PrivMail: A Privacy-Preserving Framework for Secure Emails (Full Version)[*]

Gowri R Chandran[1], Raine Nieminen[1], Thomas Schneider[1], and Ajith Suresh[2]

[1] ENCRYPTO, Technical University of Darmstadt, Germany
{chandran,nieminen,schneider}@encrypto.cs.tu-darmstadt.de

[2] Cryptography Research Centre, Technology Innovation Institute, Abu Dhabi, UAE
ajith.suresh@tii.ae

Abstract. Emails have improved our workplace efficiency and communication. However, they are often processed unencrypted by mail servers, leaving them open to data breaches on a single service provider. Public-key based solutions for end-to-end secured email, such as Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME), are available but are not widely adopted due to usability obstacles and also hinder processing of encrypted emails.

We propose PrivMail, a novel approach to secure emails using secret sharing methods. Our framework utilizes Secure Multi-Party Computation techniques to relay emails through multiple service providers, thereby preventing any of them from accessing the content in plaintext. Additionally, PrivMail supports private server-side email processing similar to IMAP SEARCH, and eliminates the need for cryptographic certificates, resulting in better usability than public-key based solutions. An important aspect of our framework is its capability to enable third-party searches on user emails while maintaining the privacy of both the email and the query used to conduct the search.

We integrate PrivMail into the current email infrastructure and provide a Thunderbird plugin[3] to enhance user-friendliness. To evaluate our solution, we benchmarked transfer and search operations using the Enron Email Dataset and demonstrate that PrivMail is an effective solution for enhancing email security.

Keywords: Email, secret sharing, outsourcing, private keyword search, secure two-party computation, private information retrieval

[*]Please cite the proceedings version of this paper published at ESORICS'23 [26].
[3]https://encrypto.de/code/PrivMail

Table of Contents

1 Introduction

Despite the widespread use of social media, text messages, and online messaging services such as WhatsApp, Signal, and Telegram, electronic mail (email) is still a popular method of communication, and it has a growing user base. In 2020, there were roughly 4 billion users of email; by 2024, it is predicted that there would be nearly 4.5 billion users, with an annual growth rate of 3% [53, 50]. The majority of email users are corporate companies and small businesses. For instance, 81% of small businesses rely on email as their primary customer acquisition channel, and 80% for retention [42].

As the use of emails has grown in popularity, it has also caught the attention of attackers who endanger security and privacy. One significant issue is related to data breaches, in which email servers are frequently targeted [124, 96]. Attackers may publicly release the breached data or attempt to profit by selling or negotiating with the email service provider [92, 45]. For example, the 'Collection #1' breach revealed 2.7 billion identification records with 773 million emails and made the data available for sale online [114]. Data breaches also threaten the economy significantly, with the average cost of a breach increasing by 15% from 2020 to 4.45 million USD in 2023 [61]. Another concern is the privacy of email content from the Service Provider (SP). Often, emails are processed without encryption by mail servers or are encrypted by the SP, requiring users to trust them completely. However, with the growing concern for individual privacy and the implementation of privacy laws like the EU General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) [112], many users are hesitate to use email for communicating sensitive information.

The aforementioned concerns led to the emergence of service providers such as ProtonMail [115] and Tutanota [119], who developed solutions for private emails using E2EE techniques. These techniques addressed privacy issues regarding the SP, while also allowing the users to perform search on emails. However, they pose other limitations, for example, only the user can perform the search [81, 118]. Furthermore, solutions such as E2EE, which keep the email content hidden from the SP, may not be enough in some situations and require other options. To better illustrate these concerns, we will use a company's email system as an example and provide more information below.

Example Use Case — Company Email System Consider PrivCorp, a company that wants to establish an email infrastructure for its employees while upholding individual data privacy. PrivCorp is concerned about the potential impact of data breaches, which have recently hit a number of enterprises. Furthermore, unlike some companies that monitor employee emails with or without their consent based on legal jurisdiction, PrivCorp is committed to implementing email monitoring in a privacy-preserving manner. In summary, PrivCorp's email infrastructure development goals include:

1. *Privacy from SP:* The email content[4] should be hidden from the Service Provider (SP). To achieve the desired goal, the company is willing to use multiple SPs, if needed.
2. *Data breach protection:* The email should remain private even if all but one of the SPs are compromised.
3. *Spam filtering:* The company should be able to analyse external emails and perform spam filtering in a privacy-preserving manner to respect the privacy of the email content.
4. *Unintended data leakage prevention:* The company should be able to monitor emails from its employees to the outside world in a privacy-preserving manner for accidental data leaks or other content that violates company policies such as defamation against the company, character assaults, and abusive content.

To address goals (1) and (2) above, a simple approach is to use E2EE methods, like PGP [6] and S/MIME [108], in which emails are encrypted and signed by the sender using public-key cryptography and decrypted and verified by the receiver. However, the strong privacy guarantees of E2EE make achieving goals (3) and (4) extremely difficult, as both require some sort of processing over the encrypted email content by an external entity.

Even when only goals (1) and (2) are considered, many of the current E2EE implementations suffer from usability issues [107, 105]. For instance, a usability study by Reuter et al. [102] showed that approximately

[4]Mostly the subject and content fields but not other meta data.

60% of the participants have never even heard of PGP and 64% never of S/MIME. In addition, some email clients do not even support S/MIME, making adoption difficult. Furthermore, these methods have been found to be vulnerable to practical attacks [110]. These attacks exploit outdated cryptographic primitives and force email clients to use a back-channel to extract the plaintext of encrypted emails [99] (see §2.1 for more details).

In terms of goals (3) and (4), the company wants to perform some sort of processing, such as a keyword search, on the email content in a private manner. While SSE [113] appears to be a promising choice for this task, it is less suited to our usecase for two main reasons. Firstly, the use of SSE would necessitate key establishment and management by the email sender and receiver, and the search could only be performed by these two parties. Secondly, practical SSE methods have limitations in terms of information leakage, which can compromise the privacy of the encrypted data [69, 91, 54]. Email services such as ProtonMail [115] and Tutanota [119] have developed mechanisms to permit the receiver to search over encrypted emails, similar to SSE systems [81, 118]. Even so, these approaches are not applicable to our scenario because they do not offer search by an external agent.

To achieve the aforementioned four goals simultaneously and efficiently, we combine MPC [52, 16] and PIR [31, 38] techniques. Figure 1 depicts an overview of our approach utilizing two MPC servers, which can easily be extended to a larger number of servers.

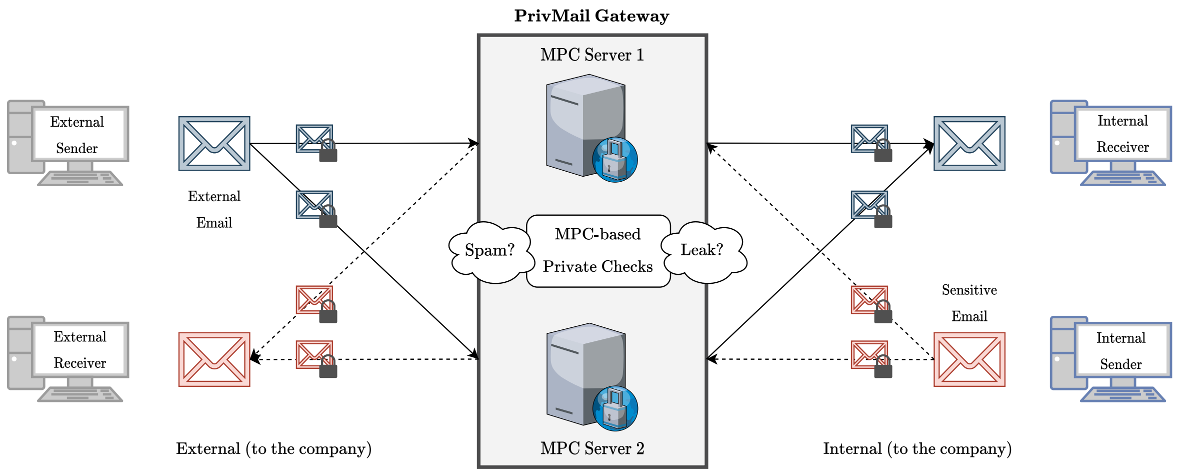


Fig. 1: Illustration of PrivMail as a secure gateway within a company's system to effectively monitor external spam emails and prevent the unintentional transmission of sensitive data from internal systems to external sources. In the event that spam or leakage is detected, the gateway stops email transmission to the destination.

Overview of Our Solution At a high level, the idea behind PrivMail is to use multiple email Service Providers (SPs) and secret sharing techniques to ensure that no individual SP sees the email content in the clear. Our idea was inspired by the observation that most people already have multiple email accounts for varied purposes such as personal and professional correspondence, with the average user having 1.75 email accounts [53, 121]. Figure 1 illustrates how PrivMail can be used to establish an email infrastructure that meets the goals of PrivCorp. For simplicity, we assume that two non-colluding MPC servers serve as email service providers. Such a distributed system has proven practical in several real-world deployments, including the widely-used private aggregate statistics service by Mozilla Firefox Telemetry [18], Brave [35] and ISRG.[5]

In our approach, rather than encrypting emails with cryptographic keys, we secret share them (both subject and content) between multiple service providers using MPC techniques. For example, if PrivCorp

[5]https://divviup.org

uses two SPs, say `gmail.com` and `outlook.com`, then each employee with id `empid` is assigned two email addresses, e.g., `empid@gmail.com` and `empid@outlook.com`. The sender splits the email content into two secret shares, each of which is sent as a regular email to one of the two email addresses, and the employee can reconstruct the shares locally to retrieve the content. We provide additional optimizations to ensure that the total communication is nearly equivalent to that of sending a single unencrypted email. With this approach, the users are only required to exchange email addresses except some simple splitting and reconstruction operations (cf. §4). In contrast to exchanging cryptographic keys or certificates needed with PGP and S/MIME [53, 121], such email addresses can be easily shared by the users (e.g., by writing on business cards or websites).

Regarding PrivCorp's goals, data privacy from SP is achieved because each SP sees only a secret share of the email, leaking no information about the actual content. Moreover, an attacker must compromise both servers in order to gain any meaningful information, making it resistant to data breaches. Service providers can perform privacy-preserving MPC computations on secret shared emails to accomplish goals (3) and (4), such as utilizing MPC for privacy-preserving implementation of machine learning-based spam filtering systems [93, 56, 88]. However, we cannot assure privacy if a government agency compels access to all servers. In this situation, the most effective approach is to choose servers located in various jurisdictions, ensuring that at least one server remains uncompromised.

Our Contributions In this paper, we propose PrivMail, a privacy-preserving system for emails. PrivMail provides a secure way to transfer and store emails without requiring to use keys or certificates. Our main idea is to secret share emails between multiple (non-colluding) mail servers (e.g., Gmail, Outlook, etc.), thereby keeping the data on each server private. Our approach has the advantage that privately sending and receiving emails can directly run on the existing email infrastructure *without server-side modifications*. PrivMail offers resilience against data breaches since the attacker must breach all of the email service providers involved in order to obtain any useful information. Furthermore, it reduces the usability issues that have been observed for schemes such as PGP and S/MIME [53, 121].

A key feature of PrivMail is its support for privacy-preserving server-side processing on secret shared emails. We propose privacy-preserving drop-in replacements for the standard Internet Message Access Protocol (IMAP) SEARCH and FETCH commands, allowing a search agent to securely and efficiently search for keyword(s) over secret shared emails and retrieve the results. Our scheme combines techniques from Secure Multi-Party Computation (MPC) and Private Information Retrieval (PIR) while avoiding leakages and the key management required by schemes like Searchable Symmetric Encryption (SSE) [69, 91, 54].

We provide a Proof-of-Concept (PoC) proxy service, which implements our system seamlessly in any arbitrary email client. On top of this, we provide a PoC Thunderbird[6] plugin, which makes a seamless user experience possible, since our framework is activated via a single button and could be even made as the default setting. We want to emphasize that our PoC software are simple and possible to code without using any advanced cryptography (only PRFs), which decreases the likelihood of serious security flaws being introduced during the development. We also simulate Simple Mail Transfer Protocol (SMTP) servers and run extensive benchmarks on private keyword search over the Enron Email Dataset [74]. We are able to demonstrate practical performance, which can encourage real-world email service providers to incorporate PrivMail's private search functionalities into their existing feature set.

The contributions in this paper are summarised as follows:

1. We propose PrivMail, a privacy-preserving framework for emails that enhances usability and data breach resilience without needing keys or certificates.
2. PrivMail offers privacy-preserving server-side processing on secret shared emails, facilitating private searches and retrieval while avoiding key management issues and leakages. We also propose multiple efficient keyword search techniques, utilizing specific properties of email text and language.
3. We published an open-source prototype implementation of PrivMail[7] and demonstrated its practicality via benchmarks on private keyword searches on the Enron Email Dataset [74].

[6]https://www.thunderbird.net
[7]https://encrypto.de/code/PrivMail

2 Related Work & Background Information

In this section, we discuss several works that are closely related to our work. We focus on the following topics: End-to-End Encryption (§2.1),Searchable Symmetric Encryption (§2.2), Secure Multi-Party Computation (§2.3), Secure Pattern Matching (§2.4), Private Set Intersection (§2.5), and Private Information Retrieval (§2.6).

2.1 End-to-End Encryption (E2EE)

End-to-End Encryption (E2EE) allows users to send emails privately by encrypting the mail at the sender's end s.t. only the receiver with the corresponding decryption key can decrypt the mail. This technique prevents any third party from reading an intercepted mail including the email service provider. There are several solutions for sending E2EE mails. Using End-to-End (E2E) encryption schemes such as S/MIME [108] and PGP [6] is a simple way to achieve private email transfer. Moreover, there has been an explosion in E2EE solutions, where more and more email services, for instance ProtonMail [115] and recently Gmail [43], are providing support for E2EE email transfers.

However, since Whitten and Tygar's seminal "Why Johnny Can't Encrypt" paper [123], there has been a long-standing debate about the usability of the schemes like S/MIME and PGP, owing primarily to the requirement for a more complex initial configuration [123, 107]. While works like [7, 9, 106] were curious about the role of key exchange, trust, and transparency in PGP usability, [102] used online surveys and user testing to find out. According to these surveys, a large number of users do not use these solutions due to the requirement of tricky configurations. Concerns have also been raised about the impact of E2E encryption on the ability to monitor citizen communications for national security threats, prompting several countries to prohibit applications that use end-to-end encryption [39]. These concerns stem from the fact that server-side searches on E2E encrypted mails can not be performed easily. Furthermore, in cases where one of the users (the sender or the receiver) is not using the same E2EE solution or none at all, then the mails can no longer be encrypted. The E2EE solution proposed by Gmail [43] does not encrypt the mail in case the receiver does not have the same system setup. Even though ProtonMail [115] provides support for encrypting incoming mails from non-encrypting senders, it does not have solution for sending to a receiver without E2EE setup. Tutanota [119], on the other hand, provides a solution for this issue where the receiver is taken to the Tutanota website to enter a password which then opens the mail in clear. Although this feature is now enabled with some services, it is still not widespread and the existing solutions are cumbersome for the receiver.

2.2 Searchable Symmetric Encryption (SSE)

Another method for implementing server-side mail processing is to use SSE, which allows searches on encrypted data. The first SSE scheme was presented for the problem of secure searches on mail servers [113]. Since then, the field of SSE has been extensively researched, with notable works including [70, 25, 49]. Practical SSE frequently leaks information about the search query, which can be exploited to reveal information about the encrypted data. Two main sources of leakage in these schemes are access patterns and search patterns. Even though later works hide the access pattern, it was shown that it is not sufficient to prevent the attacks [91]. Several attacks on such SSE systems have been devised on these pattern leakages [66, 24, 127, 69, 17, 91]. Although recent schemes, such as TWINSSE [8], are shown to be resistant to some of the attacks [24, 130], they still leak certain search patterns, which may in the future be susceptible to new attacks. A recent work by Gui et al. [54] gives even more insight on how difficult it is to build secure and efficient SSE schemes in practice underlining the fundamental problems related to SSE. Besides, schemes based on Oblivious RAM (ORAM) [46] that completely hide the search pattern typically incur very high communication cost making them unusable for practical purposes. Furthermore, the use of SSE would still necessitate key setup and management by the parties. As a result, we avoid employing SSE algorithms for keyword searches in our work.

2.3 Secure Multi-Party Computation (MPC)

MPC [52, 16] allows N parties to jointly compute a public function $f(x_1, \ldots, x_N)$ on their private inputs $x_1, \ldots, x_N$ while maintaining input privacy. MPC protocols are classified as either high-throughput [52, 34, 72] or low-latency [126, 12]. The low-latency protocols are built with Garbled Circuits (GCs) and produce constant-round solutions [76, 129, 14, 104]. For high-throughput protocols, Secret Sharing (SS)-based solutions have been used, but they require a number of communication rounds linear in the multiplicative depth of the circuit [4, 3, 85, 29, 22, 30, 94]. To achieve practical efficiency, several MPC protocols resort to the *preprocessing* model, where the input-independent computations (function-independent in general) are performed beforehand to facilitate a fast input-dependent online phase [11, 21, 15, 77, 78].

2.4 Secure Pattern Matching (SPM)

SPM [117, 57, 128, 122] entails a server with text $x \in \Sigma^n$ (over some alphabet Σ) and a receiver with pattern $p \in \Sigma^m$ with $m \leq n$. Without revealing additional information, the receiver learns where the pattern occurs as a substring of the server's text. SPM is a highly researched field with applications in various areas such as database search [47, 28], network security [87], and DNA analysis [90]. Circuit-based approaches for pattern matching techniques have been proposed in [67, 71]. These circuits are designed primarily for genomic computing and DNA matching, thus they aren't directly applicable in our context of keyword search (cf. §5.2). Later works such as [10, 58, 128] presented techniques based on homomorphic encryption. However, in these works, one of the parties involved owns the keyword while another (or a group of parties) owns the database, so they are not directly applicable to our case. As shown in §5.2, we create custom circuits for our use cases in PrivMail.

2.5 Private Set Intersection (PSI)

PSI [82, 60, 59, 64, 98, 86] allows users to compute the intersection of their private sets without revealing any information about the other elements in the set. PSI is used for various applications like contact discovery [68], Google's join and compute [64] for gaining aggregated insight about other party's data, and many more. PSI can also be used to compute keyword searches in email databases. This can be achieved by computing the intersection between the set of words in the mail and the set of search keywords. If the intersection is empty, there were no matches. Otherwise at least one match was found. However, this technique does not possess the flexibility that our search techniques, especially the circuit-based technique (§5.2), offer. For example, partial matches will no longer be possible when using PSI, and composing multiple queries together would incur higher cost in PSI. Therefore, we refrain from using PSI for our search instantiations.

2.6 Private Information Retrieval (PIR)

PIR [31, 37, 83, 38] allows a client to privately query a database $\mathfrak{D}$ without the server hosting the database $\mathfrak{D}$ learning which entry was searched. The trivial solution of sending the whole $\mathfrak{D}$ to the client requires too much communication, so PIR asks for communication complexity sub-linear in the size of the database. The first PIR scheme was introduced in [31] in an information theoretic setting by considering multiple servers. Currently, PIR is available in two deployment models: multi-server and single-server. In multi-server PIR [31, 13, 51, 89, 37, 38], the database $\mathfrak{D}$ is replicated over two or more non-colluding servers, and the client sends a query to each server and then combines all replies. In [79], single server PIR was introduced, eliminating the need for replicating the database among multiple servers. Single-server PIR [23, 27, 48, 83] protocols require substantially more computation than multi-server PIR as they must rely on cryptographic hardness assumptions. However, they avoid assuming non-colluding servers. We require PIR in PrivMail to enable search agents to securely retrieve the required emails, without which an attacker can mount statistical analysis, resulting in information leakage [69]. We do, however, modify the standard 2-server PIR with replicated database to accommodate secret shared databases (cf. §5.4).

3 Preliminaries

The generic design of PrivMail allows to seamlessly use any MPC protocol for secure computation. However, in this work, we focus on the well-explored setting with two servers ($\mathsf{n} = 2$), and the scheme is explained using this setting in the majority of the sections. Our system is comprised of four entities: the sender ($\mathcal{S}$) and receiver ($\mathcal{R}$) of an email, a collection of MPC servers, and a search agent ($\mathcal{A}$).

Threat Model All entities in PrivMail are considered to be *semi-honest*. The two MPC servers are assumed to be *non-colluding*. This is justifiable given that the MPC servers in our case are well-established email service providers such as Gmail and Outlook. Because their reputation is at stake, these service providers have strong incentives to follow the protocol and not conspire with other service providers to leak their information. They could even operate in different countries with different legislations. Such setup of non-colluding servers have already been deployed in the real-world for services such as Firefox telemetry [18], privacy-preserving machine learning [63], cryptocurrency wallets [111, 44], and COVID-19 exposure notification analytics [2]. The search agent $\mathcal{A}$ will be either the email sender or receiver, or a pre-consented third party (for instance, in the use-case mentioned in §1, the company PrivCorp's email filtering service) and is expected to semi-honestly follow the protocol specifications.

Notations We use the following logical gates: XOR ($\oplus$), AND ($\wedge$), OR ($\vee$), and NOT ($\neg$). Since XOR and NOT gates can be evaluated locally in SS-based MPC, an OR gate can be realised at the cost (communication) of an AND gate as $\mathsf{a} \vee \mathsf{b} = \neg(\neg\mathsf{a} \wedge \neg\mathsf{b})$. Given a set of m bits $\mathsf{b}_1, \ldots, \mathsf{b}_m$, $\bigwedge_{i=1}^{m} \mathsf{b}_i = \mathsf{b}_1 \wedge \mathsf{b}_2 \wedge \cdots \wedge \mathsf{b}_m$ represents the cumulative AND and the other logic operators follow similarly. When performing boolean operations with a single bit b and a binary string $\mathsf{v} \in \{0,1\}^{\ell}$, we assume that the same bit b is used to perform the operation with each bit in the string v.

The values in PrivMail are secret shared between the two MPC servers via *boolean sharing* [52]: For a secret x, the i-th server, for $i \in \{1, 2\}$, holds the randomly chosen share $\langle \mathsf{x} \rangle_i$ such that $\mathsf{x} = \langle \mathsf{x} \rangle_1 \oplus \langle \mathsf{x} \rangle_2$. We sometimes abbreviate the notation and use x_i instead of $\langle \mathsf{x} \rangle_i$ for the sake of brevity.

Existing Email Architecture In a standard email communication as depicted in Figure 2, the sender $\mathcal{S}$ sends a message to the receiver $\mathcal{R}$ via a Mail User Agent (MUA) such as Thunderbird as follows. First, the MUA converts the message to email format and sends it to a Mail Transfer Agent (MTA). Upon receiving the email, the MTA transmits it to the receiver's Mail Delivery Agent (MDA) via the Simple Mail Transfer Protocol (SMTP). Note that a single email can pass through multiple MTAs before arriving at its final destination. Finally, the MDA delivers the mail to $\mathcal{R}$'s mailbox. We leave out low-level details such as domain validation and refer to [73] for specifics.

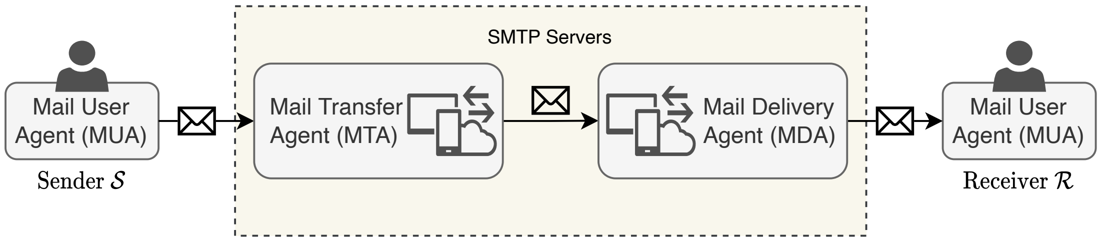
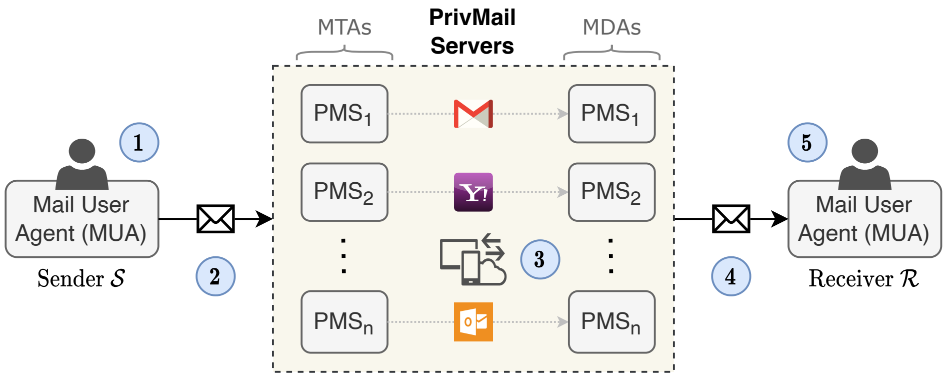


Fig. 2: Overview of existing email architecture.

On the receiver's side, $\mathcal{R}$ uses the Internet Message Access Protocol (IMAP) [33][8] to retrieve the email from the receiving server. In more detail, the two functionalities IMAP *SEARCH* [33, §6.4.4] and *FETCH* [33, §6.4.5] are used to accomplish this. SEARCH provides a comprehensive search interface similar to the Structured Query Language (SQL), where the search criteria can be single or combinations of multiple keywords, and the response contains a list of message sequence numbers corresponding to those messages that match the search criteria. FETCH retrieves all the emails from the mail server corresponding to the list returned by the SEARCH functionality. Therefore, opening a mailbox can be viewed as an invocation of SEARCH with criteria set to *ALL*, followed by a FETCH using the list returned by SEARCH.

4 PrivMail Architecture

In this section, we specify the architecture of our PrivMail framework. To keep things simple, we start with the assumption that all the senders and receivers are using PrivMail for email communication. In §4.1, we will demonstrate how to integrate our framework into the existing email infrastructure. Figure 3 depicts the email communication in PrivMail, which operates on $\mathtt{n}$ different email providers. In concrete terms, each mail provider owns an email path that connects the sender's Mail Transfer Agent (MTA) to the receiver's Mail Delivery Agent (MDA). This means that every PrivMail user will be registered with each of the $\mathtt{n}$ email providers.



① Sender submits the email. ② Email (Subject and Body) is split into $\mathtt{n}$ shares. ③ Each share is communicated via a different email provider. ④ The email shares are combined at the receiver's end. ⑤ Receiver obtains the intended email.

Fig. 3: PrivMail: Communication Phase.

At a high level, the sender $\mathcal{S}$ splits the original email (`Subject, Body`) into $\mathtt{n}$ shares and sends them to the receiver $\mathcal{R}$ via the $\mathtt{n}$ SPs, denoted by PrivMail Servers (PMS). The SPs are not required to perform any additional work to support our email transfer because the email shares are simply treated as *regular emails*, thus standard SMTP servers are sufficient for this. The email is retrieved by the receiver $\mathcal{R}$ by locally reconstructing the shares received from the SPs. Unlike an end-to-end encrypted email system (like PGP or S/MIME), this approach does not require either the sender or the receiver to handle any cryptographic keys or certificates. This also eliminates the challenges associated with implementing PGP or S/MIME in practice, where the users have to setup, manage and exchange keys and certificates [108, 107, 105]. PrivMail guarantees confidentiality, integrity and correctness. Confidentiality and correctness of PrivMail are ensured by the security of the underlying MPC protocols, while its integrity is assured as the MPC servers are assumed to be semi-honest. These security properties are analysed in detail in §4.3.

[8]The older Post Office Protocol (POP) downloads the email from the server and optionally deletes it from the server, but in contrast to IMAP provides no server-side search.

Two Server Case: Now, we explain the communication phase using the most simple example, where both the sender $\mathcal{S}$ and the receiver $\mathcal{R}$ have mail accounts with two different *non-colluding* mail providers. As illustrated in Figure 4, the first step is to split the email $\mathsf{E} = (\texttt{Subject},\texttt{Body})$ into secret shares as per the underlying MPC protocol. The secret sharing can be done either at $\mathcal{S}$'s Mail User Agent (MUA) using a custom plug-in for mail clients like Thunderbird or Outlook, or using a Sender Client Proxy (SCP) service between $\mathcal{S}$'s MUA and MTAs. In this work, we use the latter method as it is independent of the specific MUA. The email is shared as boolean shares, i.e., $\mathsf{E} = \mathsf{E}_1 \oplus \mathsf{E}_2$ with $\mathsf{E}_1 \in_R \{0,1\}^{|\mathsf{E}|}$.[9] Each of the email shares are now treated as *independent emails* and are sent using the sender's respective mail accounts using the regular email procedure. Similar to the SCP, we use a Recipient Client Script (RCS) at the receiver's end to reconstruct the original email from the shares. We provide more details on our implementations of SCP and RCS in §A.

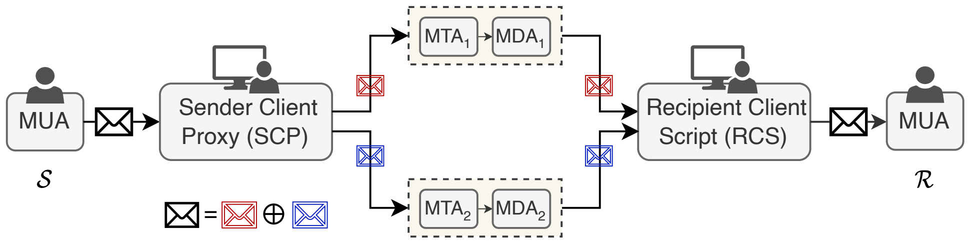


Fig. 4: PrivMail Communication: 2 Server Case.

4.1 Integration with the Existing Email Infrastructure

So far, we have assumed that each user sends and receives emails through a fixed set of distinct SPs. This assumption, however, may take some time to be adopted in practice, so we will now discuss how our scheme can be integrated with the existing email infrastructure.

The basic goal is to provide a private alternative on top of the existing email system such that the users can choose to communicate their emails either via PrivMail or via the existing email system. Let the sender $\mathcal{S}$ be registered to $\mathsf{N}_\mathcal{S}$ SPs and let $\mathsf{PMS}^\mathcal{S}$ be the set of these outgoing mail servers. Similarly, $\mathsf{PMS}^\mathcal{R}$ denotes the set of incoming email servers of receiver $\mathcal{R}$ of size $\mathsf{N}_\mathcal{R}$. Furthermore, we assume that both $\mathcal{S}$ and $\mathcal{R}$ have chosen their respective PMS servers in such a way that not all of the servers in their respective set will collude. We will now discuss approaches to integrate PrivMail to the existing email infrastructure, each with its own set of trade-offs in terms of communication and privacy.

The naïve approach In this method (Figure 5), the sender $\mathcal{S}$ splits the email into $\mathsf{N}_\mathcal{S} \cdot \mathsf{N}_\mathcal{R}$ shares using an n-out-of-n secret sharing scheme. Each server $\mathsf{PMS}_i^\mathcal{S}$ on the sender's end will receive $\mathsf{N}_\mathcal{R}$ shares, one for each of the $\mathsf{N}_\mathcal{R}$ servers on the receiver's end. When $\mathsf{PMS}_i^\mathcal{S}$ receives the email shares, it sends them to the corresponding servers at the receiver's end, treating them as *regular mails*. This setting corresponds to having a path between each pair of $\mathsf{PMS}_i^\mathcal{S}$ and $\mathsf{PMS}_j^\mathcal{R}$.

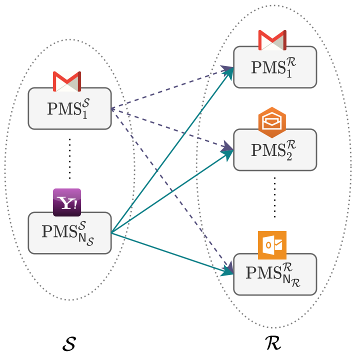


Fig. 5: Naïve Approach

We emphasize that standard SMTP servers are able to perform all the needed steps here without any modifications. In order to save storage, we can let $\mathsf{PMS}_i^\mathcal{S}$ and $\mathsf{PMS}_j^\mathcal{R}$ perform an XOR of all the $\mathsf{N}_\mathcal{R}$ shares and save only the result as its share for the mail. However, this is naturally not a standard operation for a SMTP server, and therefore we give it here just as an optimization idea. We also highlight that the communication and storage costs reduce dramatically anyway when utilizing our sharing optimization using PRF keys (see §4.2), which is compatible with standard SMTP servers.

[9] Later in §4.2 we describe an optimization to send a seed for a PRF instead of the whole share E_1.

Before looking deeper into the security of this approach, we define the term *secure path* in the context of email servers at the sender's and recipient's ends.

Definition 1. *A path connecting outgoing mail server* $\mathsf{PMS}_i^{\mathcal{S}} \in \mathsf{PMS}^{\mathcal{S}}$ *and incoming mail server* $\mathsf{PMS}_j^{\mathcal{R}} \in \mathsf{PMS}^{\mathcal{R}}$ *is said to be "secure" if neither* $\mathsf{PMS}_i^{\mathcal{S}}$ *nor* $\mathsf{PMS}_j^{\mathcal{R}}$ *colludes with the other servers in the set* $\mathsf{PMS}^{\mathcal{S}} \cup \mathsf{PMS}^{\mathcal{R}}$*. Here,* $\mathsf{N}_{\mathcal{S}}, \mathsf{N}_{\mathcal{R}} \geq 2$*, where* $\mathsf{N}_{\mathcal{S}} = |\mathsf{PMS}^{\mathcal{S}}|$ *and* $\mathsf{N}_{\mathcal{R}} = |\mathsf{PMS}^{\mathcal{R}}|$.

Privacy: Let $\mathsf{N}_{\mathsf{min}} = \min(\mathsf{N}_{\mathcal{S}}, \mathsf{N}_{\mathcal{R}})$. Considering the entire set of servers, i.e., $\mathsf{PMS}^{\mathcal{S}} \cup \mathsf{PMS}^{\mathcal{R}}$, the approach is secure as long as the adversary compromises no more than $\mathsf{N}_{\mathsf{min}}$-1 servers. This is due to the fact that in this case, there will be at least one secure path (cf. Definition 1) from $\mathsf{PMS}^{\mathcal{S}}$ to $\mathsf{PMS}^{\mathcal{R}}$, and one share out of the $\mathsf{N}_{\mathcal{S}} \cdot \mathsf{N}_{\mathcal{R}}$ shares will be communicated via this path. Because the adversary has no knowledge of this share, the email content's privacy is guaranteed due to the privacy guarantee of the underlying n-out-of-n secret sharing scheme.

We observe that it is preferable to consider security at $\mathcal{S}$'s and $\mathcal{R}$'s ends separately rather than combining them, since in real-world scenarios, the servers in $\mathsf{PMS}^{\mathcal{S}}$ and $\mathsf{PMS}^{\mathcal{R}}$ may be from different legal jurisdictions. In such cases, the number of servers at the clients side does not ensure additional security. As a result of treating the servers separately, the scheme's security is preserved at the $\mathcal{S}$'s (or $\mathcal{R}$'s) end as long as not all servers in $\mathsf{PMS}^{\mathcal{S}}$ (or $\mathsf{PMS}^{\mathcal{R}}$) are compromised. In such cases, even if $\mathsf{N}_{\mathcal{S}} < \mathsf{N}_{\mathcal{R}}$, an adversary may find it difficult to corrupt the servers in $\mathsf{PMS}^{\mathcal{S}}$ when compared to $\mathsf{N}_{\mathcal{R}}$. As a result of treating the servers separately, the scheme's security is preserved at the sender's end as long as not all servers in $\mathsf{PMS}^{\mathcal{S}}$ are compromised. Similarly, security at the receiver's end is maintained as long as there exists at least one non-colluding server in $\mathsf{PMS}^{\mathcal{R}}$.

Note that the naïve approach requires a communication of $\mathsf{N}_{\mathcal{S}} \cdot \mathsf{N}_{\mathcal{R}}$ email shares. We now present an optimized approach that reduces the communication between $\mathsf{PMS}^{\mathcal{S}}$ and $\mathsf{PMS}^{\mathcal{R}}$ to $\max(\mathsf{N}_{\mathcal{S}}, \mathsf{N}_{\mathcal{R}})$ email shares.

Optimized Approach In this method (Figure 6), the sender splits the email into $\mathsf{N}_{\mathsf{max}} = \max(\mathsf{N}_{\mathcal{S}}, \mathsf{N}_{\mathcal{R}})$ shares as per the underlying MPC protocol. Without loss of generality, consider the case where $\mathsf{N}_{\mathcal{S}} < \mathsf{N}_{\mathcal{R}}$ and the other case follows similarly. Once the email shares are generated, SCP at $\mathcal{S}$'s end will compute a mapping from the servers in $\mathsf{PMS}^{\mathcal{S}}$ to $\mathsf{PMS}^{\mathcal{R}}$. If there are common servers, i.e., $\mathsf{PMS}^{\mathcal{S}} \cap \mathsf{PMS}^{\mathcal{R}} \neq \emptyset$, then a mapping is formed between the corresponding servers for each server in the intersection (e.g., in Figure 6, Gmail server $\mathsf{PMS}_1^{\mathcal{S}}$ is mapped to Gmail server $\mathsf{PMS}_1^{\mathcal{R}}$). The remaining servers are then assigned a random mapping so that each of the servers in $\mathsf{PMS}^{\mathcal{R}}$ receives exactly one email share. When $\mathsf{N}_{\mathcal{S}} \geq \mathsf{N}_{\mathcal{R}}$, the mapping is such that each server in $\mathsf{PMS}^{\mathcal{S}}$ is assigned exactly one email share, whereas servers at the receiver's end may receive multiple shares.

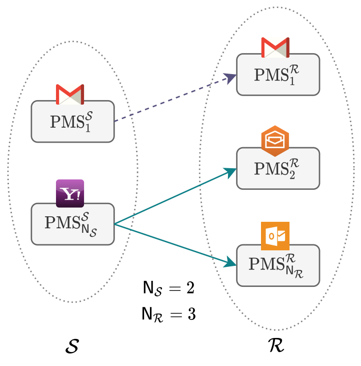

$\mathsf{PMS}_1^{\mathcal{S}}$
$\mathsf{PMS}_{\mathsf{N}_{\mathcal{S}}}^{\mathcal{S}}$
$\mathsf{PMS}_1^{\mathcal{R}}$
$\mathsf{PMS}_2^{\mathcal{R}}$
$\mathsf{PMS}_{\mathsf{N}_{\mathcal{R}}}^{\mathcal{R}}$
$\mathsf{N}_{\mathcal{S}} = 2$
$\mathsf{N}_{\mathcal{R}} = 3$
$\mathcal{S}$
$\mathcal{R}$

Fig. 6: Optimized Approach

Privacy: Similar to the naïve approach, the privacy of the optimized approach is ensured as long as there is at least one secure path between the servers at the $\mathcal{S}$'s and $\mathcal{R}$'s ends. As a side note, if we are only concerned with overall privacy that allows collusion of at most $\mathsf{N}_{\mathsf{min}}$-1 servers (cf. privacy of Approach I), then this scheme can be modified to split the email content into only $\mathsf{N}_{\mathsf{min}}$ shares and consider the remaining shares to be zero. This will reduce the communication from $\mathsf{N}_{\mathsf{max}}$ to $\mathsf{N}_{\mathsf{min}}$ emails.

4.2 Sharing Optimization using PRF Keys

In the two approaches mentioned in §4.1, each share is of the same size as the original mail. Therefore, the total communication required for sending a mail of size $|\mathsf{E}|$ would be $\max(\mathsf{N}_{\mathcal{S}}, \mathsf{N}_{\mathcal{R}}) \cdot |\mathsf{E}|$ for the optimized approach.

To further optimize the communication and storage, we can adopt techniques similar to those used in multi-server PIR schemes [31, 37, 38], that also use n-out-of-n secret sharing to split the PIR query. In these works, the query is partitioned into several *chunks*. In line with this technique, we divide each mail E into n chunks of size $|\mathsf{E}|/\mathsf{n}$ each. Each chunk is then shared among the servers PMS_i using boolean sharing such that $\mathsf{chunk}_j = \mathsf{flip}_j \oplus_{i=n,i\neq j}^{n} \mathsf{rnd}_i^j$, where flip is a boolean share and $\mathsf{rnd}_i^j = \mathrm{PRG}(\mathsf{key}_i)[j]$, where key_i is a 128-bit symmetric key. Server i then receives the tuple $(\mathsf{flip}_i, \mathsf{key}_i)$ as its share of the mail which has size $|\mathsf{E}|/\mathsf{n} + 128$ *bits*. The mail can thus be reconstructed by concatenating all the chunks together, i.e., $\mathsf{chunk}_1 \| \mathsf{chunk}_2 \| \cdots \| \mathsf{chunk}_\mathsf{n}$. For the correctness of this sharing, we set the number of chunks $\mathsf{n} = \max(\mathsf{N}_\mathcal{S}, \mathsf{N}_\mathcal{R})$. This reduces the total communication of the optimized approach to $|\mathsf{E}| + \max(\mathsf{N}_\mathcal{S}, \mathsf{N}_\mathcal{R}) \cdot 128$ *bits*, which is about the size of the original email $|\mathsf{E}|$.

4.3 Security Guarantees of PrivMail

In this section, we go over the security guarantees offered by PrivMail. We consider confidentiality, integrity, and authenticity (non-repudiation) in our analysis.

Confidentiality: PrivMail achieves confidentiality at the sender's end as long as there is one server in $\mathsf{PMS}^\mathcal{S}$ that is not colluding with the other servers in the set. The guarantee for the receiver follows similarly. When looking at the entire set of email servers ($\mathsf{PMS}^\mathcal{S} \cup \mathsf{PMS}^\mathcal{R}$), the email content is secure as long as there is a *secure path* (cf. Definition 1) between $\mathsf{PMS}^\mathcal{S}$ and $\mathsf{PMS}^\mathcal{R}$. We note that in cases where email providers use external MX servers to relay emails, there may be instances where one MX server handles emails for multiple email providers. In such cases, the security of our scheme is maintained as long as the group of compromised MX servers does not control all of the n shares.

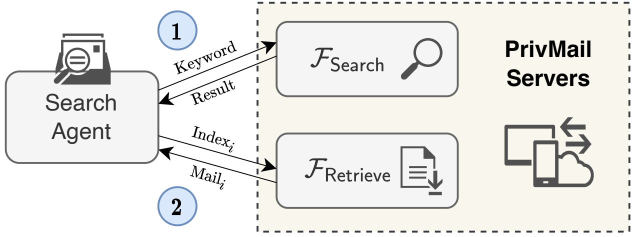
Integrity: The integrity of the email content is ensured because we assumed the MPC servers to be semi-honest in PrivMail. Here we assume that the email in transit is not modifiable by an adversary. However, if considering such an adversary then adding TLS security for the transmission of mails will ensure the integrity of the mails. In the case of malicious servers (which we leave as future work), the underlying MPC protocol is responsible for providing integrity, which is typically achieved using authentication tags for protocols in the dishonest-majority setting. In the case of emails, however, an honest sender can simply append a salted (to make two sharings of the same email appear different) SHA256 hash of the email content together with the salt to each shared email. When the email is reconstructed, the receiver can verify that it matches both hashes/salts. This is secure as long as one of the servers is not corrupted and the sender is honest.

Authenticity: PrivMail ensures the correctness of email transfer once the email shares are generated by the sender (due to the correctness of the MPC protocol). However, it does not address the issue of authenticity or non-repudiation. As our scheme is fully compliant with S/MIME and PGP, one of these can be added on top to ensure authenticity and non-repudiation when the receiver knows the sender's certified public key.

5 Private Queries using MPC

The main advantage of PrivMail is that private server-side processing of the mails is easily possible because the emails are secret shared among the servers. This allows commonly used functions like keyword search to be implemented using various MPC techniques. Furthermore, keyword searches can be utilised for other functionalities, such as checking for data leakage in outgoing emails or detecting spam in incoming emails, as in our PrivCorp's use-case in Figure 1. Given the ubiquitous usage of keyword search in the existing email infrastructure, we present multiple private keyword search techniques and discuss optimizations and extensions. Note that the search agent $\mathcal{A}$ can be either of the email users, i.e., $\mathcal{S}$ or $\mathcal{R}$, or a third party with prior consent, e.g., a company mail service.

The discussion that follows assumes that the communication phase (cf. §4) has been completed and that the email content has been secret shared among the PrivMail servers (PMSs). Furthermore, we consider keyword search over the email shares among the servers at the recipient's end ($\mathsf{PMS}^\mathcal{R}$). The same method can be used to search through the email shares of $\mathsf{PMS}^\mathcal{S}$. Consider a mailbox containing p emails that are secret shared among n non-colluding servers. Given a keyword K, the private query phase (Figure 7) proceeds using two sub-protocols: i) Private Search ($\mathcal{F}_{\mathsf{Search}}$) emulating IMAP SEARCH and ii) Private Fetch ($\mathcal{F}_{\mathsf{Fetch}}$) emulating IMAP FETCH, which are executed in the order detailed next.


1
Search Agent
Keyword
Result
Index$_i$
Mail$_i$
2
$\mathcal{F}_{\mathsf{Search}}$
$\mathcal{F}_{\mathsf{Retrieve}}$
PrivMail Servers

① Search agent submits keyword and obtains a list of email ids containing the keyword. ② Search agent retrieves the intended email by submitting the respective id.

Fig. 7: PrivMail: Private Query Phase.

1. The search agent $\mathcal{A}$ secret shares the keyword K among the n servers in accordance with the underlying MPC protocol.
2. Upon receiving the shares of K, the servers initiate the $\mathcal{F}_{\mathsf{Search}}$ functionality that enables $\mathcal{A}$ to obtain a list of indices that corresponds to emails containing the keyword. Concretely, $\mathcal{A}$ obtains a p-bit vector $\vec{\mathsf{H}} \in \{0,1\}^{\mathsf{p}}$ with $\mathsf{H}[i] = 1$ if the i^{th} email contains K and 0 otherwise. We instantiate $\mathcal{F}_{\mathsf{Search}}$ using different techniques in §5.2.
3. $\mathcal{A}$ and the n servers jointly execute $\mathcal{F}_{\mathsf{Fetch}}$ to privately fetch each email from the mailbox. We efficiently instantiate $\mathcal{F}_{\mathsf{Fetch}}$ using an extension of 2-server PIR to a secret shared database (see §5.4).

To summarize, $\mathcal{F}_{\mathsf{Search}}$ and $\mathcal{F}_{\mathsf{Fetch}}$ provide a privacy-preserving drop-in replacement for the standard Internet Message Access Protocol (IMAP) [33] functionalities SEARCH and FETCH respectively. We note that the recent revision of IMAP in [84] has only minor modifications on the most interesting parts on the keyword search for PrivMail, and for simplicity, we keep referring to [33] throughout this paper. We begin with the details of the $\mathcal{F}_{\mathsf{Search}}$ functionality.

5.1 Private Search $\mathcal{F}_{\mathsf{Search}}$

Recall from §4 that our basic approach secret shares the email's subject and body fields in their original form. One disadvantage of this strategy is that the private search becomes case-sensitive. Working over the actual shares with MPC incurs additional computation and communication to provide a solution similar to the standard IMAP SEARCH, which is case-insensitive. A simple way to avoid this overhead is to let the sender $\mathcal{S}$ secret share a lowercase version of the email as well, which doubles the size of the email. The $\mathcal{F}_{\mathsf{Search}}$ function will then be performed over these shares, and the actual shares will be used to reconstruct the original email (in $\mathcal{F}_{\mathsf{Fetch}}$). The search's efficiency can be further improved by employing a special encoding described next.

Special Encoding We define our own special 6-bit character encoding without losing much information for standard emails in English text that use 7-bit ASCII character encoding, similar to the SixBit ASCII code by AIS [101]. The encoding space is sufficient for all the lowercase alphabets and numbers (0–9) along with 28 special characters.[10] For this, we omit all ASCII control characters as well as a small set of special characters that are uncommon for a keyword. We call this 6-bit encoded email a *compact email* from now on. As shown later in §5.2, the bit length reduction due to this encoding also helps to improve the performance of the $\mathcal{F}_{\mathsf{Search}}$ protocols.

Moreover, by adding extra *signal bits* to each character in the compact email, we can effectively reconstruct the original email. For instance, these signal bits can indicate whether a letter is uppercase or not, and

[10] We define a special *padding character* for later usage (cf. §5.2). Hence, the scheme can encode at most 63 characters for the keyword search.

for special characters, they can represent the omitted ones in the compact version. Consequently, keyword searches would exclusively target the compact emails while disregarding the signal bits.

Computing Servers (CSs) Now that $\mathcal{F}_{\mathsf{Search}}$ operates over shares of the compact email and these operations are computationally intensive (as will be evident from the subsequent sections), the PMS servers can use powerful dedicated servers for this task, especially for scalability. Henceforth, we refer to the servers for $\mathcal{F}_{\mathsf{Search}}$ as Computing Servers (CSs). Note that the PMS servers can be used as the CSs as well.

Figure 8 depicts the case in which each of the PMS servers hires one CS. These additional servers essentially act as proxy for the PMS servers. This approach will not hamper the basic architecture discussed in §4 because communication with both $\mathcal{S}$ and $\mathcal{R}$ will continue to go through the PMS servers.

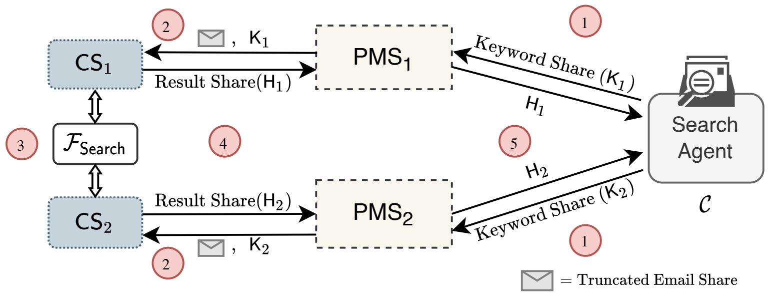


Fig. 8: Private Query—Individual Computation Servers.

As shown in Figure 9, the PMS servers could also agree and hire a set of N servers to perform the computation. The main difference between this approach and the previous one is that these outsourced N servers will act as another instance of MPC rather than a proxy. One advantage of this approach is that the servers can be instantiated using a more efficient MPC protocol in settings like honest majority over three [95, 77] and four [78] servers.

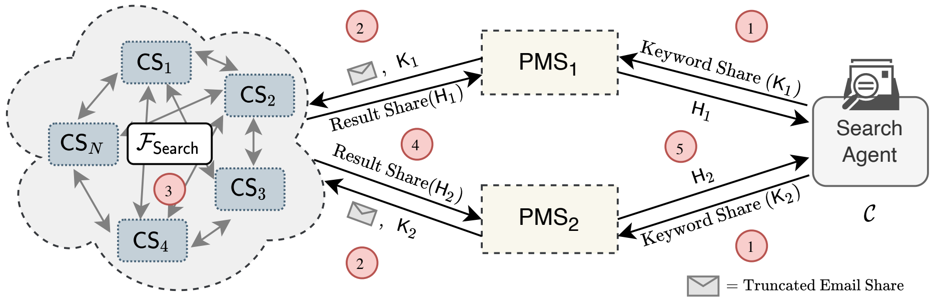


Fig. 9: Private Query—Outsourced Computation Servers.

The values secret shared among the PMS servers must be *re-shared* among these outsourced servers in order for computation to be possible. However, in our case, this is simple because each of the PMS servers can distribute its share (corresponds to the Secure Two-Party Computation (STPC) protocol in our case) among the outsourced servers using the appropriate MPC protocol, and these shares can be locally added to obtain re-sharing of the same secret.

$\mathcal{F}_{\mathsf{Search}}$ *for the 2-Server Case* As before, we resort to a simple setting of two servers and use a Secure Two-Party Computation (STPC) protocol for our computations, where the CSs enact the role of MPC servers for the private query phase. The ideal functionality $\mathcal{F}_{\mathsf{Search}}$, depicted in Figure 10, takes the secret shares of the keyword being searched and the emails as input and returns the shares of the p-bit binary vector $\vec{\mathsf{H}}$. Looking ahead, in the protocol, the shares of the keyword are generated by the search agent $\mathcal{A}$ and sent to the corresponding servers (the PRF optimization from §4.2 can be used here as well for long keywords). Furthermore, both shares of the result of $\mathcal{F}_{\mathsf{Search}}$ are given to $\mathcal{A}$, who reconstructs the result locally.

Functionality $\mathcal{F}_{\mathsf{Search}}$

Consider a mailbox of p emails $\mathsf{E}_1, \ldots, \mathsf{E}_\mathsf{p}$ and a s-character keyword K.

Input: CS$_i$ provides $(\langle \mathsf{K} \rangle_i, (\langle \mathsf{E}_1 \rangle_i, \ldots, \langle \mathsf{E}_\mathsf{p} \rangle_i))$ for $i \in \{1, 2\}$.
Computation: $\mathcal{F}_{\mathsf{Search}}$ proceeds as follows:

1. Reconstruct $\mathsf{K} = \langle \mathsf{K} \rangle_1 \oplus \langle \mathsf{K} \rangle_2$ and $\mathsf{E}_j = \langle \mathsf{E}_j \rangle_1 \oplus \langle \mathsf{E}_j \rangle_2$, for $j \in \{1, \ldots, \mathsf{p}\}$.
2. Compute vector $\vec{\mathsf{H}} \in \{0,1\}^\mathsf{p}$: $\mathsf{H}[j] = 1$, if K is a substring of E_j, and 0 otherwise.
3. Sample random $\langle \vec{\mathsf{H}} \rangle_1 \in_R \{0,1\}^\mathsf{p}$ and set $\langle \vec{\mathsf{H}} \rangle_2 = \vec{\mathsf{H}} \oplus \langle \vec{\mathsf{H}} \rangle_1$.

Output: $\mathcal{F}_{\mathsf{Search}}$ sends $\langle \vec{\mathsf{H}} \rangle_i$ to server CS$_i$.

Fig. 10: Ideal Functionality for Keyword Search

Remarks When the search agent $\mathcal{A}$ is the email user ($\mathcal{S}$ or $\mathcal{R}$), it may be preferable to download the entire mailbox and perform keyword search locally over the cleartext. Though this naïve solution appears to be much cheaper for the CSs, it may not always be an ideal solution from the perspective of the email user due to factors such as limited (mobile) bandwidth, local storage, battery usage, or cross-device accessibility.

5.2 Instantiating $\mathcal{F}_{\mathsf{Search}}$

Here, we look at concrete instantiations of the $\mathcal{F}_{\mathsf{Search}}$ functionality. First, we discuss a generic instantiation, called Circuit-Based search, that can be applied with keywords of any length and frequency. Later, we present two optimizations to the Circuit-Based search: i) Bucketing-Based, and ii) Indexing-Based searches, where the Bucketing-Based search is more efficient for single word keywords, while the Indexing-Based search is more efficient for keywords with higher frequency of appearance in the text. Each of these approaches are discussed primarily in the context of a two-server setting, but can be generalized to multiple servers.

Circuit-Based Search Consider a text $W = w_1 \| \cdots \| w_t$ of length t and a keyword $\mathsf{K} = k_1 \| \cdots \| k_s$ of length s, with $s \le t$. Let b denote the bit-length of characters (e.g., $b = 7$ for ASCII characters and $b = 6$ for compact emails, cf. §5.1). At a high level, the strategy is as follows: beginning with the i-th character position of W, a block of s characters denoted by $\widetilde{W}_i$ is derived and compared to the keyword K. We use an equality test functionality $\mathcal{F}_{\mathsf{EQ}}$ over ℓ-bit inputs to compare K and $\widetilde{W}_i$, defined as $\mathcal{F}^{\ell}_{\mathsf{EQ}}(x, y) = 1$ if $x = y$, and 0 otherwise. We use the EQ circuit of [76, 75, 109] to instantiate $\mathcal{F}_{\mathsf{EQ}}$ which is defined as follows:

$$\mathsf{EQ}^{\ell}(x, y) = \bigwedge_{i=1}^{\ell} \neg(x_i \oplus y_i). \tag{1}$$

The EQ protocol essentially XORs the strings x and y and checks for a zero string as the result. This is equivalent to checking whether the result's flipped bits are all ones and can be computed using a cumulative AND operation.

There is a total of $t - s + 1$ blocks ($\widetilde{W}_i$) with one EQ protocol executed per block. All of these executions are independent and can be carried-out in parallel. Once the results have been evaluated, a cumulative OR of the results can be used to find at least one matching block. To summarize, the search circuit SC is defined as

$$\mathsf{SC}(\underbrace{\mathsf{K}, W}_{\text{(Keyword, Text)}}) = \bigvee_{i=1}^{\overbrace{t-s+1}^{\#\ \text{blocks}}} \mathsf{EQ}^{\underset{\text{bit length}}{sb}}(\mathsf{K}, \widetilde{W}_i). \tag{2}$$

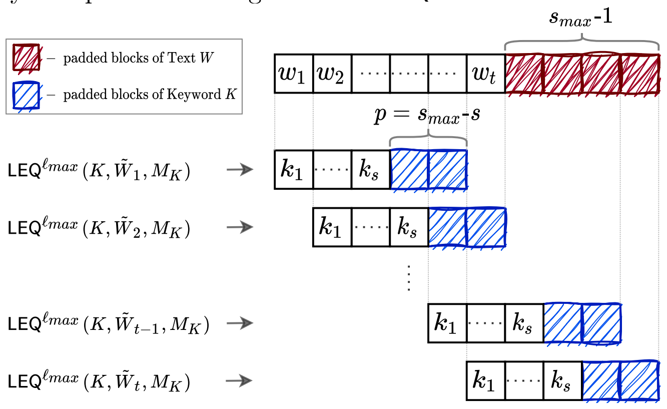
Complexity: An instance of EQ over ℓ bit inputs require a total of $\ell - 1$ AND gates (cf. Eq. (1)) and has a multiplicative depth of $\lceil \log_2 \ell \rceil$ when evaluated as a tree. The SC consists of $t - s + 1$ such EQ circuits and additionally $t - s$ AND gates (as OR can be implemented via AND). Moreover, another $\lceil \log_2(t - s + 1) \rceil$ rounds are required to compute the final result. Hence, for $\ell = sb$ in our case, the SC circuit has a total of $(t+1)sb - s^2 b - 1$ ($\approx tsb$, when $t \gg s, b$) AND gates and a depth of $\lceil \log_2 sb \rceil + \lceil \log_2(t - s + 1) \rceil$. Note that the depth of the circuit can be further reduced at the expense of increased communication using multi-input AND gates [94, 78, 40, 20].

The method described above assumes that the lengths of the text and the keyword are known to all Computing Servers (CSs). In our case, because the subject and body of an email are the text, hiding its length during computation is impractical for efficiency reasons. However, we can hide the length of the s-length keyword by padding it to a fixed length, which results in the following modifications to the above approach.

Length-Hiding Keyword Search Given two $\ell_{\max}$-bit values x, y, let the functionality $\mathcal{F}_{\mathsf{LEQ}}$ be defined as $\mathcal{F}_{\mathsf{LEQ}}^{\ell_{\max},\ell}(x, y) = 1$ if $x[i] = y[i]$ for $i \leq \ell$ and 0 otherwise. For a given $\ell \leq \ell_{\max}$, $\mathcal{F}_{\mathsf{LEQ}}$ returns 1 if the first ℓ-bits of x match with y and 0 otherwise. To hide the length $\ell = sb$ bits of keyword K, we use an additional $\ell_{\max}$-bit *length mask*[11] of the form $\mathrm{M}_x = m_1 \| \cdots \| m_{\ell_{\max}} = \{1\}^{\ell} \| \{0\}^{\ell_{\max} - \ell}$. Given these values, we instantiate $\mathcal{F}_{\mathsf{LEQ}}$ by using a length hiding equality test circuit LEQ, which is defined as

$$\mathsf{LEQ}^{\ell_{\max}}(x, y, \mathrm{M}_x) = \bigwedge_{i=1}^{\ell} \neg((x_i \oplus y_i) \wedge m_i). \tag{3}$$

The logic of the LEQ circuit is similar to that of EQ (Eq. (1)) in that the bits of x, y are compared, but the result for the padded bits (last $\ell_{\max} - \ell$ bits) is discarded by ANDing them with the zero bits of the M_x value. This only adds a layer of parallel AND gates to the EQ circuit.



– padded blocks of Text W
– padded blocks of Keyword K
s_{max}-1
w_1 w_2 w_t
$p = s_{max}$-s
$\mathsf{LEQ}^{\ell_{max}}(K, \tilde{W}_1, M_K)$
$\mathsf{LEQ}^{\ell_{max}}(K, \tilde{W}_2, M_K)$
$\mathsf{LEQ}^{\ell_{max}}(K, \tilde{W}_{t-1}, M_K)$
$\mathsf{LEQ}^{\ell_{max}}(K, \tilde{W}_t, M_K)$
k_1 k_s

Fig. 11: Length-Hiding Keyword Search.

To hide the keyword length s, the search circuit SC (cf. Eq. (2)) must now consider blocks $(\widetilde{W}_i)$ starting at every character position of the text W until the end of the last character. As a result, the number of LEQ circuits in SC is t. However, this raises an issue with the blocks of W that begin after $t - s_{\max} + 1$ characters, where $s_{\max}$ denotes the length of the padded keyword. In these cases, the length of the block will be less than the length of the keyword, making a direct comparison impossible. A simple solution would be to pad the text W with $s_{\max} - 1$ characters before the protocol's execution as shown in Figure 11. These characters can be random and do not affect the computation's correctness. However, by searching over fewer characters and using the mask's information, the number of AND gates required for these corner cases can be reduced even further. Furthermore, the masking as mentioned above can be modified to allow *wildcard* keyword search as well. The details of these approaches are provided in §B.1.

[11]The agent $\mathcal{A}$ generates the shares of the mask and sends them with the keyword shares.

Optimizations to the Circuit-Based Search: We further optimize the Circuit-Based search using different properties of the mail text and keywords. If the keyword contains no spaces, i.e., is a single word, it is beneficial to "jump" over *spaces* in the target text to avoid unnecessary comparisons. However, checking for spaces in the Circuit-Based Search is difficult as the entire email is treated as a single string. Including the logic to check for spaces within the search circuit SC (cf. Eq. (2)) is costly. Therefore, for our optimizations, each mail is considered as a collection of distinct words rather than a continuous string of characters. This also gives us advantages when the mail contains repeated words, since now we can omit the duplicates from the search.

Bucketing-Based Search To search on distinct words, we let the sender $\mathcal{S}$ secret share each word in the email text individually rather than the entire text. This, however, would expose structural information in the original text, which an adversary could exploit. To address this, we recognize that the order of the words is irrelevant to the search and could be randomized. Furthermore, repeated words can simply be omitted, and the lengths of the remaining words can be masked by padding them to some fixed length.

Padding for length obfuscation should be done with caution, as it is a trade-off between efficiency and privacy. Padding every word to a large fixed length would yield maximum privacy. However, the efficiency may now be even worse than that of the Circuit-Based Search. On the other hand, the padding can be removed completely to maximize efficiency. As a compromise, we propose a *bucketing* technique that allows users to tailor the trade-off between efficiency and privacy to their specific needs.

A naïve way of instantiating a search given a list of secret shared words and a keyword would be to use a circuit-based PSI protocol [59, 97, 86] (see §2.5). However, linear complexity PSI does exact but not sub-string matches, and thus it is not possible for the user to add padding to the words for increased privacy. Therefore, we adopt a strategy similar to our Circuit-Based Approach (see §5.2) to search through the list of words in the email.

The idea behind bucketing is to select buckets for different ranges of character length and do the padding accordingly. For example, if we define a bucket for words with lengths ranging from 1 to 4 characters, each word of that range in the bucket is padded to 4 characters. Furthermore, the search keyword must be padded in accordance with the bucketing scheme. To facilitate substring matches, the search must be performed over the bucket defined for the keyword length as well as the buckets for longer words. As a result, the bucketing technique is more efficient for longer keywords as the shorter buckets can be ignored. In order to hide the actual length of the keyword, we must use the LEQ circuit, see Eq. (3) for the equality tests. The bucketing idea described above can be extended to multi-word search queries and is discussed in §B.2.

Indexing-Based Search All search approaches described before were primarily concerned with searching over each email individually. Multiple emails in a mailbox, on the other hand, are likely to have many words in common. With this observation, we further optimize the Bucketing-Based Search for single-word search and build a *search index* for *all* emails in a mailbox. We give a description of the structure of the search index and its operations and a detailed discussion is provided in §B.3.

Consider a mailbox containing p emails $\{\mathsf{E}_1, \ldots, \mathsf{E}_\mathsf{p}\}$ and let d be the number of distinct words in those emails. As shown in Table 1, each distinct word $\mathcal{W}_i$ forms one row of the table and is associated with a p-bit string. The vector is referred to as *occurrence bit-string* and has the following format: $\vec{\mathsf{B}}^{\mathcal{W}_i} = \mathsf{B}_1^{\mathcal{W}_i} \| \cdots \| \mathsf{B}_\mathsf{p}^{\mathcal{W}_i}$. Here, $\mathsf{B}_j^{\mathcal{W}_i} = 1$ denotes the presence of word $\mathcal{W}_i$ in the email E_j and 0 denotes its absence.[12] The index table is then secret shared among the Computing Servers (CSs).

Remember that in the previous approaches, a bit value is returned for each email immediately following the execution of the search circuit SC. On the other hand, in the indexing-based approach, a keyword is first searched against all the distinct words in the table and a d-bit string $\vec{\mathsf{u}} = \mathsf{u}_1 \| \cdots \| \mathsf{u}_d$ is generated, where d is the number of distinct words. The final search result is nothing but a cumulative OR of all the occurrence

[12]The keyword can be a part of multiple distinct words in the index table as a substring.

Table 1: Search index for emails $\{\mathsf{E}_1, \ldots, \mathsf{E}_\mathsf{p}\}$ with d distinct words. $\mathsf{B}_j^{\mathcal{W}_i} = 1$, if $\mathcal{W}_i \in \mathsf{E}_j$, and 0 otherwise.

Distinct Words	Occurrence Bit-String
$\mathcal{W}_1$	$\vec{\mathsf{B}}^{\mathcal{W}_1} = \mathsf{B}_1^{\mathcal{W}_1} \Vert \mathsf{B}_2^{\mathcal{W}_1} \Vert \cdots \Vert \mathsf{B}_\mathsf{p}^{\mathcal{W}_1}$
$\mathcal{W}_2$	$\vec{\mathsf{B}}^{\mathcal{W}_2} = \mathsf{B}_1^{\mathcal{W}_2} \Vert \mathsf{B}_2^{\mathcal{W}_2} \Vert \cdots \Vert \mathsf{B}_\mathsf{p}^{\mathcal{W}_2}$
$\vdots$	$\vdots$
$\mathcal{W}_d$	$\vec{\mathsf{B}}^{\mathcal{W}_d} = \mathsf{B}_1^{\mathcal{W}_d} \Vert \mathsf{B}_2^{\mathcal{W}_d} \Vert \cdots \Vert \mathsf{B}_\mathsf{p}^{\mathcal{W}_d}$

bit-strings corresponding to the matched rows. Formally, the p-bit result of the search is

$$\text{Search Result} = \bigvee_{i=1}^{d} \mathsf{u}_i \cdot \vec{\mathsf{B}}^{\mathcal{W}_i}, \tag{4}$$

where the OR operations over a bit vector are simply the operator applied to each bit position.

Given the secret shares of $\vec{\mathsf{u}}$ among the CSs, one method for completing the above task is to involve the search agent $\mathcal{A}$. The vector $\vec{\mathsf{u}}$ is reconstructed by $\mathcal{A}$ and the corresponding occurrence bit strings are obtained securely using $\mathcal{F}_{\mathsf{Fetch}}$ (as will be explained later in §5.4). The agent can then perform the OR operation locally to obtain the final result. Another method that avoids agent intervention is to let the CSs compute the expression in Eq. (4) using the underlying MPC protocol.

To hide the keyword length, we use the bucketing technique (cf. §5.2) on top of the indexing approach. This basically means that each bucket has its own index table. This has advantages for longer keywords because we can safely skip index tables for smaller buckets. Furthermore, this approach makes the search complexity *independent* of the number of mails, except the final small computation of Eq. (4), making it more efficient for large mail boxes.

The process for creating the index table is discussed in §B.3. We propose two approaches: i) client updates and ii) server updates. The former focuses on using the receiver $\mathcal{R}$'s assistance while the latter focuses on minimizing the receiver $\mathcal{R}$'s involvement.

Comparison: Each of the search techniques discussed above, i.e., Circuit-Based, Bucketing-Based and Indexing-Based, have their own pros and cons depending on the search keyword length, number of mails in the mailbox and the occurrence of the distinct keywords. The user or the email client can therefore decide the most beneficial technique according to their requirements. In Table 2, we give a comparison between the different techniques for various use-cases, highlighting the most efficient techniques for each use-case.

Table 2: Efficiency comparison of the different search techniques for different types of keywords. The most efficient technique is marked in bold.

	Circuit	**Bucket**	**Index**
Longer Keywords	More efficient for longer keywords than for smaller.	**Significantly more efficient as smaller buckets will be completely skipped.**	Efficient only if the keyword is frequent in the text.
Higher Frequency Keywords	Efficiency remains the same as for any keyword.	Efficient if the keyword is of longer length.	**Very efficient for frequent words in the text.**
Partial Matches	**Very efficient as the text is consisered as a continuous string of characters.**	Would incur higher cost to implement.	Would incur higher cost to implement.
Special Words	**Efficient, as all words are treated with same priority.**	**Efficient if the word is of medium to long length.**	Not very efficient as the word won't be frequent in the text.

5.3 Query Chaining / Filtering

So far, our discussions have been limited to the email's `Subject` and `Body` fields. However, the Internet Message Access Protocol (IMAP) *SEARCH* [33, §6.4.4.] provides a comprehensive interface for filtering results using multiple keys. By default, "When multiple keys are specified, the result is the intersection (AND function) of all the messages that match those keys" [33]. The process of combining results for multiple keyword searches is referred to as *Query Chaining* in PrivMail and is discussed further below.

Case 1: Filtering non-private fields. It is simple to incorporate the standard filters provided by the IMAP *SEARCH* in our framework as the SMTP servers are regular email providers.[13] Consider the following example from the IMAP [33, §6.4.4.] specification.

```
SEARCH FLAGGED SINCE 1-Feb-1994 NOT FROM ''Smith''
```

The above filter uses the email's `Flagged, Date` and `From` fields and returns the list of all the flagged emails sent by all people except Smith beginning on 1st February, 1994.

Note that using private keyword searches in conjunction with filters like this helps to reduce the number of emails in the search domain, resulting in better efficiency. We note that the effect of narrowing down the target emails is less visible in the case of index-table based search (§5.2), because the search must still traverse the entire index. For a mailbox of p emails, the servers can safely ignore the occurrence bits corresponding to emails outside the filtered space by using a public mask string of the form $\vec{\mathsf{f}} = \mathsf{f}_1 \| \cdots \| \mathsf{f}_\mathsf{p}$. If the email E_i is in the set of filtered emails, $\mathsf{f}_i = 1$, otherwise 0. Servers will then use these masked occurrence bit strings $(\vec{\mathsf{f}} \cdot \vec{\mathsf{B}}^{\mathcal{W}_i})$ to compute the Search Result in Eq. (4).

Case 2: Hiding the query for non-private fields. The preceding case assumed that queries for non-private fields were revealed in clear to the SMTP servers. If the privacy of these queries is a concern, we can extend the $\mathcal{F}_{\mathsf{Search}}$ functionality (cf. §5.1) to these non-private fields as well. The search agent $\mathcal{A}$ proceeds by secret sharing the query among the computing servers. Servers, on the other hand, are aware of the other input and can treat it as a public value within the MPC computation. This approach will necessitate fewer AND computations[14] than the case with fields like `Subject, Body` which are only secret shared among the servers.

The approach is slightly different for the `Date` field, because the search queries do not always look for a substring match, but instead perform comparison operations such as before, after, or exactly on. To accomplish this, the equality circuit EQ (cf. Eq. (1)) is replaced with a comparison circuit (e.g., based on [109] or [94]). Instead of directly comparing the `Date` field, it can be converted to an integer for greater efficiency. For e.g., an integer representation using Unix timestamp[15] preserves the date and time with excellent accuracy.

Case 3: Chaining multiple queries. Assume that the search agent $\mathcal{A}$ wants to find the emails in a mailbox of p emails that contain both the keywords `security` and `conference` in the email body. Two independent $\mathcal{F}_{\mathsf{Search}}$ instances are used for this, with the keywords $\mathsf{q}_1 =$ `security` and $\mathsf{q}_2 =$ `conference`. Remember from Eq. (2) that this will result in the generation of two p-bit vectors, $\vec{\mathsf{H}}^{\mathsf{q}_1}$ and $\vec{\mathsf{H}}^{\mathsf{q}_2}$. The final result is easily obtained by performing a bitwise AND operation over these vectors denoted by $\vec{\mathsf{H}}^{\mathsf{q}_1} \wedge \vec{\mathsf{H}}^{\mathsf{q}_2}$, i.e., $\mathsf{H}^{\mathsf{q}_1}[i] \wedge \mathsf{H}^{\mathsf{q}_2}[i]$ for all $i \in \{1, \ldots, \mathsf{p}\}$. Similarly, for emails containing either of the two keywords, the same procedure could be used, with the exception that the final result is computed as $\vec{\mathsf{H}}^{\mathsf{q}_1} \vee \vec{\mathsf{H}}^{\mathsf{q}_2}$. Finally, when the query is for emails containing `security` but not `conference`, the result is simply $\vec{\mathsf{H}}^{\mathsf{q}_1} \wedge \neg(\vec{\mathsf{H}}^{\mathsf{q}_2})$. Note that this approach extends easily when searching across multiple fields as well.

The approaches mentioned above can be further modified to reduce information leakage in relation to the query. Alternatively, we can use standard Private Function Evaluation (PFE) techniques [120, 80, 1] to hide how the sub-queries are combined which we leave for future investigation.

[13] Concretely, SMTP servers can first filter out the results locally and then pass this information to the CS servers, if they are distinct (see Figure 8).

[14] Due to the linearity of the underlying MPC scheme

[15] https://www.unixtimestamp.com

Case 4: Hiding the query structure. Here, we concentrate on completely hiding the structure of the query chaining. The chaining is accomplished solely through the use of AND and OR functions, with NOT being used to flip the result for some of the queries. To hide the structure, we make the chaining appear to be made up of only AND operations and the other two operators (OR and NOT) will be obfuscated.

Consider a set of γ queries denoted by $\vec{\mathsf{q}} = \mathsf{q}_1, \ldots, \mathsf{q}_\gamma$ and the respective responses be $\vec{\mathsf{H}}^{\mathsf{q}} = \vec{\mathsf{H}}^{\mathsf{q}_1}, \ldots, \vec{\mathsf{H}}^{\mathsf{q}_\gamma}$. The NOT operations are handled with a *flip-bit* string denoted by $\vec{\mathsf{flip}} \in \{0,1\}^\gamma$ with $\mathsf{flip}[i] = 1$ indicating that the NOT operation should be performed on result of the i^{th} query q_i, denoted as $\vec{\mathsf{H}}^{\mathsf{q}_i}$. Agent $\mathcal{A}$ generates a secret sharing (boolean) of this vector among the CSs and they compute the result locally as $\vec{\mathsf{H}}^{\mathsf{q}} \oplus \vec{\mathsf{flip}}$.

For AND and OR operations, we associate a control bit vector $\vec{\alpha} \in \{0,1\}^{\gamma-1}$ generated by the agent $\mathcal{A}$ (one bit per chaining operator). In this case, $\alpha_i = 1$ indicates that the corresponding operator should be an OR and 0 represents AND. For two queries $\mathsf{q}_1, \mathsf{q}_2$ with one control bit α, the servers will now compute

$$\text{chain}(\mathsf{q}_1, \mathsf{q}_2) = \alpha \oplus ((\alpha \oplus \vec{\mathsf{H}}^{\mathsf{q}_1}) \wedge (\alpha \oplus \vec{\mathsf{H}}^{\mathsf{q}_2})) \ .$$

Note that the above expression evaluates to $\vec{\mathsf{H}}^{\mathsf{q}_1} \wedge \vec{\mathsf{H}}^{\mathsf{q}_2}$ when $\alpha = 0$ and $\vec{\mathsf{H}}^{\mathsf{q}_1} \vee \vec{\mathsf{H}}^{\mathsf{q}_2}$ when $\alpha = 1$. Similarly, for three queries, $\text{chain}(\mathsf{q}_1, \mathsf{q}_2, \mathsf{q}_3)$ is given as

$$\alpha_1 \oplus ((\alpha_1 \oplus \vec{\mathsf{H}}^{\mathsf{q}_1}) \wedge (\alpha_1 \oplus (\alpha_2 \oplus ((\alpha_2 \oplus \vec{\mathsf{H}}^{\mathsf{q}_2}) \wedge (\alpha_2 \oplus \vec{\mathsf{H}}^{\mathsf{q}_3})))))$$

and Table 3 summarises the result for various choices of the control bit.

Table 3: Computing chained query results using the results of individual queries (using AND and XOR).

α_1	α_2	$\text{chain}(\mathsf{q}_1, \mathsf{q}_2, \mathsf{q}_3)$
0	0	$\vec{\mathsf{H}}^{\mathsf{q}_1} \wedge \vec{\mathsf{H}}^{\mathsf{q}_2} \wedge \vec{\mathsf{H}}^{\mathsf{q}_3}$
0	1	$\vec{\mathsf{H}}^{\mathsf{q}_1} \wedge \vec{\mathsf{H}}^{\mathsf{q}_2} \vee \vec{\mathsf{H}}^{\mathsf{q}_3}$
1	0	$\vec{\mathsf{H}}^{\mathsf{q}_1} \vee \vec{\mathsf{H}}^{\mathsf{q}_2} \wedge \vec{\mathsf{H}}^{\mathsf{q}_3}$
1	1	$\vec{\mathsf{H}}^{\mathsf{q}_1} \vee \vec{\mathsf{H}}^{\mathsf{q}_2} \vee \vec{\mathsf{H}}^{\mathsf{q}_3}$

Similar to $\vec{\mathsf{flip}}$, $\mathcal{A}$ prepares and secret shares the control bit vector $\vec{\alpha}$ among the CSs. Combining the use of $\vec{\mathsf{flip}}$ and $\vec{\alpha}$ completely hides the query chaining structure as desired. Note that this method currently does not support the use of brackets symbol present in the IMAP SEARCH and is left for future exploration.

Case 5: Hiding full query. Let $\{W_1, \ldots, W_f\}$ denote a collection of f target fields on which an agent $\mathcal{A}$ can perform a query. This case aims at hiding the field on which a query is made from the CSs. The basic idea is to use $\mathcal{F}_{\mathsf{Search}}$ to search all possible fields and then use a selection bit vector to filter out the results for the desired field. We define a *select bit-string* of the form $\vec{\beta} = \beta_1 \| \cdots \| \beta_f$, where $\beta_i = 1$ if W_i is the desired target field, and 0 otherwise. For a query q, let $\vec{\mathsf{H}}_i^{\mathsf{q}}$ denote the result obtained from $\mathcal{F}_{\mathsf{Search}}$ over the field W_i for $i \in \{1, \ldots, f\}$. Now, the output can be calculated as

$$\bigoplus_{i=1}^{f} \vec{\mathsf{H}}_i^{\mathsf{q}} \wedge \beta_i \ .$$

This method adds an extra round to the total. Performing a search over the `Body` field of the email every time is expensive, so that field can be treated separately and avoided from the approach described above. Also, the keyword's length can reveal information about the actual target field. As a result, categorizing the fields and performing the search only on the relevant category using the approach described above will provide a better trade-off between efficiency and privacy. In our case for emails, some possible candidates for categories include: i) flags, ii) date, and ii) address. We anticipate that these approaches will be more useful when our techniques are extended to other domains, such as document processing.

5.4 Private Fetch $\mathcal{F}_{\mathsf{Fetch}}$

The $\mathcal{F}_{\mathsf{Fetch}}$ functionality (Figure 12), enables the search agent $\mathcal{A}$ to privately retrieve emails from the Computing Servers (CSs). $\mathcal{F}_{\mathsf{Fetch}}$ takes the p-bit string as input and returns the mails corresponding to the bit positions with value 1. In theory, $\mathcal{A}$ can use the standard IMAP FETCH [33, §6.4.5] command to retrieve the desired email shares from each server. As the email was originally secret shared, this method ensures the confidentiality of the retrieved email while only revealing the email number to the servers. However, in the long run, statistical analysis of which emails are frequently retrieved can reveal details of the search queries to the servers, particularly the keywords used [69]. As a result, to instantiate $\mathcal{F}_{\mathsf{Fetch}}$, we adapt multi-server Private Information Retrieval (PIR) [31, 37, 38, 55], where the database is replicated among multiple non-colluding servers.

Functionality $\mathcal{F}_{\mathsf{Fetch}}$

Consider a mailbox of p emails $\mathsf{E}_1, \ldots, \mathsf{E}_\mathsf{p}$.

Input: Server CS$_i$ provides $(\langle \mathsf{E}_1 \rangle_i, \ldots, \langle \mathsf{E}_\mathsf{p} \rangle_i)$ for $i \in \{1, 2\}$; search agent $\mathcal{A}$ provides an index $\mathsf{ind} \in \{1, \ldots, \mathsf{p}\}$.

Output: $\mathcal{F}_{\mathsf{Fetch}}$ sends $(\langle \mathsf{E}_{\mathsf{ind}} \rangle_1, \langle \mathsf{E}_{\mathsf{ind}} \rangle_2)$ to search agent $\mathcal{A}$.

Fig. 12: Ideal Functionality for Fetching Email

At a high level, to enable multi-server PIR on a secret shared (email) database, we let each CS symmetrically encrypt its share and send that to the other CS. Moreover, the decryption keys are known to the secret agent $\mathcal{A}$. We extend the multi-server PIR protocol RAID-PIR [37], to work on our secret shared email database as detailed below.

Consider a database $\mathfrak{D}$ containing p emails with the server CS$_i$ holding the share $\mathfrak{D}_i = \{\langle D_1 \rangle_i, \ldots, \langle D_\mathsf{p} \rangle_i\}$, where each $\langle D \rangle_i = (\mathsf{flip}_i \| \mathsf{key}_i)$ for $i \in \{1, 2\}$ (see §4.2). First, the client $\mathcal{A}$ samples two symmetric keys sk_1, sk_2 for a symmetric-key encryption scheme $\mathsf{Enc}()$ and sends key sk_i to server CS$_i$. The next step, on a high level, is that each CS$_i$ will encrypt its share (corresponding to each email) with sk_i and send this to the other server. Let $\widetilde{\mathfrak{D}}_i$ denote the encrypted database share of CS$_i$. The encrypted database is now defined as $\widetilde{\mathfrak{D}} = \widetilde{\mathfrak{D}}_1 \| \widetilde{\mathfrak{D}}_2$. Then the client $\mathcal{A}$ sends the shares of the selection bit vector $\vec{\mathsf{b}}$ to each server and the servers compute $\vec{\mathsf{b}}_i \cdot \widetilde{\mathfrak{D}}$ and send the result to $\mathcal{A}$. The client $\mathcal{A}$ then combines these results by XORing them and decrypts the result using the encryption keys to obtain the queried email.

The communication between the client and the servers can be further optimised by precomputing the results of one of the servers [55]. For this, the client $\mathcal{A}$ sends a seed s to a PRG to CS$_2$. The server CS$_2$ *pre*-computes its result with the random bit vector generated using this seed. Later, in the *online* phase, the client computes the bit vector share $\vec{\mathsf{b}}_1$ such that $\vec{\mathsf{b}} = \vec{\mathsf{b}}_1 \oplus \mathrm{PRG}(\mathsf{s})$. Then $\mathcal{A}$ sends $\vec{\mathsf{b}}_1$ to the server CS$_1$ and the server computes its result as discussed above. The combining of the results from the servers can be done in one of two ways, 1) server CS$_2$ sends its results to CS$_1$, then CS$_1$ sends the combined result to the client, or 2) CS$_1$ and CS$_2$ send their results directly to the client $\mathcal{A}$ and $\mathcal{A}$ combines the two results. As the final step, the client decrypts the combined result using its encryption keys and retrieves the queried email.

Remarks: It is specified in $\mathcal{F}_{\mathsf{Fetch}}$ that the agent $\mathcal{A}$ provides an index ind in the range of 1 to p as the functionality's input. In the protocol, however, $\mathcal{A}$ creates a bit vector $\vec{\mathsf{b}} \in \{0, 1\}^\mathsf{p}$ with $\mathsf{b}[\mathsf{ind}] = 1$ and all other bits set to zero. $\mathcal{A}$ then secretly distributes this bit vector to the servers CS$_1$ and CS$_2$.

For external searches, search agent $\mathcal{A}$ is assumed to be semi-honest and is only supposed to fetch emails for which the corresponding keyword search is a hit rather than a miss. A malicious agent, on the other hand, can send a malformed bit string $\vec{\mathsf{b}}$ as input to $\mathcal{F}_{\mathsf{Fetch}}$ in order to retrieve an unintended email from the mailbox (or some function on the unintended emails, depending on the $\mathcal{F}_{\mathsf{Fetch}}$ instantiation). To prevent this type of attack, servers can use the keyword search result as follows: remember that the output of $\mathcal{F}_{\mathsf{Search}}$ (Figure 10) is a p-bit vector $\vec{\mathsf{H}} \in \{0, 1\}^\mathsf{p}$, with $\mathsf{H}[i] = 1$ indicating that the i^{th} email contains K and 0 otherwise. The servers can use $\vec{\mathsf{b}} \wedge \vec{\mathsf{H}}$ instead of $\vec{\mathsf{b}}$ as the input to $\mathcal{F}_{\mathsf{Fetch}}$. This will set all unintended positions of $\vec{\mathsf{b}}$ to zero.

Now, to instantiate the symmetric key encryption scheme $\mathsf{Enc}()$, we can encrypt each email individually using a standard symmetric encryption cipher such as Advanced Encryption Standard (AES) in Counter Mode (CTR). This, however, requires a new *nonce* (a.k.a. Initialization Vector (IV)) for the encryption of each $\langle D_i \rangle_j$, $i \in \{1, \ldots, \mathsf{p}\}$. Since email numbers are unique, the email number i concatenated with the counter part can be used as the counter block.[16] Note that the key schedule in AES only needs to be run once, and the encryption (and decryption) can be done in parallel. Furthermore, the approach described above eliminates the need to communicate the nonce used in the encryption to the client $\mathcal{A}$. When new emails arrive, the servers can simply encrypt and distribute the shares corresponding to the new emails, as described above. When the client $\mathcal{A}$ is an external agent other than the receiver $\mathcal{R}$, the servers can still use an already prepared encrypted database. For this, the servers simply send the corresponding symmetric keys (provided by $\mathcal{R}$) to $\mathcal{A}$, and the rest of the procedure is the same as before.

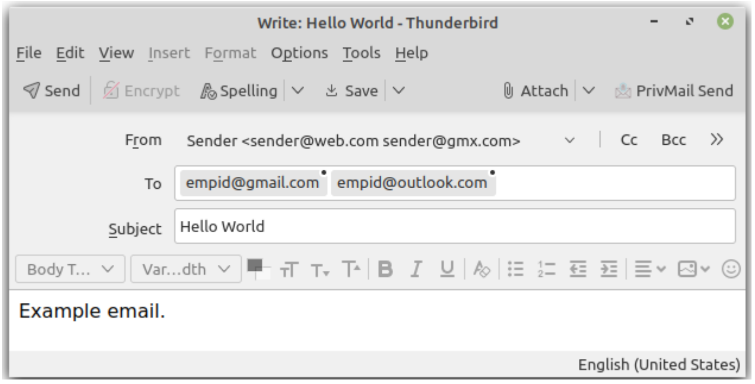
6 Implementation and Benchmarks

In this section, we describe our implementation of PrivMail and evaluate the performance of reconstruction and keyword search over secret shared emails using our approaches from §5.1.

6.1 Email Transfer

The implementation of PrivMail consists of several parts, which were discussed in §4. Our implementation of the Sender Client Proxy (SCP) runs in a Docker container [41] and works with any Mail User Agent (MUA) that allows the user to manually specify the outgoing SMTP server (i.e., basically any email client). Additionally, we implemented a plugin for Thunderbird to showcase the ease of use of PrivMail (see §6.1 for more details). We also implemented a simple Recipient Client Script (RCS) for email reconstruction at the receiver's end and use it for performance evaluation. We simulate the SMTP servers in our local network, where the roles of a Mail Transfer Agent (MTA) and a Mail Delivery Agent (MDA) are combined into a single entity for convenience. We use the YAML data-serialization language [125] to store the emails in the filesystem.

Thunderbird Plugin The PrivMail Thunderbird plugin [26] integrates PrivMail with the current email infrastructure as described in §4.1. Our implementation demonstrates how easy it is to use PrivMail in practice, since an email is sent securely via a single button "PrivMail Send" as seen in Figure 13.



From Sender <sender@web.com sender@gmx.com>

To empid@gmail.com empid@outlook.com

Subject Hello World

Example email.

Fig. 13: Thunderbird "Write" view with PrivMail plugin installed. The "PrivMail Send" button on the right initiates secret sharing and securely sends the email to the receiver.

[16]Since the nonce is not random in our case, the nonce and counter should be concatenated (e.g., storing the nonce in the upper 64 bits and the counter in the lower 64 bits of a 128-bit counter block) and not added or XORed.

The sender defines at least two sender accounts[17] (the "From" field) and at least two receiver addresses (the "To" field). The software attempts to find matching servers based on the domain names to find at least one secure path (see Definition 1) and proceeds with the optimized approach in §4.1. If the sender and receiver use a completely different set of email providers, the software uses the naïve approach (see §4.1).

The reconstruction of the shares to the original email is enabled with a single button as well, which is shown whenever a received email is opened. The only assumption for the reconstruction is that all the shares are in some of the accounts of the currently open Thunderbird profile.

Reconstruction Performance The reconstruction of the shares consists of three steps: fetching of the email shares from the (IMAP) servers, pairing, and finally combining the shares. For the Thunderbird plugin (§6.1), the retrieval is handled as for any regular email, i.e., automatically based on the Thunderbird settings (by default, fetches new emails periodically or when the inbox is opened). We pair the shares using the Unique Identifier (UID) (see §A) and Thunderbird `query()` API [116]. More precisely, the command structure is `messenger.messages.query( 'subject': UID );`, which is a fast lookup over the subject fields of all the emails. After all the shares are found based on the UID, the shares are combined and the original email is displayed on a new Thunderbird tab. For two inboxes containing 500 emails each, the pairing and combining a single email takes under 10 milliseconds on a regular laptop using Intel Core i7–8565U.

For our Recipient Client Script (RCS), the retrieval is handled using the `imaplib` module [62] and after each fetch, the email is stored in a dictionary, where the email's UID is the key and the share is the value. In the second step, the dictionaries are combined together. Since the dictionaries are hash maps in Python and the keys are random looking identifiers, the lookup for each email is a constant time operation [100]. The runtime to pair and combine 500 emails from two servers takes only around 0.235 seconds, giving us a throughput of 2,127 emails per second. The runtime of the fetching step depends on the IMAP server capacity, geographic location, and network setup, but is likely to be dominant (throughput was only 20 emails per second in our experiments). Thus the overhead introduced by PrivMail compared to fetching and viewing regular emails is negligible.

6.2 Email Search

We implement the private queries described in §5 using the mixed-protocol Secure Multi-Party Computation (MPC) framework MOTION [19]. We use the boolean GMW protocol [52, 36] between two parties for our performance evaluation, but our implementation can also be used in the N-party setting with full threshold. Our code is publicly available under the MIT License [26].[18]

Benchmark Settings for Search We tested all three approaches from §5.2 for private search (Circuit-Based, Bucketing-Based, and Index-Based) on a real-world dataset with varying parameters, using our special encoding from §5.1. For the Bucketing-Based approach, we chose to create four buckets, each of size 5 characters, i.e., buckets for words with $(1\text{-}5, 6\text{-}10, 11\text{-}15, 16\text{-}20)$ characters. All words that are more than 20 characters long are ignored. We instantiate the MPC protocols with computational security parameter 128 and statistical security parameter 40.

We run experiments against subsets of the publicly available Enron Email Dataset [74], which contains over 500,000 emails, with each email containing on average 1,607 characters and 237 words. On examining the distribution of distinct words in this database, we conclude that the bucket size distribution for our chosen buckets are $(18.8\%, 50.6\%, 21.7\%, 8.9\%)$. This implies that, on average, more than half of the words are between 6 and 10 characters long. Our benchmark subsets are drawn from Kenneth Lay's inbox.

The benchmarks are run on two dedicated simulation machines, each with an Intel Core i9–7960X (16 physical cores @ 2.8 GHz) processor and 8×16 GB DDR4 RAM. We simulate a Wide Area Network (WAN)

[17]For convenience and testing purposes, we enabled the use of only one sender account but display a warning message, since the outgoing server can reconstruct the original email from all the shares in this case. This setting can be disabled by changing `strictSecurity` value to `true`.

[18]https://encrypto.de/code/PrivMail

setting with bandwidth limited to 100 Mbit/s, and Round-Trip Time (RTT) of 100 ms. To ensure consistency, we iterate each simulation 5 times and compute the mean for the final result.

Search Performance We begin with the Circuit-Based Search from §5.2. This is the simplest method of searching, but reveals the keyword length. Table 4 summarizes our benchmarks for search across four different keyword lengths, $s \in \{3, 8, 13, 18\}$ (corresponding to the average of our bucket sizes), on email sets of sizes 100 and 200. We parallelize each equality test circuit (see Eq. (1)) with Single Instruction, Multiple Data (SIMD) operations, which results in an almost linear total runtime with respect to the keyword length. The minor difference in online runtime is caused by runtime fluctuations in our WAN simulation and can be evened out with additional iterations. The remaining cumulative OR in Eq. (2) dominates the online runtime, giving a nearly constant runtime.

Table 4: Evaluation of (non-length hiding) circuit-based search (§5.2). Runtime (Time) is in seconds and communication (Comm.) between the servers in mebibytes (MiB).

Keyword Length s	Phase	# Emails			
		100		200	
		Time (s)	Comm. (MiB)	Time (s)	Comm. (MiB)
3	Online	4.80	1.22	11.67	2.72
	Total	12.19	63.42	33.45	145.28
8	Online	5.20	3.12	10.15	7.03
	Total	20.96	174.25	48.65	399.61
13	Online	5.12	5.03	11.35	11.33
	Total	25.33	284.15	60.43	652.05
18	Online	4.31	6.89	9.40	15.52
	Total	32.47	393.07	73.74	902.54

While the approach is practical in the online phase, we find that the total communication overheads in Table 4 are prohibitively high and do not scale for large settings. However, as evident from the table, the overheads for the online phase are significantly lower than the total, and the setup phase is the bottleneck in our case. For example, in the largest test case online communication accounts for only 1.7% of the total communication, and online runtime accounts for 12.7% of the total runtime. We point out that this is due to the Secure Two-Party Computation (STPC) protocol that we used in our benchmarking, and not a problem with the approach. For example, evaluating each AND gate using the ABY [36] protocol necessitates the use of two instances of Oblivious Transfer (OT) during the setup phase. This translates to roughly 64× more communication in the setup than online when instantiated with the OT extension protocol of IKNP [65, 5]. This can be overcome by using recent optimizations for STPC, such as Silver [32] for OT in the setup phase and ABY2.0 [94] for the online phases.

Analysis of Different Approaches We now compare our three search methods from §5.2 in terms of practical performance. Since the Bucketing and Indexing-Based approaches already provide keyword length hiding, we use the Circuit-Based search's length hiding variant for a fair comparison. The Circuit-Based approach thus requires an additional layer of AND gates over the keyword length non-hiding variant. We only focus on the online phase, as we will subsequently address various methods to minimize overhead during the setup phase. Detailed benchmarks concerning setup costs will also be discussed later in this section.

Table 5 compares the performance of our three search methods in the online phase across various keyword lengths. Except for keyword length $s = 3$, we find that the Indexing-Based Approach outperforms the other two approaches in all cases. This is justified because the search domain for both the bucketing and Indexing-Based Approaches shrinks for long keywords due to the omission of buckets for short keywords.

Table 5: Evaluation of online phase of keyword length-hiding search methods: circuit, bucketing and indexing. Runtime (Time) is in seconds and communication (Comm.) between the servers in mebibytes (MiB). Best results are in bold. The trade-offs between the methods are discussed in §5.2.

Keyword Length (s)	Method	# Emails			
		100		200	
		Time (s)	Comm. (MiB)	Time (s)	Comm. (MiB)
3	Circuit	9.63	**2.61**	18.19	**5.80**
	Bucketing	9.15	16.28	14.07	35.76
	Indexing	**3.57**	6.41	**6.58**	11.22
8	Circuit	13.68	4.62	25.96	10.41
	Bucketing	7.51	7.09	12.98	15.91
	Indexing	**3.19**	**3.73**	**5.22**	**6.97**
13	Circuit	13.78	6.59	23.73	14.93
	Bucketing	4.59	1.27	10.09	2.91
	Indexing	**2.48**	**0.63**	**4.26**	**1.48**
18	Circuit	15.38	8.58	27.51	19.47
	Bucketing	4.19	0.16	8.97	0.45
	Indexing	**2.68**	**0.06**	**3.96**	**0.25**

For keyword length $s = 3$, we note that the use of the SIMD instruction in our implementation tends to increase the communication overhead of bucketing and indexing-based approaches over circuit-based approaches. In particular, while our implementation parallelizes the equality test circuits as previously mentioned, the number of operations we can pack in an SIMD instruction for small buckets is relatively small, undermining the effect. This is not a problem for the circuit-based search because the entire text acts as a single long word. The reason mentioned above justifies the higher communication of the bucketing-based over the circuit-based approach for the case of $s = 8$ as well.

Bucketing vs. Indexing: In Table 5, we see that the Indexing-Based Approach improves by $\approx 2\times$ over the Bucketing-Based Approach in terms of both runtime and communication. Furthermore, the Indexing-Based method is expected to be more and more efficient compared to the other methods for a larger number of emails, since it eliminates duplicate words across the entire email set. We remark that in settings where the number of target documents is large and the document format is more regular, our indexing-based search can be orders of magnitude more efficient than bucketing-based search.

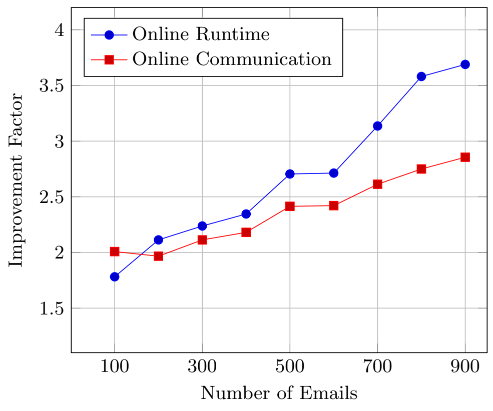
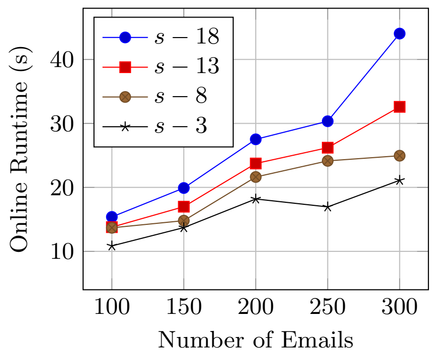


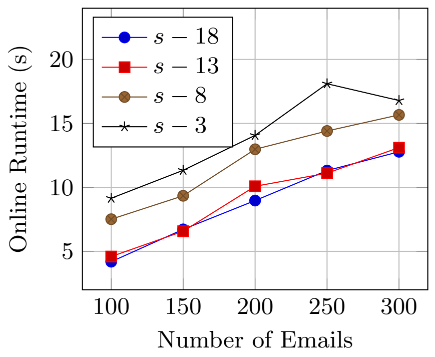
Fig. 14: Improvement (bucketing/indexing) in online runtime and communication for $s = 13$ character keyword.

However, it is important to note that the values reported for the indexing-based approach do not include the costs for building the search index (see §5.2). As previously stated, the cost of creating a search index is determined by the index table updating mode (local or server-side) as well as the frequency of the updates.[19] To get a clearer picture, we plot in Figure 14 the improvement of the runtime and communication in the online phase of the indexing-based approach over the bucketing-based approach for the case of keyword length $s = 13$ over 900 emails.

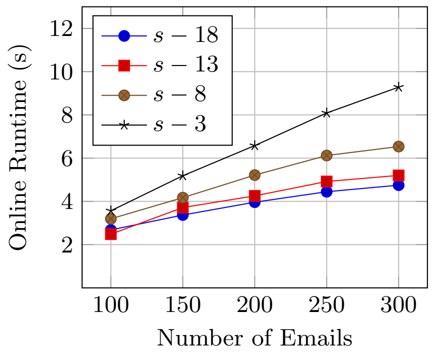
We can see that the bucketing-based search is twice as slow and uses twice as much communication for 200 emails. For 900 emails, the improvement factor rises to $3\times$. It is expected to improve more quickly as the number of emails increases because we expect to see fewer and fewer new words in the emails, ideally halting the index's growth entirely. However, we cannot see this in Figure 14, owing to the fact that a set of realistic email texts is diverse by nature, and our samples do not contain enough emails. We remark that in settings where the number of target documents is large and the document format is more regular, our indexing-based search can be orders of magnitude more efficient than bucketing-based search. For example, a company's human resources department will have documents that are mostly in predefined formats, with personal information that varies from document to document.



(a) Circuit-based (§5.2) (length-hiding)



(b) Bucketing-based (§5.2)



(c) Indexing-based (§5.2)

Fig. 15: Online runtimes for different search approaches over keyword lengths $s \in \{3, 8, 13, 18\}$.

In Figure 15, we plot the online runtimes for the three search approaches across different keyword lengths. All of the runtimes increase roughly linearly, as expected. In general, we can conclude that the runtime of the circuit-based search quickly exceeds practical limits for large keywords. When the keyword is long, bucketing and indexing-based search produces more practical and efficient results. This is due to the bucketing scheme used, which can be customized for various applications.

[19] Prior to the start of the computation, we used a Python script to create the search index from the emails. This is similar to the local updates method of building a search index (see §B.3).

Benchmarks in Detail We provide a comprehensive set of benchmarks in Table 6.

Table 6: Runtime (in seconds) and communication between the servers (Comm. in MiB) for different search approaches over keywords of various length s. The overheads are divided into setup and online phases.

Circuit-based search (non length-hiding) (§5.2)

Keyword Length s	Number of Emails											
	100				200				300			
	Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)	
	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
3	7.39	4.80	62.20	1.22	21.78	11.67	142.57	2.72	31.05	17.13	215.19	4.10
8	15.76	5.20	171.13	3.12	38.50	10.15	392.58	7.03	54.24	15.68	592.62	10.60
13	20.21	5.12	279.12	5.03	49.08	11.35	640.73	11.33	75.53	15.20	967.27	17.08
18	28.16	4.31	386.19	6.89	64.34	9.40	887.02	15.52	94.02	14.73	1,339.16	23.39

Circuit-based search (length-hiding) (§5.2)

Keyword Length s	Number of Emails											
	100				200				300			
	Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)	
	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
3	14.44	9.63	128.21	2.61	36.80	18.19	293.87	5.80	51.95	21.11	443.60	8.74
8	24.56	13.68	255.27	4.62	51.59	25.96	585.52	10.41	77.12	24.95	883.89	15.70
13	36.37	13.78	381.25	6.59	77.27	23.73	874.99	14.93	108.42	32.59	1,320.92	22.52
18	43.22	15.38	506.14	8.58	95.18	27.51	1,162.29	19.47	145.36	44.04	1,754.73	29.37

Bucketing-based search (§5.2)

Keyword Length s	Number of Emails											
	100				200				300			
	Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)	
	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
3	9.95	9.15	90.72	16.28	20.42	14.07	201.58	35.76	30.48	16.79	302.87	53.63
8	7.01	7.51	59.77	7.09	14.49	12.98	136.47	15.91	24.65	15.66	206.40	24.12
13	2.34	4.59	10.81	1.27	5.42	10.09	26.80	2.91	10.21	13.11	39.46	4.33
18	1.28	4.19	0.89	0.16	2.46	8.97	3.86	0.45	4.03	12.78	5.12	0.69

Indexing-based search (§5.2)

Keyword Length s	Number of Emails											
	100				200				300			
	Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)		Runtime (s)		Comm. (MiB)	
	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online	Setup	Online
3	4.90	3.57	41.09	6.41	8.07	6.58	74.19	11.22	10.96	9.28	99.18	14.89
8	4.42	3.19	35.15	3.73	7.29	5.22	67.75	6.97	9.88	6.53	91.89	9.44
13	1.85	2.48	7.98	0.63	2.99	4.26	19.40	1.48	3.80	5.20	26.68	2.05
18	1.11	2.68	0.85	0.06	1.61	3.96	3.77	0.25	1.87	4.75	4.84	0.32

Implementation Optimizations The search approaches in PrivMail are implemented using the STPC protocols of ABY [36] previously mentioned during the analysis of circuit-based approach and the setup phase uses the IKNP [65, 5] Oblivious Transfer (OT) extension protocol. However, several optimizations for both of these protocols were later proposed, from which we selected Silver [32] and ABY2.0 [94] as the best

Table 7: Empirical evaluation of communication for search methods using optimizations from Silver[32] and ABY2.0[94]. Searches are performed using a keyword of length $s = 8$ over 300 emails. * corresponds to protocols implemented in PrivMail (IKNP + ABY).

Method	Techniques	Communication (MiB)	
		Setup	Online
circuit	IKNP + ABY*	883.89	15.70
	Silver + ABY	4.95	15.70
	Silver + ABY2.0	18.45	7.85
bucket	IKNP + ABY*	206.40	24.12
	Silver + ABY	1.16	24.12
	Silver + ABY2.0	4.51	12.06
index	IKNP + ABY*	91.89	9.44
	Silver + ABY	**0.51**	9.44
	Silver + ABY2.0	1.90	**4.72**

candidates for the OT extension part in the setup and online phases, respectively. We then considered the case of 8-length character keywords over 300 emails and estimated the communication costs for the search approaches if they had been implemented with these protocols, with the results tabulated in Table 7.[20] We used our implementation as the baseline for the analysis and is denoted by IKNP + ABY* in Table 7. We now consider the following two cases:

Replacing IKNP [65] with Silver [32]: In this case, instead of IKNP, the Silver protocol is used to perform the OT computations during the setup phase. While generating $m = 2^{20}$ Correlated Oblivious Transfers (C-OTs), Silver [32, Fig.6] demonstrated $\approx 178\times$ improvement in communication over IKNP [65]. We note that our protocols can gain the same benefit, and the impact is significant. For example, as shown in Table 7, the setup communication of the circuit-based approach in our protocol is 883.89 MiB. This will be reduced drastically to 4.95 MiB by using Silver, demonstrating the scheme's practicality. It is worth noting that computing an increased number of OTs in a single shot will increase Silver's efficiency over IKNP even further. To be more specific, Silver [32, Fig.6] requires only $2130\times$ less communication than IKNP to generate $m = 2^{20}$ C-OTs. This is especially useful in cases like ours, where the search circuit corresponding to the circuit-based approach contains over 27 million AND gates (each AND requires 2 C-OTs) for the given set of parameters. The benefit of Silver over IKNP is clearly evident in other search approaches as well.

Replacing ABY [36] with ABY2.0 [94]: ABY2.0 improves on ABY's online communication by using input-dependent preprocessing while maintaining the same cost for the setup phase. In concrete terms, ABY requires 4 bits of online communication per AND gate, whereas ABY2.0 requires only 2. Hence implementing PrivMail using ABY2.0 will result in a $2\times$ improvement in the online communication. However, as shown in Table 7, this comes at the expense of increased setup costs. This is justified because in our analysis, we consider the generation of random OTs using Silver, whereas ABY2.0 demanded OTs on predefined inputs. To bridge this gap, we used the standard technique of preprocessing OTs [11] which resulted in an additional 3 bits of communication per random OT. However, we are not ruling out the possibility of generating the necessary amount of OTs for ABY2.0 using Silver without incurring this overhead. Despite the increased setup communication, combining Silver with ABY2.0 results in a protocol with better overall communication than when Silver is used with ABY, for both the bucketing and index-based search approaches. This is primarily due to the fact that online communication dominates communication during the setup phase for these approaches.

[20]We stress that the calculations are only intended to provide a rough estimate for the comparison, and that the results may vary significantly when implemented and benchmarked.

MPC with Trusted-Setup Our keyword search implementation (cf. §6.2) scales to the N-party setting while providing full threshold, meaning that the security is guaranteed even if $N-1$ parties are corrupted. Although this ensures strong privacy, there are variants that can be more efficient. We evaluate the closest variant, called MPC with trusted-setup [103], in this section to determine its impact on runtime and communication overhead. We emphasize that it is also easy to incorporate other practical settings with PrivMail, such as the honest-majority setting [85, 29, 78].

We ran experiments using the settings described in §6.2 over the circuit-based search (§5.2) without the length-hiding feature. The results of these experiments, presented in Table 8, demonstrated a 3.2× improvement in runtime and 52.2× reduction in communication. This reduction in communication is especially noteworthy, as it decreased from almost 1 gigabyte of communication to 18.85 MiB. The runtime was also reduced down by roughly 1/3 offering more acceptable overhead for real-time performance. Overall, these results advocates the practicality of PrivMail in scenarios where low communications is necessary and the security model allows a trusted third party to accelerate the protocols.

Table 8: Comparison of total runtime (in seconds) and communication between the servers (Comm. in MiB) of our protocol and MPC using trusted-setup [103] for a keyword of length 13 characters over 300 emails.

	Runtime (s)	Comm. (MiB)
Circuit-based §5.2	90.73	984.35
Trusted-Setup [103]	28.19	18.85
Improvement	3.2×	52.2×

7 Conclusion

In this paper, we presented PrivMail, an efficient and usable solution for secure email communication. PrivMail uses multiple email addresses and frees users from having to handle any cryptographic keys or certificates, as is the case with existing methods like PGP and S/MIME. Our scheme offers flexibility in privacy options, easy integration with the existing email infrastructures, and provides private server-side searches and filtering. Additionally, our system supports external searches for pre-approved or consented parties, which we hope will encourage email service providers to adopt our scheme. Overall, PrivMail is a promising alternative for more accessible and secure email communication.

Acknowledgments

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

References

[1] M. Y. Alhassan, D. Günther, Á. Kiss, and T. Schneider. Efficient and Scalable Universal Circuits. *J. Cryptology*, 2020.

[2] Apple and Google. Exposure Notification Privacy-Preserving Analytics (ENPA) white paper, 2021.

[3] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In *S&P*, 2017.

[4] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *CCS*, 2016.

[5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*, 2013.

[6] D. Atkins, W. Stallings, and P. Zimmermann. PGP Message Exchange Formats. RFC 1991, 1996. https://www.rfc-editor.org/rfc/rfc1991.txt.

[7] E. Atwater, C. Bocovich, U. Hengartner, E. Lank, and I. Goldberg. Leading Johnny to Water: Designing for Usability and Trust. In *SOUPS*, 2015.

[8] A. Bag, D. Talapatra, A. Rastogi, S. Patranabis, and D. Mukhopadhyay. TWo-IN-one-SSE: Fast, Scalable and Storage-Efficient Searchable Symmetric Encryption for Conjunctive and Disjunctive Boolean Queries. In *PETS*, 2023.

[9] W. Bai, M. Namara, Y. Qian, P. G. Kelley, M. L. Mazurek, and D. Kim. An Inconvenient Trust: User Attitudes toward Security and Usability Tradeoffs for Key-Directory Encryption Systems. In *SOUPS*, 2016.

[10] J. Baron, K. E. Defrawy, K. Minkovich, R. Ostrovsky, and E. Tressler. 5PM: Secure Pattern Matching. In *SCN*, 2012.

[11] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*, 1991.

[12] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *STOC*, 1990.

[13] A. Beimel and Y. Ishai. Information-Theoretic Private Information Retrieval: A Unified Construction. In *ICALP*, 2001.

[14] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing Semi-Honest Secure Multiparty Computation for the Internet. In *CCS*, 2016.

[15] A. Ben-Efraim, M. Nielsen, and E. Omri. Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing. In *ACNS*, 2019.

[16] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*, 1988.

[17] L. Blackstone, S. Kamara, and T. Moataz. Revisiting Leakage Abuse Attacks. In *NDSS*, 2020.

[18] M. S. Blog. Next steps in privacy-preserving Telemetry with Prio, 2019. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

[19] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko. MOTION - A Framework for Mixed-Protocol Multi-Party Computation. *ACM TOPS*, 2021.

[20] A. Brüggemann, R. Hundt, T. Schneider, A. Suresh, and H. Yalame. FLUTE: Fast and Secure Lookup Table Evaluations. In *S&P*, 2023.

[21] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of Hybrid Protocols for Practical Secure Computation. In *CCS*, 2018.

[22] M. Byali, H. Chaudhari, A. Patra, and A. Suresh. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PoPETs*, 2020.

[23] C. Cachin, S. Micali, and M. Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. In *EUROCRYPT*, 1999.

[24] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *CCS*, 2015.

[25] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO*, 2013.

[26] G. R. Chandran, R. Nieminen, T. Schneider, and A. Suresh. PrivMail: A Privacy-Preserving Framework for Secure Emails. In *ESORICS*, 2023. https://encrypto.de/code/PrivMail.

[27] Y. Chang. Single Database Private Information Retrieval with Logarithmic Communication. In *ACISP*, 2004.

[28] M. Chase and E. Shen. Substring-Searchable Symmetric Encryption. *PoPETs*, 2015.

[29] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *CCSW@CCS*, 2019.

[30] H. Chaudhari, R. Rachuri, and A. Suresh. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS*, 2020.

[31] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *FOCS*, 1995.

[32] G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In *CRYPTO*, 2021.

[33] M. Crispin. Internet Message Access Protocol - Version 4rev1. RFC 3501, 2003. https://rfc-editor.org/rfc/rfc3501.txt.

[34] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*, 2012.

[35] A. Davidson, P. Snyder, E. B. Quirk, J. Genereux, B. Livshits, and H. Haddadi. STAR: Secret Sharing for Private Threshold Aggregation Reporting. In *ACM CCS*. ACM, 2022.

[36] D. Demmler, T. Schneider, and M. Zohner. ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*, 2015.

[37] D. Demmler, A. Herzberg, and T. Schneider. RAID-PIR: Practical Multi-Server PIR. In *CCSW*, 2014.

[38] D. Demmler, M. Holz, and T. Schneider. OnionPIR: Effective Protection of Sensitive Metadata in Online Communication Networks. In *ACNS*, 2017.

[39] O. o. P. A. Department of Justice. International Statement: End-To-End Encryption and Public Safety, 2020. https://www.justice.gov/opa/pr/international-statement-end-end-encryption-and-public-safety.

[40] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner. Pushing the communication barrier in secure computation using lookup tables, 2017.

[41] Docker Container. https://www.docker.com.

[42] ETail Emarsys WBR SMB Report. Adapting to the pace of omnichannel commerce, 2016. https://emarsys.com/learn/white-papers/adapting-to-the-pace-of-omnichannel-commerce/.

[43] ExtremeTech. Google to Introduce End-to-End Gmail Web Encryption, 2022. https://www.extremetech.com/internet/341671-google-to-introduce-end-to-end-gmail-web-encryption.

[44] Fireblocks. MPC Wallet as a Service Technology, 2022. https://www.fireblocks.com/platforms/mpc-wallet/.

[45] L. Franceschi-Bicchierai. T-Mobile says hacker accessed personal data of 37 million customers. https://techcrunch.com/2023/01/19/t-mobile-data-breach/, 2023.

[46] S. Garg, P. Mohassel, and C. Papamanthou. TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption. In *CRYPTO*, 2016.

[47] R. Gennaro, C. Hazay, and J. S. Sorensen. Text Search Protocols with Simulation Based Security. In *PKC*, 2010.

[48] C. Gentry and Z. Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *ICALP*, 2005.

[49] M. George, S. Kamara, and T. Moataz. Structured Encryption and Dynamic Leakage Suppression. In *EUROCRYPT*, 2021.

[50] N. Gilbert. Number of Email Users Worldwide 2022/2023: Demographics & Predictions. https://financesonline.com/number-of-email-users/, 2022.

[51] I. Goldberg. Improving the Robustness of Private Information Retrieval. In *S&P*, 2007.

[52] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game: A Completeness Theorem for Protocols with Honest Majority. In *STOC*, 1987.

[53] T. R. Group. Email Statistics Report, 2019-2023, 2018. https://www.radicati.com/wp/wp-content/uploads/2018/12/Email-Statistics-Report-2019-2023-Executive-Summary.pdf.

[54] Z. Gui, K. G. Paterson, and S. Patranabis. Rethinking Searchable Symmetric Encryption. In *S&P*, 2023.

[55] D. Günther, M. Heymann, B. Pinkas, and T. Schneider. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *USENIX Security*, 2022.

[56] I. U. Haq, P. Black, I. Gondal, J. Kamruzzaman, P. A. Watters, and A. S. M. Kayes. Spam Email Categorization with NLP and Using Federated Deep Learning. In *ADMA*, volume 13726, pages 15–27, 2022.

[57] C. Hazay and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *J. Cryptology*, 2010.

[58] C. Hazay and T. Toft. Computationally Secure Pattern Matching in the Presence of Malicious Adversaries. *J. Cryptology*, 2014.

[59] Y. Huang, D. Evans, and J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*, 2012.

[60] B. A. Huberman, M. K. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *EC*, 1999.

[61] IBM Security. Cost of a Data Breach Report 2023. https://www.ibm.com/reports/data-breach, 2023.

[62] Imaplib. https://docs.python.org/3/library/imaplib.html#imaplib.IMAP4.fetch.

[63] Inpher. XOR Secret Computing Engine, 2022. https://inpher.io/xor-secret-computing/.

[64] M. Ion, B. Kreuter, E. Nergiz, S. Patel, S. Saxena, K. Seth, D. Shanahan, and M. Yung. Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions. *Cryptology ePrint Archive, Report*, 2017.

[65] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, 2003.

[66] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS*, 2012.

[67] S. Jha, L. Kruger, and V. Shmatikov. Towards Practical Privacy for Genomic Computation. In *S&P*, 2008.

[68] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert. Mobile Private Contact Discovery at Scale. In *USENIX Security*, 2019.

[69] S. Kamara, A. Kati, T. Moataz, T. Schneider, A. Treiber, and M. Yonli. SoK: Cryptanalysis of Encrypted Search with LEAKER - A framework for LEakage AttacK Evaluation on Real-world data. In *Euro S&P*, 2022.

[70] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS*, 2012.

[71] J. Katz and L. Malka. Secure text processing with applications to private DNA matching. In *CCS*, 2010.

[72] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *CCS*, 2013.

[73] D. J. C. Klensin. Simple Mail Transfer Protocol. RFC 5321, 2008. https://rfc-editor.org/rfc/rfc5321.txt.

[74] B. Klimt and Y. Yang. The Enron Corpus: A New Dataset for Email Classification Research. In *ECML*, 2004. https://www.cs.cmu.edu/~./enron/.

[75] V. Kolesnikov, A. Sadeghi, and T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS*, 2009.

[76] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, 2008.

[77] N. Koti, M. Pancholi, A. Patra, and A. Suresh. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *USENIX Security*, 2021.

[78] N. Koti, A. Patra, R. Rachuri, and A. Suresh. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*, 2022.

[79] E. Kushilevitz and R. Ostrovsky. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *FOCS*, 1997.

[80] H. Liu, Y. Yu, S. Zhao, J. Zhang, W. Liu, and Z. Hu. Pushing the Limits of Valiant's Universal Circuits: Simpler, Tighter and More Compact. In *CRYPTO*, 2021.

[81] M. Martinoli. Behind the scenes of ProtonMail's message content search, 2022. https://proton.me/blog/engineering-message-content-search.

[82] C. A. Meadows. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In *S&P*, 1986.

[83] C. A. Melchor, J. Barrier, L. Fousse, and M. Killijian. XPIR: Private Information Retrieval for Everyone. *PoPETs*, 2016.

[84] A. Melnikov and B. Leiba. Internet Message Access Protocol (IMAP) - Version 4rev2. RFC 9051, 2021. https://rfc-editor.org/rfc/rfc9051.txt.

[85] P. Mohassel and P. Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *CCS*, 2018.

[86] P. Mohassel, P. Rindal, and M. Rosulek. Fast Database Joins and PSI for Secret Shared Data. In *CCS*, 2020.

[87] K. S. Namjoshi and G. J. Narlikar. Robust and Fast Pattern Matching for Intrusion Detection. In *INFOCOM*, 2010.

[88] T. Nguyen, N. Karunanayake, S. Wang, S. Seneviratne, and P. Hu. Privacy-preserving spam filtering using homomorphic and functional encryption. *Comput. Commun.*, 197:230–241, 2023.

[89] F. G. Olumofin and I. Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *FC*, 2011.

[90] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI - A System for Secure Face Identification. In *S&P*, 2010.

[91] S. Oya and F. Kerschbaum. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. In *USENIX Security*, 2021.

[92] C. Page and Z. Whittaker. It's All in the (Lack of) Details: 2022's badly handled data breaches. https://techcrunch.com/2022/12/27/badly-handled-data-breaches-2022/, 2022.

[93] M. A. Pathak, M. Sharifi, and B. Raj. Privacy Preserving Spam Filtering. *CoRR*, abs/1102.4021, 2011.

[94] A. Patra, T. Schneider, A. Suresh, and H. Yalame. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security*, 2021.

[95] A. Patra and A. Suresh. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*, 2020.

[96] N. Perlroth. Yahoo Says Hackers Stole Data on 500 Million Users in 2014, 2016. https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html.

[97] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient Circuit-Based PSI with Linear Communication. In *EUROCRYPT*, 2019.

[98] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*, 2018.

[99] D. Poddebniak, C. Dresen, J. Müller, F. Ising, S. Schinzel, S. Friedberger, J. Somorovsky, and J. Schwenk. Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels. In *USENIX Security*, 2018.

[100] Python Time Complexity. https://wiki.python.org/moin/TimeComplexity.

[101] E. S. Raymond. AIS Payload Data Types, 2017. https://gpsd.gitlab.io/gpsd/AIVDM.html.

[102] A. Reuter, K. Boudaoud, M. Winckler, A. Abdelmaksoud, and W. Lemrazzeq. Secure Email - A Usability Study. In *FC Workshops*, 2020.

[103] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *ASIACCS*, 2018.

[104] M. Rosulek and L. Roy. Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits. In *CRYPTO*, 2021.

[105] S. Ruoti, J. Andersen, L. Dickinson, S. Heidbrink, T. Monson, M. O'Neill, K. Reese, B. Spendlove, E. Vaziripour, J. Wu, D. Zappala, and K. E. Seamons. A Usability Study of Four Secure Email Tools Using Paired Participants. *ACM TOPS*, 2019.

[106] S. Ruoti, J. Andersen, S. Heidbrink, M. O'Neill, E. Vaziripour, J. Wu, D. Zappala, and K. E. Seamons. "We're on the Same Page": A Usability Study of Secure Email Using Pairs of Novice Users. In *CHI*, 2016.

[107] S. Ruoti, J. Andersen, D. Zappala, and K. E. Seamons. Why Johnny Still, Still Can't Encrypt: evaluating the usability of a modern PGP client. *CoRR 1510.08555*, 2015.

[108] J. Schaad, B. C. Ramsdell, and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification. RFC 8551, 2019. https://rfc-editor.org/rfc/rfc8551.txt.

[109] T. Schneider and M. Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In *FC*, 2013.

[110] J. Schwenk. *Attacks on s/mime and openpgp*. In *Guide to Internet Cryptography: Security Protocols and Real-World Attack Implications*. Springer International Publishing, 2022, pages 431–445.

[111] Sepior. Advanced MPC wallet™, 2022. https://sepior.com/products/advanced-mpc-wallet/.

[112] D. Simmons. 17 Countries with GDPR-like Data Privacy Laws. https://insights.comforte.com/countries-with-gdpr-like-data-privacy-laws, 2022.

[113] D. X. Song, D. A. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *S&P*, 2000.

[114] V. Song. Mother of All Breaches Exposes 773 Million Emails, 21 Million Passwords. https://gizmodo.com/mother-of-all-breaches-exposes-773-million-emails-21-m-1831833456, 2019.

[115] P. Technologies. ProtonMail Security Features and Infrastructure, 2016. https://protonmail.com/docs/business-whitepaper.pdf.

[116] Thunderbird Query API. https://webextension-api.thunderbird.net/en/stable/messages.html#query-queryinfo.

[117] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. U. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *CCS*, 2007.

[118] Tutanota. Searching encrypted data is now possible with Tutanota's innovative feature, 2017. https://tutanota.com/blog/posts/first-search-encrypted-data.

[119] Tutanota. Secure email made for you. https://tutanota.com/security.

[120] L. G. Valiant. Universal Circuits (Preliminary Report). In *STOC*, 1976.

[121] T. Watson. The number of email addresses people use [survey data], 2019. https://www.zettasphere.com/how-many-email-addresses-people-typically-use.

[122] X. Wei, M. Zhao, and Q. Xu. Efficient and secure outsourced approximate pattern matching protocol. *Soft Computing*, 2018.

[123] A. Whitten and J. D. Tygar. Why Johnny Can't Encrypt: A usability evaluation of PGP 5.0. In *USENIX Security*, 1999.

[124] Wikipedia. List of Data Breaches. https://en.wikipedia.org/wiki/List_of_data_breaches.

[125] YAML Data Serialization Language. https://yaml.org.

[126] A. C.-C. Yao. How to Generate and Exchange Secrets. In *FOCS*, 1986.

[127] J. Yao, Y. Zheng, Y. Guo, and C. Wang. SoK: A Systematic Study of Attacks in Efficient Encrypted Cloud Data Search. In *SBC@ASIACCS*, 2020.

[128] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba. Privacy-Preserving Wildcards Pattern Matching Using Symmetric Somewhat Homomorphic Encryption. In *ACISP*, 2014.

[129] S. Zahur, M. Rosulek, and D. Evans. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT*, 2015.

[130] Y. Zhang, J. Katz, and C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In *USENIX Security*, 2016.

A Communication Phase in PrivMail

Sending and receiving emails in PrivMail necessitates the use of software that splits the email into shares on the sender's side and reconstructs the original email content on the receiver's side from the shares. When compared to S/MIME or PGP, this software is much simpler and does not rely on certificates. This section contains more information about the software's requirements as well as information about our implementations of Sender Client Proxy (SCP) and Recipient Client Script (RCS).

Sending emails in PrivMail: As described in §4, the subject and body of the email are split into n parts and communicated via different email providers. This means that the sender must have access to n different outgoing mail servers and know at least n distinct email addresses of the receiver. In our SCP, we have a simple configuration file where the sender can define a bijective mapping from the servers to the email addresses for each recipient. However, there are alternative ways to construct the map. For example, the sender could have a pool of outgoing servers, which can be automatically used with any recipient. This approach is more user friendly especially if the sender is using a custom plugin software on the mail client instead of a proxy solution (like our SCP). The automatic selection must make sure that the privacy is not jeopardized by having a single entity to control multiple paths (see §4.1 for more details). This is easy to do by first mapping the same outgoing servers and email address domains together, e.g., use Google's outgoing server for a Gmail address.

The split parts of the email are binary data, thus we use the Base64 encoding and put the body inside a block with specific starting and ending lines as in the example below:

```
-----BEGIN SECRET SHARE BLOCK Ver1.0-----
MQJ5HFRyE1sHHWsNHjFRM2gdLiJ8GFBMCTNfbmpDL
RmQKNDRhbA1+TFQnEhkcURx4VEBtZVgJBC9OCFQyG
LOMXbiYrAjQ6IO4rSlUnDH5/dkpGdhguUH8SSQBOU
QiVdNiNd
-----END SECRET SHARE BLOCK Ver1.0-----
```

This approach was inspired by how S/MIME [108] and PGP [6] include encrypted data in emails.

The split subject parts are included as Base64 in the subject with no extra lines. However, we include a 48-bit UID at the start of the subject to assist the receiver in reconstructing the original content using the correct parts. We also note that if the receiver does not know how many parts are to be used for the reconstruction, this information must be included in the emails. It is possible to use the same "block approach" as described above for this. This method is also used to include different versions of the original body of the email that are later used in the private queries (see §5).

Receiving emails in PrivMail: The receiver can fetch emails from its mailboxes as usual, since PrivMail does not break the regular email format. In order to reconstruct the PrivMail emails, the receiver needs to first recognize them and group the parts correctly. With our RCS, we can fetch a set of emails (e.g., unread) from any number of mailboxes we have access to. Then we simply check if the body of the email contains the specific starting and ending lines in order to determine if we have a PrivMail email part. We should also check that the encoding of the block and subject field are valid. Then we use the UID to group the corresponding email parts together and finally reconstruct the original content of the email.

B Private Queries using MPC (§5)

Here we provide additional details on the private search $\mathcal{F}_{\mathsf{Search}}$ discussed in §5.

B.1 Length-Hiding Keyword Search (§5.2)

Recall that the length mask for two $\ell_{\max}$-bit strings x, y is of the form $\mathrm{M}_x = m_1 \| \cdots \| m_{\ell_{\max}} = \{1\}^{\ell} \| \{0\}^{\ell_{\max}-\ell}$. In our case, x and y are both character strings with $s_{\max}$ characters each, and the bit length will be $\ell_{\max} = b \cdot s_{\max}$. As a result, having only one mask bit per character is sufficient because the same mask bit can be used to mask all of the b bits of it. This reduces the size of M_{K} (corresponding to keyword K) from $\ell_{\max}$ to $s_{\max}$, resulting in a $b\times$ improvement.

Another optimization concerns the number of AND gates required for the LEQ circuit. The LEQ circuits operate over $\ell_{\max}$-bit inputs in all the t instances, as shown in Figure 11. Consider the last instance, $\mathsf{LEQ}^{\ell_{\max}}(\mathsf{K}, \widetilde{W}_t, \mathrm{M}_{\mathsf{K}})$, in which the last character of W (w_t) is compared to the first character of K (k_1). If the second character k_2 is part of the actual keyword and not a random pad, the output should be 0 regardless of whether $w_t = k_1$ or not. If, on the other hand, k_2 is a random pad, this corresponds to K being a single character keyword and we can safely take the result of comparing w_t with k_1 as the final result and discard the rest. The problem now boils down to determining whether or not k_2 is a part of the actual keyword K. To our advantage, the length mask M_{K} contains precisely this information. In particular, the second bit of the mask $m_2 = 1$ indicates that k_2 is a part of K, and 0 otherwise. Hence, the last instance can be computed as

$$\mathsf{LEQ}^{\ell_{\max}}(\mathsf{K}, \widetilde{W}_t, \mathrm{M}_{\mathsf{K}}) :\Leftrightarrow \mathsf{LEQ}^{b}(k_1, w_t, m_1) \wedge (\neg m_2) \ . \tag{5}$$

The LEQ operates over b bits rather than $\ell_{\max}$, resulting in a $t_{\max}$ times improvement. The m_2 part, on the other hand, necessitates the use of an additional AND gate. Following the same approach, the $(t-1)^{\text{th}}$ instance $\mathsf{LEQ}^{\ell_{\max}}(\mathsf{K}, \widetilde{W}_{t-1}, \mathrm{M}_{\mathsf{K}})$ can be computed as

$$\mathsf{LEQ}^{2b}(k_1 || k_2, w_{t-1} || w_t, m_1 || m_2) \wedge (\neg m_3) \ . \tag{6}$$

A similar optimization can be done for all the LEQ instances from $t - s_{\max} + 2$.

Wildcard Keyword Search: The length masking scheme can be slightly tweaked to enable wildcard search on the keywords at no extra cost. Remember from §5.2 that setting the mask bit for a character position to 0 effectively forces the search circuit SC to discard the comparison result at that position. We were interested in checking the equivalence of the first s characters in our case, so the mask was set as a vector of s 1s followed by 0s. As a result, it is sufficient to place the 0s of the mask at appropriate positions for the wildcard search. For example, if the keyword is '`secret`' and the mask is '101101', it is equivalent to searching for '`s*cr*t`', where '`*`' denote wildcard entries.

B.2 Multi-Word Search for Keywords

This section provides a high-level overview of approaches for extending the optimizations for single-word keywords in Bucket-based (§5.2) and Index-based (§5.2) searches to multi-word searches. We emphasize that by multi-word search, we mean a keyword that contains space character(s), and thus contains multiple words.

One straightforward approach might involve creating buckets for combinations of multiple words. While this concept theoretically applies to any quantity of words, it becomes notably unfeasible as the number of words grows. Also, index-based search has the advantage of only requiring a single search for texts (written in the same language) that contain the same words multiple times. Unfortunately, this advantage dissipates when dealing with combined multiple words.

Another option is to divide the multi-word search query into single-word searches and then combine the results. One disadvantage of this method is that the word order cannot be preserved, resulting in additional incorrect results. We note that although IMAP [33] supports multi-word searches, several Mail User Agents (MUAs), particularly for mobile clients, implement the search interface in such a way that multi-word keywords are split into separate single-word queries and any match is considered good, i.e., the queries are combined using the OR key.

A complete privacy-preserving multi-word search solution would necessitate the sender $\mathcal{S}$ secret sharing the *word position number* as well as the buckets. This should then be stored in the search index along with every occurrence bit, significantly increasing the index's size. However, single-word searches can now be combined with a circuit that verifies that the word position numbers are sequential. For the time being, we leave a more formal description of this solution as future work and omit a more precise examination. We anticipate that when the number of emails is not enormous, the regular multi-word search using circuit-based search described in §5.2 will work more efficiently.

B.3 Index Table Updates (§5.2)

Here, we provide details for creating the index table required for the Index-based search discussed in §5.2.

Local Updates In this approach, the CSs seek the assistance of receiver $\mathcal{R}$ in building the index table. To begin with, $\mathcal{R}$ creates a *base index table* with the most common words in the selected language, and the corresponding occurrence bit-strings $\vec{\mathsf{B}}^{\mathcal{W}_i}$ are set to empty strings. This table is then secret shared among the servers (CSs). The receiver will download the index table's shares at regular intervals (e.g., once per night), reconstruct and update it based on the emails received since the last update. This can be viewed as $\mathcal{R}$ keeping a local copy of the index table corresponding to the emails received after the last table update and updating the main table at regular intervals.

Consider the case where p' new emails have been received since the last index table update. Allow these emails to contain a total of d' distinct words, where d'_{old} represents the number of words already present in the main index table. It is obvious that the updated index table will contain $(d' - d'_{old})$ more rows. Furthermore, the size of each occurrence bit (corresponding to the table's column size) will be increased by p' bits. In order to hide the number of new words added in each update $(d' - d'_{old})$, the base index table contains several "dummy" words (e.g., using padding characters), which are later replaced with the new words by the receiver. If the index becomes full, i.e., there are no more dummy words, $\mathcal{R}$ doubles the size of the table and adds new dummy entries to it.

In cases where it is acceptable to reveal the number of new words added, communication with $\mathcal{R}$ can be optimized further. When $\mathcal{R}$ requests an update, it first downloads all the shares of the distinct words $\{\mathcal{W}_1, \ldots, \mathcal{W}_d\}$ from the servers, rather than the entire table. It computes the respective occurrence bits for all the new p' emails for all the words that are already in the table. If there are no new distinct words $(d' = d'_{old})$, (only) these occurrence bits are secret shared among the servers. If not, $\mathcal{R}$ performs a *full update*, which includes additionally downloading the shares of the occurrence bit-strings, adding rows for new words, randomly permuting the index rows, and finally secretly sharing the new index table with the servers. The rows are randomly permuted to avoid the servers map the newly added words to the respective email set.

Server-side Updates Here we focus on reducing receiver $\mathcal{R}$'s intervention by offloading more tasks to the servers. For ease of explanation, consider a new email E_j containing ξ distinct words $\vec{\mathcal{K}}^j = \{\mathcal{K}^j_1, \ldots, \mathcal{K}^j_\xi\}$ and let $\mathcal{K}^j_q$ be the q^{th} word in the list for $q \in \{1, \ldots, \xi\}$. Also, remember from the previous section that the index table already has d distinct words $\{\mathcal{W}_1, \ldots, \mathcal{W}_d\}$ in it.

Step 1: Servers evaluate $\xi \cdot d$ instances of EQ circuits of the form $\mathsf{EQ}(\mathcal{K}^j_q, \mathcal{W}_i)$ for $i \in \{1, \ldots, d\}$. As these are independent, they can be evaluated in parallel. Furthermore, because we are dealing with distinct words, there can only be *at most one* i for which $\mathsf{EQ}(\mathcal{K}^j_q, \mathcal{W}_i) = 1$ for a given q, and vice versa.

Step 2: The email E_j's occurrence bit at the i^{th} row $\mathsf{B}^{\mathcal{W}_i}_j$ should be set to 1 if any of $\mathcal{K}^j_q$ matches with $\mathcal{W}_i$ and 0 other wise. Using the observation above, it can be computed as

$$\mathsf{B}^{\mathcal{W}_i}_j = \bigoplus_{q=1}^{\xi} \mathsf{EQ}(\mathcal{K}^j_q, \mathcal{W}_i). \tag{7}$$

Step 3: To determine whether $\mathcal{K}^j_q$ is a new word not already in the index table, we use the bit $\mathsf{T}^{\mathcal{K}^j_q}$, which is defined as 1 if $\mathcal{K}^j_q \in \{\mathcal{W}_1, \ldots, \mathcal{W}_d\}$, and 0 otherwise. This can be calculated using the results from Step 1 above, in which $\mathcal{K}_q^j$ is compared against all of the d words in the index table, and is given as

$$\mathsf{T}^{\mathcal{K}_q^j} = \bigoplus_{i=1}^{d} \mathsf{EQ}(\mathcal{K}_q^j, \mathcal{W}_i). \tag{8}$$

Step 4: The servers compute $\mathsf{T}^{\mathcal{K}^j} = \neg(\mathsf{T}^{\mathcal{K}_1^j} \wedge \cdots \wedge \mathsf{T}^{\mathcal{K}_\xi^j})$ where $\mathsf{T}^{\mathcal{K}^j} = 1$ indicates that email E_j contains at least one distinct keyword not found in the index table, and 0 otherwise.

Step 5: At this point, the servers reconstruct $\mathsf{T}^{\mathcal{K}^j}$ towards the receiver $\mathcal{R}$, and the rest of the steps are similar to those described earlier for a full update. If there are no new distinct words, this approach is advantageous for $\mathcal{R}$ because it does not require updating. The computation and communication at $\mathcal{R}$'s side could be further reduced by batching the shares for several emails in one shot. $\mathcal{R}$ requires the distinct words list ($\vec{\mathcal{K}}^j$) only for emails where $\mathsf{T}^{\mathcal{K}^j} = 1$. This can be obtained securely from the CSs by invoking $\mathcal{F}_{\mathsf{Fetch}}$ (§5.4). Furthermore, $\mathcal{R}$ can skip the updates corresponding to words already present in the table, as this is already done by the servers in *Step 2*. However, re-sharing of the updated index table is required, and in this approach, the servers will only learn the number of new emails with distinct words.[21]

To completely eliminate the involvement of the receiver $\mathcal{R}$ while also preventing information leakage regarding the update, the servers will update the table entries in an oblivious manner as shown next.

Server-side Updates without Receiver In this section, we will show how to perform the index table update without the involvement of receiver $\mathcal{R}$. Remember that the servers securely computed the bit value $\mathsf{T}^{\mathcal{K}_q^j}$ (in *Step 3*), which has a value of 1 if the keyword $\mathcal{K}_q^j$ is already in the index table, and 0 otherwise.

Consider a new distinct keyword $\mathcal{K}_q$ that does not exist in the index table, i.e., $\mathsf{T}^{\mathcal{K}_q} = 0$. Remember that the index table is initially filled with dummy words in addition to the actual distinct words by the receiver. These dummy entries are used to hide the addition of new words to the table, and they will be replaced with actual distinct words as needed. The servers should now replace one of these dummy word entries with $\mathcal{K}_q$ and update the corresponding occurrence bit string. To track the first dummy word entry in the index, we define an *entry-bit* string of the form $\vec{\delta} = \delta_1 \| \cdots \| \delta_d$. It is a binary string with all 0s except a 1 at position i, which indicates that the first dummy word in the current index table is at the ith row. Initially, $\mathcal{R}$ sets $\vec{\delta} = 1 \| \{0\}^{d-1}$ and secret shares it among the servers.

In order for the approach to work, both the dummy word and its occurrence bit string in the index table must be set to a string of all 1s.[22] This has no effect on the scheme's privacy because the servers will be unable to distinguish these entries from the others due to the privacy guaranteed by the underlying MPC scheme's secret sharing.

Corresponding to each row i of the index table, we define a boolean flag $\mathsf{flag}_i = \neg\mathsf{T}^{\mathcal{K}_q} \wedge \delta_i$. Note that $\mathsf{flag}_i = 1$ indicates that the keyword $\mathcal{K}_q$ can replace the entry at the ith row. Given this flag, the word in the ith row of the table $\mathcal{W}_i$ is updated using an update function UD defined as

$$\mathsf{UD}(\mathcal{W}_i, \mathcal{K}_q, \mathsf{flag}_i) = \mathcal{W}_i \wedge \left(\mathcal{K}_q \vee \neg\mathsf{flag}_i\right) \ . \tag{9}$$

Case 1: When $\mathsf{flag}_i = 0$, the term $(\mathcal{K}_q \vee \neg\mathsf{flag}_i)$ evaluates to all ones and hence $\mathsf{UD}(\mathcal{W}_i, \mathcal{K}_q, \mathsf{flag}_i) = \mathcal{W}_i$. This means that the current entry is unaffected as desired.

Case 2: When $\mathsf{flag}_i = 1$, $\mathsf{UD}(\mathcal{W}_i, \mathcal{K}_q, \mathsf{flag}_i) = \mathcal{W}_i \wedge \mathcal{K}_q$. However, in this case, $\mathcal{W}_i$ is a dummy word and as previously stated, it is simply a string of all 1s. Hence, the result of UD is simply $\mathcal{K}_q$.

The same idea can be applied to the case of the occurrence bit string as well. The servers must update the occurrence bit string to reflect the new distinct word $\mathcal{K}_q$. If this keyword corresponds to the jth email (this

[21]This is equal to the number of $\mathcal{F}_{\mathsf{Fetch}}$ instances and leakage can be reduced by using multi-block or volume-hiding PIR to instantiate $\mathcal{F}_{\mathsf{Fetch}}$.

[22]This can also be viewed as treating a string of b 1 bits as a special dummy character.

information is known to the servers), the occurrence bit string will be all zero except at position j, which will be set to 1. Servers then locally generate a boolean sharing of this string as per the underlying MPC protocol.

Following the above set of operations with respect to a distinct word $\mathcal{K}_q$, we should also update the entry-bit string $\vec{\delta}$. If there was an update ($\mathsf{T}^{\mathcal{K}_q} = 0$), the dummy word position in the bit string should be moved by one bit position; otherwise, it should remain unchanged. Since $\vec{\delta}$ is a bit vector with exactly one bit set to 1, this can be accomplished by setting each of the bits as $\delta_i \leftarrow \delta_{i-1} \oplus \left(\mathsf{T}^{\mathcal{K}_q} \wedge (\delta_i \oplus \delta_{i-1})\right)$ with $\delta_i = 0$.

The preceding procedure is repeated for each distinct word in a new email. In addition, to monitor when the index table becomes full and needs to be extended/doubled, the servers reconstruct it and check if $\delta_d = 1$.

In the above approach of index table updates without the involvement of the receiver, the servers would have to update each entry in the index table for each set of distinct words in the new email. The problem of finding an optimized solution for this case is left open for future work.