generic MPC protocol that compares q'_i and A using Distance. The output of Search is a set of indices I_i.



Client ($\mathcal{U}$)
Server
$q'_i \leftarrow \Gamma(q_i)$
A
MPC:
$I_i \leftarrow \mathsf{Search}(A, q'_i)$
I_i

Figure 5.5: The communication flow between the client and the server in the semantic search transfer.

Recall that we created the encoded database D' by sorting datbase according to the random permutation π. To retrieve the records, the client uses π to encode the indices I'_i and then sends them to the server. The server, in possession of D', responds with the encrypted records F'_i, which the client decrypts with the keys K to derive the set of records F_i for that transfer.



Client ($\mathcal{U}$)
Server
$I'_i \leftarrow \mathsf{Encode}(I_i, \pi, \eta, \upsilon)$
$F'_i \leftarrow \mathsf{Retrieve}(D', I'_i)$
$F_i \leftarrow \mathsf{Decode}(F'_i, K)$

Figure 5.6: The communication flow between the client and the server in the file retrieval.

5.2 Database Initiation

From the proof of concept, we found that for a megabyte-sized database, it takes hours to complete a single transfer, therefore, it would be beneficial to structure our protocol so that we do most of the costly work in advance. We encode our database D in the initiation to make the transfers as efficient as possible. The overarching idea is for the client to obliviously encrypt and shuffle D with its keys K and permutation π. If we were to follow a general ORAM construction, the client would non-obliviously encrypt the records and use an oblivious sorting algorithm to shuffle them. This process would only provide privacy in one way; hence, we introduce the notion of Private Swap (PS), which yields privacy for both the client and server in the shuffling and encryption of the database. For PS, we present an efficient generic MPC protocol that uses binary circuits. We use a generic MPC-based PS protocol in combination with an oblivious sorting algorithm to implement the PreProcess algorithm. With the implementation, we perform experiments to find the largest database size we can support within a time budget τ of eight hours. After presenting the results from the experiment, we propose an OT-based PS protocol and discuss its potential efficiency.

5.2.1 The PreProcess algorithm

To pad each record is important as the length of a record could work as a direct identifier even if it is encrypted. We use the same idea as in the proof of concept to pad the records; we settle for an upper bound ω, which is the maximal length of any possible record R_ι^{chars}. We then pad all records such that they are of length ω. Later in the transfer step, we must make valid retrievals of "empty" records, therefore, we fill the database with dummy records d_i of length ω. We add the dummy records after the records such that the database from index 1 to m consists of records R_ι, and from $m+1$ to n consists of dummy records. The number of dummy records added will later determine how many transfers k can be made. This way of adding the dummy records allows the client to infer their location implicitly by the server sending it the length of the database n and the number of records m.

Algorithm 8: Server - Padding the Database

```
Input: Database $D \leftarrow \left[R_1^{\text{chars}}, R_2^{\text{chars}}, ..., R_m^{\text{chars}}\right]$
Output: Database $D' \leftarrow [R'_1, R'_2, ..., R'_m, d_1, d_2, ..., d_{n-m}]$

$D' \leftarrow [\ \ ]$
for $R_\iota^{\text{chars}}$ in $D$ do
    $R'_i \leftarrow R_\iota^{\text{chars}}$.padToLength($\omega$)
    $D'$.append($R'_i$)
end
for $i \in \{1, 2, ..., n-m\}$ do
    $D'$.append($d_i$)
end

return $D'$
```

We now have a database with records and dummy records that the client wants to shuffle. The client uses an oblivious sort algorithm Sort to sort the database D' after some random permutation π of length n. For simplicity, we say that the sorting algorithm considers two and two records at a time, denote with $i, j, b \leftarrow$ Sort, where $i, j \in \{1, 2, ..., n\}$ are indices of the records and $b \in \{0, 1\}$ is a bit to signal whether to swap or not. The challenge is how the client can request records R_i and R_j, encrypting and potentially swap them without learning the records. This might seem like a simple problem at first, as the server could mask the records before sending them away, but how is the server supposed to unmask them without the client revealing which mask goes with which record? In this sense, we need an oblivious swapping algorithm combined with two-way masking of the records, which brings us to the notion of private swap.

Algorithm 9: Client $\mathcal{U}$ - PreProcess

```
Input:
Output: Permutation $\pi$
Output: Dummy indices $\eta$
Output: Keys $K$

$K \leftarrow [\ ]$
receive $(m, n)$
$\eta \leftarrow [m+1, m+2, ..., n]$
$\pi \leftarrow$ Permute $([1, 2, ..., n])$
for $i, j, b$ in Sort $(\pi)$ do
    send $(i, j)$
    private swap:
        Input: $K, i, j, b$
        Output:
    end
end

return $\pi, \eta, K$
```

Algorithm 10: Server - PreProcess

```
Input: Database $D' \leftarrow [R'_1, ...R'_m, d_1, ..., d_{n-m}]$
Output: Database $D'$

send $(m, n)$
while sorting do
    receive $(i, j)$
    private swap:
        Input: $D'_i, D'_j$
        Output: $C_i, C_j$
    end
    $D'_i \leftarrow C_i$
    $D'_j \leftarrow C_j$
end

return $D'$
```

We implement the PreProcess algorithm by combining Sort and PS, detailed in Algorithm 9 and Algorithm 10. In PreProcess, the client to control when records are swapped and provides the keys used for encryption. At the end, we have the encrypted records in a random order. This leaves the client with the later possibility of asking the server for ciphertexts and decrypting them, so the server does not learn which records are retrieved. The algorithm starts with the server sending the client the number of records m and the database size n. The client then derives the indices of the dummy records $\eta \leftarrow [m+1, ..., n-1, n]$ and gets a random permutation π from $\mathsf{Permute}$. It then uses the permutation as the element to be sorted; the client gets two indices i, j, and a bit b for each comparison of records. i and j are sent to the server to retrieve the elements D'_i and D'_j from the database. The client then provides a set of encryption keys K, indices i and j together with the swap indicator b to the PS protocol, and the server receives the ciphertexts C_i and C_j, which are its inputs encrypted and potentially swapped by the client. In the protocol, the client stores the keys in K at the location of i and j, effectively replacing old key pairs if present. After the private swap the server updates the element in D'_i and D'_j with the new ciphertexts C_i and C_j. Once the sorting is complete, the client stores π, η, and K, and the server receives D'.

5.2.2 Generic MPC-based Private Swap Protocol

An efficient PS protocol can be realized using the $\mathsf{Encrypt}$ algorithm in counter mode CTR in combination with generic MPC. On one side, the client pre-produce the key stream s_i outside the secure computation, and on the other the server supplies the record. We use generic MPC, to combine the key stream with the record without revealing either input to the other party, giving us an OPRF and boiling the encryption part of the protocol down to XOR operations. For the swap we recall that an if-else statement is describable as $b \cdot x + (b-1) \cdot y$, for statement $b \in \mathbb{F}_2$, and outcomes $x \in \mathbb{F}_{2^z}$ and $y \in \mathbb{F}_{2^z}$. AND is equivalent to multiplication, XOR to addition and NOT to subtraction over a Galois field, which we use to transform the expression into a set of logic gate operations.

$$\begin{gathered}
P_i \leftarrow \mathsf{XOR}\left(C_i, s'_i\right) \\
P_j \leftarrow \mathsf{XOR}\left(C_j, s'_j\right) \\
R_i \leftarrow \mathsf{XOR}\left(\mathsf{AND}\left(P_j, b\right), \mathsf{AND}\left(P_i, \mathsf{NOT}\left(b\right)\right)\right) \\
R_j \leftarrow \mathsf{XOR}\left(\mathsf{AND}\left(P_i, b\right), \mathsf{AND}\left(P_j, \mathsf{NOT}\left(b\right)\right)\right) \\
C_i \leftarrow \mathsf{XOR}\left(R_i, s_i\right) \\
C_j \leftarrow \mathsf{XOR}\left(R_j, s_j\right)
\end{gathered}$$

In the later rounds of the sorting the client might need to decrypt the records before encrypting them, this is achieved by adding another set of XOR gates where the client can supply the decryption key streams s'_i and s'_j. If the client does not need to decrypt the record it simply supplies an all-zero key stream. Combining the if-else statement and the re-encryption gates we derive the following equations, where every bit in P_1 and P_2 are evaluated against b:

We use the equations to describe the computation in our PS protocol as a binary circuit $\mathcal{C}$, which is executed in generic MPC. For deriving keys and counters, we assume the existence of a PPT algorithm KeyGen. We use these new keys e and counters c, produced from KeyGen, combined with Encrypt in Counter (CTR) mode, CTR-Encrypt, to produce key streams. All this together gives us a compact PS protocol, Algorithm 11 and Algorithm 12.

Algorithm 11: Client $\mathcal{U}$ - Generic MPC-based Private Swap

```
Input: $K, i, j, b$
Output:

if $i$ in $K$ then
    $e_i, c_i \leftarrow K$.get($i$)
    $s'_i \leftarrow$ CTR-Encrypt $(e_i, c_i)$
else
    $s'_i \leftarrow \{0\}^\omega$
end
if $j$ in $K$ then
    $e_j, c_j \leftarrow K$.get($j$)
    $s'_j \leftarrow$ CTR-Encrypt $(e_j, c_j)$
else
    $s'_j \leftarrow \{0\}^\omega$
end
$e_i, c_i \leftarrow$ KeyGen ()
$e_j, c_j \leftarrow$ KeyGen ()
$s_i \leftarrow$ CTR-Encrypt $(e_i, c_i)$
$s_j \leftarrow$ CTR-Encrypt $(e_j, c_j)$
multi-party computation
    Input: $b, s'_i, s'_j, s_i, s_j$
    Output:

    $\mathcal{C}\left(b, s'_i, s'_j, s_i, s_j\right)$
end
$K$.update($i$).with($e_i, c_i$)
$K$.update($j$).with($e_j, c_j$)

return
```

Algorithm 12: Server - Generic MPC-based Private Swap

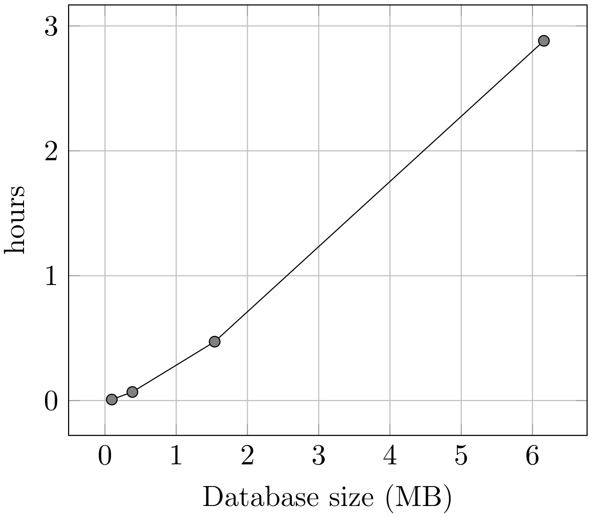
```
Input: $D'_i, D'_j$
Output: $C_i, C_j$

multi-party computation
    Input: $D'_i, D'_j$
    Output: $C_i, C_j$

    $C_i, C_j \leftarrow \mathcal{C}\left(D'_i, D'_j\right)$
end

return $C_i, C_j$
```

5.2.3 Results

The dominant time complexity stems from Sort, instantiated as bitonic sort [3] , with a complexity of $\mathsf{O}\left(n \log^2(n)\right)$. It applies to both the client and server as they perform private swaps for all the required comparisons together. However, this is without parallelization. With parallelization, bitonic sort can achieve a time complexity of $\mathsf{O}\left(\log^2(n)\right)$, given that n is finite, which for our purposes is the case. The communication complexity, regardless of parallelization, is $\mathsf{O}\left(n \log^2(n)\right)$.

Further, we instantiate Encrypt with AES, and implement the dummy records as arbitrary key streams of length ω.

Figure 5.7: PreProcess - Runtime



The experiment implements bitonic sort without parallelization and the results are presented in Figure 5.7. We find that within a time budget τ of eight hours, we can preprocess a database of size 6.16 megabytes. Most of the runtime is spent performing private swaps in order to sort the database, and a single private swap takes on average 0.33 seconds and has 3.67 megabytes of data sent by the client and 3.67 megabytes of data sent by the server.

In Appendix C, we perform a deeper analysis of our implementation of the bitonic sort algorithm. The analysis reveals that the number of private swaps required to sort the permutation π of size n follows the sequence $a(n) = \frac{n}{4} \cdot \left(\log_2^2(n) + \log_2(n)\right)$, without parallelization. We use this sequence and the average runtime of a private swap of 0.33 seconds, to approximate the total runtime for PreProcess. When we compare the approximated runtime, in seconds, to the experimental results a 10% difference is revealed, attributed to the additional overhead by network and computational processes outside MP-SPDZ.

We therefore add this additional overhead to give a better approximation:

$$P(n) = \frac{5n}{54} \cdot \left(\log_2^2(n) + \log_2(n)\right)$$

With perfect parallelization the analysis shows that PreProcess follows the sequence $b(n) = \frac{1}{2} \cdot \left(\log_2^2(n) + \log_2(n)\right)$, which we combine with our overhead to give an approximation, assuming that a private swap takes 0.33 seconds. Note that the scalar does not encapsulate the overhead introduce by parallelization:

$$PP(n) = \frac{5}{27} \cdot \left(\log_2^2(n) + \log_2(n)\right)$$

We use $PP(n)$ to estimate that a database length $n = 2^{393}$ can be initiated within the time budget. That equates to a database size of approximately $1.21 \cdot 10^{122}$ bytes, a number so large even googol could not exhaust it. Let us be clear that this also would require 2^{393} parallel cores for processing and communication channels, making it infeasible with today's technology, but it shows that solutions that implement partial parallelization will most likely lie within the range of 6 to $1.21 \cdot 10^{116}$ megabytes.

5.2.4 OT-based Private Swap protocol

In addition to the generic MPC-based PS protocol, we present a PS protocol built from OT_1^2, Algorithm 13 and Algorithm 14. In it, the server applies key streams s'_i and s'_j to its two records D'_i and D'_j such that it has two pairs of encrypted records. The first pair, denoted M_0 and M_1, is the two records masked with s'_i. The second pair, denoted N_0 and N_1, is masked with s'_j. From these four masked records, it will perform two OT_1^2 where the client, using the swap signal bit b, requests b in the first OT and $\mathsf{NOT}(b)$ in the second. The client then applies its key streams s_i and s_j to the two records, giving C'_i and C'_j, and sends them back to the server so that it receives C'_i first and then C'_j second. To finish the protocol, the server removes its key streams from the records by applying s'_i to the first it receives and s'_j to the second. Leaving it with C_j and C_j, which are the two records encrypted and potentially swapped by the client.

To assess the OT-based PS protocol with post-quantum security we assume, based on the works of [8], that we can perform 2^{20} number of 128-bit OTs in 0.5 seconds, which we simplify to 2^{20} per second as the protocol requires two OTs per execution. We combine this assumption with the sequences $a(n)$ and $b(n)$ to approximate the protocol's efficiency. When $n = 2^{20}$, we multiply with the number of blocks per record $\frac{\omega}{\beta} = 376$ by $a(n)$, and find that PreProcess requires 41 397 780 480 total number of OTs. We divide this by the efficiency assumption 2^{20}, to approximate that the runtime is about 11 hours for a database size of 15 gigabytes. With perfect parallelization, we perform the same calculation but swap $a(n)$ for $b(n)$ and find that it takes 0.0753 seconds for a database size of 15 gigabytes, assuming that the OT-extension protocol does not hinder to the parallelization of bitonic sort. Note that the approximations only include the OT parts of the protocol.

Algorithm 13: Client $\mathcal{U}$ - OT-based Private Swap

```
Input: $b, s_i, s_j$
Output:
oblivious transfer:
    Input: $b$
    Output: $M_b$
end
$b' \leftarrow \mathsf{NOT}(b)$
oblivious transfer:
    Input: $b'$
    Output: $N_{b'}$
end
$C'_i \leftarrow \mathsf{XOR}(M_b, s_i)$
$C'_j \leftarrow \mathsf{XOR}(N_{b'}, s_j)$
send $C'_i$
send $C'_j$
return
```

Algorithm 14: Server - OT-based Private Swap

```
Input: $D'_i, D'_j, s'_i, s'_j$
Output: $C_i, C_j$
$M_0 \leftarrow \mathsf{XOR}(D'_i, s'_i)$
$M_1 \leftarrow \mathsf{XOR}\left(D'_j, s'_i\right)$
oblivious transfer:
    Input: $M_0, M_1$
    Output:
end
$N_0 \leftarrow \mathsf{XOR}\left(D'_i, s'_j\right)$
$N_1 \leftarrow \mathsf{XOR}\left(D'_j, s'_j\right)$
oblivious transfer:
    Input: $N_0, N_1$
    Output:
end
receive $C'_i$
receive $C'_j$
$C_i \leftarrow \mathsf{XOR}(C'_i, s'_i)$
$C_j \leftarrow \mathsf{XOR}\left(C'_j, s'_j\right)$
return $C_i, C_j$
```

5.3 File Retrieval

We have initiated the OT protocol, so let us move to the transfer phase. Recall that we are interested in encoding the indices that the client gets from the search so that they can be used to retrieve the relevant records without leaking information about which records were transferred. This is done using what we did in PreProcess to facilitate the functionality of the Encode, Retrieve and Decode algorithms. As done previously, we implement and perform experiments with these algorithms and then discuss their efficiency.

5.3.1 The Encode, Retrieve and Decode algorithms.

The client wants to encode the set of indices I that it acquired in the search. We realize that π is a permutation of the original indices of D and, therefore, works as a translation between the indices used to derive the auxiliary information A and the indices of the encoded database D'. This means that we can use the permutation π to encode the indices I so that they are unlinkable to any specific record, in the perspective of the server. Imagine that we are interested in index ι; to encode it, we take the ι^{th} index of the permutation, $\iota' \leftarrow \pi_\iota$. This way, we can build the encoded indices I', Algorithm 15.

Algorithm 15: Client - Encode

```
Input: Record indices $I$
Input: Dummy record indices $\eta$
Input: Permutation $\pi$
Input: Requested indices $\upsilon$
Output: Encoded indices $I'$

$I' = [\ \ ]$
for $\iota$ in $I$ do
    if $\iota$ in $\upsilon$ then
        $\iota \xleftarrow{\$} \eta$
        $\eta$.remove($i$)
    end
    $i' \leftarrow \pi_\iota$
    $I'$.add($i'$)
end
for $\iota \in \{|I'|, ..., \sigma\}$ do
    $\iota \xleftarrow{\$} \eta$
    $\eta$.remove($\iota$)
    $i' \leftarrow \pi_\iota$
    $I'$.add($i'$)
end
$\upsilon$.addAll($I'$)

return $I'$
```

Further, we realize that different lengths of I could be used as a direct identifier for the results from the search. Therefore, we always want the client to request the same amount of records, and to achieve this, we pad I with a fixed length σ while never requesting the same index twice.

Recall that η contains the indices of the dummy records, meaning that we can use them to pad I. It is also important to realize that if an index from I already has been requested, then it is unnecessary to request it again, so the client simply requests a dummy record in its place. Once an index of a dummy record is requested, it is removed from η and can, therefore, no longer be used. We keep track of these requested indices with the set υ.

The indices are now encoded, and the client sends them to the server. Once received, the server picks the encrypted record from D' for all the encoded indices, $F' \leftarrow \mathsf{Retrieve}(I')$, Algorithm 16.

```
Algorithm 16: Server - Retrieve
  Input: Encoded Database $D'$
  Input: Encoded indices $I'$
  Output: Encrypted Records $F'$

  $F' = [\ ]$
  for $\iota'$ in $I'$ do
      $F'.\mathrm{add}(D'_{\iota'})$
  end

  return $F'$
```

Next, the server sends the encrypted records F' to the client, which uses the encryption keys K to decrypt the records by fetching key $K_{\iota'}$ to decrypt record $R_{\iota'}$ from F'. We discard any received dummy records $d_{\iota'}$. Giving us the implementation of Decode, Algorithm 17.

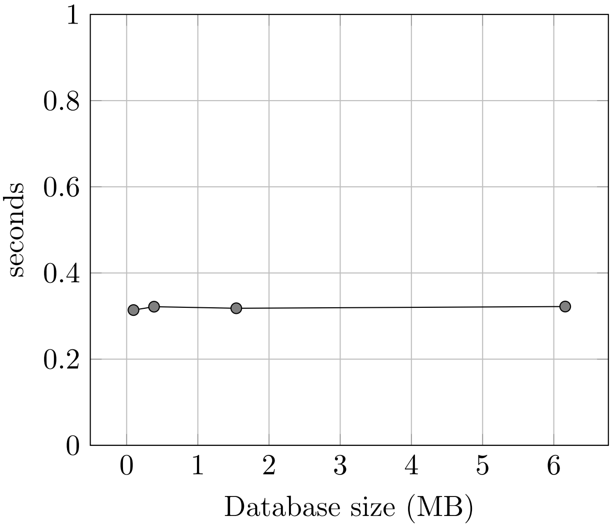
```
Algorithm 17: Client - Decode
  Input: Encrypted records $F'$
  Input: Encryption keys $K$
  Output: Transferred records $F$

  $F = [\ ]$
  for $R_{\iota'}$ in $F'$ do
      $R_\iota \leftarrow \mathsf{Decrypt}(R_{\iota'}, K_{\iota'})$
      $F.\mathrm{add}(R_\iota)$
  end

  return $F$
```

5.3.2 Results

The time complexity is dependent on the padding length σ of I'. We, therefore, fix $\sigma = 5$ so that it is constant for all executions; further, the number of legitimate records retrieved l determines computation time in Encode, and it determines how many times Decode is executed, so we, therefore, fix I' so that it consists of one legitimate index and four dummy indices, doing so fixes the number of retrieved records $l = 1$. It is essential to realize that our protocol is a $k \times l$-out-of-n OT, meaning that if we do not fix l and σ across executions, it will vary depending on the contents of the records. Essentially, fixing them mitigates data bias across transfers from different databases. k represents the number of transfers, which we fix to $k = 1$, giving us a time and communication complexity of $\mathsf{O}(1)$.

For brevity, we combine all three algorithms into one program. I', υ, η, and I are implemented as hashsets.

Figure 5.8: File Retrieval - Runtime



In the experiment, we find that each transfer takes about 0.32 seconds and is independent of the database size, thus verifying the constant time complexity. As we fixed σ, the total communication $2\sigma + \omega\sigma$, counting each index in I' as two bytes, is 30.09 kilobytes for each transfer.

5.4 Keyword Search

The first type of search we present is a keyword search, where the query is compared to the record values. We want this comparison to be done without the server learning the outcome and with the client only learning the indices of the records it is interested in. In other words, we want a PSI protocol that takes in the client's search query q and the auxiliary information A from the server, and outputs a set of record indices I to the client. We split the protocol into an initiation and transfer phase, where most of the work is done in the initiation.

5.4.1 Initiation

The goal is for the server to derive some auxiliary information A about the database D. An inverted index matrix is often used in database search as it is efficiently searched, as it structures all the values of the records in the left column and then puts the index of each record that contains said value in the right column. We think of this structure as each row being a tuple consisting of the value v_j and a set of indices I_j, (v_j, I_j). To enforce the access control the server filters values depending on the client's identity $\mathcal{U}$, when creating the inverted index matrix. This is done in Filter. Importantly, we pad every set of indices I_j, as their length is useable as a direct identifier. The padding is performed by adding as many unique dummy indices as is required such that I_j is the length of σ. We stress that these dummy indices have to be indices that are not valid indices on the database and that they are unique per set I_j, Algorithm 18.

Algorithm 18: Server - Filter

```
Input: Client's Identity $\mathcal{U}$
Input: Database $D \leftarrow \left[R_1^{\text{values}}, R_2^{\text{values}}, ..., R_m^{\text{values}}\right]$
Output: Auxiliary information $A$

$A \leftarrow [\ \ ]$
for $R_\iota^{\text{values}}$ in $D$ do
    for $v_j$ in $R_\iota^{\text{values}}$ do
        if $v_j$ allowed by $\mathcal{U}$ then
            if $v_j$ in $A$ and $\iota$ not in $I_j$ then
                $I_j$.add($\iota$)
            else
                $I_j \leftarrow [\ \ ]$
                $I_j$.add($\iota$)
                $A$.add($v_j, I_j$)
            end
        end
    end
end

return $A$
```

Garble encrypts A, with the two encryption keys, e_1 and e_2 picked by the server. We use e_1 to encrypt the hashes of the values and e_2 to derive a new ephemeral encryption key $e_{2,j}$ that encrypts the set of indices I_j. In the end, the order of the tuples in A' is randomly shuffled, Algorithm 19. The server then sends the encrypted auxiliary information A' to the client.

Algorithm 19: Server - Garble

```
Input: Auxiliary information $A$
Input: Encryption keys $e_1$, $e_2$
Output: Encrypted auxiliary information $A'$

for $I_j$ in $A$ do
    $I_j$.padToLength($\sigma$)
end
for $(v_j, I_j)$ in $A$ do
    $I'_j \leftarrow [\ ]$
    $e_{2,j} \leftarrow \mathsf{Encrypt}(\mathsf{Hash}(v_j), e_2)$
    for $\iota$ in $I_j$ do
        $\iota' \leftarrow \mathsf{Encrypt}(\iota, e_{2,j})$
        $I'_j$.add($\iota'$)
    end
    $v'_j \leftarrow \mathsf{Encrypt}(\mathsf{Hash}(v_j), e_1)$
    $A'$.add$\left(v'_j, I'_j\right)$
end
$A'$.shuffle()

return $A'$
```

5.4.2 Transfer

In the transfer phase, the client possesses the encrypted auxiliary information A', which it wants to compare with its search query q. Recall that we want to find the intersection between the values of the records and the search query. A' consists of tuples $\left(v'_j, I'_j\right)$ where the first values are the encrypted values of the records; this means that in order for the client to find if q is equal to any of the values it has to be encrypted under the same key. The key, e_1, is held by the server, and cannot reveal it to the client. This scenario can be solved with an OPRF as it allows for the oblivious evaluation of the Encrypt algorithm, where the server supplies e_1 and the client q. For simplicity, we implement this functionality through generic MPC.

Once the client has found which tuple it is interested in, it wants to decrypt the set of encrypted indices associated with that tuple. Recall that the indices I'_j were encrypted under an ephemeral key $e_{2,j}$. The client already knows v_j but needs it encrypted under the server's key e_2. Therefore, we use the same OPRF functionality above to encrypt v_j under e_2. This gives the client $e_{2,j}$ in the form of q_{e_2}, that it can use to decrypt all the indices in I'_j. If an index decrypts to anything other than a valid index on D, it is discarded as a dummy index.

Algorithm 20: Client $\mathcal{U}$ - Search

```
Input: $A'$
Input: $q$
Output: $I$

$q'_i \leftarrow$ Hash $(q_i)$
multi-party computation:
    Input: $q'_i$
    Output: $q_{e_1}$

    $q_{e_1} \leftarrow$ Encrypt $(q'_i)$
end
multi-party computation:
    Input: $q'_i$
    Output: $q_{e_2}$

    $q_{e_2} \leftarrow$ Encrypt $(q'_i)$
end
$I \leftarrow [\ \ ]$
$I'_j \leftarrow A'.\text{get}(q_{e_1})$
for $\iota'$ in $I'_j$ do
    $\iota \leftarrow$ Decrypt $(\iota', q_{e_2})$
    $I.\text{add}(\iota)$
end

return $I$
```

Algorithm 21: Server - Search

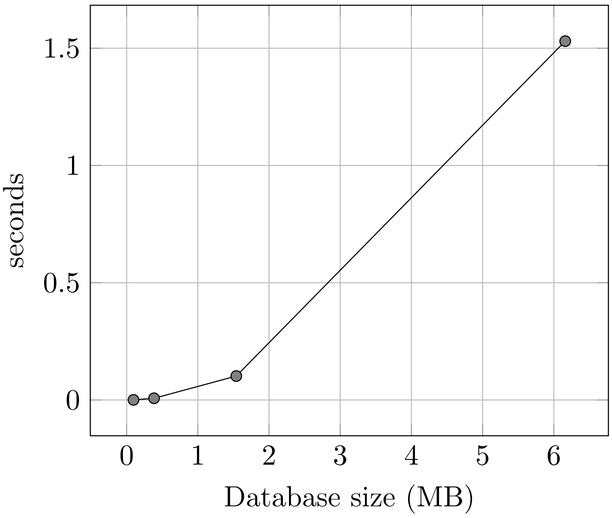
```
Input: $e_1, e_2$

Output:


multi-party computation:
    Input: $e_1$
    Output:

    Encrypt $(e_1)$
end
multi-party computation:
    Input: $e_2$
    Output:

    Encrypt $(e_2)$
end







return
```

5.4.3 Results

The time complexity of Filter depends on the number of records m and the some scalar of number of values in the record μ, thus its time complexity is $\mathsf{O}(m \cdot \mu)$. The time complexity of Garble is dependent on the padding length σ, which we keep constant across executions, and the number of unique values from all the records θ; thus, its time complexity is $\mathsf{O}(\sigma \cdot \theta)$. Search has a constant number of encryptions with inputs of constant size β, thereby making its time and communication complexity $\mathsf{O}(1)$.

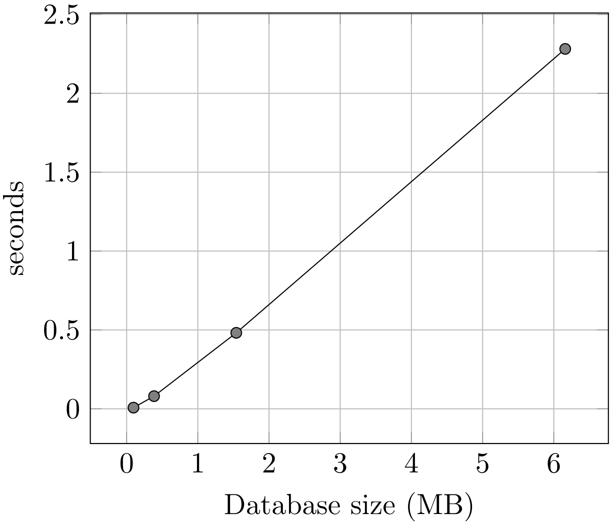
We instantiate Encrypt with AES in Electronic Code Book (ECB) mode, Hash with Secure Hash Algorithm and KECCAK (SHAKE), and the block length $\beta = 128$ bits. A and A' are implemented as a hashmaps.

Figure 5.9: Filter - Runtime



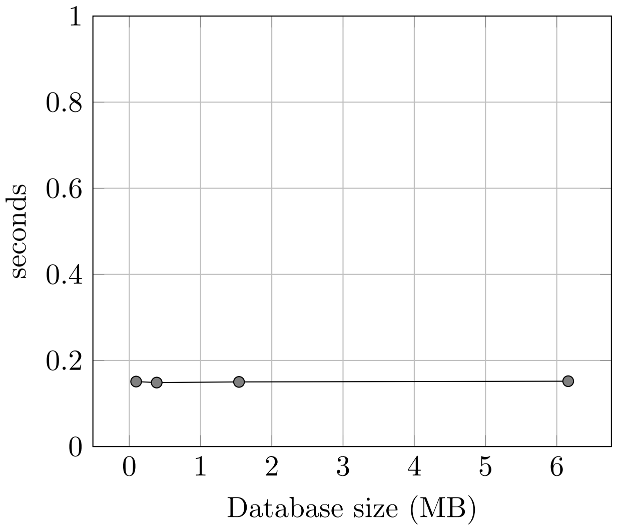
In the experiment, we kept $n = m$. From the runtimes Figure 5.9, we find that Filter exerts closer to a quadratic runtime because, on average, new records scale μ by a constant factor, implying that it scales with m.

Figure 5.10: Garble - Runtime



Garble exerts a linear runtime, which makes sense as we kept the padding length σ constant, implying that θ is scaled with n. The magnitude of the two algorithms is in seconds, so they are feasible as is. However, it is important to highlight that these results do not represent industry-standard performance as the encryption is done in software, and the generation of the inverted index matrix is done rather simplistically. The linear time complexity in Garble also backs up our assumption that new records, on average, introduce the same amount of new data.

Figure 5.11: Search - Runtime



Lastly, we verify the constant time complexity of Search, with a runtime of around 0.15 seconds and communication cost of 0.31 megabytes sent by the client and 0.31 megabytes sent by the server.

5.4.4 Score-based Search

The inverted index matrix A is not limited to two values per tuple; we can add additional elements to include new information related to the search. Imagine that we have a scoring system consisting of grades; each refers to different variants of our database where the details in the records vary. The challenge is ensuring that the client retrieves records from the correct database. A neat property of PDS is that the search and retrieval are independent; therefore, if the client performs a search and learns what grade it is qualified for, it can then send that grade to the server to gain access to the corresponding variant of the database. In this way, we further empower the server to enforce better access control of the database.

To include the grades we add a third column in A with the grade corresponding to each keyword v_j. The encryption follows the same idea as with the encryption of the indices. We pick a new encryption key e_3 that is combined with v_j to derive an ephemeral key $e_{3,j} \leftarrow \mathsf{Encrypt}\,(v_j, e_3)$, that encrypts the grade in row j. In the transfer phase, the client and server require an additional OPRF execution for the client to get its search query q encrypted under e_3, so that it can decrypt the grade in the row it is interested in. Once it has the grade, it sends it to the server, which grants it access to the corresponding database variant, and then the client continues by retrieving the relevant records.

There are challenges with this score-based search as the grades can work as direct identifies, and supporting multiple variants of the database brings with it the question of feasibility.

5.5 Semantic Search

The second type of search we present is a semantic search that uses a LLM to capture the semantics of objects by describing them as vector embeddings. To compare objects, we use the distance between them to describe their semantic likeness, meaning that a shorter distance implies similarity. In our case, we want the comparison to be done without the server learning anything about the search query q and the client only learning the set of indices I of the records with a distance shorter than some threshold t. Compared to the keyword search, this type of search is not a straightforward PSI protocol as we are not asserting whether two objects are equal but if they are similar. Therefore, we use generic MPC to compute the distance between elements. We split the protocol into an initiation and transfer phase. In the initiation, the server creates vector embeddings of all the records and store them in an indexing A. In the transfer phase, the client and server compute the distance between every vector in A and the embedding of the search query in MPC. The output is the set of indices I, which contains the index of every record with a distance shorter than some threshold t to q'. In the end, we implement the protocol and discuss the experimental results.

5.5.1 Initiation

To begin, the server use some LLM Γ to create vector embeddings of each record. Γ takes the values of a record R_ι^{values} as input and outputs a vector γ_ι. We pair the vectors with the index of the record they embed, (γ_ι, ι), to form an indexing A. To satisfy the access control policy of the client's identity, $\mathcal{U}$, the values in the records are filtered. This is done by creating a new copy of the record R'^{values}_ι where we add all the allowed values by the access control. Then, the copy of the record is feed into the LLM to produce a vector embedding that captures intended searchable semantics of the record.

```
Algorithm 22: Server - Filter
Input: Client's Identity $\mathcal{U}$
Input: Database $D \leftarrow \left[R_1^{\text{values}}, R_2^{\text{values}}, ..., R_m^{\text{values}}\right]$
Input: Large Language Model $\Gamma$
Output: Auxiliary information $A$

$A \leftarrow [\ \ ]$
for $R_\iota^{\text{values}}$ in $D$ do
    $R'^{\text{values}}_\iota \leftarrow [\ \ ]$
    for $v_j$ in $R_\iota^{\text{values}}$ do
        if $v_j$ allowed by $\mathcal{U}$ then
            $R'^{\text{values}}_\iota$.add($v_j$)
        end
    end
    $\gamma_\iota \leftarrow \Gamma\left(R'^{\text{values}}_\iota\right)$
    $A$.add($\gamma_\iota, \iota$)
end

return $A$
```

5.5.2 Transfer

```
Algorithm 23: Client $\mathcal{U}$ - Search
Input: $q' \leftarrow \Gamma\,(q)$
Output: $I$

multi-party computation:
    Input: $q'$
    Output: $I$
    $I \leftarrow [\ \ ]$
    for $(\gamma_\iota, \iota)$ in $A$ do
        $\delta \leftarrow$ Distance $(q', \gamma_\iota)$
        if $\delta < t$ then
            $I$.add($\iota$)
        end
    end
end

return $I$
```

```
Algorithm 24: Server - Search
Input: $A$
Output:

multi-party computation:
    Input: $A$
    Output:
    $I \leftarrow [\ \ ]$
    for $(\gamma_\iota, \iota)$ in $A$ do
        $\delta \leftarrow$ Distance $(q', \gamma_\iota)$
        if $\delta < t$ then
            $I$.add($\iota$)
        end
    end
end

return
```

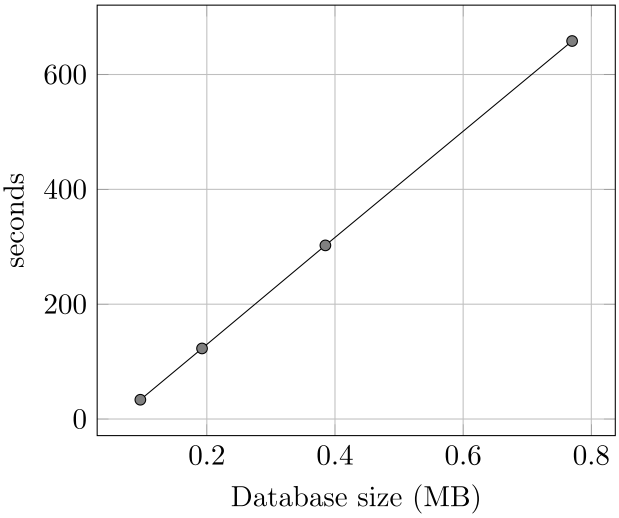
In the transfer, the client derives a vector embedding $q' \leftarrow \Gamma(q)$ for q. With generic MPC, it and the server calculate the distance δ to all the embeddings γ_ι in A. If $d \leq \tau$ then the index ι is added I which is revealed to the client.

5.5.3 Results

The time complexity of Search is $\mathsf{O}(m)$ as the search embedding q' is compared to every record embedding γ_i. Further, we notice that this time complexity is for computations done in MPC, which is especially costly.

We choose a LLM Γ with an embedding dimension of 768, meaning that our embedding vectors γ_i are of length 768, and we instantiate Distance with Euclidean distance.

Figure 5.12: Search - Runtime



In the experiment, we find that the runtime of Search indeed follows a linear time complexity. It shows feasibility for smaller-sized databases, but it quickly becomes infeasible as a doubling in size leads to a doubling in the runtime.

5.6 Protocol Assessment

We assess the proposed protocol, subprotocols, and the experimentation results in the context of our described PDS properties. We also discuss compliance with the privacy description and its practical implications. Further we elaborate on the experimental results in the context of our timely and storage imbalanced descriptions to discuss their practicability. Lastly, we discuss post-quantum security in the protocol by looking at each subprotocol.

5.6.1 Privacy

We describe privacy with three games, Algorithm 1, Algorithm 2 and Algorithm 3. To discuss privacy in the protocol, we show that an adversary cannot gain any significant advantage in any of the games. We consider each game individually, starting with the server's privacy in the search, then the retrieval, and lastly, the client's privacy in the protocol.

Server privacy in the keyword search: Let us consider Algorithm 1 for the keyword search protocol. In this game the server is the challenger and the client is the adversary. Recall that the adversary is a PPT algorithm with access to the query functionality of the Oracle. In the query functionality, the oracle uses the Garble algorithm to derive responses to the adversary. Its privacy is described such that if the adversary cannot distinguish between a response from Garble of using the auxiliary information A and some random substitute \$, with an advantage greater than $\mathcal{E}(\lambda)$, then we say that the search is private. The response to the adversary is the encoded auxiliary information A'.

Referencing the communication flow of the initiation of the keyword search, Figure 5.2, we have to show that Garble produces A' such that it perfectly hides its contents since it is sent to the client. In Garble, the encryption of v'_j is produced for every value v_j in A'. Here, it follows that every v'_j is pseudorandom and unique, as we instantiated $\mathsf{Encrypt}$ with AES and every input $\mathsf{Hash}(v_j)$ was unique. For every set I_j, an ephemeral key $e_{2,j}$ is produced by encrypting v_j with e_2, giving the encryption of the set a unique encryption key, yielding a unique encryption of every I_j as each index ι also is unique. Furthermore, we padded I_j with unique values, hence, all the values in A' are pseudorandom, padded to the same length, and therefore indistinguishable across iterations of Garble.

Next, we reference the communication flow of the transfer phase of the keyword search in the overview, Figure 5.4. We do this to show the correctness of Search and to verify to ourselves that no additional information about A' is leaked. The encryption of the client's search query q'_i under e_1 and e_2 is done with an OPRF, and the OPRF is realized using generic MPC thereby its privacy follows from it. The last thing we must show is that Search reveals no more information than a single set I_j. The client can only match q_{e_1} with one or less v'_j in A' at a time, and given that it has a positive match, it can only construct the ephemeral key $e_{2,j}$ for that set of indices I'_j. Thus, the client can, at most, decrypt one set of indices in A' per transfer. The size of A' reveals information about m and n, but recall that the client already knows these values from the protocol. Further, the values of the indices that the client learns between sessions cannot be correlated as D is shuffled in Setup.

Server privacy in the semantic search: Let us consider Algorithm 1 for the semantic search. In this game the server is the challenger and the client is the adversary.

Recall the communication flow in the initiation, Figure 5.3, shows that no data is exchanged between the client and server. In the transfer phase the communication flow, Figure 5.5, shows that Search is computed in MPC. Therefore, it follows from MPC that Search reveals no information about A (excluding the values of m and n); thus, the client only learns the set of indices I_j, showing that no additional information is leaked.

Server privacy in the retrieval: Let us consider Algorithm 2 for the OT protocol. In this game the server is the challenger and the client is the adversary. Recall that the adversary is a PPT algorithm with access to the query functionality of the Oracle. In the query functionality, the oracle uses the Retrieve algorithm to derive responses to the adversary. Its privacy is described such that if the adversary cannot distinguish between a response from Retrieve of its original query item I'_i and some random substitute \$ with an advantage greater than $\mathcal{E}(\lambda)$ then we say that Retrieve is private.

Recall that Retrieve fetches only the associated record with the indices and returns that ciphertext F'_i to the client. Hence, it follows that Retrieve reveals no other information about the database as it precisely fetches the relevant records. In PreProcess each record is encrypted using CTR-AES and shuffled randomly according to some permutation. Additional, the shuffling is done with an oblivious sorting algorithm which has the same access patterns on the records regardless of the permutation. The swaps are made private by the MPC-based PS protocol, which performs the encryption and swapping using generic MPC, so its privacy follows from there.

Client privacy in the protocol: Let us consider Algorithm 3 for the protocol. In this game the client is the challenger and the server is the adversary. Recall that the adversary is a PPT algorithm with access to the query functionality of the Oracle. In the query functionality, the oracle uses the Encode algorithm to derive responses to the adversary. Its privacy is described such that if the adversary cannot distinguish between a response of its original query item I_i and some random substitute \$ with an advantage greater than $\mathcal{E}(\lambda)$ then we say that protocol is private for the client. The response to the adversary input I'_i is a set of encoded indices, achieved by running Encode to encode them.

Referring to the communication flow in the transfer phase of both the keyword and semantic search, Figure 5.4 and Figure 5.5, the client's privacy follows from generic MPC. Leaving the remaining information the client sends to the server in the PreProcess, Figure 5.1. The PreProcess algorithm is simplified in the overview, but recall that the client uses an oblivious sorting algorithm in combination with PS to shuffle and encrypt the database. In the generic MPC-based PS protocol, Algorithm 11 and Algorithm 12, CTR-Encrypt is used, which provides IND-CPA security, thus making the encrypted records indistinguishable from the server. Following from the oblivious sorting algorithm is that the access pattern the client uses reveals no information about the swaps it performs. Thus, the random permutation π puts the records in a random order such that the original location of the records cannot be correlated with the new ones by the server. Therefore, using π to encode the set of indices I_i in Encode reveals no information about the indices. Further, recall that dummy indices are used to pad each set of encoded records in Encode. Hence, the set of encoded indices I_i' reveals no information about the records retrieved and the result of the search, given the set of dummy indices η is not exhausted.

Downsides to the privacy description: In our privacy descriptions we deviated from the IND-CPA definition, often implicitly referred to as semantic security, by introducing an encoding oracle. By doing so, we lose the multi-tenancy property that implicitly follows from PIR's privacy definition [24]. To demonstrate why, let us consider the proposed protocol in our work. In it, Encode uses a permutation to encode a set of indices sent to the server, something that works with a single client as it keeps track of its requests and never uses the same indices. With multiple clients, under the assumption that the clients do not cooperate, this becomes a problem as Encode produces the same output for clients who want to retrieve the same records, which, when sent to the server reveals that those clients have equivalent searches. Thus, the key observation is that Encode should satisfy a non-deterministic semantic security definition to allow for multi-tenancy, which implicitly brings with it that Retrieve also has to be non-deterministic. Depending on the protocol, Garble also could be non-deterministic as encoding the database deterministic could reveal information about whether the contents of the database has changed. For an instance in our protocol if the client receives an identical indexing on two different initiations of the database it can infer that no new records were added or deleted.

5.6.2 Timely

The most costly algorithm in the protocol is the PreProcess. In the experiment, we demonstrate that within a time budget τ of eight hours, we can initiate a database of megabyte size, with the use of the generic MPC-based PS protocol and without any parallelization of bitonic sort. Further, we estimate that with perfect parallelization and a time budget τ of eight hours, a database of size 2^{393} bytes could be initiated.

The limiting factor is the runtime of performing a private swap, being why the OT-based private swap protocol is an important realization. Its efficiency is directly equivalent to the efficiency of OT_1^2, which is among the most fundamental and most researched topics in secure computation. We approximate that it allows us to initiate a database in the gigabyte range if swapped with the generic MPC PS protocol in PreProcess. Providing an implementation to test the efficiency also answers the question about the efficiency of initiating the PS protocol, something we have not addressed with the generic MPC-based PS protocol, which requires a new set of beaver's triples per execution. In our experiments, we assumed that the circuits were already pre-compiled, so for simplicity, we reused the same circuit thereby not factoring in the compilation time in our runtime.

Regarding the semantic search, it does not meet our timely definition of exerting sublinear complexity in the transfers, and we demonstrated that its efficiency is limited to smaller database sizes. What we can take away from the experiment is that it is possible to support embedding dimensions from state-of-the-art LLMs. Outside the scope of this work is the accuracy of these models on our data. Therefore, the open question is, what the minimal required embedding dimension is to satisfy a privacy-preserving semantic search on the records? To answer this question is necessary for a proper efficiency assessment of a private semantic search protocol as the vector length directly affects the number of required operations.

5.6.3 Storage Imbalance

The dominant space complexity is in the PreProcess algorithm. In it, the client picks a key and a counter to produce a key stream for the encryption of every item in the database. This means that the client, at most, has to store a key and a counter for n items. The counters and keys are each of size β, which in bytes are $\frac{\beta}{8}$. We combine this with the records being of ω bytes, giving us the following fraction to calculate their difference $\frac{4 \cdot \omega}{\beta}$. By our definition of storage imbalanced, we say that the protocol is $(\frac{4 \cdot \omega}{\beta})$-imbalanced. We use our experimental data, $\omega = 6016$ and $\beta = 128$, we find that the proposed solution is 188-imbalanced, meaning the server requires 188 times more storage than the client.

5.6.4 Post-Quantum Security

One of this work's main objectives is to avoid using post-quantum insecure hardness assumptions. However, the program used in MP-SPDZ to perform generic MPC in the experiments was semi-party.x, which denotes a stripped-down version of the MASCOT [27] protocol. MASCOT performs computations with arithmetic circuits, that uses Beaver triples [5]. These triples are distributed among the parties with OT, and even though most of the OTs are realized with symmetric primitives through OT-extension, some base OTs that use asymmetric encryption are still required. Given the results from the work of Boyle et al. [8], it is fair to assume that the base OTs can be performed to the same magnitude of efficiency with a post-quantum secure hardness assumption. This suggests that the magnitude of efficiency presented in this work still holds.

In the keyword search, we implemented an OPRF using generic MPC, that can be replaced with a post-quantum secure ORPF [1].

Regarding the OT-based PS protocol, the results presented were comparable to the efficiency of the post-quantum secure OT protocol from Boyle et al. mentioned above. Thus, its results still seem promising. This is important as the PS protocol used in the experiments used generic MPC and still yielded much worse performance than we approximated the OT-based PS protocol to yield. [1]

[1] The semantic search utilizes generic MPC but is excluded due to its poor experimental results.

Chapter 6

Private Database Search Program

Our core objective is to demonstrate feasibility. Therefore, along the way, we have implemented the proposed solutions as programs and used them in experiments to extract metrics like runtime and total communication costs. So far, it might seem like these programs are stand-alone implementations, but they are, in fact, subprograms in one large complete implementation. We refer to this complete implementation as the program, and it consists of over 6000 lines of code (including docstrings) that features include the proof of concept (Chapter 4) application and private database search (Chapter 5) application.

6.1 Program Implementation

We recall from the experimental setup, Section 1.3, that we use Python 3.10 as our primary programming language and MP-SPDZ 0.3.8 to perform generic MPC. In Section 4.3 we discovered that semi-party.x gave us the quickest runtime, therefore, we instantiate MP-SPDZ with it. For communication outside MP-SPDZ, we use the socket library, a part of core Python, to establish a secure channel using Transport Layer Security (TLS) 1.3. To integrate MP-SPDZ with Python, the client and server write their input to separate local files that MP-SPDZ reads; once the computation is complete, MP-SPDZ writes the output in an output file located in the directory of the party that should learn it. Then, that file is read in Python to retrieve the output. This means that all communication between the client and server goes through either the secure communication channel or MP-SPDZ, which makes the parties independent, allowing them to persist on different systems and communicate over Ethernet / Internet Protocol (IP).

Once the client and server are online, the client can choose whether to perform a semantic search or keyword search. The client can also choose whether or not to continue from the previous session, thereby allowing the parties to preprocess the initiation phase, then go offline and return online later to perform the transfers without having to redo the initiation.

We have split the program into two phases: the initiation and the transfer phases. The initiation is done once, and once complete, we can perform k number of transfers constrained by the number of dummy records in the database.

In the initiation, the database and either the keyword search or semantic search are initialized, depending on the client's choice at the beginning. The idea in the database initiation is for the client to shuffle and encrypt the database obliviously. First, the server fills the database with new records, Section 4.1, and pads them by implementing Algorithm 8. Crucially, the order of the indices on the records are randomly shuffled. Then, the client and server perform the shuffling and encryption of the records by implementing Algorithm 9 and Algorithm 10, where we implemented Sort as bitonic sort, Permute with the core Python library random, use the generic MPC-based PS protocol, Algorithm 11 and Algorithm 12, and to generate key streams we use CTR-Encrypt as CTR-AES, provided to use by the cryptography library [13]. KeyGen is implemented by getting random bit strings from /dev/urandom to produce nonces and encryption keys. Next, either the keyword search or semantic search is initiated.

In the case of the keyword search, the server creates and encrypts an inverted index matrix where it enforces access control by filtering unwanted information when reading the records. To do this, the server implements Algorithm 18 and Algorithm 19, we use Hash as SHAKE and Encrypt as AES, both of which are provided by the cryptography library.

In the case of the semantic search, the server creates an indexing containing vector embeddings of each record. It does so by implementing Algorithm 22, and Γ we use the all-mpnet-base-v2 model from Sentence-BERT (SBERT) [39].

The communication flow between the client and server in the initiation implements Figure 5.1, Figure 5.2 and Figure 5.3, respectively.

In the transfer(s), the database search and record retrieval are executed. In the case of the keyword search, the client gets its search query encrypted under the server's keys, then performs the lookup of the encrypted inverted index matrix and decrypts one row, reavling a set of indices. The client, therefore, implements Algorithm 20, with Decrypt as AES provided by the cryptography library, and the server implements Algorithm 21. For both algorithms Encrypt uses the AES circuit provided with MP-SPDZ. In the case of the semantic search, the client and server together compute the distances between the search query and all the vector embeddings, then return the indices of embeddings that are closer than some threshold. This is achieved by the client implementing Algorithm 23 and the server implementing Algorithm 24; we use Euclidean distance for Distance.

Once the search is complete, the client wants to retrieve the records corresponding to the indices it learned. To do so, it implements Algorithm 15, where Encode uses the permutation, that it used to shuffle the database, to translate the indices to their new location, and pads it to a fixed length using dummy record indices. Then, the server implements Algorithm 16, which collects the encrypted records for the encoded indices. Lastly, the client implements Algorithm 17, where Decode reproduces the key stream with CTR-AES, provided by the cryptography library, to decrypt each record.

The communication flow between the client and server implements Figure 5.4, Figure 5.5 and Figure 5.6, respectively.

Limitations in the implementation:

- The mock records' itinerary does not have to include a Norwegian airport.
- The X.509 certificates are not signed by a trusted Certificate Authority (CA), but rather self-signed; therefore, the client and server's certificates are manually added as trusted by each other to enable authentication in TLS.
- For demonstrative purposes the semantic search returns the whole array of indices and distances to the client, then the client always picks the closes record to retrieve regardless of its distance. A simplification done as the LLM is not trained on mock records and is therefore not accurate.

The final experiment demonstrates that the program allows the client to perform private searches on a mock PNR registry, it was performed in February 2024 with members from Kripos and other competent authorities present. We showcased how this program with the keyword search is feasible to use in an investigation and discussed some of the challenges surrounding research, development, and legislative compliance. For those who were not present, similar experiments can be recreated as the entire program is made public on `https://github.com/BenjaminHansenMortensen/PrivateDatabaseSearch`. [1]

[1] We do not make any security guarantees, as this implementation is intended for private and demonstrative use.

Chapter 7

Conclusion

For many decades, secure computation researchers have focused on developing privacy-enhancing technologies, working their way from inefficient, but fascinating, solutions to practical realizations adopted in the real world. Due to increased interest in the field and improvements in computational power, we now find ourselves in an era where the question is no longer if it can be done but whether if it is feasible.

The cooperation between the security services is essential to national security, but sharing classified intelligence is not straightforward due to security concerns and its potential infringement of the rights of the natural person. We considered the case of the PNR registry, where an apparent deadlock occurs when a client wants to perform a query, containing classified information, on the registry but is incapable of doing so as it cannot share the classified information. Such a case inhibits the purpose of the registry and limits cooperation.

We showed that this apparent deadlock is resolvable with the use of MPC, demonstrating that one of our proposed solutions can support a megabyte-size database. Further, we approximate that another proposed solution should improve those results to the order of gigabytes. These solutions not only preserve the privacy of the involved parties, allowing them to take full advantage of the registry, but also increase the capability of searches on the registry, as demonstrated with the keyword search protocol. These feasible results are enabled by PDS, which facilitates combining PSI protocols with OT protocols to empower various types of searches and protocol properties. With our implementation we show that both a keyword search and semantic search are possible, but we remark that only the keyword search is feasible.

A question left open is the implication of a WLAN setting on the results; practitioners must consider that even a low network latency might not be sufficient. For example, our proposed OT protocol, Section 5.2 and Section 5.3, for record retrieval requires a quasilinear number of messages exchanged between the client and server, thus making even a 1 ms network latency too long for large database sizes. Therefore, future work should consider using that state-of-the-art SPIR protocol XSPIR [33] for the retrieval of records, to better understand the database sizes that can be supported.

Further, we emphasize the insightful observation that Encode and Retrieve needs to exert non-deterministic semantic security in order to allow for multi-tenancy and that the communication flow between the client and server was not captured in our privacy definition to suggest replacing the game-based approach with a simulation-based approach as it would better allow for describing the ideal functionality of PDS protocols, something that is key in steps towards malicious security.

All things considered, it is exciting to see that MPC is feasible for private searches on private databases, as similar problems are prevalent in processes that consider a database with personal data or classified information. We remark however that in the case of the PNR registry the additional involvement of jurists and others is necessary for defining the right requirements surrounding suitable solutions.

Bibliography

[1] Martin R. Albrecht et al. ‘Round-Optimal Verifiable Oblivious Pseudorandom Functions from Ideal Lattices’. In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, 2021, pp. 261–289. DOI: 10.1007/978-3-030-75248-4_10.

[2] David Archer et al. *Welcome to pyca/cryptography*. 2023. URL: https://cryptography.io (visited on 19/11/2023).

[3] Kenneth E. Batcher. ‘Sorting Networks and Their Applications’. In: *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*. Vol. 32. AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968, pp. 307–314. DOI: 10.1145/1468075.1468121. URL: https://doi.org/10.1145/1468075.1468121.

[4] Donald Beaver. ‘Correlated Pseudorandomness and the Complexity of Private Computations’. In: *28th Annual ACM Symposium on Theory of Computing*. Philadephia, PA, USA: ACM Press, 1996, pp. 479–488. DOI: 10.1145/237814.237996.

[5] Donald Beaver. ‘Efficient Multiparty Protocols Using Circuit Randomization’. In: *Advances in Cryptology – CRYPTO’91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[6] Michael Ben-Or, Shafi Goldwasser and Avi Wigderson. ‘Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)’. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[7] Dan Boneh et al. ‘Private Database Queries Using Somewhat Homomorphic Encryption’. In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Ed. by Michael J. Jac-

obson Jr. et al. Vol. 7954. Lecture Notes in Computer Science. Banff, AB, Canada: Springer, Heidelberg, Germany, 2013, pp. 102–118. DOI: 10.1007/978-3-642-38980-1_7.

[8] Elette Boyle et al. 'Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation'. In: *ACM CCS 2019: 26th Conference on Computer and Communications Security.* Ed. by Lorenzo Cavallaro et al. London, UK: ACM Press, 2019, pp. 291–308. DOI: 10.1145/3319535.3354255.

[9] Jan Camenisch, Maria Dubovitskaya and Gregory Neven. 'Oblivious transfer with access control'. In: *ACM CCS 2009: 16th Conference on Computer and Communications Security.* Ed. by Ehab Al-Shaer, Somesh Jha and Angelos D. Keromytis. Chicago, Illinois, USA: ACM Press, 2009, pp. 131–140. DOI: 10.1145/1653662.1653679.

[10] Jan Camenisch et al. 'Short Threshold Dynamic Group Signatures'. In: *SCN 20: 12th International Conference on Security in Communication Networks.* Ed. by Clemente Galdi and Vladimir Kolesnikov. Vol. 12238. Lecture Notes in Computer Science. Amalfi, Italy: Springer, Heidelberg, Germany, 2020, pp. 401–423. DOI: 10.1007/978-3-030-57990-6_20.

[11] Benny Chor et al. 'Private Information Retrieval'. In: *36th Annual Symposium on Foundations of Computer Science.* Milwaukee, Wisconsin: IEEE Computer Society Press, 1995, pp. 41–50. DOI: 10.1109/SFCS.1995.492461.

[12] I. Damgard et al. *Multiparty Computation from Somewhat Homomorphic Encryption.* Cryptology ePrint Archive, Report 2011/535. https://eprint.iacr.org/2011/535. 2011.

[13] Individual Contributors. Revision ea71c070. *'Bristol Fashion' MPC Circuits.* 2013. URL: https://homes.esat.kuleuven.be/~nsmart/MPC/ (visited on 13/05/2024).

[14] Shimon Even, Oded Goldreich and Abraham Lempel. 'A Randomized Protocol for Signing Contracts'. In: *Advances in Cryptology – CRYPTO'82.* Ed. by David Chaum, Ronald L. Rivest and Alan T. Sherman. Santa Barbara, CA, USA: Plenum Press, New York, USA, 1982, pp. 205–210.

[15] Michael J. Freedman, Kobbi Nissim and Benny Pinkas. 'Efficient Private Matching and Set Intersection'. In: *Advances in Cryptology – EUROCRYPT 2004.* Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Interlaken, Switzerland: Springer, Heidelberg, Germany, 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1.

[16] Michael J. Freedman et al. 'Keyword Search and Oblivious Pseudorandom Functions'. In: *TCC 2005: 2nd Theory of Cryptography Conference.* Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, 2005, pp. 303–324. DOI: 10.1007/978-3-540-30576-7_17.

[17] Craig Gentry. 'Fully homomorphic encryption using ideal lattices'. In: *41st Annual ACM Symposium on Theory of Computing.* Ed. by Michael Mitzenmacher. Bethesda, MD, USA: ACM Press, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[18] Yael Gertner et al. 'Protecting Data Privacy in Private Information Retrieval Schemes'. In: *30th Annual ACM Symposium on Theory of Computing.* Dallas, TX, USA: ACM Press, 1998, pp. 151–160. DOI: 10.1145/276698.276723.

[19] Yael Gertner et al. 'The Relationship between Public Key Encryption and Oblivious Transfer'. In: *41st Annual Symposium on Foundations of Computer Science.* Redondo Beach, CA, USA: IEEE Computer Society Press, 2000, pp. 325–335. DOI: 10.1109/SFCS.2000.892121.

[20] Oded Goldreich. 'Towards a Theory of Software Protection and Simulation by Oblivious RAMs'. In: *19th Annual ACM Symposium on Theory of Computing.* Ed. by Alfred Aho. New York City, NY, USA: ACM Press, 1987, pp. 182–194. DOI: 10.1145/28395.28416.

[21] Oded Goldreich, Silvio Micali and Avi Wigderson. 'How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority'. In: *19th Annual ACM Symposium on Theory of Computing.* Ed. by Alfred Aho. New York City, NY, USA: ACM Press, 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[22] Oded Goldreich and Rafail Ostrovsky. 'Software Protection and Simulation on Oblivious RAMs'. In: *Journal of the ACM* 43.3 (1996), pp. 431–473. DOI: 10.1145/233551.233553.

[23] Vipul Goyal et al. 'ATLAS: Efficient and Scalable MPC in the Honest Majority Setting'. In: *Advances in Cryptology – CRYPTO 2021, Part II.* Ed. by Tal Malkin and Chris Peikert. Vol. 12826. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, Germany, 2021, pp. 244–274. DOI: 10.1007/978-3-030-84245-1_9.

[24] Ryan Henry. 'Tutorial: Private Information Retrieval'. In: *ACM CCS 2017: 24th Conference on Computer and Communications Security.* Ed. by Bhavani M. Thuraisingham et al. Dallas, TX, USA: ACM Press, 2017, pp. 2611–2612. DOI: 10.1145/3133956.3136069.

[25] Yuval Ishai and Eyal Kushilevitz. 'Private Simultaneous Messages Protocols with Applications'. In: *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings.* IEEE Computer Society, 1997, pp. 174–184. DOI: `10.1109/ISTCS.1997.595170`. URL: `https://doi.org/10.1109/ISTCS.1997.595170`.

[26] Marcel Keller. 'MP-SPDZ: A Versatile Framework for Multi-Party Computation'. In: *ACM CCS 2020: 27th Conference on Computer and Communications Security.* Ed. by Jay Ligatti et al. Virtual Event, USA: ACM Press, 2020, pp. 1575–1590. DOI: `10.1145/3372297.3417872`.

[27] Marcel Keller, Emmanuela Orsini and Peter Scholl. 'MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer'. In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security.* Ed. by Edgar R. Weippl et al. Vienna, Austria: ACM Press, 2016, pp. 830–842. DOI: `10.1145/2976749.2978357`.

[28] Marcel Keller, Valerio Pastro and Dragos Rotaru. 'Overdrive: Making SPDZ Great Again'. In: *Advances in Cryptology – EUROCRYPT 2018, Part III.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. Lecture Notes in Computer Science. Tel Aviv, Israel: Springer, Heidelberg, Germany, 2018, pp. 158–189. DOI: `10.1007/978-3-319-78372-7_6`.

[29] Joe Kilian. 'Founding Cryptography on Oblivious Transfer'. In: *20th Annual ACM Symposium on Theory of Computing.* Chicago, IL, USA: ACM Press, 1988, pp. 20–31. DOI: `10.1145/62212.62215`.

[30] Ilan Komargodski, Moni Naor and Eylon Yogev. 'How to Share a Secret, Infinitely'. In: *TCC 2016-B: 14th Theory of Cryptography Conference, Part II.* Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, 2016, pp. 485–514. DOI: `10.1007/978-3-662-53644-5_19`.

[31] Eyal Kushilevitz and Rafail Ostrovsky. 'Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval'. In: *38th Annual Symposium on Foundations of Computer Science.* Miami Beach, Florida: IEEE Computer Society Press, 1997, pp. 364–373. DOI: `10.1109/SFCS.1997.646125`.

[32] Benoît Libert et al. 'Adaptive Oblivious Transfer with Access Control from Lattice Assumptions'. In: *Advances in Cryptology – ASIACRYPT 2017, Part I.* Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Hong Kong, China: Springer, Heidelberg, Germany, 2017, pp. 533–563. DOI: `10.1007/978-3-319-70694-8_19`.

[33] Chengyu Lin, Zeyu Liu and Tal Malkin. 'XSPIR: Efficient Symmetrically Private Information Retrieval from Ring-LWE'. In: *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part I.* Ed. by Vijayalakshmi Atluri et al. Vol. 13554. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, Heidelberg, Germany, 2022, pp. 217–236. DOI: `10.1007/978-3-031-17140-6_11`.

[34] Dahlia Malkhi et al. 'Fairplay - Secure Two-Party Computation System'. In: *USENIX Security 2004: 13th USENIX Security Symposium.* Ed. by Matt Blaze. San Diego, CA, USA: USENIX Association, 2004, pp. 287–302.

[35] Catherine Meadows. 'A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party'. In: *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986.* IEEE Computer Society, 1986, pp. 134–137. DOI: `10.1109/SP.1986.10022`. URL: `https://doi.org/10.1109/SP.1986.10022`.

[36] Moni Naor and Benny Pinkas. 'Oblivious Transfer with Adaptive Queries'. In: *Advances in Cryptology – CRYPTO'99.* Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1999, pp. 573–590. DOI: `10.1007/3-540-48405-1_36`.

[37] Rafail Ostrovsky. 'Efficient Computation on Oblivious RAMs'. In: *22nd Annual ACM Symposium on Theory of Computing.* Baltimore, MD, USA: ACM Press, 1990, pp. 514–523. DOI: `10.1145/100216.100289`.

[38] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer.* Cryptology ePrint Archive, Report 2005/187. `https://eprint.iacr.org/2005/187`. 2005.

[39] Nils Reimers and Iryna Gurevych. 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks'. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, Nov. 2019. URL: `https://arxiv.org/abs/1908.10084`.

[40] Neil James Alexander Sloane. *The On-Line Encyclopedia of Integer Sequences (OEIS).* 1964. URL: `https://oeis.org/` (visited on 12/05/2024).

[41] Andrew Chi-Chih Yao. 'How to Generate and Exchange Secrets (Extended Abstract)'. In: *27th Annual Symposium on Foundations of Computer Science.* Toronto, Ontario, Canada: IEEE Computer Society Press, 1986, pp. 162–167. DOI: `10.1109/SFCS.1986.25`.

Glossary

\$ A random substitute for a variable.

α Additional time spent on landing and take-off when flying between two airports.

β The block size.

χ The length of the search query q. $\chi \in \mathbb{Z}^{+}$.

δ Distance between two vectors.

η The set of dummy record indices.

Γ A large language model that outputs vector embeddings.

ι An index, $i \in \{1, 2, ..., n\}$.

λ The security parameter.

ν The length of the indices I, $\nu \in \mathbb{Z}^{+}$.

ω The upper bound number of characters of any possible record.

π A random permutation.

ρ The length of the auxiliary information A. $\rho \in \mathbb{Z}^{+}$.

σ The length of the encoded set of indices I'. $\sigma \in \mathbb{Z}^{+}$.

τ A time budget for the protocol initiation.

υ A set of already retrieved indices.

Ξ A tuple of system parameters of the server.

γ_i A vector embedding produce with Γ.

ζ_i A server with a share of a replicated database.

A Auxiliary information about the database, $A \in \{0, 1\}^{\rho}$.

A' The encoded auxiliary information A.

C'_i/C'_j A masked ciphertext.

C_i/C_j A ciphertext.

D The database.

D' The encoded database.

E An encryption scheme.

F The retrieved records, $F \subset D$.

F' The retrieved encrypted records.

H A history of responses.

$I/I_i/I_j$ A set of indices, $I \in \{1, 2, ..., n\}^{\nu}$.

$I'/I'_i/I'_j$ A set of encoded indices, $I' \in \{1, 2, ..., n\}^{\sigma}$.

K The client's encryption keys and other encryption varliables.

M_0/M_1 A masked record, $M_0, M_1 \in (0, 1)^{\omega}$.

N_0/N_1 A masked record, $N_0, N_1 \in (0, 1)^{\omega}$.

O An intersection between two sets.

P_i A party.

X An arbritrary set of elements.

Y A set of elements.

b A single bit, 0 or 1.

b' The adversary's prediction bit, 0 or 1.

c_i/c_j A counter.

d_i A dummy record.

$e_i/e_{i,j}$ An encryption encryption key.

i An index or iteration, $i \in \{1, 2, ..., n\}$.

i' An encoded index, $i' \in \{1, 2, ..., n\}$.

k The number of rounds of transfers, $k \in \{1, 2, ..., n\}$.

m The number of records in the database, $m \in \mathbb{Z}^+$.

n The length of the database. $n \in \mathbb{Z}^+$.

p A probability, $p \in (0, 1)$.

q The client's search query, $q \in \{0, 1\}^{\chi}$.

q' The client's encoded search query.

q_{e_i} The client's search query encrypted with some ephemeral key e_i.

s'_i/s'_j A masked key stream, $s'_i, s'_j \in (0, 1)^{\omega}$.

s_i/s_j A key stream, $s_i, s_j \in (0, 1)^{\omega}$.

t The vector distance threshold, $t \in \mathbb{R}^+$.

w A wire in or out to a logic gate.

x A finite field element.

y A finite field element.

z A natural number.

$\mathcal{A}$ A probabilistic polynomial-time algorithm used to denote an adversary.

$\mathcal{C}$ A circuit description of a function.

$\mathcal{E}(\lambda)$ A negligible function $\mathcal{E} : \mathbb{N} \mapsto \mathbb{R}^+$ that takes the security parameter λ.

$\mathcal{G}$ A garbled circuit.

$\mathcal{k}$ A key or label for a bit in a garbled gate.

μ The combined number of values in all the records $R_{\iota}^{\text{values}}$.

R_i/R_j A records that can take the form of R_{ι}^{chars} and $R_{\iota}^{\text{values}}$.

R_{ι}^{chars} An array of the character that makes up a record, each may be of different length denoted $|R_{\iota}^{\text{chars}}|$.

$R_{\iota}^{\text{values}}$ An array of the values in a record, each may be of different length denoted $|R_{\iota}^{\text{values}}|$.

θ The combined number of unique values in all the records $R_{\iota}^{\text{values}}$.

$\mathcal{U}$ The identity of the user of the client.

v/v_j A value in a record $R_{\iota}^{\text{values}}$.

Acronyms

AES Advanced Encryption Standard.

CA Certificate Authority.

cPDS computational Private Database Search.

CPU Central Processing Unit.

CTR Counter.

DDH Decisional Diffie-Hellman.

ECB Electronic Code Book.

FFI Norwegian Defence Research Establishment.

IFI Department of Informatics.

IND-CPA Indistinguishable under Chosen Plaintext Attack.

IND-CCA Indistinguishable under Chosen Ciphertext Attack.

IP Internet Protocol.

LAN Local Area Network.

LLM Large Language Model.

MPC (Secure) Multi-Party Computation.

OEIS The On-line Encyclopedia of Integer Sequences.

OPRF Oblivious Pseudorandom Function.

ORAM Oblivious Random Access Machine.

OT Oblivious Transfer.

PDQ Private Database Query.

PDS Private Database Search.

PIR Private Information Retrieval.

PNR Passenger Name Record.

PPT Probabilistic Polynomial-Time.

PS Private Swap.

PSI Private Set Intersection.

RAM Random Access Memory.

SBERT Sentence-BERT.

SHAKE Secure Hash Algorithm and KECCAK.

SHE Somewhat Homomorphic Encryption.

SPIR Symmetric Private Information Retrieval.

TLS Transport Layer Security.

UiO University of Oslo.

WLAN Wireless Local Area Network.

XOR Exclusive OR.

Appendices

Appendix A: Sample Mock Passenger Name Record

The following is a sample mock passenger name record, presented on JSON format.

```
{
    "PNR Number": 1,
    "Payment Information": {
        "Ticket Number ": 331390749,
        "Date": "26/11/2025",
        "Name": "Matthew Georgia Tim Carson",
        "Address": {
            "City": "Fayetteville",
            "Zip Code": "72701",
            "Street": "2017 North Hartford Drive"
        },
        "Phone Number": "+47 40489687",
        "Email": "matthew.georgia.tim.carson.@outlook.com",
        "Vendor": "Mastercard",
        "Type": "Credit",
        "Bonus Program": "Gold"
    },
    "Airline": "SAS",
    "Travel Agency": "Balslev",
    "Travel Plan": {
        "Destination 1": {
            "IATA Code": "NWH",
            "Airport Name": "Parlin Field",
            "City": "Newport",
            "Time": "15/12/2025, 01:55:00"
        },
        "Destination 2": {
            "IATA Code": "BEK",
            "Airport Name": "Bareli",
            "City": "Bareli",
            "Time": "15/12/2025, 23:35:00"
        }
    },
    "Passengers": {
        "Passenger 1": {
            "Name": "Glen Gompf",
            "Status": {
                "Destination 1": "no show",
                "Destination 2": "-"
            },
            "Seat": "6B",
            "Luggage": {
                    "Cabin": [
                        "*unknown*"
                    ],
                    "Checked": [
                        14.2
                    ],
                    "Special": []
                }
            },
            "Passenger 2": {
                "Name": "Jeffrey Bailey Cavallaro",
                "Status": {
                    "Destination 1": "showed",
                    "Destination 2": "showed"
                },
                "Seat": "6A",
                "Luggage": {
                    "Cabin": [
                        "*unknown*"
                    ],
                    "Checked": [
                        17.2,
                        15.4,
                        20.0,
                        16.3,
                        19.6
                    ],
                    "Special": []
                }
            },
            "Passenger 3": {
                "Name": "Matthew Georgia Tim Carson",
                "Status": {
                    "Destination 1": "showed",
                    "Destination 2": "showed"
                },
                "Seat": "27F",
                "Luggage": {
                    "Cabin": [
                        "*unknown*"
                    ],
                    "Checked": [
                        16.6,
                        20.4
                    ],
                    "Special": []
                }
            }
        }
    }
```

Appendix B: Proof of concept Search and Retrieve

The implementation of the combination of Search and Retrieve in MP-SPDZ in the proof of concept program.

Algorithm 25: Proof of concept - MP-SPDZ - Search & Retrieve

```
Input: Search query $q' \leftarrow \mathsf{Hash}(q)$
Input: Database
    $D \leftarrow \begin{bmatrix} 1 & R^{\text{chars}}_{1,1} & R^{\text{chars}}_{1,2} & \dots & R^{\text{chars}}_{1,\omega} \\ 2 & R^{\text{chars}}_{2,1} & R^{\text{chars}}_{2,2} & \dots & R^{\text{chars}}_{2,\omega} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m & R^{\text{chars}}_{m,1} & R^{\text{chars}}_{m,2} & \dots & R^{\text{chars}}_{m,\omega} \end{bmatrix}$
Input: Auxiliary information
    $A \leftarrow \begin{bmatrix} 1 & R^{digests}_{1,1} & R^{digests}_{1,2} & \dots & R^{digests}_{1,\sigma} \\ 2 & R^{digests}_{2,1} & R^{digests}_{2,2} & \dots & R^{digests}_{2,\sigma} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m & R^{digests}_{m,1} & R^{digests}_{m,2} & \dots & R^{digests}_{m,\sigma} \end{bmatrix}$
Output: Retrieved records $\vec{D}$

    $I \leftarrow \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$ of size $m$
for $i \in \{1, 2, \dots, m\}$ do
    for $j \in \{2, 3, \dots, \sigma + 1\}$ do
        $I_i \leftarrow I_i + A_{i,1} \cdot (q' = A_{i,j}) \cdot (I_i = 0)$
    end
end

    $\vec{D} \leftarrow \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}$ of size $m \times \omega$
for $i \in \{1, 2, \dots, m\}$ do
    for $j \in \{2, 3, \dots, \omega + 1\}$ do
        $\vec{D}_{i,j-1} \leftarrow D_{i,j} \cdot (I_i = D_{i,1})$
    end
end
return $\vec{D}$
```

Appendix C: Bitonic Sort Secquence Analysis

Referencing our implementation of bitonic sort on Github, we want to predict the number of calls to the PS protocol.

We run the program for a few different values of n and have the code count the number of calls to the PS protocol.

Database length n	Number of Private Swaps
2^1	1
2^2	6
2^3	24
2^4	80
2^5	240
2^6	672
2^7	1792
2^8	4608
2^9	11520
2^{10}	28180

Table 7.1: Bitonic Sort PS Protocol Calls with No Parallelization

Using The On-line Encyclopedia of Integer Sequences (OEIS) [40] we identify the sequence A001788 that follows the patterns, and we rewrite it to $\log_2$ such that it can take a database length n as an input, yielding the sequence:

$$a(n) = \frac{n}{4} \cdot \left(\log_2^2(n) + \log_2(n)\right)$$

We use the same approach to find the sequence that fits the fully parallelized implementation of bitonic sort. Here we collapse parallelized loops into a single call to the PS protocol,

Database length n	Number of Private Swaps
2^1	1
2^2	3
2^3	6
2^4	10
2^5	15
2^6	21
2^7	28
2^8	36
2^9	45
2^{10}	55

Table 7.2: Bitonic Sort PS Protocol Calls with Perfect Parallelization

We identify the sequence A000217 and rewrite it, yielding the sequence:

$$b(n) = \frac{1}{2} \cdot \left(\log_2^2(n) + \log_2(n)\right)$$

We now have $a(n)$ and $b(n)$ to approximate the number of private swaps required to sort a database of length n with bitonic sort.