University of California
Santa Barbara

Bridging Strong Privacy with Scalability for Online Data Services

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Ishtiyaque Ahmad

Committee in charge:

Professor Divyakant Agrawal, Co-Chair
Professor Trinabh Gupta, Co-Chair
Professor Amr El Abbadi

September 2024

The Dissertation of Ishtiyaque Ahmad is approved.

Professor Amr El Abbadi

Professor Divyakant Agrawal, Committee Co-Chair

Professor Trinabh Gupta, Committee Co-Chair

September 2024

Bridging Strong Privacy with Scalability for Online Data Services

by

Ishtiyaque Ahmad

To my family.

Acknowledgements

I would like to express my deepest gratitude to all those who have supported and guided me throughout my PhD journey.

I am immensely grateful to my advisors, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta—not only for their instrumental mentorship throughout my PhD but for the lasting impact they have had on my growth. Their invaluable guidance, insights, patience, and belief in me—most importantly, an optimal balance between providing direction and fostering intellectual freedom—have shaped me into the person I am today. Any success I achieve in the future, whether academic or professional, will be a testament to their influence.

I am also thankful to all my professors at UCSB, whose knowledge and wisdom have profoundly influenced me through their courses, discussions, and interactions. A heartfelt thanks to my labmates Erwan, Fuheng, Lawrence, Mohammad, Sujaya, and Victor, for their camaraderie and support, both intellectually and personally. Their constant help, feedback, and encouragement have made this journey much smoother.

To my friends, thank you for your continuous encouragement, and to all the teachers in my life, I am deeply indebted and grateful for the lessons, both academic and otherwise, that you have imparted along the way.

Finally, to my parents, whose unwavering love, encouragement, and belief in me have been my greatest source of strength.

Curriculum Vitæ
Ishtiyaque Ahmad

Education

2024	Ph.D. in Computer Science (Expected), University of California, Santa Barbara.
2019	M.Sc. in Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.
2015	B.Sc. in Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh.

Publications

1. **Ishtiyaque Ahmad**, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. *Pantheon: Private retrieval from public key-value store.* International Conference on Very Large Data Bases (VLDB 23), Vancouver, Canada, August 2023.
2. **Ishtiyaque Ahmad**, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. *Private Information Retrieval in Large Scale Public Data Repositories.* International Conference on Very Large Data Bases (VLDB 23), Vancouver, Canada, August 2023.
3. Ling Liang, Jilan Lin, Zheng Qu, **Ishtiyaque Ahmad**, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. *SPG: Structure-private graph database via SqueezePIR.* International Conference on Very Large Data Bases (VLDB 23), Vancouver, Canada, August 2023.
4. Jilan Lin, Ling Liang, Zheng Qu, **Ishtiyaque Ahmad**, Liu Liu, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. *INSPIRE: In-Storage Private Information Retrieval via Protocol and Architecture Co-design.* International Symposium on Computer Architecture (ISCA 22), New York City, New York, USA, June 2022.
5. **Ishtiyaque Ahmad**, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. *Coeus: A system for oblivious document ranking and retrieval.* ACM Symposium on Operating Systems Principles (SOSP 21), Virtual event, October 2021.
6. **Ishtiyaque Ahmad**, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. *Addra: Metadata-private voice communication over fully untrusted infrastructure.* 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), Virtual event, July 2021.
7. Sujaya Maiyya, **Ishtiyaque Ahmad**, Divyakant Agrawal, and Amr El Abbadi. *Samya: A Geo-Distributed Data System for High Contention Aggregate Data.* IEEE International Conference on Data Engineering (ICDE 21), Virtual event, April 2021.

8. **Ishtiyaque Ahmad**, Md Anwar Parvez, and Anindya Iqbal. *TypoWriter: A Tool to Prevent Typosquatting.* IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC 19) Milwaukee, WI, USA, July 2019.
9. **Ishtiyaque Ahmad**. *Automatic Detection of Diabetic Retinopathy from Fundus Images using Image Processing and Artificial Neural Network.* M.Sc. Engineering Thesis, Department of CSE, BUET, Bangladesh (January 2019).
10. Md Tarikul Islam Papon, **Ishtiyaque Ahmad**, Nazmus Saquib, and Ashikur Rahman. *Non-invasive Heart Rate Measuring Smartphone Applications using On-board Cameras: A Short Survey.* The 1st International Conference on Networking Systems and Security (NSysS 2015), January 5-7, 2015, Dhaka, Bangladesh.
11. Nazmus Saquib, Md Tarikul Islam Papon, **Ishtiyaque Ahmad**, and Ashikur Rahman. *Measurement of Heart Rate Using Photoplethysmography.* The 1st International Conference on Networking Systems and Security (NSysS 2015), January 5-7, 2015, Dhaka, Bangladesh.

Abstract

Bridging Strong Privacy with Scalability for Online Data Services

by

Ishtiyaque Ahmad

In today's data-centric world, most of the global data resides in the cloud, accessible through various online services. Whether it's browsing medical advice on WebMD, referencing information on Wikipedia, managing investments through online stock brokers, or streaming content from platforms like Netflix and YouTube, our online interactions are vast. Ensuring privacy in these contexts is crucial because the knowledge of a user's accessed content could potentially reveal sensitive private information about the user. However, existing privacy measures employed by online services often fall short, leading to incidents where providers misuse and share users' sensitive data with external parties. Moreover, powerful entities, including nation-states and government agencies, frequently engage in mass surveillance by collecting such data. The only way to defend against such powerful privacy risks is to design these services such that the service provider is oblivious to our data. Although recent cryptography research offers techniques for such systems, their adoption in mass Internet applications remains limited due to scalability issues. As a result, a significant tension has prevailed between *privacy* and *scalability* for online services over the past few decades. This dissertation centers on the following fundamental question: *can we break this tension and build data systems that guarantee strong privacy to the users while ensuring both real-world scalability and practical performance?*

This dissertation explores the tension between privacy and scalability in three application areas— voice communication, accessing Wikipedia articles, and retrieving content from a cloud service provider's key-value database. Through the development of three

new systems — Addra, Coeus, and Pantheon — it substantially improves the tension in these application areas. These systems are designed on the fundamental principle of composing cryptographic building blocks in a manner that efficiently leverages their diverse strengths and weaknesses. Consequently, they represent a notable advancement in this field.

Addra is a system that facilitates voice communication while hiding the associated metadata, such as the knowledge of who is communicating with whom, as well as the time and duration of the call. This metadata contains rich information about people's lives and therefore is a prime target for powerful adversaries such as nation states. Existing systems that hide voice call metadata either require trusted intermediaries in the network or scale to only tens of users. Addra is the first system for voice communication that hides metadata over fully untrusted infrastructure and scales to tens of thousands of users. At a high level, Addra follows a template in which callers and callees deposit and retrieve messages from private mailboxes hosted at an untrusted server. However, Addra improves message latency in this architecture, which is a key performance metric for voice calls. First, it enables a caller to push a message to a callee in two hops, using a new way of assigning mailboxes to users that resembles how a post office assigns PO boxes to its customers. Second, it innovates on the underlying cryptographic machinery and constructs a new private information retrieval scheme, FastPIR, that reduces the time to process oblivious access requests for mailboxes. An evaluation of Addra on a cluster of 80 machines on AWS demonstrates that it can serve 32K users with a 99-th percentile message latency of 726 ms—a 7× improvement over a prior system for text messaging in the same threat model.

Coeus addresses a fundamental abstract problem termed as *oblivious document ranking and retrieval*. The problem is stated as follows: given a confidential string q and a remote server containing a collection of public documents $\mathcal{D}$, how can one effectively

select and access one of the top K most relevant documents to q within $\mathcal{D}$ without revealing any information about q or the chosen document, even to the server itself? At a high level, Coeus composes two cryptographic primitives: secure matrix-vector product for scoring document relevance using the widely-used term frequency-inverse document frequency (tf-idf) method, and private information retrieval (PIR) for obliviously retrieving documents. However, Coeus reduces the time to run these protocols, thereby improving the user-perceived latency, which is a key performance metric. Coeus first reduces the PIR overhead by separating out private metadata retrieval from document retrieval, and it then scales secure matrix-vector product to tf-idf matrices with several hundred billion elements through a series of novel cryptographic refinements and an efficient workload distribution strategy. For a corpus of English Wikipedia containing 5 million documents, a keyword dictionary with 64K keywords, and on a cluster of 143 machines on AWS, Coeus enables a user to obliviously rank and retrieve a document in 3.9 seconds—a 24× improvement over a baseline system.

The third contribution of this dissertation, Pantheon, addresses the problem of preserving client privacy in a scenario where a cloud server manages a key-value store and offers a private query service to clients. Ensuring client privacy in this setting is difficult because the key-value store is *public*, and a client cannot encrypt or modify it. Therefore, privacy in this context implies hiding the accesses pattern of a client. Pantheon cryptographically allows a client to retrieve the value corresponding to a key from a *public* key-value store without allowing the server or any adversary to know any information about the key or value accessed. Pantheon devises a single-round retrieval protocol which reduces server-side latency by refining its cryptographic machinery and massively parallelizing the query execution workload. Using these novel techniques, Pantheon achieves a 93× improvement for server-side latency over a state-of-the-art solution.

Contents

Chapter 1

Introduction

Privacy is a major concern for individuals in this digital age, where the use of everyday internet services exposes users to growing privacy risks. While recent advances in privacy and cryptography have developed techniques that can effectively protect against these risks, their implementation in large-scale internet services remains limited. A key reason is that a straightforward application of these techniques results in significant performance overhead, making them impractical when applied to mass internet services, which demand high throughput and low latency. This raises an important research question: *how can we design systems that not only provide strong privacy guarantees but are also scalable enough for integration into widely used internet applications?*

1.1 Problem Statement and Motivation

The vast amount of data we share with cloud services is vulnerable to privacy risks in a variety of ways. External attackers or hackers may attempt to steal our data [106, 116], while rogue employees at service providers could access private information [176], such as emails [154], without authorization. There have been numerous incidents where

service providers, who have full access to user data, have misused it [38, 70], either for their own benefit or by handing over sensitive information to third parties. In addition to these risks, powerful entities such as nation-states or government agencies are increasingly involved in the collection of sensitive data for mass surveillance [79, 66]. These entities are more dangerous than individual hackers or rogue employees because they have the resources to compromise the entire network infrastructures or coerce service providers into revealing data about ordinary citizens, often on a large scale [112]. The alarming nature of these privacy risks is exacerbated by the fact that current internet services are not designed to protect against such powerful adversaries.

As a concrete example, consider Wikipedia, a popular platform for exploring topics of interest. When users search for topics and read related articles, the current state-of-the-art privacy protection is secure communication, which encrypts search terms and retrieved content during transmission to prevent network adversaries from accessing this information. However, this protection is limited to external threats. A rogue employee at the server or a powerful entity that gains control of the server can still access a user's search history and the articles they view. This exposure can reveal sensitive information about a person's religious beliefs, preferences, medical conditions, financial status, or other private matters.

A second example is voice calls, a common form of communication in daily life. Although the state-of-the-art privacy measure in this domain—end-to-end encryption [72]—ensures that the content of the call is protected from network adversaries, it does not safeguard other associated information, such as the time, duration, and participants of the call—collectively known as voice call metadata. This metadata can be extremely revealing [146, 149, 59]. In some cases, governments have required telecommunications providers to retain metadata, including the participants and timing of calls. Analysis of this metadata has shown that it can be used to trace connections between journalists

and their sources, potentially compromising the anonymity of whistleblowers [113].

From these examples, it is clear that to provide strong privacy for users, we need to build systems where the service provider itself remains oblivious to user data. This raises the natural question: is it even possible to create such solutions? Can a service provider still offer useful functionalities without accessing certain user information? Fortunately, recent advancements in cryptography, including techniques such as homomorphic encryption [81, 83] and secure multiparty computation [88], demonstrate that the answer is yes. These tools allow meaningful computation to be performed on encrypted data, ensuring that the service provider cannot access the underlying information. Significant research has been dedicated to developing solutions using these techniques. However, these solutions are not yet widely adopted because they either fail to deliver strong enough privacy guarantee or are unable to scale for real-world applications.

This highlights a persistent tension between *privacy* and *scalability* in existing research. While it is impossible to eliminate this tension entirely—since privacy always comes with a cost—this dissertation seeks to explore how this tension can be minimized. In summary, it aims to investigate how we can build systems that (1) provide provable privacy, (2) tolerate entire infrastructure compromise, and (3) are scalable for practical use.

The primary approach of this dissertation is to evaluate existing cryptographic techniques, considering both their strengths and weaknesses, and then tailor system designs to maximize their advantages. This involves efficiently redesigning systems to leverage these strengths while addressing potential shortcomings. Additionally, the dissertation focuses on refining the underlying cryptographic machineries to address performance bottlenecks and enhance overall efficiency.

1.2 Dissertation Organization

This dissertation is organized as follows:

Chapter 2 introduces a new system called Addra [7], designed to hide voice call metadata—such as the identities of participants, as well as the time and duration of calls—even in the presence of a powerful adversary capable of compromising both the service provider and the entire communication infrastructure. Voice call metadata holds valuable insights into individuals' lives and is a key target for powerful entities, including nation-states. Existing systems that protect voice call metadata either rely on trusted intermediaries within the network or are limited in scalability, typically supporting only tens of users. In contrast, Addra is the first system that can hide voice call metadata over fully untrusted infrastructure while scaling to tens of thousands of users. At a high level, Addra follows a template in which callers and callees deposit and retrieve messages from private mailboxes hosted at an untrusted server. However, Addra improves message latency in this architecture, which is a key performance metric for voice calls, with two key idea. First, it enables a caller to push a message to a callee in two hops, using a new way of assigning mailboxes to users that resembles how a post office assigns PO boxes to its customers. Second, it innovates on the underlying cryptographic machinery and constructs a new private information retrieval scheme, FastPIR, that reduces the time to process oblivious access requests for mailboxes. An evaluation of Addra on a cluster of 80 machines on AWS demonstrates that it can serve 32K users with a 99-th percentile message latency of 726 ms—a 7× improvement over a prior system for text messaging in the same threat model.

Chapter 3 describes Coeus [6], a system that addresses the *oblivious document ranking and retrieval problem*. The problem is as follows: given a private string q and a remote server that holds a set of public documents $\mathcal{D}$, how can one of the K most relevant doc-

uments to q in $\mathcal{D}$ be selected and viewed without anyone (not even the server) learning anything about q or the document? Coeus is designed to solve this challenge by ensuring complete privacy during the document ranking and retrieval process. At a high level, Coeus composes two cryptographic primitives: secure matrix-vector product for scoring document relevance using the widely-used term frequency-inverse document frequency (tf-idf) method, and private information retrieval (PIR) for obliviously retrieving documents. However, Coeus reduces the time to run these protocols, thereby improving the user-perceived latency, which is a key performance metric. Coeus first reduces the PIR overhead by separating out private metadata retrieval from document retrieval, and it then scales secure matrix-vector product to tf-idf matrices with several hundred billion elements through a series of novel cryptographic refinements. For a corpus of English Wikipedia containing 5 million documents, a keyword dictionary with 64K keywords, and on a cluster of 143 machines on AWS, Coeus enables a user to obliviously rank and retrieve a document in 3.9 seconds—a 24× improvement over a baseline system.

Chapter 4 introduces Pantheon [5], a system that addresses the problem of *private retrieval from a public key-value store*. In this scenario, a cloud server maintains a key-value store and offers a private query service to its clients. Preserving client privacy in this context is challenging because the key-value store is *public*, meaning clients cannot encrypt or modify it. Therefore, privacy must be ensured by hiding the client's access patterns. Pantheon provides a cryptographic solution that enables clients to retrieve the value associated with a key from a public key-value store without revealing any information about the key or value to the server or any adversary. Pantheon devises a single-round retrieval protocol which reduces server-side latency by refining its cryptographic machinery and massively parallelizing the query execution workload. Pantheon overcomes previous limitations by supporting dynamic databases, achieving a 93× latency improvement over a state-of-the-art solution, and becoming the first system to

support sub-second private retrieval from a million-row key-value database.

Chapter 5 provides a comprehensive summary of the key contributions and findings of the dissertation, highlighting the advancements made in building privacy-preserving systems. Chapter 6 explores potential avenues for future research that stem from the ideas and techniques introduced in this dissertation. It identifies open problems and areas where further exploration could enhance the efficiency, scalability, and applicability of privacy-preserving technologies.

Chapter 2

Addra: Metadata-private voice communication over fully untrusted infrastructure

2.1 Introduction

Voice call metadata—the parties involved in the call, the duration of the call, and the time of the call—can be incredibly revealing. The former General Counsel of NSA, Stewart Baker, has said, "metadata absolutely tells you everything about somebody's life. If you have enough metadata, you don't really need content" [179, 47, 48]. Several academic studies [146, 149, 59] have confirmed the power of metadata. As an example, Mayer et al. [146] used telephone metadata to infer that a study participant "received a long phone call from the cardiology group at a regional medical center, talked briefly with a medical laboratory, ... and made brief calls to a self-reporting hotline for a cardiac arrhythmia monitoring device." The authors confirmed that the participant had a cardiac arrhythmia. A study of whistle-blowers also revealed that metadata can identify

a journalist's sources [113].

Given the information contained in metadata, a significant question is: how can one make a voice call without revealing to anyone the metadata associated with the call? Fortunately, several systems have tackled this problem [135, 133, 86, 194, 25, 203] (§2.7). Although these systems hide metadata and keep message latency low, they either restrict scalability to only tens of users [86, 194], or are vulnerable to attacks by requiring trusted intermediaries in the communication infrastructure [25, 135, 133, 203]. An example of a trust assumption is that the system guarantees security only if the adversary can compromise at most a fraction (20%) of the servers that route user calls [133]. Trusting intermediaries can be risky as powerful adversaries like nation states are the ones that try to collect metadata. Such adversaries have been known to wield their vast political, technical, and financial power to gain access to metadata [138, 18, 180, 157]. A system that can withstand strong adversaries while serving more than tens of users is Pung [16, 13]. Pung makes *no* assumptions about the communication infrastructure—the adversary may compromise a part or all of the infrastructure. However, Pung targets applications such as email and chat with long-lived messages that are retrieved asynchronously. Indeed, a Pung client makes $\lceil \log_2(n+1) \rceil$ round trips to a remote server to obliviously search and retrieve a message (n is the number of users), thereby incurring several seconds of message latency (§2.6.1). In contrast, voice calls have a strict time budget. If a user sends a packet every few hundred milliseconds, then each hop in the communication infrastructure must not spend longer than this time period to process and forward the packet, to avoid an unbounded packet build up.

We present Addra, the first system that provably hides metadata for voice calls, makes no assumptions about the underlying infrastructure, and scales to tens of thousands of users. In terms of privacy guarantees, Addra provides relationship unobservability—an adversary cannot detect whether a relationship (voice call) exists between any two

users of the system [167] (§2.2.1). These privacy guarantees are achieved with practical latency performance of under 750 ms, and for low-bandwidth voice synthesis at a rate of 1.6 Kbit/s as in the Mozilla LPCNet voice codec [206, 205, 151]. Addra, like Pung, relies on a set of mailboxes hosted at an untrusted server. Callers deposit messages and callees retrieve messages from these mailboxes using a private information retrieval (PIR) cryptographic protocol [126, 44, 46] (§2.3.2). This protocol ensures that the untrusted server does not learn which mailbox a callee is accessing, thereby unlinking the callee from the caller. However, Addra must address two challenges in this architecture to support low-latency voice calls (§2.2.3). First, it must reduce the number of round trips a caller or callee makes to the server to transfer or retrieve a voice packet. Second, Addra must reduce the time the server takes to process caller and callee requests, particularly, the PIR requests.

Addra addresses the first challenge through a new, and remarkably simple, use of mailboxes (§2.3). When someone rents a conventional post office box, or PO box, at a post office, they get a mailbox with a unique and fixed address into which the mailman deposits incoming mail. Addra inverts this architecture. In Addra, a caller (rather than a recipient or callee) gets a dedicated mailbox with a fixed address or "phone number". The caller deposits its outgoing messages into this mailbox—independent of who the caller is calling. (Thus, an adversary cannot tell whom the caller is calling.) Meanwhile, a callee retrieves a message from the mailbox tied to the caller's phone number using a PIR protocol. Crucially, to transmit a message, a caller makes one push request to the server, and the server makes one push request to the callee—a hop count of two. In contrast, prior work requires multiple round trips between the server and the callees.

Addra addresses the second challenge mentioned above, of reducing server-side processing time for PIR, by two means. First, it parallelizes PIR processing across multiple server machines and multiple CPU cores on a machine. The fact that PIR is parallelizable

is known and studied [62, 99]. Second, and more saliently, Addra constructs a new PIR scheme, FastPIR, that fundamentally reduces the server-side PIR processing time relative to prior state-of-the-art schemes [3, 13] (§2.4). Even though FastPIR was motivated by Addra, it can be used for other applications of PIR [150, 93, 31, 63, 140, 142].

FastPIR builds on the homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [33, 75] (§2.4.1) and leverages two of its features. First, it uses the single instruction, multiple data (SIMD) capability of BFV ciphertexts to compute on compressed PIR requests. Prior state-of-the-art schemes [13, 3] also exploit SIMD capabilities but not in a way that keeps PIR requests compressed in memory. Meanwhile, such compression improves memory utilization, reduces CPU time, and eliminates the time to uncompress requests (§2.4.2). However, working over compressed requests naively increases PIR response size. So, second, FastPIR uses homomorphic rotation operations in BFV to pack multiple pieces of a PIR response, thereby reducing response size. Further, FastPIR reduces both the CPU time per rotation and the number of calls to this operation (§2.4.3, §2.4.4).

For completeness, Addra includes a dialing protocol that allows a callee to detect that a caller is calling and learn the caller's phone number (mailbox address). For this purpose, Addra uses the dialing protocol from Pung (§2.5).

We have implemented (§2.5) and evaluated (§2.6) a prototype of Addra. Our prototype runs on Amazon EC2 where the server runs in the US East region, and the clients (callers and callees) run geographically apart in the US West region. When the server uses 80 machines, Addra supports 32K clients communicating with each other with a 99-th percentile message latency of 726 ms. In contrast, Pung (the only other system that works at scale over completely untrusted infrastructure) transmits messages for the same number of users with a message latency of 5.2 seconds. Besides, Addra requires a network download bandwidth of 1.46 Mbps and an upload bandwidth of 30 Kbps for

every client. Although Addra achieves low message latency for a few tens of thousands of users, it does not currently scale to hundreds of thousands or a few million users due to the overhead of PIR which grows quadratically with the number of users. Furthermore, although its instantaneous bandwidth requirements are modest, the total network transfers are high as a client must remain online even if it is not participating in a call to hide call initiation patterns. Thus, Addra assumes clients with unlimited data plans. Nevertheless, Addra demonstrates, for the first time, that even over completely untrusted infrastructure, metadata for voice calls can be hidden at scale for tens of thousands of users.

2.2 Goals, threat model, and challenges

Addra's goal, at a high level, is to enable its users to make peer-to-peer voice calls while hiding metadata from a powerful adversary that may compromise the entire communication infrastructure.

2.2.1 Goals

Performance and scalability. Voice calls require the communication infrastructure to transmit messages with low latency. Addra targets a sub-second message latency due to the feasibility of voice calls under such a setting [133]. Thus, if Alice sends a voice packet to Bob, then Bob should receive it within one second. Additionally, the infrastructure must not queue up voice packets indefinitely. For instance, if Alice generates a voice packet every 500 ms, then every hop in the infrastructure must spend no more than 500 ms to process the packet before sending it forward towards Bob. Addra must also provide sufficient throughput so that the transmitted voice is understandable. For this purpose, Addra targets the LPCNet voice codec [206, 205], which specializes in low-

bandwidth voice synthesis at a rate of 1.6 Kbit/s. Finally, we want Addra to scale to a large number of users (for example, tens of thousands on a cluster of hundred machines).

Content privacy. Addra must ensure that only the caller and callee of a voice call can comprehend the content of the voice packets they send to each other.

Metadata privacy. Addra, similar to Pung [16], targets the guarantee of *relationship unobservability* as defined by Pfitzmann and Hansen [167]. Relationship unobservability states that it is undetectable whether a relationship (voice call) exists between a sender (caller) and a recipient (callee), unless the sender or the recipient are compromised. If either the caller or the callee is compromised, then offering privacy guarantees has little value, as the compromised party can trivially reveal the existence of communication (or lack thereof).

2.2.2 Threat model and assumptions

As motivated in the introduction (§2.1), Addra assumes an adversary who can compromise the entire communication infrastructure, including routers, switches, and middleboxes. The adversary can observe network traffic, perform traffic analysis, and manipulate traffic: reorder, replay, change, and inject network packets.

Callers and callees trust their own devices. More generally, the adversary can compromise a subset of end user devices. In this case, Addra must provide content and metadata privacy to the users of non-compromised devices.

The adversary may not break standard cryptographic primitives such as public-key and symmetric-key encryption.

The adversary may mount a denial-of-service attack: bring down the entire communication infrastructure or selectively drop traffic. In such cases, Addra cannot guarantee

voice communication but must continue to guarantee privacy.

2.2.3 Challenges

Meeting the performance and privacy goals stated above under the threat model just described is challenging. Indeed, prior work either relaxes the threat model or does not meet the performance goals. For instance, Yodel [133] is a metadata-private voice communication system that scales to several million users but assumes that a server in the communication infrastructure is compromised with only a 20% chance. On the other hand, Pung [16, 13] works in the stronger threat model. However, it cannot push frequent messages from a caller to a callee. As mentioned earlier (§2.2.1), if a caller samples voice every 500 ms, then each hop of the communication infrastructure must process a voice packet within 500 ms before the arrival of the next packet to avoid packet build up. This time budget entails that a caller or a callee cannot make multiple round trips to a server in the communication infrastructure to send or receive a single packet. But Pung requires message recipients to make multiple such trips to its server.

Addra addresses these challenges and meets the performance requirements for tens of thousands of users without making any trust assumptions, as described next.

2.3 Architecture and overview of design

2.3.1 Architecture

Figure 2.3 shows Addra's architecture. Addra consists of a server and user (participant) devices. The server runs over untrusted infrastructure. It is logically centralized but physically distributed over multiple machines. The server's role is to facilitate communication among the user devices in a privacy-preserving manner.

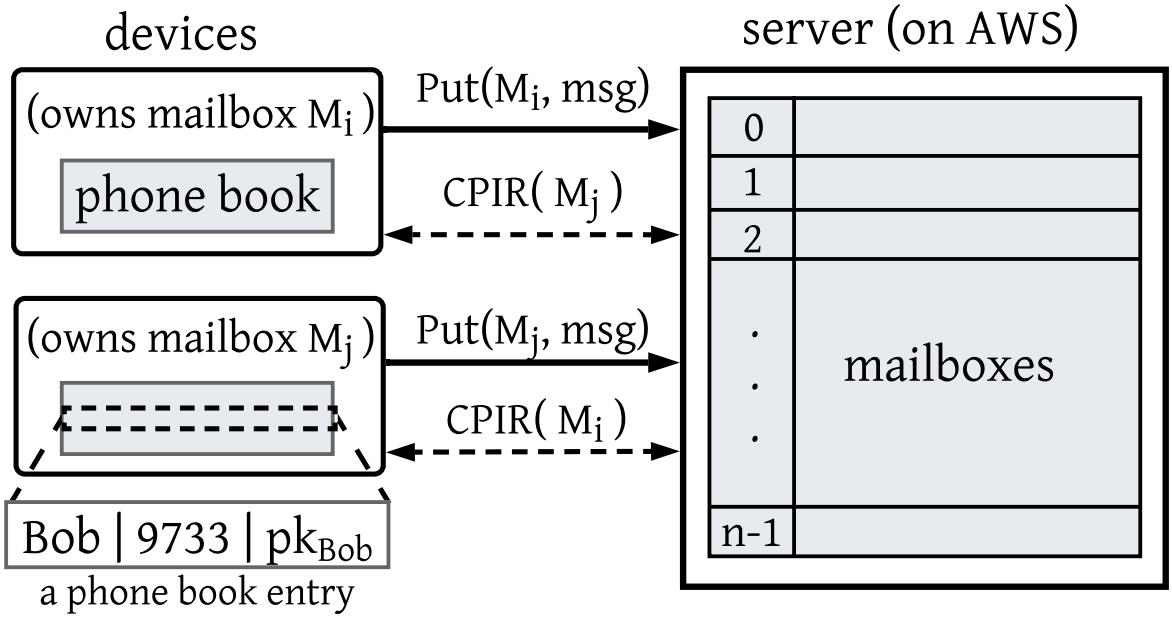


Figure 2.1: High-level architecture of Addra. The server runs over untrusted infrastructure and exposes mailboxes that user devices read from or write into. The mailbox identifiers (integers 0 to $n-1$) play the role of "phone numbers". A device stores phone numbers of the device owner's contacts in a local phone book. CPIR refers to the private information retrieval cryptographic primitive (§2.3.2).

The server exposes *mailboxes*. Specifically, it exposes n mailboxes, where n is the number of user devices using the system. Each mailbox can store one message and it has an ID, which is a number between (and inclusive of) 0 and $n-1$. As we will describe later (§2.4), it is helpful to view the n mailboxes as a matrix with n rows and m columns, where each row is an individual mailbox, and the m pieces of a message are m elements of a matrix row.

The user devices run logic to enable users to initiate, pick up, and participate in calls. Each device gets assigned a mailbox ID, which acts as its phone number. Each device also contains a *phone book*, which stores information on device owner's contacts. Each phone book entry is a tuple of a phone number of the contact, a cryptographic public key

belonging to the contact, and other standard information such as the contact's name, work place, and photograph. We assume that a device owner either knows this information or can obtain it privately through out-of-band means such as in-person meetings or personal websites.

2.3.2 Protocol

Addra relies on a cryptographic protocol called *private information retrieval* or *PIR* [44, 126]. We begin with a short background on PIR; Section 2.4 describes a new PIR scheme.

A primer on PIR. A PIR protocol [44, 126] runs between a user device and the server in Addra, where the device is interested in retrieving the message in the idx-th mailbox at the server without revealing the value of idx.

A PIR protocol has three procedures: Query, Answer, and Decode. Query is run by a device. It takes as input the index idx between 0 and $n-1$ and returns a query, q. Typically, q is an encryption of a suitable encoding of idx. Answer is run by the server; it takes as input the query q and the set of n mailboxes, and returns an encoding of the message in the idx-th mailbox (without learning the value of idx). Finally, Decode is run by the device; it takes the output of Answer and returns the idx-th mailbox message.

Addra's protocol. User devices in Addra participate in a round-based protocol consisting of a one-time registration step followed by synchronous rounds, each consisting of a dialing phase followed by a communication phase consisting of multiple subrounds of communication. In more detail, initially a device performs a one-time *registration step* to register itself with the server and obtain its phone number (mailbox ID). This is followed by a sequence of rounds. Each round starts with a *dialing phase*, where the device initiates a call to another device or picks an incoming call. The dialing phase is followed

```
1: function RECV (key key, resp resp)
2:     //resp is the output of ANSWER PIR procedure
3:     c ← DECODE(resp)                                  // DECODE is a PIR procedure
4:     msg ← AES.DEC(key, c)
5:     play msg to user
6: function SEND (mailbox M, token t, message msg, key key)
7:     c ← AES.ENC(key, msg)
8:     send (M, t, c) to server
9: function MAIN ( )
10:    //Register device and obtain a mailbox ID and unique token
11:    (M_self, tkn, n) ← REGISTERDEVICE()
12:    While True
13:        //Run dialing phase. k_enc is for encrypting content
14:        (M_peer, k_enc) ← DIAL/PICKUP()
15:        q ← QUERY(M_peer, n)                          // QUERY is a PIR procedure
16:        send q to server
17:        //Asynchronously listen for server responses
18:        register callback RECV(k_enc, ...) for server responses
19:        //Run communication phase consisting of t subrounds
20:        for r = 0 to t − 1
21:            wait for message generation interval
22:            call SENDM_self, tkn, msg_r, k_enc
```

Figure 2.2: Pseudocode for a user device in Addra. n is the number of mailboxes at the server. QUERY, ANSWER, DECODE are procedures of a PIR scheme (§2.3.2, §2.4).

by the *communication phase*, consisting of multiple subrounds, where each device sends exactly one message to the server and receives one message from the server. Notably, a device always writes a message to its assigned mailbox, while it receives a message from its peer's mailbox.

We now describe Addra's protocol in more detail. Figure 2.2 shows the pseudocode for a user device. A device starts executing the MAIN function (line 9 in Figure 2.2).

One-time registration step. When a user device joins Addra, it registers itself with the server and obtains three pieces of information: a mailbox ID, a unique authentication token tkn, and the number n of mailboxes (line 11 in Figure 2.2). As mentioned above, the mailbox ID acts as the phone number assigned to the device. Meanwhile, the

authentication token is a 128-bit uniformly generated string shared between the server and the device that enables the server to verify that a device is writing messages to its assigned mailbox (and not to a mailbox assigned to another device). One may use digital signatures instead of authentication tokens, but Addra prefers the *symmetric* tokens due to their better efficiency. The number of mailboxes n may increase if new devices join the system; when this happens, the server broadcasts an updated value of n.

Addra's server is untrusted and may assign mailbox IDs or authentication tokens incorrectly; for instance, it may reassign a previously assigned mailbox ID. Besides, it may distribute different values of n to different devices. The privacy guarantees of Addra's protocol do not depend on the server assigning correct values for these items. However, a malicious server can deny service to system participants, which is not prevented by our threat model (§2.2.2). A service provider who runs the server will likely be incentivized to provide a continuous service to keep its customer base.

Dialing phase. Once registered, a user device, who we refer to using its phone number, M_{self}, executes the round-based protocol. At the beginning of each round, M_{self} initiates a call or picks up an incoming call (line 14 in Figure 2.2). If the device initiates a call, it selects the phone number of the peer device it is calling, M_{peer}, and an encryption key, k_{enc}, to hide the content of the messages it will send. On the other hand, if the device picks up an incoming call then it learns the phone number of the caller and its content encryption key. For now, we leave out the details of how a device picks up a call till later (§2.5). After initiating or picking up a call, M_{self} generates a PIR query $q \leftarrow \textsc{Query}(M_{peer}, n)$ for the peer's mailbox, and sends q to the server (lines 15 and 16 in Figure 2.2). The PIR query indicates, without revealing the value of M_{peer}, that M_{self} is interested in receiving messages deposited into M_{peer}'s mailbox. The device M_{self} then registers an asynchronous callback to process PIR responses from the server (line 18 in

Figure 2.2). Meanwhile, the server stores the PIR queries from all devices and uses them across all subrounds of the round's communication phase.

Communication phase. In each subround of the communication phase, (1) a device deposits an encrypted message into its assigned mailbox at the server, (2) the server processes PIR queries from all devices and pushes the results to devices who registered these queries, and (3) each device decodes its PIR response from the server. In more detail, at the beginning of a subround, a device encrypts the message it wants to send to its peer with the key k_{enc} to create a ciphertext c. It sends the tuple (M_{self}, tkn, c) to the server (line 22 in Figure 2.2), where tkn is the device's assigned authentication token obtained during the registration step. The server uses the token to validate that the messages being written to mailboxes indeed come from devices that own the mailboxes. After performing these checks, the server runs the Answer PIR procedure for all PIR queries. That is, for a query q sent by a device during the dialing phase, the server runs $resp \leftarrow \textsc{Answer}(\text{mailboxes}, q)$ and pushes the PIR response $resp$ to the device. Finally, on receiving a response, a device invokes the callback it registered during the dialing phase (line 1 in Figure 2.2). This callback decodes the PIR response using the Decode PIR procedure, decrypts the underlying message sent by the device's peer M_{peer}, and delivers the message to the user.

Dummy participation and messages. The protocol described so far does not address the case when a device owner does not participate in a call. During such idle periods, like prior systems for strong metadata privacy (e.g., [133, 16]), a device adds *cover traffic* (also called chaff). In particular, if a device does not initiate or pick up a call in a round's dialing phase, it calls itself: inputs M_{self} into Query (line 15 in Figure 2.2). Besides, if a device does not have a message to send during a subround, it writes an

encryption of a random message into its mailbox. Sending cover traffic is necessary, as otherwise an adversary can learn connections between users by monitoring if they join and leave at similar times.

Security analysis. Addra's protocol satisfies relationship unobservability, meaning that an adversary cannot detect the existence of relationships between system users (§2.2.1). We provide a rigorous proof in Appendix A.1. Briefly, Addra's protocol meets the property because the protocol a user device executes is independent of whom the user is communicating with or the behavior of the (malicious) server. First, a user device encrypts messages using a content encryption key known only to its peer. Further, it always writes outgoing messages at fixed intervals to its own mailbox—independent of whether the device is engaged in a call, or the identity of its peer, or the behavior of the server who may or may not deliver incoming messages to the device, or who may replay messages. Second, the security property of PIR ensures that an adversary cannot tell the IDs of the mailboxes from which devices are retrieving messages. Again, the server may process PIR queries incorrectly, or broadcast an incorrect value n for the number of mailboxes, but a user device always registers a PIR query for one of the n mailboxes, no matter the value of n. Thus, the adversary cannot detect whether a user Alice is communicating with Bob or Charlie or someone else, or even communicating at all (i.e., retrieving messages from its own mailbox).

Performance characteristics. Addra's protocol exhibits two key characteristics that set it on the path to meeting its performance goals (§2.2.1). First, the protocol pushes messages from senders to recipients in two hops—independent of the number of users in the system. Specifically, in each subround, a sender pushes a message to the server, who then processes the PIR query provided beforehand by the recipient, and pushes the

PIR response to the recipient. This two-hop communication pattern is crucial for voice calls which require low latency. Second, the protocol amortizes the cost of generating and transferring a PIR query across subrounds of a round (our prototype runs a round every five minutes, and a subround every 480 ms; §2.6). Thus, the server does not have to deal with PIR query management (and certain preprocessing of query) during the time-sensitive subrounds. Nevertheless, the server must complete computing Answer for all PIR queries in a time smaller than the voice packet generation interval (that is, the duration of a subround) to avoid packet build up. Besides, the network transfers from the server to the devices are dictated by the size of the output of Answer. Thus, a low cost of the Answer PIR procedure is key for Addra's performance.

2.4 FastPIR: A new CPIR scheme

As described above (§2.3.2), a critical component of Addra's protocol is the Answer PIR procedure. It not only dictates Addra's message latency but also the resource consumption (both CPU and network) imposed by Addra.

PIR schemes are of two types: computational PIR (CPIR) [126] and information-theoretic PIR (IT-PIR) [44, 46]. CPIR schemes assume a single (untrusted) server and rely only on cryptographic assumptions; in contrast, IT-PIR schemes are more efficient but require two or more non-colluding servers. In Addra, we use a CPIR scheme as its trust assumptions are in line with Addra's goal of not trusting the communication infrastructure (§2.2.2).

One can plug in an existing CPIR scheme, either XPIR [3] or SealPIR [13], which are the state-of-the-art CPIR schemes, into Addra's protocol (§2.3.2). However, these schemes exhibit a tension between the CPU time to run Answer (and thus the wall-clock time for Answer) and the output size of Answer (and thus the network overhead).

Suppose a CPIR client wants to privately retrieve the idx-th message from a library L of n messages (mailboxes) held at a server. In prior work, a typical way to construct a CPIR query is to treat the library as a matrix with n rows and generate a ciphertext for every row of L.[1] The ciphertext for the idx-th row encrypts the value 1, and the ciphertexts for the other rows encrypt 0. However, this strategy creates large queries with a number of ciphertexts that is proportional to the value of n (e.g., XPIR's query size is ≈33 MiB for n=2^{15}, and ≈1 GiB for n=2^{20}; §2.6.5). When the server processes larger queries, it consumes more memory and CPU cycles to read them into CPU caches, which slows down query processing. (SealPIR compresses the query while transferring it on the network, but expands it to the larger query at the server).

A popular technique due to Stern [198] to address the query-size issue is called *recursion*. This technique is parameterized by a depth parameter d. A value of $d = 2$ or higher shrinks the query—it contains $d \cdot \sqrt[d]{n}$ ciphertexts instead of n—by rearranging the library as a d-dimensional hypercube. However, this rearrangement increases the CPIR Answer output size exponentially with d. Thus, if we plug in existing CPIR schemes (XPIR or SealPIR), then Addra would compromise on either server CPU time or network bandwidth.

Our CPIR scheme, FastPIR, works without recursion and thus keeps the smaller CPIR answer size. However, it optimizes the computation time for Answer. In fact, FastPIR takes less time than both XPIR and SealPIR (with or without recursion) to run Answer, particularly when the number of messages n in the library is greater than a threshold (≈20K; §2.6.5), thereby improving the scalability and message latency of Addra. FastPIR may be a good fit for other applications of CPIR where costs are dominated by those of the CPIR Answer procedure.

[1] A technique called aggregation [3, 17] further combines multiple rows (messages) into wider rows, resulting in a matrix with n/a rows, where the value a depends on the size of each message.

FastPIR, like SealPIR [13], builds on the lattice-based homomorphic encryption scheme of Brakerski/Fan-Vercauteren (BFV) [33, 75]. BFV offers superior efficiency than a traditional number-theoretic homomorphic encryption scheme such as Paillier [163], resists attacks by quantum computers, is implemented in mature and actively maintained codebases [164, 187], and is in the preliminary stages of being standardized (e.g., with ISO/IEC) [10]. We start with a necessary background on BFV (§2.4.1), and then delve into the details of FastPIR (§2.4.2–§2.4.4).

2.4.1 Background: The BFV cryptosystem

We focus here on describing the more efficient vectorized variant of BFV in which a single homomorphic operation operates over multiple plaintext inputs (single instruction, multiple data or SIMD; also called batching in the literature).

In this BFV variant, a plaintext is a vector of dimension N, where the parameter N equals a power of two and is at least 2^{10} for the security of the BFV scheme [10]. Each component of the plaintext is an integer in $\mathbb{Z}_p = \{0, \ldots, p-1\}$, the set of integers modulo p. Sometimes, we will view a BFV plaintext as a matrix with two rows and $N/2$ columns rather than a vector with dimension N.

A BFV ciphertext is also a vector but of dimension $2 \cdot N$. Each of its component is an element of $\mathbb{Z}_q$, where $q \gg p$.

The BFV encryption procedure, BFV.Enc, adds noise when it converts a plaintext vector into a ciphertext vector. This noise grows as homomorphic operations are performed on the ciphertext. If the noise grows beyond a threshold, then the ciphertext decryption procedure BFV.Dec does not produce the correct plaintext. Hence, $q \gg p$ for enough noise budget.

The size of the plaintext vector, N, the size of the domain of each component of

the plaintext, p, and the size of the domain of each component of the ciphertext, q, are all tunable parameters. Typically, one picks a combination of p, q, N depending on the application, the required noise budget, and the desired security level; we discuss concrete values for these parameters for Addra in §2.5.

BFV supports the following homomorphic operations that are used in FastPIR:

- **BFV.Add**(c_0, c_1) takes as input encryptions c_0 and c_1 of plaintext vectors v_0 and v_1, and outputs an encryption of $v_0 + v_1$ (component-wise vector addition).
- **BFV.ScMult**(v_0, c_1) takes as input a plaintext vector v_0 and an encryption c_1 of a plaintext vector v_1, and produces an encryption of the product $v_0 \odot v_1$, where the operator $\odot$ denotes component-wise multiplication.
- **BFV.RowRotate**(c_0, i) takes as input an encryption c_0 of a plaintext v_0 and an integer $0 < i < N/2 - 1$, and produces an encryption of v_0 rotated right by i positions cyclically row-wise. For instance, if plaintext dimension is $N = 8$ and v_0 is $((a, b, c, d), (e, f, g, h))$ in its matrix representation, then a right rotation by $i = 1$ produces an encryption of $((d, a, b, c), (h, e, f, g))$.
- **BFV.ColRotate**(c_0) takes as input an encryption c_0 of a plaintext v_0 and returns an encryption of a plaintext produced by swapping the two rows of v_0.

  For the example above, the result is an encryption of $((e, f, g, h), (a, b, c, d))$.

The BFV homomorphic operations require public keys generated by a key generation procedure. In particular, the rotation procedures require a set of rotation keys. While BFV.ColRotate requires one key, the size of the set of keys for BFV.RowRotate can vary. On the one extreme, this set can be configured to contain one key that rotates the plaintext vector by one position. Thus, to perform a rotation by $i > 1$ positions, BFV.RowRotate calls itself i times, incurring i times the cost of one BFV.RowRotate operation. On the other extreme, the set can contain $N/2 - 1$ keys for all possible

values of i between 0 and $N/2$. This extreme reduces CPU time for BFV.RowRotate as it does not call itself recursively, but this configuration increases the key size. For the BFV parameters we choose (§2.5), each rotation key is 128 KiB, and the set of all possible rotation keys is 256 MiB. Thus, in practice, one generates $\log_2(N/2)$ keys for all powers-of-two between 0 and $N/2-1$, and each invocation of BFV.RowRotate calls itself recursively up to $\log_2(N/2)$ times.

2.4.2 The FastPIR scheme

Recall the CPIR scenario (§2.4): a server holds a library L of n messages where each message has m components, while a client holds an integer $0 \leq idx \leq n-1$ and wants to retrieve the idx-th library message without revealing idx to the server.

To build intuition for FastPIR, suppose that L is an $N \times 1$ matrix consisting of N unit length messages, where N is the plaintext vector dimension in BFV. Then, the client constructs the CPIR query q for the idx-th message by encrypting a BFV plaintext whose idx-th entry is one and the rest are zeros (this is called *one-hot encoding* of idx). For instance, if $N = 4$ and $idx = 1$, the client encrypts the BFV plaintext $(0, 1, 0, 0)$. The server multiplies this encryption q with L by computing BFV.ScMult(L, q) to obtain an encryption of the idx-th entry of L. For the example above, if L is (a_0, a_1, a_2, a_3), BFV.ScMult produces an encryption of $(0, a_1, 0, 0)$ as the multiplication is component-wise. The client receives the output and decrypts it to get a_1.

The advantage of this strategy is that a query consumes only a component of a ciphertext for each of the n rows of L (instead of a ciphertext per row). However, a challenge is that this strategy generates one output ciphertext for each of the m columns of L. FastPIR addresses this challenge by combining ciphertexts for m columns into a single ciphertext using the BFV rotation operations (BFV.RowRotate and BFV.ColRotate),

1: **function** QUERY(index idx, n)
2: //Create a one-hot encoding of idx
3: **for** $i = 0$ to $n - 1$
4: $f_i \leftarrow (i == idx)\ ?\ 1 : 0$
5: //Split and encrypt the one-hot vector
6: **for** $i = 0$ to $(n/N) - 1$ // N is BFV plaintext dimension
7: $q_i = \text{BFV.ENC}(pk, (f_{i \cdot N}, \ldots, f_{(i+1) \cdot N - 1}))$
8: **return** $q = (q_0, \ldots, q_{(n/N)-1})$

Figure 2.3: QUERY procedure for a basic version of FastPIR. pk is a public key for the BFV scheme (§2.4.1).

thereby reducing CPIR answer sizes.

Before describing the details of rotation, we remark that the use of vectorized operations (SIMD capabilities of BFV) is common. In fact, both XPIR and SealPIR use vectorized operations. The difference is that these prior CPIR schemes apply vectorization across columns of the matrix while FastPIR applies it across rows of the matrix, which is a more efficient use of vectorization in the PIR context (§2.6.5).

Details. Figures 2.3, 2.4, and 2.5 show the different functions of FastPIR scheme. They assume that n is a multiple of N, i.e., $n = k \cdot N$ for some $k \geq 1$, and $m \leq N$. If these constraints do not hold, then the server pads L with empty rows and splits L into sets of N columns.

The QUERY (Figure 2.3) procedure and the top half of ANSWER (Figure 2.4 until line 9) follow the intuition described above. That is, QUERY creates a one-hot encoding of idx (line 4 in Figure 2.3), splits the encoding into multiple BFV plaintexts, and encrypts each plaintext separately (line 7 in Figure 2.3). The top half of ANSWER multiplies the $k = n/N$ plaintext column vectors of each column of L with the corresponding ciphertexts in the query (line 8 in Figure 2.4), and adds the k output ciphertexts to get one ciphertext per column of L (line 9 in Figure 2.4). For instance, if $n = 8$, $N = 4$, $idx = 1$, and a column of L is $(a_0, a_1, \ldots, a_7)$, then ANSWER computes encryptions of $(0, a1, 0, 0)$ and

```
1: function Answer(library L, query q = (q_0, ..., q_{(n/N)-1}))
2:     //Represent L as a matrix of elements in Z_p: L ∈ Z_p^{n×m}
3:     //q is an output of Query
4:     for j = 0 to m − 1
5:         sum_j = BFV.Enc(pk, 0)
6:         for i = 0 to (n/N) − 1
7:             p_{i,j} ← SubMat(L, i · N, (i + 1) · N − 1, j, j)
8:             t_{i,j} = BFV.ScMult(p_{i,j}, q_i)
9:             sum_j = BFV.Add(sum_j, t_{i,j})
10:    //Combine outputs from all columns
11:    Initialize s_top, s_bot to encryptions of zero vectors
12:    for j = 0 to m − 1
13:        if j < N/2
14:            sum_j ← BFV.RowRotate(sum_j, j)
15:            s_top ← BFV.Add(s_top, sum_j)
16:        else
17:            sum_j ← BFV.RowRotate(sum_j, j − N/2)
18:            s_bot ← BFV.Add(s_bot, sum_j)
19:    return BFV.Add(s_top, BFV.ColRotate(s_bot))
```

Figure 2.4: Answer procedure for a basic version of FastPIR. pk is a public key for the BFV scheme (§2.4.1) SubMat extracts a sub-matrix of a matrix.

$(0, 0, 0, 0)$ in line 8 of Figure 2.4, and adds them to get an encryption of $(0, a1, 0, 0)$ in line 9 of Figure 2.4.

The bottom half of Answer packs together outputs from each column into a single ciphertext (lines 11–19 in Figure 2.4). Suppose the number of columns is $m = 4$ and the outputs corresponding to them are encryptions of $(0, a_1, 0, 0)$, $(0, b_1, 0, 0)$, $(0, c_1, 0, 0)$, and $(0, d_1, 0, 0)$, or equivalently encryptions of $((0, a_1), (0, 0))$, $((0, b_1), (0, 0))$, $((0, c_1), (0, 0))$, and $((0, d_1), (0, 0))$, when the underlying plaintexts are viewed in their matrix form. Then, Answer uses the BFV.RowRotate and BFV.Add operations to produce encryptions of $((b_1, a_1), (0, 0))$ and $((d_1, c_1), 0, 0))$ (lines 12– 18 in Figure 2.4), before column rotating the second ciphertext, and adding the result to the first ciphertext to obtain an encryption of $((b_1, a_1), (d_1, c_1))$ (line 19 in Figure 2.4). Using rotations to pack outputs from multiple columns into a single ciphertext is crucial as otherwise a CPIR answer size can contain

```
1: function DECODE(answer ans, index idx)
2:     //ans is an output of ANSWER
3:      ans_pt ← BFV.DEC(sk, ans)
4:     if idx mod N > N/2
5:         ans_pt ← PTCOLROTATE(ans_pt)
6:     return ans_pt ← PTROWROT(ans_pt, N/2 − (idx mod N/2))
```

Figure 2.5: DECODE procedure for a basic version of FastPIR. sk is a private key for the BFV scheme (§2.4.1). PTROWROT and PTCOLROTATE are like BFV.ROWROTATE and BFV.COLROTATE, respectively except they operate on BFV plaintexts rather than BFV ciphertexts.

multiple ciphertexts (instead of one).

DECODE (Figure 2.5) is straightforward; it decrypts the output of ANSWER and then rotates the plaintext depending on the value of the requested index. For the example above, DECODE first obtains the plaintext matrix $((b_1, a_1), (d_1, c_1))$, and then performs a rotation on this matrix by $idx = 1$ to obtain $((a_1, b_1), (c_1, d_1))$.

2.4.3 Reducing the cpu cost of rotations

Recall that one goal of FastPIR is to optimize the CPU time of ANSWER procedure (§2.3.2). A source of inefficiency in what is described above is the cost of BFV.ROWROTATE (lines 14 and 17 in Figure 2.4), as the CPU time taken by it depends on the value of i—the positions by which the underlying plaintext is rotated. When i is a power of two, then BFV.ROWROTATE is fast, whereas when i is a not a power of two, BFV.ROWROTATE calls itself up to $\log_2(i+1)$ times (§2.4.1). For example, a call to BFV.ROWROTATE with an input $i = 7$ translates into three rotations by amounts one, two, and four—powers of two that add to seven.

FastPIR eliminates the calls to expensive rotations whose input rotation amount is not a power of two. As intuition, suppose that the ANSWER procedure (Figure 2.4) needs to make two calls to BFV.ROWROTATE—one for rotating a vector by two positions and

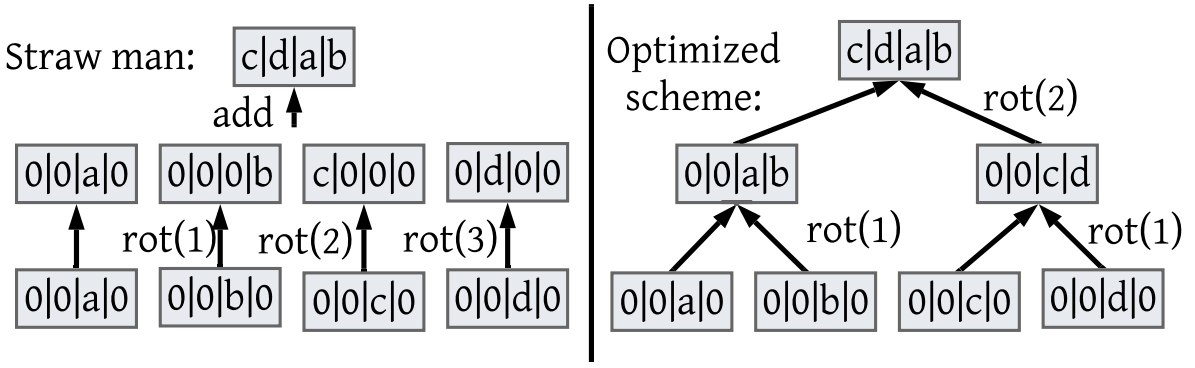


Figure 2.6: Illustration of optimized rotations in FastPIR. The straw man (left) performs a mix of slow rotations (with rotations amounts that are not powers of two) and fast rotations (with rotation amounts that are powers of two) to combine multiple vectors. FastPIR's optimized scheme combines vectors using fast rotations only.

the other for rotating a vector for another matrix column by three positions. Then, the straw man design presented in the previous subsection treats each rotation separately. Particularly, it breaks down the rotation by three positions into a rotation by one position followed by a rotation by two positions. Instead, FastPIR first rotates the second vector by one position and adds the result to the first vector. Then, it rotates the combined vector once by two positions, thereby rotating only by powers-of-two amounts.

Figure 2.6 illustrates the idea for our running example with $m = 4$ matrix columns, where the FastPIR processing for each column produces a ciphertext. FastPIR arranges the vectors to be combined as leaf nodes of a tree; it then builds up to the root of the tree. When producing a parent at a given height h of the tree, FastPIR rotates the right child by 2^{h-1} positions and adds the rotated vector to the left child. The effect is that FastPIR combines m ciphertexts in lines 14 and 17 in Figure 2.4 using m fast rotations.

2.4.4 Reducing the number of rotations

This optimization reduces the number of calls to BFV.RowRotate by a factor of two, and eliminates the call to BFV.ColRotate, thereby further reducing the CPU cost

of Answer. The trade-off is a 2× increase in CPIR query size. The key idea is to exploit the matrix representation of a BFV plaintext (§2.4.1) and retrieve two elements of a matrix row (instead of one) at a time.

As motivation, suppose that the matrix L is of dimension $N/2 \times 2$, and the client wants the idx-th row. Then, the client sends an encryption of a vector whose idx-th and $idx + N/2$-th entries are one (and the rest are zeros). For instance, if $N = 4$ and $idx = 1$, then the client sends an encryption of $(0, 1, 0, 1)$, or equivalently, $((0, 1), (0, 1))$. The server multiplies this query with L to get an encryption of a vector whose idx-th and $idx + N/2$-th entries are the desired elements from the two columns of L. As an example with $idx = 1$, say L is $((a_0, a_1), (b_0, b_1))$, then the multiplication operation produces an encryption of $((0, a_1), (0, b_1))$.

Figure 2.4.4 shows the procedures of FastPIR with this optimization. The procedures assume that n is a multiple of $N/2$, i.e., $n = k \cdot (N/2)$ for some $k \geq 1$, and m is even and $\leq N$. As before (§2.4.2), if these constraints do not hold, then the server appropriately pads and splits L.

The Query procedure encrypts a set of vectors that in total contain two non-zero entries (line 6 in Figure 2.4.4). The Answer procedure multiplies k parts of every pair of columns of L with the k ciphertexts in the query, and adds the results to get one ciphertext for every pair of columns. Then, Answer packs these outputs using the optimized scheme to combine ciphertexts described previously (§2.4.3). The Decode procedure decrypts the output of Answer and performs a rotation on the plaintext output.

Security analysis. The security of a CPIR scheme requires the output of Query to not reveal any information about the requested index [126, 46]. FastPIR meets this property because its Query procedure (i) produces semantically-secure BFV ciphertexts, and (ii) outputs $n/(N/2)$ ciphertexts independent of the value of the desired index idx.

function QUERY(index idx, n)
  for $i = 0$ to $n - 1$
    $f_i \leftarrow (i == idx)\ ?\ 1 : 0$ // one-hot encoding
  for $i = 0$ to $n/(N/2) - 1$
    $v \leftarrow (f_{i \cdot N/2}, \ldots, f_{(i+1) \cdot N/2 - 1})$
    $q_i = \text{BFV.ENC}(pk, v||v)$ // || denotes concatenation
  return $q = (q_0, \ldots, q_{n/(N/2)-1})$
function ANSWER(library L, query $q = (q_0, \ldots, q_{n/(N/2)-1})$)
  //Represent L as a matrix of elements in $\mathbb{Z}_p$: $L \in \mathbb{Z}_p^{n \times m}$
  //q is an output of QUERY
  for $j = 0$ to $(m/2) - 1$
    $sum_j = \text{BFV.ENC}(pk, 0)$
    for $i = 0$ to $n/(N/2) - 1$
      $p_{i,j} \leftarrow \text{SUBMAT}(L, i \cdot N/2, (i+1) \cdot N/2 - 1, 2j, 2j+1)$
      $t_{i,j} = \text{BFV.SCMULT}(p_{i,j}, q_i)$
      $sum_j = \text{BFV.ADD}(sum_j, t_{i,j})$
  //Combine outputs from all pairs of columns
  return ROTATEANDCOMBINE($sum_0, \ldots, sum_{m/2-1}$)
function DECODE(answer ans, index idx)
  //ans is an output of ANSWER
  $ans_{pt} \leftarrow \text{BFV.DEC}(sk, ans)$
  return PTROWROT($ans_{pt}, N/2 - (idx \bmod N/2)$)

Figure 2.7: QUERY, ANSWER, and DECODE procedures for FastPIR. (pk, sk) is a (public, private) key pair for the BFV scheme (§2.4.1). SUBMAT extracts a sub-matrix of a matrix. ROTATEANDCOMBINE refers to the optimized procedure to combine ciphertexts (§2.4.3). PTROWROT is like BFV.ROWROTATE except that it operates on BFV plaintexts rather than BFV ciphertexts.

2.5 Implementation details

FastPIR. Our prototype of FastPIR is ≈1000 lines of C++ and is available at https://github.com/ishtiyaque/FastPIR. We used the Microsoft SEAL library v3.5 [187] for the underlying cryptographic operations of the BFV scheme. Recall that FastPIR configures BFV so that it supports vectorized operations (§2.4.1). For vectorization, the plaintext modulus p has to be a prime number congruent to 1 (mod $2N$), where N is the vector dimension of a BFV plaintext and equals 2^{10} or a higher power of two (§2.4.1). Moreover, one needs to choose $p \ll q$ to ensure correct decryption. For Addra, we choose

$N = 2^{12}$, p a 19-bit prime 270337, and q a 109-bit composite that is the product of a 54-bit prime (18014398509309953) and a 55-bit prime (36028797018652673). These parameters provide a 128-bit security level as guided by the homomorphic encryption standard [10]. (One may choose different parameters for FastPIR based on application requirements.)

Master-worker architecture for Addra. We implemented Addra server using a master-worker architecture with many worker machines to distribute the PIR workload. Specifically, during the dialing phase of a round in Addra's protocol (§2.3.2), the master receives CPIR queries from all devices and shards them across the workers, where a worker gets a subset of the queries. Then, during the communication phase, the master initiates each subround at a fixed schedule. During each subround, it waits to receive messages from the clients, compiles them into a message library, and broadcasts the entire message library to the workers. In case a laggard client fails to get its message to the master during the time period the master waits for incoming messages, the master buffers the laggard's message for the next subround. If more than one message arrives at the master from a client for the same subround, the master retains the latest message. Meanwhile, to process CPIR queries, each worker computes the output of Answer on its assigned subset of the queries and pushes the outputs to the client devices who registered the queries.

Dialing protocol. Addra uses Pung's protocol to initiate calls [17, Chapter 4.5.3] (which in turn is based on Alpenhorn [134]). Briefly, a caller sends "hello" messages encrypted with the callee's public key to the server, who then broadcasts the set of "hello" messages from all callers to all user devices. A callee decrypts the ciphertexts using its private key and learns the content encryption key and the caller's phone number

(which are inside the hello message). This protocol is not efficient as the server broadcasts the ciphertexts to the participants (although the server could use a CDN or multicast protocols), and a callee decrypts ciphertexts from all users. Thus, Addra runs this protocol infrequently (every five minutes; §2.6.3). A more efficient dialing protocol in Addra's threat model is still an open problem.

Options for which call to pick. A device may receive multiple incoming calls, or may make an outgoing call at the same time a call comes in. In such scenarios, Addra exposes all options to the device owner and lets them pick the call they want to participate in. However, depending on which option a user chooses, they could leak some information to the users who are on the other end in the non-chosen options. For instance, if Alice receives a call from both Bob and Charlie, and decides to pick Bob's call, then Charlie may infer that Alice is busy. This leakage is not specific to Addra but applies to any metadata-private system [15, 14]. As efficient solutions to this problem become available, one could enhance the options-based approach currently implemented in Addra.

Other libraries and lines of code. Our prototype of Addra (`https://github.com/ishtiyaque/Addra`) is ≈2,000 lines of C++ on top of existing libraries, including Fast-PIR. Our implementation of the dialing protocol uses the libscapi [141] library for public-key encryption using the Cramer-Shoup scheme [54] with a key size of 3072 bits which provides 128 bits of security. It also uses AES-CBC implementation from OpenSSL with a 128-bit key for end-to-end content encryption with 128 bits of security. It implements the message library broadcasting mechanism from master to workers using rpclib [178]. Finally, we use the open source implementation of LPCNet [151] for speech encoding/decoding.

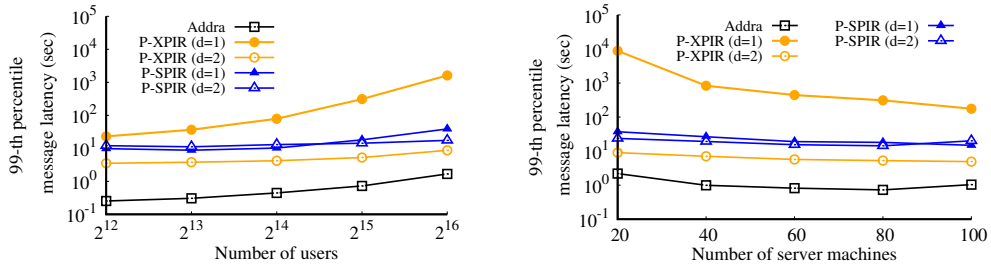


Figure 2.8: (Left) Message latency with a varying number of users for eighty server worker machines. (Right) Message latency with a varying number of server worker machines for 32,768 users. Messages are 96 bytes in size. The y-axis is log-scaled. d denotes CPIR recursion depth, where $d = 1$ denotes no recursion and $d = 2$ enables recursion. Addra does not use recursion (§2.4).

2.6 Evaluation

Our evaluation answers the following questions:

1. What is Addra's message latency, and how does it vary with the number of users and server machines?

2. How much resource overhead (CPU, network upload and download) does Addra impose on its server and users?

3. How does Addra compare to Pung [16, 13, 17], which is the state-of-the-art prior system for metadata-private communication over completely untrusted infrastructure?

4. How does FastPIR compare to the state-of-the-art CPIR schemes, XPIR [3] and SealPIR [13]?

A highlight of our evaluation results is as follows:

- Addra's 99-th percentile message latency is 726 ms for 32,768 users and 80 server machines. For the same configuration, Pung's message latency is 5.2 seconds.

- Addra's server consumes 22.3 minutes of CPU time for a subround with 32,768 users, where a subround corresponds to 480 ms of voice call. Translated to provisioning burden, each user requires the server to provision 0.085 CPU for its call. In contrast, Pung consumes 77.1 minutes of CPU time (3.45× higher) per subround.
- An Addra user downloads and uploads 55.1 and 1.08 MiB of data for each round when 32,768 users use Addra, where a round corresponds to five minutes of voice call. Thus, translated into bandwidth, Addra requires a download and upload bandwidth of 1.46 Mbps and 30 Kbps, respectively. In contrast, a Pung client downloads and uploads 250 MiB (4.6× higher) and 313 MiB (289× higher) for five minutes of voice call data.
- FastPIR has a smaller server-side CPU time *and* a smaller response size relative to XPIR and SealPIR, particularly when the number of messages in the PIR library is greater than a threshold ($\approx$ 20K).

Setup and method. We compare Addra to two variants of Pung: Pung-XPIR (P-XPIR) and Pung-SealPIR (P-SPIR). The former is the original Pung system from OSDI 2016 [16] that instantiates CPIR with the XPIR scheme [3]. The second variant replaces the XPIR scheme with the SealPIR CPIR scheme [13]. We include both variants as there is no clear winner between them across all performance metrics. Further, we evaluate these variants without ($d = 1$) and with CPIR recursion ($d = 2$). We do not experiment with a recursion depth $d > 2$ as the server CPU time and the network transfers from the server to the clients, which are the two key overhead metrics, grow significantly with depth greater than two [13].

We configure Addra and Pung to provide a security level of 128-bits. Also, we configure Pung to use its BST retrieval scheme in which a message recipient obliviously searches through a tree while retrieving one message from the Pung server. This scheme

is the most scalable retrieval scheme for Pung especially as the number of system users increase; we discuss other retrieval schemes Pung supports in the related work section (§2.7). For all of the systems, we deploy the server on a cluster of machines in AWS EC2 US East region (Ohio). Addra requires a master machine and a set of worker machines (§2.5). For the master, we use a machine of type `c5.24xlarge` (96 vCPU, 192 GiB of RAM and 25 Gbps of network bandwidth) which provides a high network bandwidth to enable the master to broadcast the message library (the mailboxes) to the workers. For the workers, we use the compute-optimized machines of type `c5.12xlarge` (48 vCPU, 96 GiB of RAM, and 12 Gbps of network bandwidth). Pung does not have a master and therefore we use machines of type `c5.12xlarge` as its workers. To compensate for the extra master machine assigned to Addra (relative to Pung), we assign two additional worker machines of type `c5.12xlarge` to Pung.

Addra is required to process queries from all clients in every subround to meet its security goals. Since we cannot run tens of thousands of clients in our infrastructure, we employ a combination of real and simulated clients. We deploy 256 geographically distant real clients in a machine of type `c5.24xlarge` in AWS US West (N. California). The mean network RTT, as measured by Ping, between the server and these clients is 51 ms. During each round and subround, real clients send their queries and messages to the server, and the server inserts the queries and messages of the remaining simulated clients.

We configure Addra to run a round every five minutes and a subround every 480 ms. This configuration results in a fixed message size of 96 bytes at each subround as the LPCNet voice codec encodes a 40 ms audio frame into 8 bytes (§2.2.1) [206, 205]. We vary the number of users (from 4,096 to 65,536) and the number of worker machines (from 20 to 100). We repeat experiments for 10 trials. To account for tail latency, we process the queries from real clients only after processing the queries from all simulated

clients. Then, we measure the 99-th percentile latency observed by the real clients over the 10 trials, the CPU time consumed by the server and the real clients, and the amount of data uploaded and downloaded by the real clients.

2.6.1 Message latency

Variation with the number of users. Figure 2.8 (left) shows the 99-th percentile message latency with a varying number of users when the server has 80 worker machines.

Addra's message latency is 254 ms for 4,096 users and increases to 1678 ms for 65,536 users. This increase is due to three reasons. First, as the number of users increases, so does the number of mailboxes and the time to broadcast their content from the master to the workers (§2.5). Second, the number of CPIR queries the server processes every subround equals the number of users (§2.3.2). Third, the time to process a CPIR query increases with the number of mailboxes, so each worker takes longer to generate CPIR responses. For 32,768 users, the latency is 726 ms, of which 398 ms is for CPIR query processing at the workers, 186 ms is for broadcast of mailbox content from the master to the workers, and the rest is for network transfers between the client and the server. However, for 65,536 users, the latency increases to 1,678 ms, of which 1,186 ms is for CPIR query processing alone. This processing time is higher than the 480 ms subround time budget and thus voice packets start queuing up at the server for these many users.

Addra's message latency is lower than Pung's, specifically, that of Pung-XPIR by a factor of 7.2× for 32,768 users, due to two reasons. First, a sender in Addra pushes a message to the server, who performs CPIR processing and pushes the response to the recipient—in total, the message traverses two hops (§2.3.2). In contrast, while the sender in Pung pushes a message to the server in one hop, a recipient has to make $\lceil \log_2(n+1) \rceil$ sequential round-trips to the server to fetch a message, where n is the number of users.

Second, Addra uses FastPIR, which has lower server-side CPIR answer generation time than XPIR or SealPIR used in Pung; we will expand on this difference shortly (§2.6.2, §2.6.5).

Variation with the number of worker machines. Figure 2.8 (right) shows the 99-th percentile message latency as a function of the number of worker machines when the number of users is fixed to 32,768. Latency decreases for all systems with an increase in the number of worker machines due to increased parallelization for CPIR answer generation, but only up to an inflection point. Beyond this inflection point, adding workers does not improve latency as the time to replicate mailboxes from the master to the workers goes up, while the CPU on the workers starts to become idle. Thus, an immediate scalability bottleneck in Addra is the time to broadcast mailboxes from the master to the workers. Distributing the master or reducing the number of workers by extracting more efficiency from each may further push out the inflection point.

2.6.2 Server-side cpu consumption

Figure 2.9 shows that server-side CPU time increases with the number of users. This is expected as both the number of CPIR queries and the time to generate an answer for each query increases with the number of users (§2.3.2). Addra's CPU consumption is lower than Pung's. For instance, for 32,768 users, Addra takes 22.3 minutes while Pung (with XPIR and CPIR recursion depth $d = 2$) takes 77.1 minutes (3.45× higher). If we convert these times to CPU provisioning requirements, then for each subround lasting 480 ms or 0.48 seconds, Addra's server consumes 22.3 minutes, or 1,338 seconds, of CPU, which is provided by provisioning $1,338/0.48 = 2788$ CPUs, or 0.085 CPU per user. Similarly, each Pung user requires 0.29 CPU per user. A key reason for this difference is that FastPIR in Addra consumes lower amount of server-side CPU relative to XPIR or

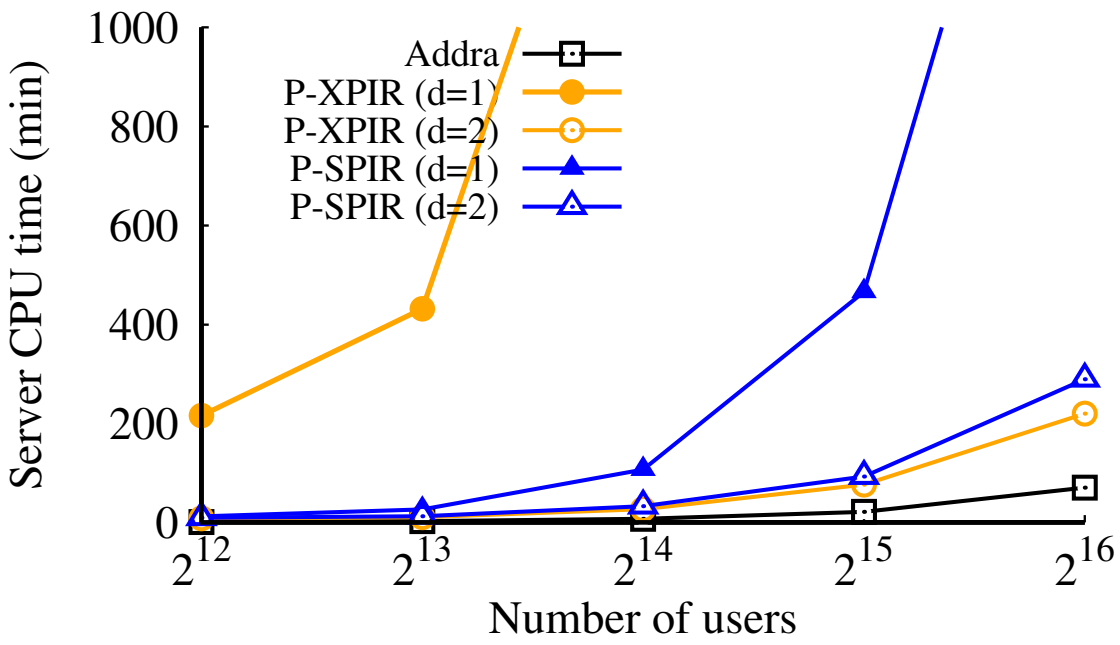


Figure 2.9: Server-side CPU time per subround with a varying number of users. A subround corresponds to 480 ms of voice call; in a subround, each user sends and receives one 96 byte message.

SealPIR in Pung (§2.6.5). We note that even though a recursion depth of $d = 2$ reduces CPU consumption relative to no recursion ($d = 1$), increasing depth further ($d = 3$) does not reduce CPU consumption [13]. Furthermore, a higher depth increases network overhead (§2.6.3). Thus, as mentioned earlier (§2.6), we restrict our experiments to a depth of $d = 2$.

2.6.3 Client-side resource overheads

Network transfers. Figure 2.10 shows the amount of data a client downloads and uploads for one round of communication (a round corresponds to five minutes of voice call).

An Addra user downloads ≈55.1 MiB in a five-minute round when 32,768 users use

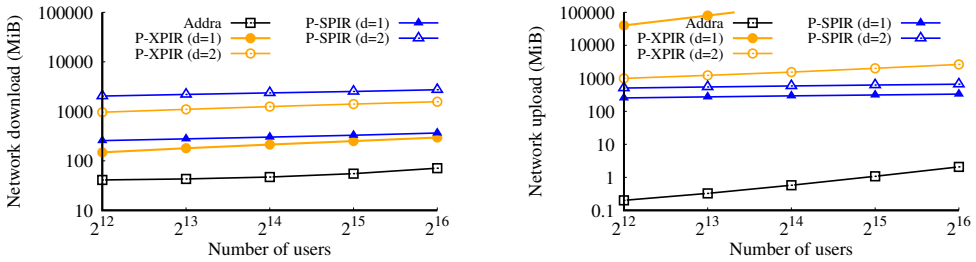


Figure 2.10: Data downloaded and uploaded by a user per round with varying number of users. A round corresponds to five minutes of voice call.

Addra. That is, each user requires 1.46 Mbps of network download bandwidth. Of the 55.1 MiB, ≈39 MiB is due to the communication phase of the round while the rest is due to the dialing phase (§2.3.2). Further, the former is independent of the number of system users, while the latter depends linearly on the number of users.

Relative to a non-private baseline which does not hide metadata, Addra's network overhead is significantly higher due to the use of CPIR, which encrypts messages into BFV ciphertexts. For example, if the non-private baseline uses LPCNet which encodes 480 ms of speech in 96 bytes of data, then a user's network download bandwidth will be 1.56 Kbps. In contrast, Addra encrypts the 96 bytes into a 64 KB ciphertext, which is a 682× increase.

However, relative to Pung, an Addra user downloads less data, by 4.5–45.7×, depending on the Pung variant. The improvement is due to two reasons. First, Pung requires a message recipient to make multiple CPIR queries with the server to search through the message library that is organized as a tree. Second, CPIR answer size increases with a higher CPIR recursion depth ($d = 2$ versus $d = 1$). Addra's FastPIR, on the other hand, operates at $d = 1$ to keep CPIR answer sizes, and thus the downloads, smaller (§2.4).

An Addra user uploads one CPIR query per round during its dialing phase. The Addra server then reuses the query across subrounds (§2.3.2). Even though the query size for Addra is larger compared to that of Pung (§2.6.5), unlike Pung, this cost is

amortized over multiple subrounds of the communication phase. As a result, Addra's upload network transfers are small: ≈1.1 MiB per round, or 30 Kbps.

We remark that even though Addra's instantaneous network overhead (1.46 Mbps download and 30 Kbps upload) appears manageable, it adds up over time due to the involvement of a client in dummy calls (§2.3.2). Thus, Addra requires certain conditions such as unlimited network downloads for its clients to be deployable. We anticipate that in the future, as the need for privacy increases, so will advances in network technology that will provide options for unlimited data.

cpu time. An Addra client consumes ≈27.5 seconds of CPU time per a five-minute round when the number of users is 32,768. 94% of this time is from the dialing protocol (§2.5). For the same configuration, a Pung client consumes 1.7–63× higher CPU, primarily due to multiple CPIR queries with the server for transmitting each message.

2.6.4 Discussion on voice quality

The quality of voice calls and user experience depends on several factors including message transmission latency, jitter (the inconsistencies among packet arrival intervals [123]), and the effectiveness of the voice encoder that converts human speech into a digital signal. This section briefly discusses Addra's performance on these metrics.

We reported Addra's message transmission latency in §2.6.1. Specifically, latency varies with the number of users, and is lower for a lower number of users. For example, for 8K users, the latency is 306 ms, which is below the ITU-G.114 recommended value of 400ms [119]. As the number of users increases, Addra's latency crosses the recommended value, but stays below one second for a significant number of users (32,768); this value of one second is critical as it is possible to make voice calls at this latency [133].

To measure jitter, we ran Addra for one round (i.e., 5 minutes of voice call) with 80

worker machines and a varying number of users. Ideally, a user should receive a voice packet every 480 ms, which is the duration of one subround. We measured the interval between consecutive packet arrival timestamps and calculated the absolute deviation of this value from 480 ms as jitter. Addra's mean jitter is 4.1 ms for 4,096 users and increases to 36.8 ms for 32,768 users. This increase with the number of users is correlated with higher CPU and network load at the server.

Finally, the effectiveness of voice encoding is a property of the encoder. Addra's current prototype uses the LPCNet [151] encoder developed by Mozilla. Conducting a user experience study on LPCNet's quality is outside the scope of this paper, but we refer the reader to the original paper on LPCNet that discusses a subjective assessment of LPCNet's quality based on an experiment with one hundred human listeners [205].

2.6.5 Comparison of CPIR schemes

A core component of Addra and Pung is the CPIR cryptographic primitive. Pung uses either XPIR or SealPIR, which are also the state-of-the-art schemes. Addra uses FastPIR (§2.4). This section compares the cost of these CPIR schemes in isolation. Besides, since CPIR applies to several other contexts [150, 93, 31, 63], this section sheds light on which scheme could be better for which application.

We microbenchmarked the XPIR, SealPIR, and FastPIR libraries on a single CPU of an AWS instance of type `c5.12xlarge` (48 vCPU, 3.6 GHz, 96 GiB RAM). We configured all three libraries for a 128-bit security level. However, XPIR does not set parameters from the homomorphic encryption standard [10], and its parameters are smaller relative to those for SealPIR and FastPIR.

We varied the number of messages in the library ($n \in \{2^{13}, \ldots, 2^{20}\}$) and the size of each message ($m \in \{96\text{B}, 256\text{B}, 1024\text{B}\}$). The lowest value of n captures a small

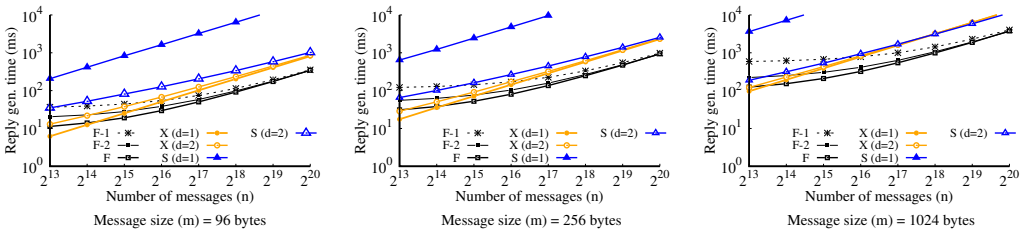


Figure 2.11: CPU time to run the ANSWER CPIR procedure for XPIR (X), SealPIR (S), and three variants of FastPIR (F) with a varying number of messages n in the server library and a varying size m of each message. F-1 and F-2 are intermediate baselines for FastPIR: F-1 leaves out both optimizations for the rotation operations (§2.4.3 and §2.4.4), while F-2 leaves out only the optimization in §2.4.4. Both axes are log-scaled. d denotes recursion depth (§2.4). FastPIR does not use recursion (sets $d = 1$, which means no recursion). For both XPIR and SealPIR, an optimization to aggregate multiple small messages into a larger one (called aggregation in the literature) is enabled.

library with a few thousand messages, while the other extreme of $n = 2^{20}$ demonstrates how FastPIR scales with the number of messages relative to the other CPIR libraries. Similarly, the different message sizes demonstrate performance for scenarios with small messages (for example, Addra) and also larger messages.

We measure and report both CPU and network overhead for query generation (QUERY), answer generation (ANSWER), and answer decode (DECODE) CPIR procedures, for 10 trials. Given that the CPIR cost in Addra is dominated by the cost to run the ANSWER procedure, we describe the results while focusing on ANSWER. At a high level, FastPIR keeps both the CPU cost for ANSWER and the size of ANSWER output small, while XPIR and SealPIR sacrifice one of the two.

cpu time for Answer. Figure 2.11 shows the CPU time for the ANSWER procedure for different values of n and m. Beyond a threshold n, and for all values of m, FastPIR consumes the least amount of CPU time for ANSWER independent of whether the baselines use recursion or not ($d = 1$ is no recursion, and $d = 2$ enables it). For instance, when n=2^{20} and m=256B, ANSWER in FastPIR takes 2.5× less time than XPIR ($d = 2$) and

	n = 32,768					n = 1,048,576				
	X($d=1$)	X($d=2$)	S($d=1$)	S($d=2$)	F($d=1$)	X($d=1$)	X($d=2$)	S($d=1$)	S($d=2$)	F($d=1$)
query size (KiB)										
m = 96 bytes	33,856	2,112	32	64	1,024	1,082,432	11,776	928	64	32,768
m = 256 bytes	95,328	3,520	96	64	1,024	3,050,432	19,776	2,752	64	32,768
m = 1024 bytes	524,288	8,192	512	64	1,024	16,777,216	46,368	16,384	64	32,768
answer size (KiB)										
$m \in \{96B, 256B, 1024B\}$	32	256	32	320	64	32	288	32	320	64
client cpu costs (ms)										
QUERY (m = 96B)	118.6	7.4	0.7	1.4	21.3	3801.8	41.5	19.2	1.4	679.0
QUERY (m = 256B)	335.2	12.4	2.0	1.4	21.4	10711.3	69.8	56.9	1.4	678.6
QUERY (m = 1024B)	1841.6	28.8	10.6	1.4	21.4	58990.8	164.2	338.8	1.4	678.7
DECODE ($m \in \{96B, 256B, 1024B\}$)	0.1	0.41	0.19	1.88	0.36	0.1	0.37	0.2	1.86	0.41

Figure 2.12: Network costs and client-side CPU costs for XPIR (X), SealPIR (S), and FastPIR (F) with a varying number of messages (n) and the size of each message (m) in the server library. d denotes recursion depth (§2.4). FastPIR does not use recursion (sets $d = 1$, which means no recursion). For both XPIR and SealPIR, an optimization to aggregate multiple small messages into a larger one (called aggregation in the literature) is enabled.

2.7× less time than SealPIR ($d = 2$).

The figure also shows the impact of FastPIR's optimizations (§2.4.3, §2.4.4) in reducing its CPU overhead. For smaller values of n, the impact of these optimizations is significant. For instance, for n=2^{15} and m=256B, FastPIR without the two optimizations (F-1 in the figure) is 2.73× more expensive than the full-fledged FastPIR, while FastPIR without its last optimization in §2.4.4 (F-2 in the figure) is 1.45× more expensive than FastPIR will all optimizations enabled. But, as n increases the lower CPU time benefit of the optimizations diminishes. This trend is expected as for larger n the cost for the Answer procedure is dominated by the time to run the BFV.ScMult and BFV.Add operations rather than the rotation operations, which is what the optimizations focus on (§2.4.2–§2.4.4).

Output size of Answer. Figure 2.12 shows the size of the CPIR response generated by the Answer procedure for the three CPIR schemes. When the schemes do not use recursion ($d = 1$), their answer output sizes are smaller relative to when they use recursion, although FastPIR's response size is double the size of XPIR and SealPIR. However,

$d = 1$ is not a viable solution for either XPIR or SealPIR. For XPIR, the query size is large for $d = 1$, which increases network bandwidth and CPU time for processing of CPIR queries (Figure 2.9). For SealPIR, the compressed query is smaller on the wire, but the expanded query has comparable size to that of XPIR. Furthermore, the cost to expand adds significant CPU time for SealPIR $d = 1$.

When the schemes use recursion ($d = 2$), both XPIR and SealPIR do not have the query-size drawback, but increase answer output size, by 8 to 10 times, relative to the $d = 1$ setting. Overall, FastPIR produces smaller responses (answer outputs) without large queries (XPIR with $d = 1$) or significant addition to computation time (SealPIR with $d = 1$).

Query-related overheads. Query generation time and query sizes are significantly larger in FastPIR than SealPIR (especially when the latter uses recursion). For instance, query size for 2^{15} items in SealPIR with $d = 2$ is 17 times smaller than the query size in FastPIR (with $d = 1$). However, FastPIR's query sizes are either smaller or comparable to those for XPIR, depending on recursion depth and message size.

Summary. If ANSWER is invoked frequently for an application with a library that has over several tens of thousands of messages, then FastPIR is a better fit. However, if the application cannot be designed such that its costs are dominated by those of ANSWER, then SealPIR or XPIR may be a better fit.

2.7 Related work

Onion-routing. Systems such as Tor [203], which are based on onion-routing [90, 173], can support anonymous VoIP calls with low message latency. However, they do not

provide strong guarantees. Indeed, a network adversary, such as an ISP, can learn call metadata via traffic analysis [37, 127, 165, 152, 111].

Mix-nets. Chaum introduced a mix-net: a network of nodes in which each node (called a mix) batches incoming messages and releases them in a permuted order [41]. A mix-net based system fundamentally requires at least one mix to be trusted [209, 204, 128, 132, 130, 169, 129, 133, 135, 136]. Yodel [133] is a state-of-the-art system based on mix-nets that specifically targets voice calls. Yodel scales to a few million users while providing a sub-second message latency. However, Yodel assumes that a fraction of the mixes it uses (80%) are not compromised. As one relaxes this assumption, say to make the fraction of trusted mixes to be 70% or lower, Yodel increases the latency between a caller and a callee.

DC-nets. Unlike a mix-net, a dining cryptographers network (DC-net) provides unconditional security using a technique that requires broadcasting of messages between network participants [40]. Due to the broadcasting requirement, earlier systems based on DC-nets scaled to only tens of participants [86, 194, 51]. Later systems [222, 52] improved scalability but at the cost of relaxing the threat model. For instance, Dissent in numbers [222] scales to 5000 clients with 600 ms latency for 600-client groups, but runs a DC-net among a (smaller) group of servers while assuming that one of them is trusted.

PriFi [25] is the latest DC-net based system. It improves latency for a LAN setting of a small organization with a few hundred users (latency is 100 ms for 100 users). PriFi does not scale to thousands or tens of thousands of users. It also assumes that one of its servers in the group of servers is trusted.

Private mailboxes. Systems based on private mailboxes either obliviously write to [50, 73] or read from [16, 13, 183, 31, 125] mailboxes hosted over untrusted servers. The state-of-the-art system based on this strategy that works over completely untrusted infrastructure is Pung [16, 13] (rest of the systems assume non-colluding servers).

We empirically compared Addra to Pung, particularly to its scalable tree-based message retrieval scheme called BST (§2.6). Pung offers two other retrieval schemes: one called explicit retrieval and the other based on Bloom filters. The explicit scheme requires two round trips between a message recipient and the server, and incurs comparable server-side CPU overhead as the BST scheme. However, it is not viable in terms of network overhead as the server has to frequently broadcast a mapping comparable in size to the entire message library. For instance, for 32K users, the server pushes 625 MiB of mapping data every five minutes to every user, thus adding a bandwidth requirement of 16.6 Mbps per user.

The Bloom filter scheme significantly lowers the network overhead relative to the explicit scheme. However, its overhead is still linear in the number of objects (so it is not a viable solution as the system scales up to hundreds of thousands of users). Besides, it works probabilistically: a message recipient is not guaranteed to download the message sent by the sender, thus degrading the quality of service by a non-zero amount.

Although Addra supports synchronous voice calls at scale, and Pung does not (§2.6.1), Addra does not replace Pung, which is designed for asynchronous applications such as email and chat. Indeed, Addra cannot retrieve long-lived messages from the server, which is a requirement for such applications.

Private information retrieval (PIR). Chor et al. [46, 44] introduced the problem of PIR over multiple non-colluding servers, while Kushilevitz and Ostrovsky [126] introduced CPIR over a single untrusted server. Since these decades old seminal works, there

have been numerous improvements to concrete constructions of PIR. For instance, some schemes reduce PIR overheads [198, 62, 13, 3, 61], while others improve answer recovery against a byzantine server [159, 64]. In this paper, we introduced FastPIR, a new CPIR scheme that reduces the server-side computation overhead relative to the state-of-the-art CPIR schemes [13, 3] (§2.6.5).

2.8 Summary and future work

Metadata from voice calls contains rich information about people's lives, and is a prime target for powerful adversaries such as nation states. Prior work that hides metadata either requires trusted intermediaries or does not scale to more than tens of users for low-latency voice calls. This paper described Addra, the first system that hides metadata for voice calls over completely untrusted infrastructure for tens of thousands of users. Addra's current prototype supports 32,768 users on a cluster of 80 machines with a message latency of 726 ms and a voice synthesis rate of 1.6 Kbps. Addra provides its performance and privacy properties through a new, simple, and efficient protocol to access private mailboxes hosted on an untrusted server (§2.3), and a new private information retrieval (PIR) scheme, FastPIR (§2.4).

Our future work involves further scaling Addra from tens of thousands of users to hundreds of thousands or a few million users. To accelerate CPIR computation, a promising direction could be to explore efficient implementations of the master-worker architecture of Addra's server, as well as increased efficiency for the workers using GPUs and FPGAs. For the latter, one would have to address challenges related to running PIR on a heterogeneous system. Finally, a full-fledged Addra system would require extending its support from peer-to-peer voice calls to group calls.

Chapter 3

Coeus: A System for Oblivious Document Ranking and Retrieval

3.1 Introduction

As a motivating example, consider Ziv, who identifies with a non-binary gender, chooses to keep this preference secret from a conservative family, and considers Wikipedia a reliable source of information. Ziv wants to attend a gender-specific event and wishes to read about the event's history before attending it. As usual, Ziv opens Wikipedia, enters a search query (e.g., "History of ____ event in San Francisco"), and selects one of the links to get the desired information. However, this time Ziv feels concerned about privacy due to recent, high-profile data breaches, via insider attacks [84], external hacks [26, 110, 147], mass surveillance by an ISP [18], and even financial pressure [180]. *Can we enable Ziv to search for and retrieve documents from Wikipedia, or more generally any public document repository, privately? Furthermore, can Ziv get peace-of-mind with provable privacy guarantees?*

Ziv's situation is one example of a fundamental problem this paper addresses: *the*

oblivious document ranking and retrieval problem. An abstract formulation of the problem is as follows. A user holds a search query q containing multiple keywords, while a server holds a set of public documents $\mathcal{D}$. The user enters q in a web browser (or app), which interacts with the server to enable the user to select and view one of the K documents that have highest relevance to q. The privacy requirement is that nobody besides the user (neither the server nor a network eavesdropper) must learn any information about q or the document viewed by the user.

We emphasize that this problem is quite different from the problem of searching and ranking on encrypted data that has received much attention in the literature (e.g., [195, 32, 39, 216, 104, 60, 58, 114, 199, 233, 65, 215, 103, 161, 161, 213, 120, 193, 4, 200, 226]; §3.7). In searching on encrypted data, the data is private (owned by the user), while in our problem the documents are public and known to the server (for example, the Wikipedia server owns the documents). This difference in setting enables fundamentally different techniques; for example, if the documents are private, then the owner may encrypt and arrange them in a tree data structure, as in oblivious RAMs [89, 197]. Such encryption is not possible if the documents are public.

Instead, oblivious document ranking and retrieval is more closely related to the problem of private information retrieval (PIR) [44, 126], although with significant differences. PIR, in its most basic form, allows a user to privately retrieve a document by specifying an index in a list (e.g., retrieve the 34-th document from the list of 1,000 documents). In contrast, in our problem a user specifies a multi-keyword search query and not an index. Two extensions to PIR, namely PIR-by-keywords [45] and private stream searching [162, 28], allow the user to retrieve documents that match keywords—but without consideration of ranking. As an example, suppose the user's string is "Cristiano Ronaldo". Then, with PIR-by-keywords, the user will get *one* of the many articles that contain the name of the famous soccer player. On the other hand, with private stream searching, the user

will get *all* documents that mention Ronaldo, without any ranking or ordering, possibly overwhelming the user.

This paper describes Coeus, the first system for oblivious document ranking and retrieval over public documents under a strong threat model that does not make assumptions about the server. Coeus ranks a document's relevance given a user query using the term frequency-inverse document frequency (tf-idf) statistical method [181, 185] (§3.3.1), which is used commonly in text-based recommender systems in digital libraries. Coeus imposes a latency of a few seconds on a user while providing provable guarantees.

At a high-level, Coeus composes the secure matrix-vector product primitive [87, 101, 102, 121] with PIR. One natural way to do their composition is a two-round protocol, where in the first round the user securely multiplies the query q with the tf-idf matrix to obtain scores for all the documents, and then in the second round retrieves the top-K documents obliviously using PIR. The challenge though is the high server-side overhead, imposed by both secure matrix-vector product and PIR. Fundamentally, if the server must learn no information about the user query or the retrieved document, then it must process its *entire* state comprising the tf-idf matrix and the document library; else, the server will learn information about keywords that are *not* in the query, or the documents that are *not* retrieved by the user (§3.2.3).

Coeus responds to this challenge in two ways. First, at the protocol-design level, instead of using the natural two-round protocol, Coeus uses a new three-round protocol that separates out metadata retrieval from document retrieval. In the first round, as in the two-round protocol, a user converts the query q into an encrypted vector, sends it to the server, and obtains encrypted relevance scores for the documents by securely multiplying the vector with the tf-idf matrix. Then, in the second round, the user retrieves short descriptions and title (metadata) for top-K scoring documents from a metadata library using multi-retrieval PIR that can concurrently retrieve multiple objects [13, 108].

Finally, in the third round, the user retrieves a single document that the user wants to view in detail using a single-retrieval PIR [13, 3].

Coeus's three round protocol reduces PIR overhead relative to the two-round protocol. Not only does a user retrieve K smaller metadata instead of K documents, but the separation of metadata from document retrieval enables the server to pack variable-sized documents and compress the document library, thereby reducing PIR compute time.

Coeus's second idea further improves the overhead of the first round through a new secure matrix-vector product primitive that fundamentally reduces server-side work (§3.4.2, §3.4.3), and distributes this work efficiently across a cluster of machines (§3.4.4). Coeus starts from the state-of-the-art construction of Halevi and Shoup that works for a square matrix block with few thousand rows and the same number of columns [101, 102] (§3.3.2). First, Coeus observes and eliminates redundancy in the calls to an underlying homomorphic rotation operation; this optimization reduces overhead by a constant factor of approximately four (§3.4.2). Second, Coeus amortizes the overhead of homomorphic rotations across multiple blocks of the tf-idf matrix (§3.4.3). Third, Coeus efficiently distributes the computation for thousands of matrix blocks (the tf-idf matrix is large consisting of several hundred billion elements) onto a cluster of machines arranged in a master-worker-aggregator architecture. During workload distribution, Coeus preserves the benefits of amortization while keeping in check the network transfer overhead, by including an optimizer that finds the optimal shape of the submatrices at the worker nodes (§3.4.4). Although Coeus's secure matrix-vector product scheme is designed keeping Coeus's scale in mind, it may find uses in other applications especially where matrices are large.

We have implemented (§3.5) and evaluated (§3.6) a prototype of Coeus. On an Amazon EC2 cluster (97 machines for document relevance scoring, 7 for metadata retrieval, and 39 for document retrieval), and for a document library consisting of a corpus of

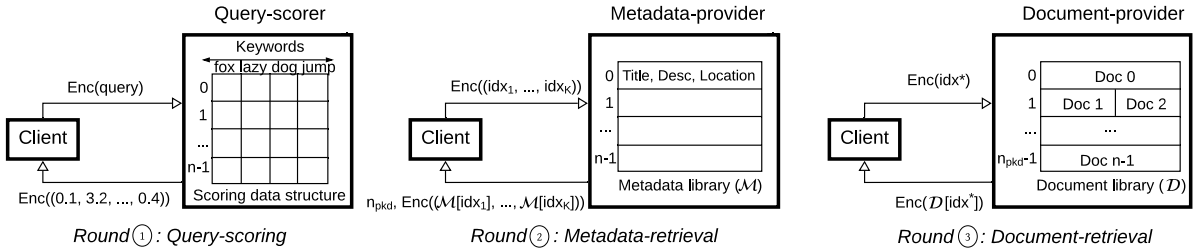


Figure 3.1: An overview of Coeus's three-round protocol: query-scoring, metadata-retrieval, and document-retrieval.

English Wikipedia with 5M documents, Coeus's latency is 3.9s for oblivious document ranking and retrieval. In contrast, without Coeus's two techniques, its latency over the same cluster would be 93.9s—thus an improvement of 24×. If Coeus's resource overheads are converted to dollars, then it costs 6.5 cents per request, in contrast to 1.62 dollars for the baseline.

Coeus's absolute overheads are substantial: each request keeps a cluster of machines busy for up to a few seconds. Thus, it may not be used for every request. However, Coeus scales horizontally, as one can replicate its setup, for example, at various CDNs. But more importantly, Coeus shows that Ziv could *choose* to get strong privacy guarantees while retrieving documents from Wikipedia, without waiting for tens of seconds for the webpage to load, and without draining wallet balance (e.g., hundred private requests per month would cost Ziv 6.5 dollars rather than 162 dollars).

3.2 Architecture and overview

Coeus is designed for private retrieval of public documents. Abstractly, a user holds a multi-keyword query q and a server holds a library of n documents and their metadata (information such as the document title and a short text description). Similar to how search engines work, Coeus takes as input the query q and enables the user to select and

view one of the $K \geq 1$ documents that rank highest for q. In the process, an adversary who may compromise the server hosting the library or the network learns no information about q.

3.2.1 Approach and architecture

An approach to realize the picture described above is to incorporate fully homomorphic encryption (FHE) [81]: the user encrypts q using FHE and sends it to the server, who homomorphically ranks and sends the top-K documents back to the user. On the plus side, the user retrieves the documents in a single round of communication, but on the negative side, the server's computational work is prohibitively high due to the large expense of the homomorphic comparison operation [**lou2019glyph**, 115].

An alternative to the single-round approach is to split document ranking and retrieval into separate protocol rounds. In the first round, the user retrieves scores for each of the n documents, and *locally* compares them to learn indices for the top-K documents. Then, in the second round, the user obliviously retrieves the K documents from the server's library. A downside of this two-round protocol is that the user's device downloads K documents rather than the one document the user eventually views in detail.

Coeus instead follows an approach consisting of three rounds of *query-scoring*, *metadata-retrieval*, and *document-retrieval* that run in succession. These rounds are depicted in the three sub-figures of Figure 3.1, that also shows Coeus's client-server architecture, and the server's three components: a *query-scorer*, a *metadata-provider*, and a *document-provider*.

In the query-scoring round, Coeus's client, running on a user's device, encodes the user query q into a suitable format (for example, a Boolean vector), encrypts it, and sends it to the query-scorer component of the server. The query-scorer maintains a data structure to score documents against user queries and returns an encrypted vector whose i-th com-

ponent contains the query's score for the i-th document in the server's library. The client then locally processes the score vector to obtain the K indices $\{idx_1, idx_2, \ldots, idx_K\}$ for the K vector entries that have the highest values.

Next, in the metadata-retrieval round, the client takes the K indices, encodes and encrypts them in a specific way that enables oblivious retrieval of the metadata, and sends them to the metadata-provider (the middle diagram in Figure 3.1). The metadata-provider sends back the entries in the metadata library $\mathcal{M}$ corresponding to the K indices. The client presents the metadata of the top-K documents to the user and asks the user to select one of the documents.

Finally, in the document-retrieval round, the client uses the metadata from the previous round to get a document from the document-provider component of the server. Since document sizes vary and Coeus must not reveal the length of the retrieved document, the document-provider packs the n documents in the document library $\mathcal{D}$ into $n_{pkd} \leq n$ equal-sized *objects*. Such packing is possible as Coeus can add a document's location (e.g., the index of the object into which a document is packed) to the metadata of the document that is retrieved before the document. The user's device downloads an entire object and locally selects the required document.

3.2.2 Assumptions and guarantees

Threat model. Coeus assumes a strong adversary who may arbitrarily compromise the server or the network. For instance, it may log and process network packets, or the requests received and the responses sent by the server.

We assume that the adversary cannot break standard cryptographic assumptions, such as the semantic security of encryption. We also assume that the adversary does not compromise the user device.

Although we consider server-side side channels (disk access patterns, memory access patterns, etc.), we do not consider side channels that exist due to a client's participation in the system. In particular, we let the adversary learn the number of queries a user makes, the wall-clock times at which the user makes these queries, and the time the user spends in selecting one of the K documents to view in detail. A user who wishes to hide this information can send queries at a fixed schedule, and send dummy queries (e.g., "Cristiano Ronaldo") if needed, as in communication metadata hiding systems [7, 16, 132, 204].

Privacy guarantee. Coeus guarantees *query privacy*. Informally, an adversary learns no information about the user query q (which also means it learns no information about the metadata or the document returned by the server). This notion of privacy is formalized via a security game between a challenger and an adversary, in which the adversary supplies two queries, the challenger simulates Coeus's protocol for one of them, and the adversary guesses which query the challenger picked. In Coeus, the adversary cannot identify the query choice with probability significantly better than that of random guessing ($\frac{1}{2}$).

Non-guarantees. Coeus does not guarantee content integrity that undermines correctness but not privacy. Indeed, the server may compute scores incorrectly, or return documents that do not match the requested indices. Coeus could be extended to add protection against these attacks through additional techniques such as verifiable computation [36, 166].

3.2.3 Challenges

Coeus's three round protocol already improves over alternatives such the one-round

or the two-round protocol (§3.2.1). But still, Coeus must manage the high server-side compute overhead. This challenge is fundamental and best illustrated by an example. Suppose that a client makes a query through Coeus. Then, the three server components, namely, the query-scorer, the metadata-provider, and the document-provider must process their *entire* state (the data structure for scoring, and the libraries $\mathcal{M}$ and $\mathcal{D}$) to service the user query. Indeed, if the server were given an information that would allow it to process a subset of the scoring data structure or the libraries (say leaving out a particular document of $\mathcal{D}$), then the server would learn information about the query keywords or the document that the user is *not* interested in. Although, we cannot break this fundamental lower bound [27], our goal is to improve the concrete efficiency and provide low-latency, affordable ranking and document retrieval.

3.3 Background and protocol

This section describes the scoring method Coeus uses to determine a document's relevance given a user's query (§3.3.1), cryptographic primitives Coeus builds on (§3.3.2), and Coeus's protocol (§3.3.3) that composes these primitives to provide query privacy (§3.2.2). This protocol is an intermediate design point for Coeus, as one of the protocol components requires further optimizations (§3.4).

3.3.1 Term frequency-inverse document frequency

Coeus uses the term frequency-inverse document frequency (tf-idf) measure [181, 238, 185] to determine document relevance given a user query. This method is used popularly in the information retrieval community. It also expresses the scoring function as a matrix-vector product, which is a linear computation that can be performed somewhat efficiently over encrypted data (§3.3.2). Given that tf-idf is well-studied, we do not go into its lower-

level details, but instead focus on the matrix-vector computation structure.

The main idea behind tf-idf is to assign a weight to each (term, document) pair, where a term is, a keyword or a phrase, and the weight reflects how important or relevant the term is to a document in a collection of documents. Thus, a corpus of documents is represented by a tf-idf matrix, where the matrix rows correspond to the documents in the corpus, and the columns correspond to terms in the corpus.

With this arrangement, a common way to score a document d for a query q is to add the tf-idf weights for all terms in the query. This computation can be expressed as a matrix-vector product. The query is converted to a binary vector, whose j-th component is 1 if the j-th term in the corpus is present in the query. Then, the score of a document is the dot product of the query vector with the row vector for the document in the tf-idf matrix. More generally, the scores for all documents are computed by taking the matrix-vector product of the tf-idf matrix and the query vector.

3.3.2 Cryptographic building blocks

Coeus obliviously performs the scoring computation and retrieves the best matching documents and their metadata (§3.2.1), using two cryptographic primitives: secure matrix-vector product [87] and private information retrieval (PIR) [44, 126]. The state-of-the-art constructions of these primitives [13, 101, 102] in turn rely on an underlying homomorphic encryption (HE) scheme based on lattices. The literature offers many lattice-based HE schemes [143, 75, 33, 81, 34]; we use and describe the BFV scheme [75, 33] due to its maturity [187] and involvement as a leading candidate in homomorphic encryption standardization efforts [10].

BFV homomorphic encryption scheme. In the more efficient vectorized version of BFV, a plaintext is a vector with N components and a ciphertext is a vector with $2N$

components, where N is of the form 2^x in $\{2^{11}, \ldots, 2^{15}\}$ [10]. For the plaintext, each component is an element of $\mathbb{Z}_p$, which is the set of integers modulo p. Meanwhile, each component of the ciphertext vector is an element of $\mathbb{Z}_{p\prime}$. The parameters N, p, p' can be tuned for a desired security level [10](§3.5).

The BFV encryption algorithm adds "noise" when encrypting a plaintext into a ciphertext. [1] This noise grows as homomorphic operations are performed on the ciphertext. To ensure that the noise does not grow to a point where the ciphertext cannot be decrypted, $p\prime \gg p$ must be ensured.

The BFV scheme supports three homomorphic operations, BFV.Add, BFV.ScMult, and BFV.Rotate, that are used in the higher-level secure matrix-vector product and PIR primitives.

- **BFV.Add** takes as input two ciphertexts c_1 and c_2 and produces a ciphertext c_{out} that decrypts to the component-wise sum of the plaintext vectors in c_1 and c_2.
- **BFV.ScMult** takes as input a plaintext vector s (of same domain as a BFV plaintext) and a ciphertext vector c and produces a ciphertext c_{out} that decrypts to the component-wise product of s with the plaintext vector in c.
- **BFV.Rotate** takes as input a ciphertext c, an integer $1 \leq i \leq N-1$, and a set of *rotation keys* RK, and produces a ciphertext c_{out} that decrypts to the plaintext in c rotated left cyclically by i positions. For instance, if c encrypts the plaintext (a, b, c, d), then a rotation by $i = 3$ produces a ciphertext that decrypts to (d, a, b, c).

The set of rotation keys RK is configurable and the rotation is performed as a combination of the rotation keys, each of which indicates the number of positions to rotate. On the one extreme, $RK = \{rk_1\}$ contains a single rotation key, where rk_1 performs rotations by one position. In this configuration, each call to BFV.Rotate resolves into i single position

[1] For the readers familiar with differential privacy [71], we remark that the noise in the context of homomorphic encryption is semantically much different from the noise added for differential privacy.

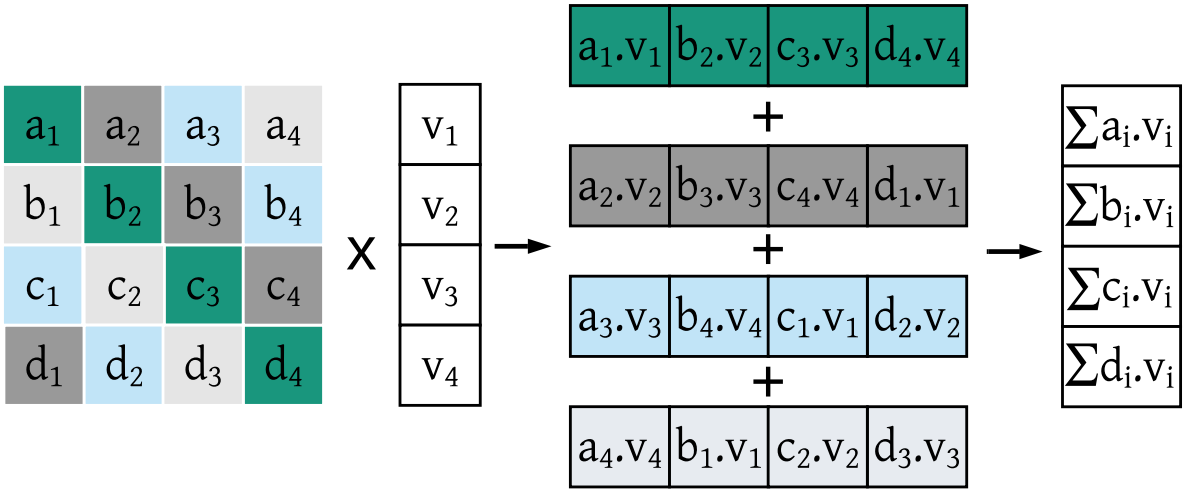


Figure 3.2: An illustration of secure matrix-vector product construction of Halevi and Shoup [101, 102] for a 4×4 matrix.

rotations. Although the size of RK is small, this configuration is impractical due to significant noise growth. On the other extreme, the set $RK = \{rk_1, \ldots, rk_{N-1}\}$ contains $N-1$ keys, and BFV.Rotate calls a single i position rotation with the key rk_i. This configuration keeps noise growth in check (and reduces CPU time of BFV.Rotate), but drastically increases the size of RK needed for BFV.Rotate (with our parameters, all $N-1$ keys in RK would be ≈1.5 GiB). So we assume the default configuration implemented in the state-of-the-art library for BFV [187], where $RK = \{rk_{2^0}, rk_{2^1}, \ldots, rk_{2^{\log(N)-1}}\}$ contains $\log(N)$ keys for all powers of two between 1 and $N-1$, and rotations by i are performed using the rotation keys corresponding to positions of 1s in i's binary representation. Thus, rotation by i uses as many keys as the number of 1's in i's binary representation (i.e., i's Hamming weight); we call such internal calls to a primitive rotation operation that rotates by a power-of-two amount as BFV.PRot. Further, since the set of rotation keys is fixed, we will assume the set of rotation keys is implicit when specifying the rotation operation.

Secure matrix-vector product. A protocol for secure matrix-vector product runs between a client and a server, where the client has a vector, the server has a matrix, and at the end of the protocol the client learns the result of the matrix-vector product. In the process, the server learns no information about the values in the client vector.

The literature on cryptography offers many constructions for secure matrix-vector product (e.g., [87, 118, 101, 121, 12, 68, 102]). The state-of-the-art construction is that of Halevi and Shoup [101, 102]. It operates over square matrices of dimension $N \times N$ (where N is the number of components in plaintext vectors of a lattice-based HE scheme).

The main idea of the Halevi-Shoup construction is illustrated in Figure 3.2. The client starts by encrypting its vector of dimension $N \times 1$ using a lattice-based HE scheme and calling its function. The server then performs the product of the vector with its plaintext matrix using the BFV.ScMult homomorphic operation. The key point here is that the server multiplies the *diagonals* of the matrix with the *rotations* of the client vector. For instance, say $N = 4$ and the matrix is as shown in Figure 3.2. Then, the server first scalar-multiplies the client vector that encrypts (v_1, v_2, v_3, v_4) with the matrix's main diagonal (a_1, b_2, c_3, d_4) to get a ciphertext that encrypts $(a_1 \cdot v_1, b_2 \cdot v_2, c_3 \cdot v_3, d_4 \cdot v_4)$. Then, the server rotates the client vector by one position using BFV.Rotate and multiplies the rotated vector with the matrix diagonal adjacent to the main diagonal to get encryption of $(a_2 \cdot v_2, b_3 \cdot v_3, c_4 \cdot v_4, d_1 \cdot v_1)$. And so on. Finally, the server adds (using BFV.Add) all the intermediate ciphertexts to get one ciphertext containing the result of the matrix-vector product.

We emphasize that the Halevi-Shoup method is much more efficient than naive matrix-vector multiplication that multiplies the input vector with the *rows* of the matrix (rather than its diagonals). In the naive scheme, the server would have to perform $\log(N)$ rotations for each row to add all components of the dot product and allocate the result correctly in the output vector. The Halevi-Shoup construction reduces these $\log(N)$ ro-

tations per-row down to 1 by performing multiplications in diagonal order. In total, the construction makes N calls each to BFV.ScMult, BFV.Add, and BFV.Rotate.

One can trivially support matrices larger than $N \times N$, say of dimension $(m \cdot N) \times (\ell \cdot N)$, by partitioning it into square blocks of size $N \times N$. (In case the original matrix dimensions are not multiples of N, then the matrix can be padded.) In this case, the aforementioned costs get multiplied by the number of blocks $m \cdot \ell$ in the larger matrix.

Private information retrieval (PIR). A PIR protocol [44, 126] runs between a client and a server, where a client has an index i between 1 and n, and the server holds a set of n items. The protocol allows the client to retrieve the i-th item while hiding the value of i from the server.

PIR exists in two flavors: computational PIR (CPIR) [126] and information-theoretic PIR (ITPIR) [44]. CPIR protocols are computationally more expensive but make no assumptions about the server (except standard cryptographic assumptions). On the other hand, ITPIR protocols are more efficient, but require non-colluding servers. For Coeus, we use a CPIR protocol due the alignment of CPIR assumptions with Coeus's threat model (§3.2.2).

Although a PIR protocol allows a client to retrieve one document, it can be extended to retrieving $K > 1$ documents without naively running K parallel instances of a single-retrieval PIR protocol. These more efficient schemes for multiple retrievals are called multi-retrieval PIR [13, 108].

3.3.3 Coeus's protocol

Coeus composes secure matrix-vector product with PIR. Specifically, the query-scorer in Coeus's server (§3.2.1) maintains a tf-idf matrix, and during the query-scoring round of Coeus's protocol, uses secure matrix-vector product to score documents against a user

query. This is possible as the scoring computation with tf-idf is a matrix-vector product (§3.3.1).

In rounds two and three, a Coeus client and the server use PIR. Specifically, in round two, Coeus runs multi-retrieval PIR between the client who has K indices and the metadata-provider who has the metadata library. For round three, the client and the document-provider use single-retrieval PIR.

A subtle issue for the third round is that of document sizes, which can vary. But PIR expects all objects in the server's library to be of the same size. Coeus addresses this issue by using a mix of concatenation and zero-padding, while taking inspiration from prior work on PIR with variable document sizes [99, 108]. In particular, Coeus uses bin packing to pack multiple documents into the least number of bins such that the "capacity" of each bin is equal to the size of the largest document in the document library. After bin packing, Coeus fills unfilled space in each bin with zeros. A consequence of packing is that a Coeus client needs start and end offsets of a document to extract it from a larger (binned) object. Coeus includes this information in the metadata for each document.

We remark that had Coeus not used a three-round protocol that separates out metadata retrieval from document retrieval, Coeus would have had to forego the packing technique described above. Instead, to make document sizes uniform, the natural option would have been to pad each document to the size of the largest document, thereby increasing the size of the document library and the overhead of PIR.

Security analysis. Appendix A.2 contains a rigorous proof that Coeus's protocol provides query privacy (§3.2.2). Briefly, during round one, the client sends an encrypted vector after converting a query into a binary vector and encrypting it. Thus, the server learns no information about the query due to the semantic security of encryption. For rounds two and three, the security of PIR ensures that the server learns no information

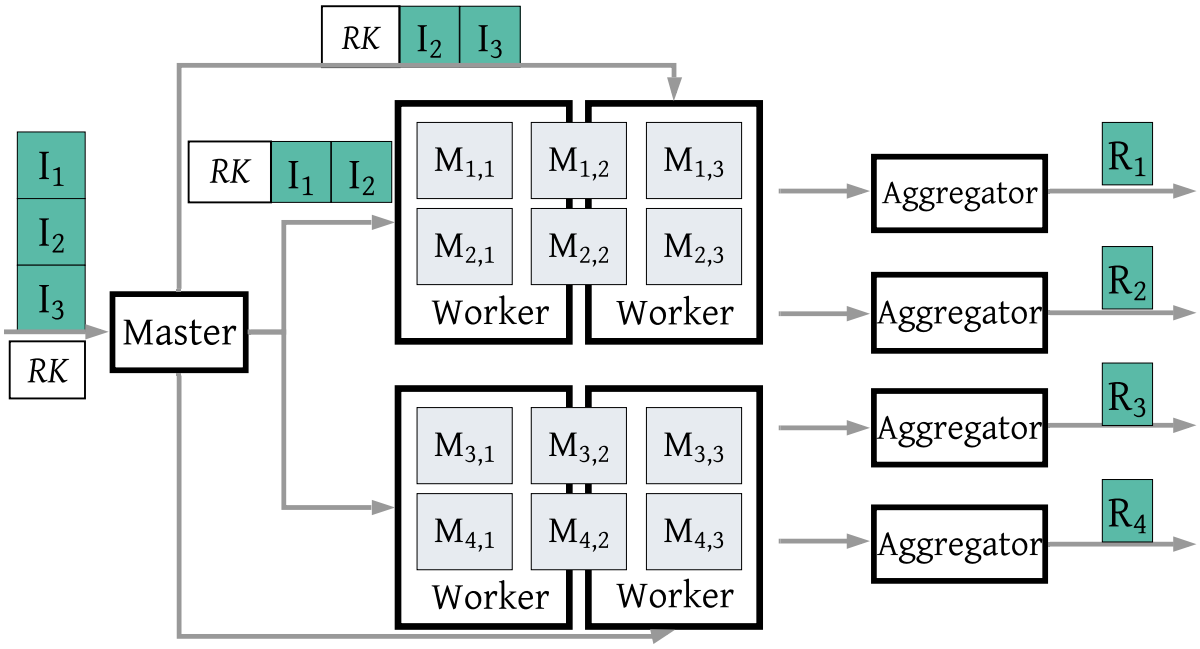


Figure 3.3: How Coeus partitions secure matrix-vector product onto a single master node, and a set of worker and aggregator nodes. I is the input vector from the client containing ℓ ciphertexts, one for each block along the width of the matrix. M is the matrix with $m \times \ell$ blocks. R is the result vector containing m ciphertexts. RK is the set of cryptographic keys for the BFV.Rotate homomorphic operation.

about the indices for which the client is retrieving objects from the metadata or the document library.

3.4 Large-scale secure matrix-vector product

The server-side scalability of PIR has received significant attention recently [7, 13, 16]. Besides, the metadata and document libraries are not large, at least in relation to the tf-idf matrix. But the tf-idf matrix can have millions of rows and tens of thousands of columns—a total of several hundred billion elements—corresponding to the documents and keywords in the server's document library. Thus, a fundamental question Coeus must answer is: how can its server compute the secure matrix-vector product with the

tf-idf matrix while keeping the client-perceived latency small?

One option is to process the tf-idf matrix block-by-block using the Halevi-Shoup construction (§3.3.2), where each block is of dimension $N \times N$, and N is of the form 2^x for some integer $x \in \{11, \ldots, 15\}$ for the security of the underlying homomorphic encryption scheme [10]. This solution, however, does not meet the small latency requirement. First, the processing time for each block is several seconds even on a machine with tens of CPUs (§3.6.3). This expense is due to the high cost of the underlying homomorphic operations, particularly BFV.Rotate (§3.3.2). Second, a tf-idf matrix with billions of elements comprises of thousands of blocks. Naturally, we do not want to provision thousands of machines for the computation. Thus, how should Coeus scale the secure matrix-vector product?

Coeus reduces the work the server has to perform (§3.4.2, §3.4.3), and distributes this work efficiently over a cluster of machines (§3.4.4). We begin with an abstract overview of Coeus's scheme that will help set the stage for the optimizations.

3.4.1 Overview

Computation. Coeus's server multiplies a matrix M of dimension $(m \cdot N) \times (\ell \cdot N)$ consisting of $m \cdot \ell$ blocks each of dimension $N \times N$, with a client input vector I comprising of ℓ ciphertexts (recall each ciphertext itself encrypts a vector of dimension N) to produce a result vector R comprising of m ciphertexts. The i-th ciphertext in R is computed as

$$R_i = \sum_{j=1}^{\ell} \textsc{Block-Mult}(M_{i,j}, I_j, RK),$$

where the sum operation is the homomorphic BFV.Add operation, Block-Mult is a block-level secure matrix-vector multiplication algorithm, $M_{i,j}$ is a matrix block, I_j is a ciphertext in the client input vector, and RK is a set of client-supplied keys for the

BFV.ROTATE homomorphic operation.

Architecture. Coeus projects this computation onto a *master* node, and a set of *worker* and *aggregator* nodes (Figure 3.3). The master receives I and RK from the client. It then copies the keys RK to every worker. It also distributes one or more ciphertexts in I to each worker. The workers together compute BLOCK-MULT$(M_{i,j}, I_j, RK)$ for all $i \in \{1, \ldots, m\}$ and for all $j \in \{1, \ldots, \ell\}$. Each worker, however, performs only part of this computation—corresponding to a *submatrix* of M. An aggregator produces one or more ciphertexts in R by adding outputs from one or more workers.

Division of matrix into submatrices. If Coeus were performing a plain matrix-vector product, it could partition the matrix into submatrices arbitrarily: a submatrix could be a single cell of dimension 1×1, or the entire matrix of dimension $(m \cdot N) \times (\ell \cdot N)$, or any dimension in between. However, the Halevi-Shoup block multiplication algorithm that Coeus builds on imposes certain restrictions due the vectorization of the underlying homomorphic operations: each diagonal of a $N \times N$ matrix block is encoded into a single, indivisible unit (§3.3.2). This means that although the submatrix width w can be any value between 1 and $\ell \cdot N$, the submatrix height h must be a multiple of N. One way to visualize this constraint is to imagine that each matrix block is transformed by taking its diagonals one-by-one and putting them as columns of the block; after this transformation, one can slice the block vertically but not horizontally.

A toy example of the computation. Suppose the matrix M has dimension 4×3 in terms of blocks. Then, the client input I has three ciphertexts, and the result vector R has four ciphertexts. Also, suppose that one of the workers gets assigned the submatrix consisting of block $M_{1,1}$ and half of block $M_{1,2}$ (first $N/2$ diagonals of $M_{1,2}$). Then, this worker receives ciphertexts I_1, I_2 from the master, multiplies I_1 and I_2 with $M_{1,1}$ and the

$N/2$ diagonals of $M_{1,2}$, respectively, to obtain two ciphertexts, and sends their sum to an aggregator. This aggregator adds this ciphertext to a similar ciphertext from another worker who is responsible for the remaining half of $M_{1,2}$ and the whole of $M_{1,3}$. This final sum is R_1.

3.4.2 Reducing expense of homomorphic rotations

We first drill into the computation performed by a single worker, and further into the computation for a single block of the worker's submatrix. For now, assume that the width w and height h of the submatrix are both multiples of N so that the submatrix is an exact multiple of some number of blocks; we will relax this simplifying assumption shortly.

Consider the Halevi-Shoup computation for a block. It comprises of N steps, where each step rotates the plaintext in an input ciphertext c by one position (§3.3.2, Figure 3.2). As an example, if $N = 4$, and the input ciphertext c encrypts the plaintext (v_1, v_2, v_3, v_4), then the algorithm calls BFV.Rotate$(c, 1)$, BFV.Rotate$(c, 2)$, and BFV.Rotate$(c, 3)$ in succession. These $N - 1$ rotations consume the bulk ($\approx 90\%$) of the CPU time.

As mentioned earlier (§3.3.2), each call to BFV.Rotate resolves into a set of calls to a primitive rotation operation BFV.PRot that performs rotations with power-of-two amounts. For instance, BFV.Rotate$(c, 3)$ resolves into a call to $c' \leftarrow$ BFV.PRot$(c, 2)$ followed by a call to BFV.PRot$(c', 1)$. In total, all $N-1$ calls to BFV.Rotate in the Halevi-Shoup algorithm make $\sum_{i=1}^{N-1}$ HammingWt$(i) = (N-2) \cdot \log(N)/2$ calls to BFV.PRot, where HammingWt() returns the number of 1's in the binary representation of its input. But observe there is significant redundancy across multiple calls to BFV.Rotate. For instance, BFV.Rotate$(c, 1100_2)$ calls BFV.PRot for rotation amounts eight and four, while BFV.Rotate$(c, 1111_2)$ calls BFV.PRot over the same ciphertext for rotation

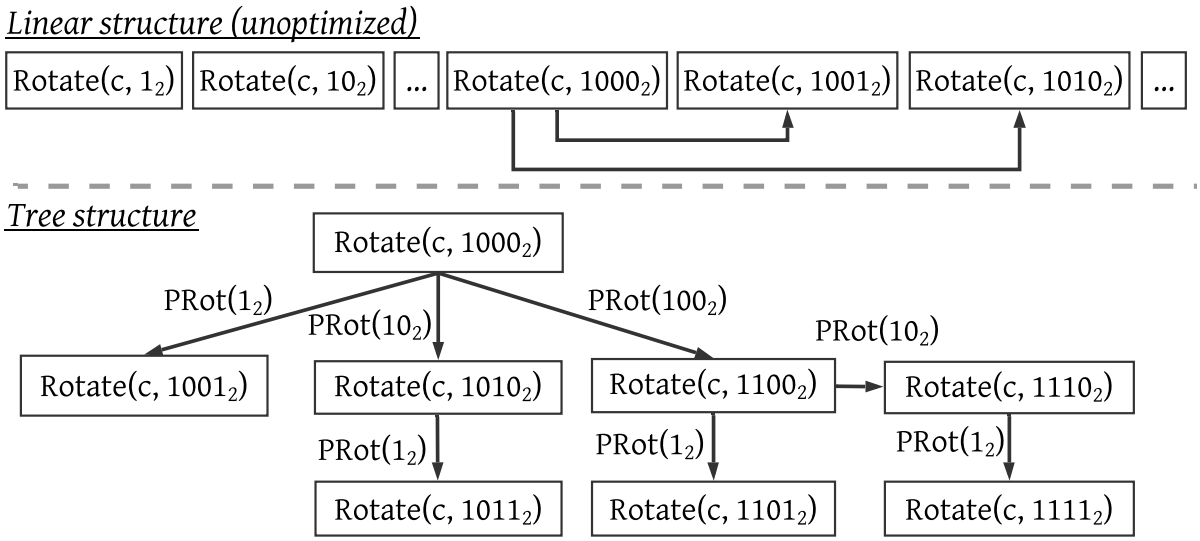


Figure 3.4: How Coeus conserves calls to the BFV.PRot operation.

amounts eight, four, two, and one. Coeus eliminates these redundant calls to BFV.PRot and resolves the $N-1$ calls to BFV.Rotate in the Halevi-Shoup algorithm into $N-1$ calls to BFV.PRot.

Details. Define Parent(i) as the logical AND of the binary representation i_2 and the negation of the smallest non-zero suffix of i_2. For example, if i is 1100_2 in binary, then its smallest non-zero suffix is 100_2, and its parent is $1100_2 \,\&\, {\sim}100_2 = 1000_2$. It is easy to see that the hamming distance between i_2 and Parent(i) is one. Thus, we can obtain $c' \leftarrow$ BFV.Rotate(c, i) by performing one BFV.PRot over the ciphertext BFV.Rotate$(c,$ Parent$(i))$, where the primitive rotation is for an amount equal to the smallest non-zero suffix of i_2.

A first-cut solution to leveraging this parent-child relationship is to generate all rotations of a ciphertext c, that is, BFV.Rotate(c, i) for all $i \in \{1, \ldots, N-1\}$, sequentially, as depicted by a toy example for $N = 16$ in the top part of Figure 3.4. In particular, we can generate BFV.Rotate$(c, i+1)$ from its parent, which is one of the ciphertexts from

BFV.Rotate$(c, 1)$ to BFV.Rotate(c, i). This solution does eliminate redundant calls to BFV.PRot, but it has a downside that it increases memory pressure as it requires storing up to N ciphertexts in memory.

However, observe in the toy example that once the ciphertext BFV.Rotate$(c, 1000_2)$ is generated, the ciphertexts from BFV.Rotate$(c, 1_2)$ to BFV.Rotate$(c, 0111_2)$ can be discarded as they cannot be parents for any ciphertext after BFV.Rotate$(c, 1000_2)$. Similarly, once BFV.Rotate$(c, 1100_2)$ is generated, all ciphertexts prior to (and including) BFV.Rotate$(c, 1011_2)$ can be discarded, and same for BFV.Rotate$(c, 1110_2)$ as the parent for the next value of $i = 1111_2$ is 1110_2.

Leveraging this intuition, Coeus collapses the linear structure into an efficient tree structure that eliminates redundant BFV.PRot without increasing memory pressure, as depicted in the bottom part of Figure 3.4. Coeus performs a depth-first traversal through the tree and at each step in the traversal, generates a child ciphertext from its parent using one call to BFV.PRot. Coeus's algorithm garbage collects any ciphertext in a branch of the tree that has been completely traversed. Hence at any given point, the maximum number of intermediate ciphertexts stored is $\log(N)$ as the height of the tree is $\log(N)$, the number of bits in N. However, further observe that once the algorithm traverses all siblings of a given ciphertext, it can also garbage collect the parent. Hence the number of stored intermediate ciphertexts further reduces to $\lceil \log(N)/2 \rceil$.

This optimization to conserve calls to BFV.PRot applies even to fractional blocks (recall the simplifying assumption at the beginning of this subsection) that contain $d < N$ adjacent diagonals and require performing up to d consecutive rotations. The computation for d diagonals maps to generating a subtree of the overall tree.

Cost savings. The original Halevi-Shoup algorithm applied to a $(m \cdot N) \times (\ell \cdot N)$ dimension matrix makes $m \cdot \ell \cdot N$ calls to the BFV.ScMult and BFV.Add homomorphic

operations (§3.3.2), and $m \cdot \ell \cdot \sum_{i=1}^{N-1} \textsc{HammingWt}(i) = m \cdot \ell \cdot (N-2) \cdot \log(N)/2$ calls to BFV.PRot. Coeus's optimization reduces the calls to the expensive BFV.PRot to $m \cdot \ell \cdot (N-1)$—an improvement by a factor of $\approx \log(N)/2$.

3.4.3 Amortizing rotations across blocks

This subsection zooms out of block-level savings, and considers the entire submatrix at a worker (§3.4.1). Having potentially many matrix blocks to process raises a natural question: can we amortize the overhead *across* blocks? It turns out that the cost of rotations can be amortized.

As with the last subsection, we begin by making a simplifying assumption that the width w and height h of the submatrix are multiples of N; we will relax this assumption towards the end of this subsection.

Consider the computation imposed by the Halevi-Shoup algorithm on a set of matrix blocks that are *vertically aligned* in the submatrix: that is, the blocks $\{M_{i,j}\}$ for a fixed j and different values of i (up to h/N values of i, which is the number of vertically-stacked blocks in a submatrix of height h). First, these blocks are multiplied by the same input ciphertext: the j-th ciphertext I_j in the client input vector I. Second, when these blocks are multiplied by I_j, the Halevi-Shoup algorithm produces the same sequence of rotations for each block: $\text{BFV.Rotate}(I_j, 0), \text{BFV.Rotate}(I_j, 1), \ldots, \text{BFV.Rotate}(I_j, N-1)$.

Coeus eliminates this redundancy in rotations by reordering homomorphic operations. If Coeus were to process each of the vertically-aligned blocks independently, then it would perform a computation structured as: for each of the h/N blocks, perform a sequence of N BFV.Rotate, N BFV.ScMult, and N BFV.Add. Instead, Coeus restructures this computation along the diagonals of the blocks: for each of the N diagonals, perform one BFV.Rotate followed by h/N BFV.ScMult's and h/N BFV.Add's for the h/N blocks.

This optimization extends to fractional blocks that are vertically aligned and contain $d < N$ diagonals each. These diagonals are multiplied by consecutive d rotations of the *same* input ciphertext. Thus, the homomorphic operations can be reordered as before to amortize the costs of rotation.

Cost savings. Let h be the height of the submatrix and w be its width. Then, the submatrix has $f = (h/N) \cdot \lfloor w/N \rfloor$ full blocks and $t = (h/N) \cdot (w - N \cdot \lfloor w/N \rfloor)$ diagonals in the fractional blocks. Without the optimization presented in this subsection, Coeus would make $f \cdot N + t$ calls to each of BFV.ScMult, BFV.Add, and BFV.PRot. With the optimization, the number of calls to BFV.PRot reduces by a factor of h/N.

3.4.4 Setting submatrix dimensions optimally

So far, we have discussed the matrix-vector product while keeping submatrix dimensions abstract: width w and height h. But, how should these values be set?

A strawman design is to partition the matrix into submatrices by using a strategy that is commonly used for *plaintext* matrix-vector multiplication. In plaintext multiplication, the compute time to process a submatrix is proportional to the area of the submatrix—and does not depend on the *shape* of the submatrix. This performance characteristic leads to a common strategy of breaking up the matrix into square submatrices [184, 218].

However, for Coeus, this strategy is sub-optimal as the compute time to process a submatrix *depends* on the shape of the submatrix: taller (but less wide) submatrices have lower compute overhead due to the amortization of rotations (§3.4.3). A downside of making submatrices less wide, however, is the increase in aggregator overhead to combine results from each worker. Thus, one needs to find a submatrix shape that minimizes the total time to compute the matrix-vector product considering both per-worker and across-worker work.

We first present an analytical model for the time to compute the matrix-vector product. This model has limitations and makes several simplifying assumptions, and thus cannot be directly used, but serves as a tool to understand the system behavior. We then use this analytical model to present Coeus's empirical method to determine the submatrix shape.

Analytical model. Our goal is to minimize the total time for computing the matrix-vector product. This time is the sum of three components, $t_{distribute}$, $t_{compute}$, and $t_{aggregate}$, which correspond to the times for the three stages of computation: distributing inputs from the master to the workers, processing each submatrix parallelly at the workers, and aggregating worker outputs (§3.4.1, Figure 3.3).

The first component $t_{distribute}$ is the sum of the time for copying rotation keys RK from the master to each worker, and copying parts of the input vector I as needed to the workers. If the total number of workers is $n_{workers}$, and the time to transfer one copy of RK out of the master is $t_{key\_transfer}$, then the total time for the copying of keys is $n_{workers} \cdot t_{key\_transfer}$. For the remaining cost of copying parts of the input vector I, observe that for a submatrix of width w, a worker needs $\lceil w/N \rceil$ ciphertexts. Thus, if $t_{ct\_transfer}$ is the time to transfer one ciphertext, the total time for input distribution phase is

$$t_{distribute} = n_{workers} \cdot (t_{key\_transfer} + \cdot \lceil w/N \rceil \cdot t_{ct\_transfer}). \tag{3.1}$$

The second component of the total time, $t_{compute}$, is the time taken by a worker to process its submatrix. This time follows from the number of per-worker homomorphic operations executed. This number was analyzed in the previous subsection (§3.4.3). If t_{add}, t_{mult}, and t_{rot} are the times to perform one homomorphic BFV.ScMult, BFV.Add,

and BFV.PRot, then

$$t_{compute} = (h \cdot w)/N \cdot (t_{mult} + t_{add}) + w \cdot t_{rot}. \tag{3.2}$$

Finally, the aggregation time $t_{aggregate}$ equals the sum of the times to transfer intermediate ciphertexts from workers to the aggregators, and the time each aggregator takes to add the ciphertexts. The former equals $m \cdot \lceil (\ell \cdot N)/w \rceil \cdot t_{ct\_transfer}$, and the latter equals $m \cdot \lceil (\ell \cdot N)/w \rceil \cdot t_{add}/n_{agg}$, where m is the number of blocks across the height of the original matrix M, and n_{agg} is the number of aggregators. The rationale is that the matrix has $\lceil (\ell \cdot N)/w \rceil$ vertical partitions (recall that matrix dimensions are $(m \cdot N) \times (\ell \cdot N)$), and each generates m ciphertexts. Thus,

$$t_{aggregate} = m \cdot \lceil (\ell \cdot N)/w) \rceil \cdot (t_{ct\_transfer} + t_{add}/n_{agg}). \tag{3.3}$$

Observe that $t_{distribute}$ and $t_{compute}$ depend linearly on the value w ($h \cdot w$ in Equation 3.2 is the area of each submatrix and is fixed depending on the total area of M and $n_{workers}$). Thus, wider submatrices increase input distribution and computation time. In contrast, $t_{aggregate}$ depends inversely on w, and reduces with the width of the submatrix. Due to these opposing forces, the total time is a convex function of w.

Ideally, we would like to derive an optimal value for w (the lowest point of the convex function) that would minimize the total time. However, there are two issues. First, the model uses uniform values for network transfer times for both keys and ciphertexts that do not account for load, network conditions, and the topology in which workers and aggregators are connected. Second, the total time function is not continuous and differentiable. Hence, in Coeus we develop an empirical method to determine the submatrix width value.

Coeus's empirical method. One tempting option is to configure and deploy a prototype of Coeus for all possible values of w and measure the total time to compute the matrix-vector product. But observe that the total time is a convex function. Thus, we can perform a more efficient directional search inspired by gradient descent in machine learning [91]. Coeus starts by measuring the time for any value of w, say w_{start}; then takes a step in an increasing or decreasing direction of w and measuring the time for a new value of w; then, if the time decreases, it continues in the same direction; otherwise, it goes back to w_{start} and takes a step in the opposite direction. Coeus repeats this process until steps in both directions increase time. Besides following this search approach, Coeus explores only select values of w such that either N is divisible by w, or $\ell \cdot N$ is divisible by w (when $w > N$). These constraints allow Coeus to more easily deal with the boundary conditions due to the ceil function.

3.5 Implementation details

Query-scorer. Coeus's query-scorer (§3.2) is written in ≈2200 lines of C++. Its main piece is a distributed implementation of Coeus's secure matrix-vector product (§3.4) that uses the state-of-the-art Microsoft SEAL library [187] for BFV homomorphic encryption. Recall that the BFV scheme has three parameters: the bound p on each component of the plaintext vector, the dimension N of this vector, and the bound $p\prime$ on each component of the ciphertext vector (§3.3.2). We set p as a 46-bit prime (`0x3FFFFFF84001`), $p\prime$ as a product of three 60-bit primes {`0xFFFFFFFFFFD8001`, `0xFFFFFFFFFFE8001`, and `0xFFFFFFFFFFFC001`}, and N as 2^{13}. These values provide 128-bit security [10]. Furthermore, they satisfy the constraint $p\prime \gg p$ such that the query-scorer can perform the required number of homomorphic operations for the large tf-idf matrix while staying within the noise budget (§3.3.2).

tf-idf matrix preparation and encoding. The query-scorer converts a document library into a tf-idf matrix (§3.3.1) using the Gensim Python library for natural language processing [174, 80]. The query-scorer must also encode the tf-idf matrix into plaintext vectors in the BFV scheme (§3.3.3). One way to perform this encoding is to map each matrix element individually into a single component (of size $\log(p)$) of the plaintext vector. However, this method is wasteful as p is a 46-bit prime and tf-idf values are within a small range. Instead, Coeus uses the standard ideas of quantization [85] and input packing [100, 2] to map multiple (three in our prototype) matrix elements into a single component of the plaintext. For example, if a_1, b_1, and c_1 are the beginning elements of the first three rows of the tf-idf matrix, then Coeus first quantizes each one to one of 2^{10} levels, and then packs them into the value $a_1 \cdot d^2 + b_1 \cdot d + c_1$ made of three "digits" of size $\log d = 15$ bits each. As long as the number of keywords in user queries is less than 2^5, this arrangement ensures that additions of packed values happen digit-wise without overflow.

Metadata and document providers. Coeus's metadata and document provider are written in ≈1200 and ≈1000 lines of C++, respectively. Underneath, the metadata-provider contains our implementation of the multi-query PIR protocol of Angel et al. [13], which in turn builds on the state-of-the-art SealPIR PIR library [188]. Meanwhile, the document-provider directly uses the SealPIR library (which, by default, provides single-retrieval capability). Both the metadata and document provider use a master-worker architecture for the PIR server, where the master receives client request and distributes work to the workers. Similarly, both providers configure SealPIR to provide 128-bit security. Finally, the document-provider implements the first-fit-decreasing bin packing algorithm to pack the set of variable-sized documents into a PIR library with equal-sized objects (§3.3.3).

3.6 Evaluation

Our evaluation focuses on highlighting Coeus's latency for a user request, Coeus's resource overheads (CPU, network, and dollars) for both its server and clients, and the benefits of Coeus's techniques in reducing these overheads. A summary of our main results is as follows:

- For a corpus of 5M documents from English Wikipedia and a dictionary with 65,536 keywords, Coeus's latency is 2.81 s, 0.55 s, and 0.54 s for its three protocol rounds of query-scoring, metadata-retrieval, and document-retrieval (§3.2.1). For the same configuration, a baseline system with two rounds incurs a total latency of 93.9 seconds.
- For 5M documents and 65,536 keywords, Coeus's resource consumption (CPU and network) is substantial. However, when converted to a dollar amount, this cost is 6.5 cents per request. In contrast, the baseline costs 1.62 dollars.
- Both system-level design techniques (§3.2.1, §3.3.3) and optimizations to secure matrix-vector product (§3.4.2–§3.4.4) significantly improve Coeus's performance.

Baselines. We compare Coeus to two baseline systems. *B1* composes the secure matrix-vector product construction of Halevi and Shoup for query-scoring with PIR (specifically, SealPIR [188]) for document retrieval to form a two-round protocol (§3.2.1, §3.3.3). *B2* improves on B1 by incorporating Coeus's technique of splitting the document retrieval round into separate rounds for metadata and document retrieval (§3.3.3). Notably, both B1 and B2 apply the Halevi-Shoup algorithm for query-scoring to the tf-idf matrix block-by-block, and distribute this computation onto a cluster of machines by assigning square, equal-sized submatrices to each worker machine. The difference between B2 and Coeus is the improvements to secure-matrix vector product (§3.4.2–§3.4.4).

Dataset. Our seed corpus is an English Wikipedia articles dump from Feb 1, 2021 [219]. It contains ≈6M articles. However, Coeus's topic modeling library Gensim [174, 80] (§3.5)

removes small re-directional articles, which leaves $4,965,789$ articles. We form a keyword dictionary from these articles by picking keywords that have the highest idf (specificity) (§3.3.1).

Experiment configurations. We vary the number of documents (n) in the server's document library, the number of keywords in the dictionary, and the number of machines assigned to the server. To vary n, we sample documents from the seed corpus uniformly at random. This sampling dictates the size of the document library. For the baseline B1, we pad each sampled document to the size of largest document in the set of sampled documents. In contrast, for B2 and Coeus, we pack (concatenate) smaller documents before padding (§3.3.3). Each document's metadata is 320 bytes, which includes 255 bytes of title [221], and 40 bytes of a short description [220], among other information such as the document's location in the (packed) document library in the case of B2 and Coeus (§3.3.3). For the baseline B1, we set $K = 16$ as the number of documents the client retrieves in the second protocol round, while for B2 and Coeus, K equals the number of documents for which the client receives metadata in the second protocol round. Finally, we set the number of tf-idf matrix columns equal to the number of keywords, and the number of rows of the matrix equal to $\lceil n/3 \rceil$ after taking into account tf-idf matrix preparation from the document data (§3.5).

Testbed. We run Coeus's server and a client over a set of machines in the US East (Ohio) AWS EC2 data center. Each component of the server (query-scorer, metadata-provider, and document-provider) uses one machine of type `c5.24xlarge` (96 vCPU, 192 GiB RAM, and 25 Gbps network bandwidth) to host its master, and a variable number of machines of type `c5.12xlarge` (48 vCPU, 96 GiB RAM, 12 Gbps network bandwidth) to run its workers. For the query-scorer, we also run an aggregator on each of the worker machines. The client uses a single vCPU of a machine of type `c5.12xlarge`.

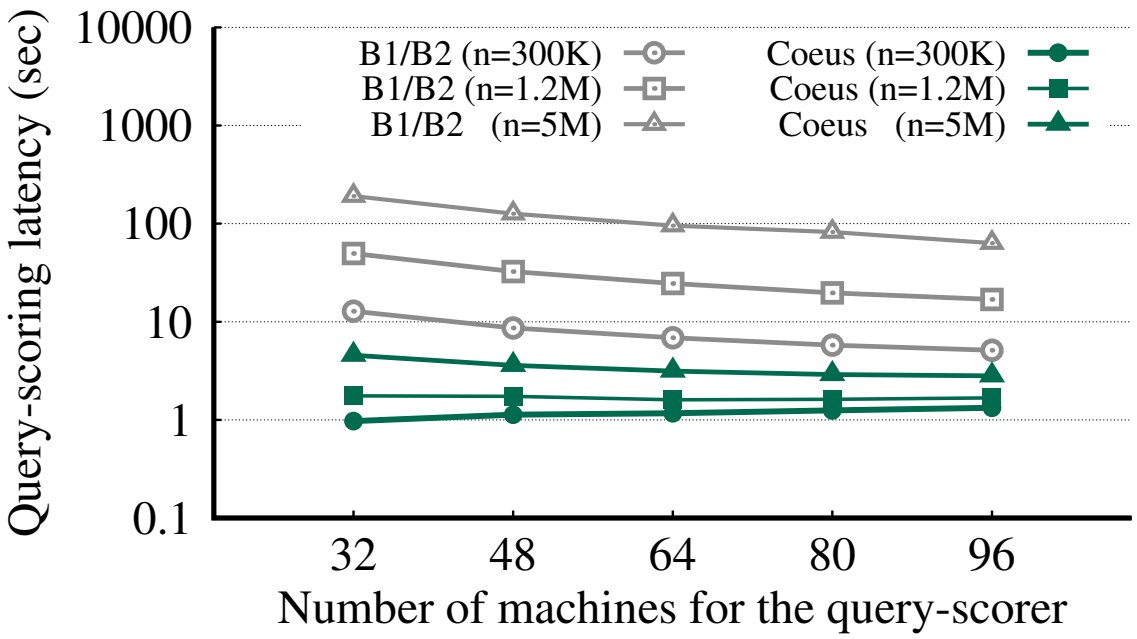


Figure 3.5: User-perceived latency for Coeus's query-scoring round. n is the number of documents in the document library. The number of keywords is set to 65,536.

3.6.1 Latency performance of Coeus

We first focus on the query-scoring round of Coeus's protocol as it is different for Coeus and both the baselines, and then on the other two rounds (metadata and document retrieval), which are different for Coeus and only the B1 baseline.

Coeus versus the baselines for query scoring. Figure 3.5 shows the user-perceived latency of Coeus and the baselines for their query-scoring round, while keeping the number of keywords fixed to 65,536 but varying both the number of documents n in the document library and the number of worker machines for the query-scorer. Coeus's latency is, in general, much lower than the baseline latency. For example, for 5M documents and 96 machines, Coeus's latency is 2.8s, while the baseline's latency is 63.4s, which is 22.6× higher. These improvements are due to Coeus's optimizations to secure matrix-vector product that fundamentally reduce, and efficiently distribute, the server's

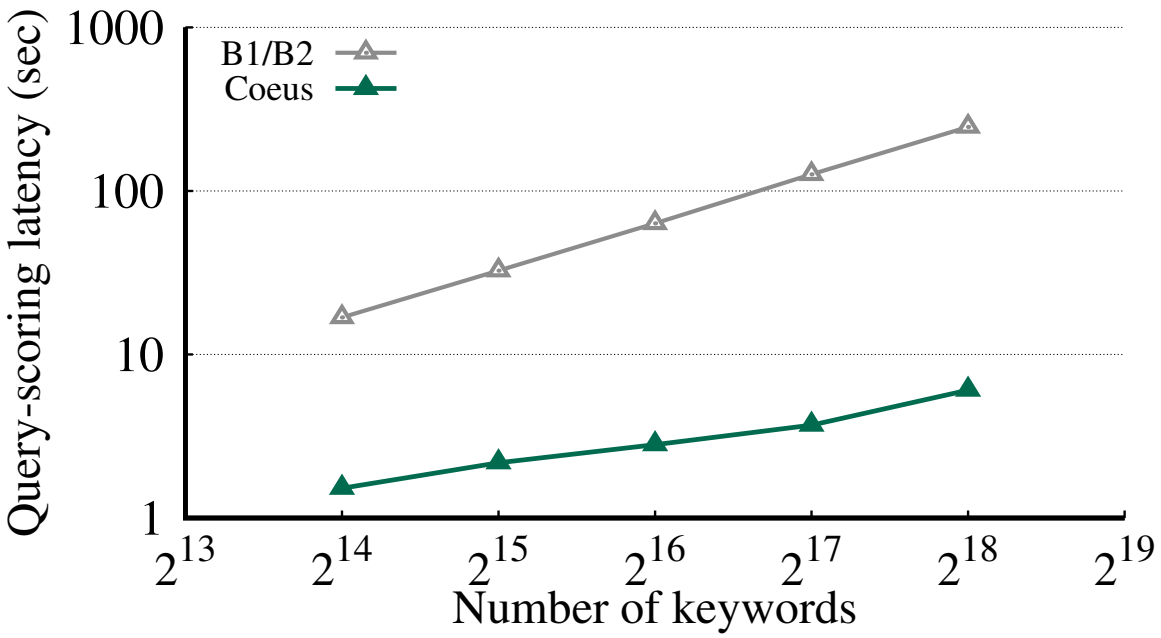


Figure 3.6: User-perceived latency for Coeus's query-scoring round with the number of keywords. The number of documents is set to 5M, and the number of machines for the query-scorer is 96.

work (§3.4.2–§3.4.4). We will evaluate these optimizations individually in §3.6.3.

Variation with the number of machines. The latency of query-scoring initially decreases with the number of machines for the query-scorer, then reaches an inflection point, and then increases with more machines. This trend is most clear to see for Coeus when $n = 1.2$M: the latency is 1.75s for 32 machines, decreases to 1.60s for 64 machines, and then increases to 1.68s for 96 machines. The reason is that although the per-machine compute time decreases with an increase in the number of machines due to a reduction in the size of the submatrix assigned to a machine, the overhead of aggregating intermediate outputs increases (§3.4.4). Thus, adding more machines does not necessarily improve latency. (For $n = 300$K and $n = 5$M, the curves for Coeus are to the right and left of the inflection point, respectively.)

Variation with the number of documents. Coeus's latency for query-scoring increases with the number of documents, but not linearly. This is due to the amortization of the cost of BFV.Rotate operations across matrix blocks (§3.4.3). For instance, for 32 server machines, latency for Coeus grows from 0.97s for 300K documents to 1.75s for 1.2M documents—an increase of 1.8×. In contrast, the corresponding latency for the baselines increases from 12.8s to 49.7s (an increase of 3.88×). This linear growth for the baselines is expected as they perform the secure matrix-vector product block-by-block, without any amortization of costs across blocks.

Variation with the number of keywords. Figure 3.6 shows how Coeus's query-scoring latency changes with the number of keywords when $n = 5$M and the query-scorer runs over 96 worker machines. Coeus's latency increases linearly with the number of keywords with a slope smaller than one. For instance, it increases by 4.1× from 1.5s to 6.1s when the number of keywords increase by 16× from 2^{14} to 2^{18}. The reason the latency does not increase sixteen times (even though the matrix increases by that factor) is that Coeus readjusts submatrix dimensions to make submatrices taller, which reduces server's work by further amortizing the cost of BFV.Rotate operations (§3.4.4, §3.4.3). In contrast, the baseline latency increases with a slope of ≈ 1 as the baseline secure matrix-vector product computation time increases linearly with the width of the matrix.

Latency for metadata and document retrieval. Figure 3.7 shows user-perceived latency for Coeus and the baselines for the rounds of metadata-retrieval (if applicable) and document-retrieval. (For completeness, the figure also shows query-scoring latency from Figure 3.5.)

The baseline B1 does not have an explicit metadata-retrieval round. It uses 48 worker machines to retrieve metadata and data together for $K = 16$ documents. This choice of

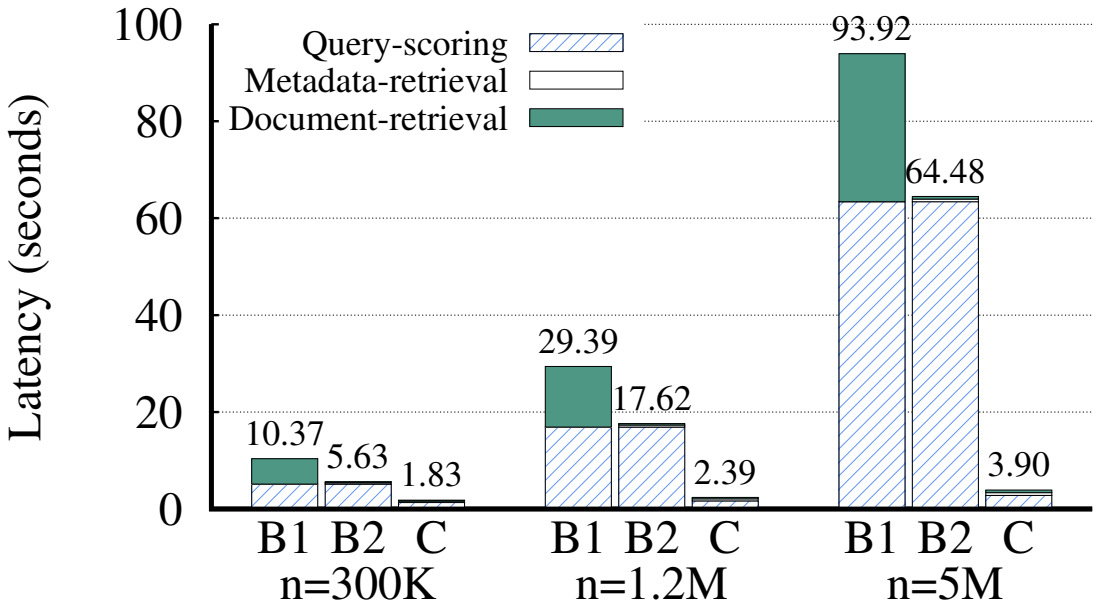


Figure 3.7: User-perceived latency for Coeus (C) and the baseline systems (B1 and B2) with a varying number of documents (n) in the document library. The number of keywords is 65,536. The text provides details of the machines for these experiments.

48 machines is based on parameters for SealPIR and the size of the document library. For instance, SealPIR's multi-retrieval scheme requires partitioning the document library into a number of buckets that is a multiple of K. We choose 48 buckets and assign each bucket to a distinct worker machine. In contrast to B1, the baseline B2 and Coeus have an explicit metadata-retrieval round. We configure these systems to use 6 worker machines for the metadata-provider and 38 machines for the document-provider. Again, these choices are based on SealPIR parameters and the size of the metadata and document libraries. For instance, the largest object after document packing is 142.5 KiB, which encrypts into 38 BFV ciphertexts in SealPIR, where each is processed in parallel.

Coeus's (and B2's) separation of metadata retrieval from document retrieval significantly improves latency over B1. For example, for $n = 5$M, B1 takes 30.5s, while Coeus takes 0.55s for metadata retrieval and 0.54s for document retrieval. This gain is for two

	n=300K	n=1.2M	n=5M
Client cpu (sec)			
B1	4.04	4.43	5.54
B2/Coeus	0.34	0.61	1.64
Upload (MiB)			
B1	12.29	12.29	17.89
B2/Coeus	14.31	14.31	14.31
Download (MiB)			
B1	460.27	470.02	508.02
B2/Coeus	18.78	28.53	66.53

Figure 3.8: Client-side costs per request for Coeus and the baseline systems (B1 and B2) for a keyword dictionary with 65,536 keywords and a varying number of documents (n).

reasons. First, B1 retrieves $K = 16$ documents each of size 140.7 KiB privately from a document library via multi-retrieval PIR, whereas Coeus retrieves a single document and 320 byte metadata for each of the K documents. Second, B1's document library is much larger than Coeus's: 670.8 GiB versus 13.1 GiB. This is because B1 pads each document in its library to the size of the largest document (140.7 KiB), whereas Coeus packs multiple smaller documents into 96,151 objects each of size 142.5 KiB (§3.3.3).

Summary. Coeus's latency is dominated by that of query-scoring. Further, Coeus's techniques are effective: the decoupling of metadata from document retrieval reduces latency from 93.9s to 63.5s for 5M documents and 65,536 keywords, and the optimizations to secure matrix-vector product further reduce this latency to 3.9s (an improvement of 24×).

3.6.2 Resource overheads of Coeus

This section explores the overhead Coeus imposes on clients and estimates the combined overhead in terms of dollars.

Client-side overhead. Figure 3.8 shows client-side CPU time, network upload, and network download for Coeus and the baselines with a varying number of documents (n) in the server's document library. Coeus's network overhead is substantial and thus Coeus requires significant download bandwidth at the client. This is because the query-scoring response contains a score for each document, and thus grows with the number of documents (§3.2.1, §3.3.3). Meanwhile, the upload bandwidth does not change with n, as (a) the length of the input vector to query-scoring depends on the number of keywords (and not on the number of documents), and b) the size of the inputs to PIR (specifically, SealPIR) follows a step function and changes only for a value $n > 16$M.

Coeus's overheads, particularly the network downloads, are significantly lower than that of the baseline B1. The reason is that B1 privately downloads $K = 16$ documents while Coeus retrieves a single object and K smaller metadata.

Dollar cost. We convert both the network and the server-side resource overhead to a dollar amount. For the former, we use a network pricing model of \$0.05 per GiB, which is Amazon's price for bulk network downloads (Amazon does not charge for uploads) [190]. For the server's cost, we multiply the machine rent for Amazon EC2 (`c5.12xlarge` and `c5.24xlarge` machines cost \$0.744 and \$1.488 per hour, respectively [189]) with the number and type of machines we use and the time for which we use them to service a request.

For Coeus, the per-request dollar cost for the configuration of 5M documents and 65,536 keywords is 6.5 cents, of which 5.9 cents is due to query-scoring. The baseline B2 increases this cost to 1.29 dollars, of which 1.28 dollars is due to query scoring. Further, B1 increases this cost to 1.62 dollars, where the additional 34 cents is due to the more expensive document retrieval. Thus, Coeus's improvements take oblivious document ranking and retrieval a level up in affordability.

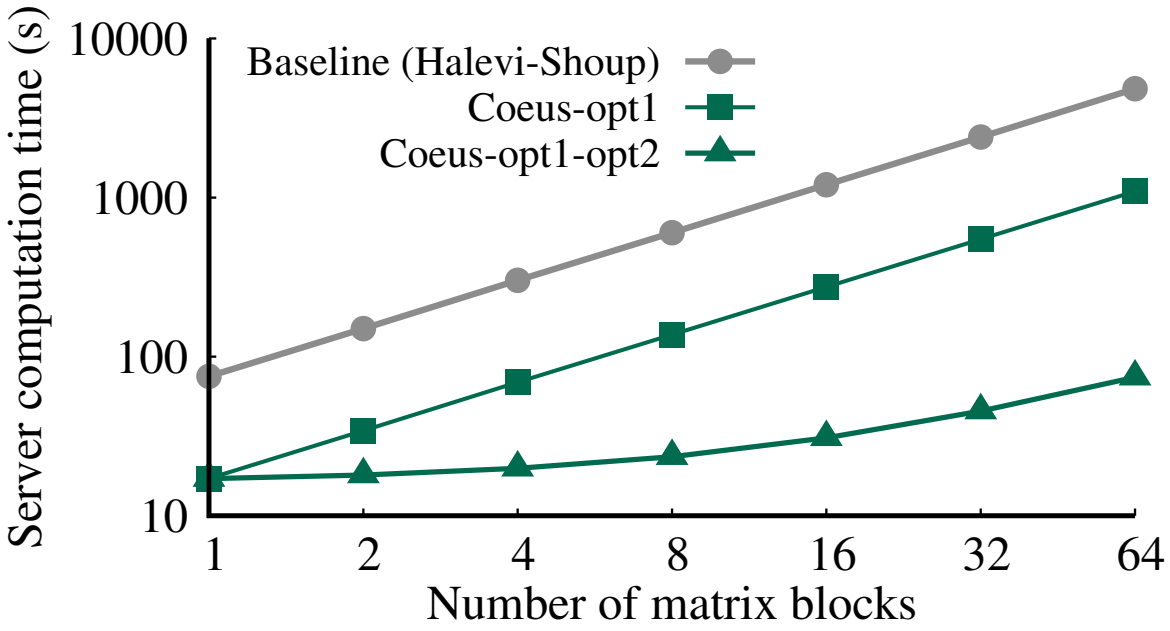


Figure 3.9: Server CPU time to perform secure matrix-vector product.

3.6.3 Performance of secure matrix-vector product

A major part of Coeus's gain over the baselines is courtesy of the improvements to secure matrix-vector product (§3.4). This section zooms into the performance of this primitive in isolation. We first focus on a matrix that fits into a single machine, and then on Coeus's distributed implementation over a cluster of machines. The results also shed light on when Coeus's construction could be beneficial to other applications.

Single machine performance. We run the server component of the secure matrix-vector product on a single CPU of an AWS machine of type `c5.12xlarge`. We compare (a) the baseline Halevi-Shoup construction extended to process multiple blocks block-by-block, (b) this baseline plus Coeus's first optimization (§3.4.2) to reduce the overhead of rotations (Coeus-opt1), and c) this previous variant extended with Coeus's technique (§3.4.3) to amortize rotation time across blocks (Coeus-opt1-opt2).

Figure 3.9 shows the CPU time to compute the product. Each block is of dimension $N \times N$, where $N = 2^{13}$, and new blocks are added vertically on top of existing blocks.

Coeus-opt1 reduces computation time by a constant factor of $\approx 4.4\times$ relative to the baseline. This reduction in time is due to the constant $\log(N)/2 = 6.5$ factor savings in the time for the BFV.Rotate operations (§3.4.2). Coeus-opt1-opt2 further reduces overhead by amortizing the cost of rotations across blocks (§3.4.3). For instance, for the baseline Halevi-Shoup construction, increasing the number of blocks from one to sixty-four increases time linearly from 75s to 4,834s (an increase of 64.4×), but for Coeus-opt1-opt2, the time increases from 17.1s to 74.2s (a factor of 4.34). Overall, for the data point with 64 blocks, the baseline Halevi-Shoup construction takes 4,834s, Coeus's first variant (Coeus-opt1) reduces that time to 1,094s, and Coeus's version with both optimizations (Coeus-opt1-opt2) reduces time to 74.2s.

Multiple machine performance. Coeus distributes secure matrix-vector product computation efficiently onto a cluster of machines, by optimally shaping the submatrices for the worker nodes (§3.4.4). We compare Coeus's performance with and without this optimization. We run the server component of the secure matrix-vector product over a cluster of 64 machines of type `c5.12xlarge` while utilizing all CPUs on each machine. We measure the wall-clock time for the computation while varying the submatrix width.

Figure 3.10 shows the wall-clock time for various phases of secure-matrix vector computation (input distribution from the master to the workers, processing of submatrices at all worker nodes, and the aggregation of intermediate outputs generated by the workers) for an example matrix with 2^{20} rows and 2^{16} columns. The figure also shows the end-to-end (total) time measured by the client.

Overall, the total time curve is convex: the time is higher than its lowest value when submatrices are either too thin (left side of the x-axis) or too wide (right side of the

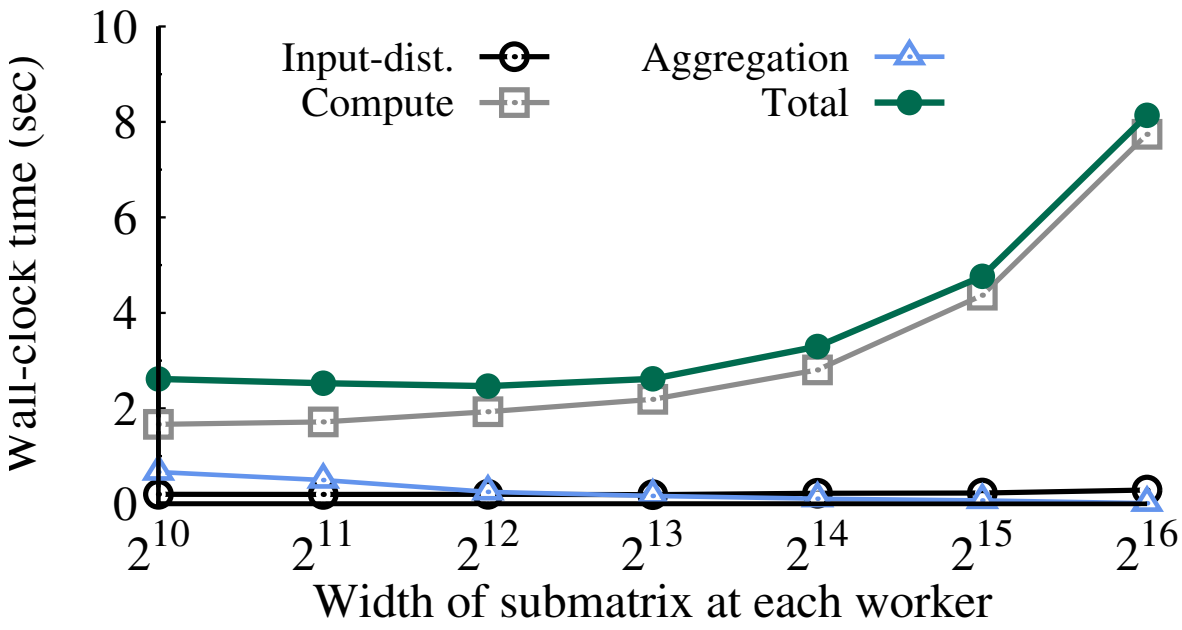


Figure 3.10: Wall-clock time for various phases of computation of Coeus's secure matrix-vector product: input distribution, computation at the workers, and aggregation of intermediate results. The curve labeled "total" is the end-to-end time measured at a client.

x-axis). This convex shape is due to two competing forces. On the one hand, the time to process the submatrices increases with width due to a reduction in the amortization of BFV.Rotate cost. (The time for input distribution also increases with width but slowly.) On the other hand, the cost of aggregation decreases with width. Coeus balances these two forces by finding and setting the optimal width for the submatrices (§3.4.4). Indeed, if Coeus had used the solution of square submatrices, then its time would be 4.76s (the point with width of 2^{15}) rather than 2.46s (width of 2^{12})—an improvement of 1.93×.

The above experiment clarifies that statically setting square submatrices is suboptimal. But could we statically set submatrices to a rectangular shape and get most of the benefit provided by Coeus's scheme? This question is especially compelling as the total time curve (Figure 3.10) changes slowly around the optimal point of 2^{12} width. Thus, as a concrete example, could one always set submatrix width to 2^{12}?

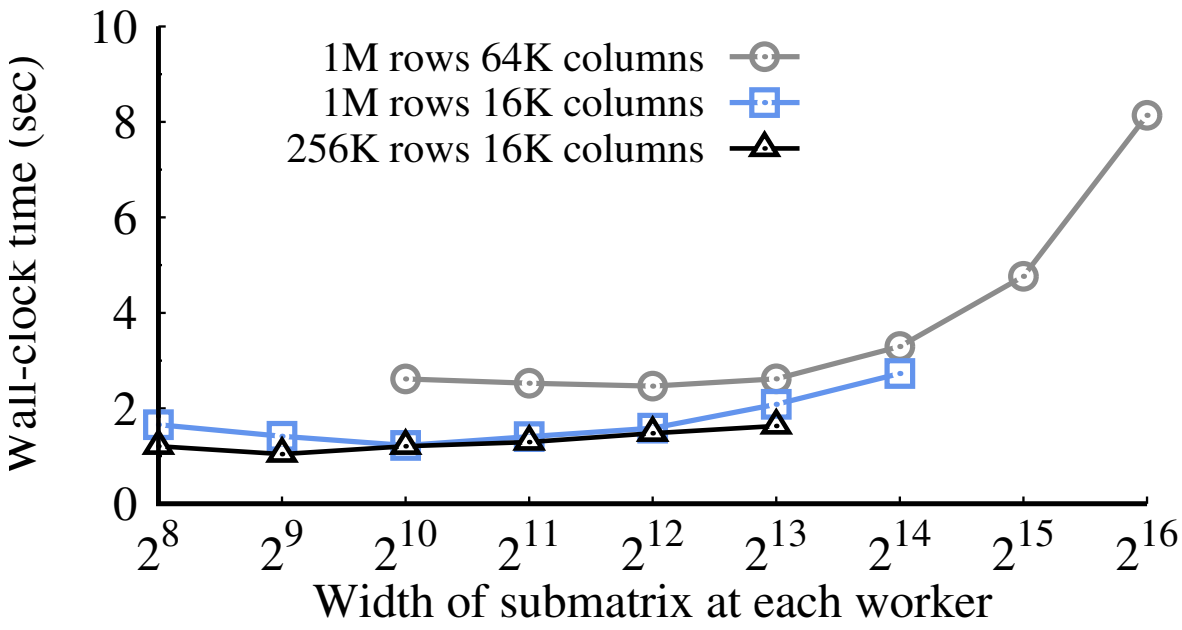


Figure 3.11: Wall-clock time for Coeus's secure matrix vector product protocol over 64 machines for different matrix dimensions and varying submatrix widths.

To answer this question, we rerun the experiment above for three different matrix dimensions: 1M rows and 64K columns, 1M rows and 16K columns, and 256K rows and 16K columns. Figure 3.11 shows the results. The inflection (optimal) point differs significantly—4096, 1024, and 512, respectively—for the three dimensions. Further, statically picking either of these widths is detrimental for the other configurations. For instance, if we pick 4096 as the submatrix width, then Coeus would incur 41% more latency (1.47s instead of 1.04s) relative to the optimal point for the matrix with 256K rows and 16K columns. On the other hand, picking a submatrix width of 512 will be optimal for this matrix with 256K rows and 16K columns but increase latency by 16% for the matrix with dimensions 1M rows and 16K columns. In general, the optimal point depends on various factors such as matrix dimensions, machine performance characteristics, and network connectivity between machines. Besides, these factors change over time due to updates to the document library and upgrades to the infrastructure.

3.6.4 Comparison of Coeus to a non-private baseline

We implemented a tf-idf based system that does not hide user query and the matched documents. This baseline implements a two-round protocol. In the first round, a client sends a query to the server in plaintext. The server computes the tf-idf scores for each document and returns metadata for the top $K = 16$ documents. In the second round, the client selects one document from the top-K and retrieves it from the server. With 5M documents and 65,536 keywords in the tf-idf matrix, and after distributing the server's workload over 48 machines of type `c5.12xlarge`, the end-to-end latency experienced by a client is ≈90ms, which is 44× lower than Coeus. The dollar cost for a single query is 0.09 cents, 72× cheaper than Coeus.

Coeus is different to a non-private baseline also in terms of expressiveness of queries. It supports tf-idf-based ranking over a multi-keyword query, but not other forms of queries such as Boolean queries with AND, OR, and NOT operators, fuzzy queries that auto-correct words that are spelled incorrectly, and wildcard and regular expression queries that enable search for patterns. Supporting these queries in Coeus requires future research, though we note that limited query processing, e.g., checking for typographical errors for fuzzy queries, could be done at the client-side.

3.7 Related work

Searching over encrypted private data. Starting with the seminal work of Song et al. [195], a large body of literature has focused on searching on encrypted *private* data held at a remote server (we refer the reader to surveys and recent papers on this problem [32, 39, 216, 104, 60, 58]).

Two characteristics differentiate this problem from the problem Coeus addresses. First, this problem considers a scenario where the documents are owned by one or more

users, while their storage is outsourced. Thus, the data owner can encrypt its documents using a *symmetric* encryption scheme (e.g., [55]) or include an encrypted index that later helps with search. As noted earlier (§3.1), such encryption is not possible when data is public. Second, schemes in this category focus on searching rather than ranking. For example, a recent system DORY [58] supports retrieval of documents exactly matching a keyword.

Ranking over encrypted private data. A body of literature extends the capability of searching on private encrypted data with the capability to rank the search results [114, 199, 233, 65, 215, 103, 161, 161, 213, 120, 193, 4, 200, 226]. Among these, the schemes of Yu et al. (two-round searchable encryption or TRSE) [233] and Strizhov and Ray [199] are related to Coeus.

Both these schemes support ranking using tf-idf over a two-round protocol that is similar to the two-round baseline B1 discussed and evaluated in this paper (§3.2.1, §3.6). In the first round, a user sends a homomorphically encrypted query to a remote server and learns relevance scores for each document. Then, in the second round, the user retrieves the top-K documents. Despite the similarities to B1, we compare Coeus to B1 rather than these existing schemes, for two reasons.

First, in these existing schemes, the tf-idf matrix is encrypted as the data is private and owned by a data owner. Thus, the remote server multiplies an encrypted matrix with an encrypted vector. In contrast, in the baseline B1 (and in Coeus), the matrix is in plaintext and only the vector is encrypted which results in cheaper server-side operations. Second, these existing schemes inefficiently compute the matrix-vector product. TRSE uses the homomorphic encryption scheme of van Dijk et al. [210] which has large parameters and lacks support for vectorized operations. Meanwhile, Strizhov and Ray's scheme uses the more efficient BGV scheme [34] but computes the product naively by

multiplying the vector with each matrix row. In contrast, B1 uses the state-of-the-art construction of Halevi and Shoup [101, 121] (§3.3.2).

Searching over public data. PIR [126, 44] and its extensions are designed for *public* data. Indeed, PIR in its basic form allows retrieval by index from a public library. With PIR-by-keywords [45, 78], a user specifies a keyword and retrieves *one* of the documents that contains the keyword. Therefore, PIR-by-keywords is most applicable to a setting where keywords are unique, for example, key-value stores [16]. SQL-PIR [158] and Splinter [214] extend the PIR-by-keywords interface to support data retrieval using a subset of SQL. However, they do not support selective, oblivious aggregation across columns as in tf-idf scoring computations. Moreover, they assume non-colluding servers unlike Coeus which does not make such assumptions about the server (§3.2.2). Finally, private stream searching [162, 28, 57] extends search to a stream of public documents such as Google News alerts [229, 232, 156, 230, 231, 77, 235]. But, as mentioned earlier (§3.1), these works do not consider ranking. In contrast to all these works, one can view Coeus as an extension to PIR that prefixes a ranking stage to the private document retrieval stage.

Other related work. Other approaches to searching or ranking privately include trusted execution environments (TEEs) such as Intel SGX [117, 109, 201, 192, 148], anonymous communication systems such as Tor [203], and obfuscation-based techniques that send dummy queries besides real queries [67, 23, 217]. These approaches are either orthogonal or do not provide strong guarantees: TEE-based solutions require trusting the manufacturer of the TEE, anonymous communication systems hide identity but reveal the personally identifiable information (PII) in the query that can in turn reveal a user's identity [24], and obfuscation-based techniques are heuristic in nature and thus suscep-

tible to attacks that separate out real queries from dummy queries [23, 217]. In contrast to these solutions, Coeus hides the content of user queries (and not user identity), and does so provably by incorporating and refining advanced primitives from cryptography.

3.8 Summary and future work

Coeus, to our knowledge, is the first end-to-end system that supports oblivious ranked retrieval over large scale public data and an untrusted infrastructure. Prior approaches either did not support ranking or only managed private data. One can view Coeus as an extension to the PIR domain that efficiently supports ranking by exploiting standard tf-idf statistical methods. At Coeus's core is a new three round protocol that separates metadata retrieval from document retrieval (§3.2.1, §3.3.3), and a novel secure and efficient matrix-vector product protocol (§3.4) based on the Halevi and Shoup method. This latter scheme, although designed primarily for oblivious document retrieval may be useful in other application contexts. Coeus demonstrates that oblivious ranked document retrieval, which up to now was practically impossible due its high overhead costs, has come to the realm of the possible. Our hypothetical, privacy conscious Ziv can now use Coeus to obliviously retrieve from Wikipedia, with its corpus of about 5 million documents, the history of any event of interest in under 4 seconds (rather than minutes) and at a cost of single digit cents (rather than dollars). Needless to say, this is not a panacea, but a significant improvement that paves the way for a practical future where privacy is within the reach of the masses.

In terms of further improvements, one avenue is to reduce the server-side compute overhead, which is still the main bottleneck. Here, accelerators such as GPUs may drive down costs for both secure matrix-vector product and PIR. The sparsity of the tf-idf matrix too presents an opportunity as it contains many zero entries. One can also

consider concurrent queries and batch processing opportunities that are not applicable with a single query. Finally, besides performance, one can improve expressiveness by adding more types of queries such as fuzzy queries, as discussed earlier (§3.6.4).

Coeus's source code is available at

https://github.com/ishtiyaque/Coeus_artifact.

Chapter 4

Pantheon: Private Retrieval from Public Key-Value Store

4.1 Introduction

Due to the widespread use of cloud applications, searching for data from a cloud server has become ubiquitous. However, accessing data stored in a cloud server comes with severe privacy concerns owing to numerous attacks and data breaches [11, 212, 95, 30]. A long line of work [171, 223, 182, 39, 172, 211, 124, 22, 237, 19, 20, 21, 202, 170, 74, 148, 94] (§4.6) addresses this privacy concern for *private data* where a client owns the data and outsources it in encrypted form to a cloud server. The server then executes client queries on the encrypted data to ensure privacy. However, none of these approaches provide privacy for querying over *public data*, where the data is owned and managed by a cloud server, and the server provides query services to many clients. A practical use case is a breached password database service, where a client may query whether her password has been breached and other relevant information. However, the client may be unwilling to disclose which password she is querying about to the service provider or any

network eavesdropper. Similarly, when customers query information about a particular ticker symbol from a stockbroker, or information about a certain disease from a medical repository, they may want to hide the query keywords to preserve their financial and medical privacy. A common feature in all these use-cases is that the server holds a key-value store and clients query about keys. Evidently, a client cannot encrypt the data, i.e., the key-value store in this setting. Therefore, information on which key a client performs queries can compromise privacy [145, 225]. Unfortunately, preserving client privacy in this application domain of *public data* has received very little attention in the literature. This paper addresses this general research problem where a cloud server owns and manages a key-value store, and the clients want to perform queries without sacrificing privacy.

The problem of private retrieval from public data is closely associated with Private Information Retrieval (PIR) [44, 46, 126]. At a high level, PIR allows a client to retrieve an element from an untrusted server without letting the server know which element the client retrieved. However, PIR requires the server to consider the data as an array of elements and the client to know the array-index of the desired element (for example, the client wants to retrieve the element at index 13 from an array of 100 elements). This requirement is a limiting factor in many practical use cases, especially for key-value stores, where the client may be interested in a particular key, but does not know the exact arrangement of the data at the server. An extension of PIR, known as keyword-PIR [45], bypasses this restriction by using multiple rounds of PIR. It allows a client to retrieve the value corresponding to a key from an untrusted server obliviously. Nevertheless, keyword-PIR requires the client to know the total number of keys (n) in the key-value store and perform $\lceil \log_2(n+1) \rceil + 1$ sequential PIR interactions with the server. As a result, it suffers from three significant limitations. First, the number of round-trips increases with the number of keys and thus creates performance and scalability bottlenecks. For

example, the keyword-PIR protocol requires 21 round-trips to retrieve a value from a key-value store containing 1M tuples. Second, the client must know the number of tuples n in the key-value store before constructing a query, creating performance overhead for a dynamic key-value store. Finally, the keyword-PIR protocol involves $O(\log n)$ round-trips between the server and client, where each round requires processing the key-value store. Therefore, the server must preserve its state across the rounds of a query to guarantee consistency. Naturally, one would prefer to retrieve the value corresponding to a key in a single-round to guarantee consistency and atomicity. In this paper, we focus on single-round solutions to the private key-value retrieval problem.

A general approach for constructing a single round solution is to use Fully Homomorphic Encryption (FHE) [81, 34]. In this approach, a client constructs a query q that cryptographically hides the desired key k using FHE and the client sends q to the server. The server obliviously checks equality between k (hidden inside q) and each key in the key-value store using a homomorphic equality operator to determine an encrypted representation of the index of k in the key-value store. Then, the server uses this hidden index information and Private Information Retrieval (PIR) [44, 46, 126] to obtain an encrypted form of the desired value as the response. The client receives the server's response and converts it into the plaintext form of the value.

A number of prior works [8, 9, 69, 82, 144] adopted this approach with different techniques for equality check and PIR. A recent work named Constant-weight Keyword PIR (CKP) [144] is so far the best known instantiation of this FHE-based single-round approach. It proposes a new homomorphic equality operator to check equality between the query key and the keys in the key-value store. Then, it uses the SealPIR [13] technique to retrieve the value using Private Information Retrieval. Even though this work offers a single-round solution to the private key-value retrieval problem, it has major limitations in terms of performance and scalability. First, their proposed equality operator, though

better than prior works, involves expensive homomorphic operations. At a high level, the constant-weight equality operator obliviously evaluates a boolean circuit, and therefore requires each bit of the operand to be encrypted separately into a different FHE ciphertext. The computation for the equality operator comprises expensive homomorphic multiplication operations, and the number of homomorphic multiplications required is a multiple of the number of tuples in the key-value store. Second, the output of the equality operator also involves one ciphertext for each result, and therefore leads to a costly PIR technique. For example, using an AWS instance containing 48 vCPU as the server, the latency for retrieving a value privately from a key-value store containing 64K tuples is 107.8 seconds (§4.5.2), out of which the equality checking takes 80.5 seconds and the PIR step takes 24.9 seconds.

This paper addresses the *performance* and *scalability* issues of privately querying over public data. We present Pantheon, a system that provides a single-round solution to the private key-value retrieval problem and scales to millions of tuples while keeping the server-side latency reasonably low. Pantheon achieves this performance and scalability with contributions in two directions - it refines the cryptographic machinery of the single-round protocol, and applies system level optimizations to support scalability. The primary cryptographic contribution of Pantheon is to present a new homomorphic equality operator using Fermat's little theorem [177]. The key advantage of Pantheon's equality operator is that it is a number theoretic technique, thus the computation is performed in an integer space rather than requiring the evaluation of a bitwise boolean circuit. In addition, Pantheon takes advantage of the SIMD batching property of FHE to pack multiple operands in the same ciphertext and thus amortizing the cost. Due to these two techniques, Pantheon's equality operator reduces the number of ciphertext multiplications compared to CKP by three orders of magnitude (§4.5.1), resulting in significantly lower computation. However, scaling Pantheon to support practical large

key-value stores still remains a monumental challenge. This challenge is due to a fundamental lower bound on the server-side computation. More specifically, while serving a client's query, the Pantheon server must process the entire key-value store; otherwise, it will learn information about the client's query. Therefore, scaling the system leads to high computational overhead on the Pantheon server. Pantheon addresses this systems level challenge by carefully distributing its workload over a cluster of machines and massively parallelizing the computation in each machine. Note that, Fermat's little theorem has been known for centuries, but Pantheon makes the first use of it to provide an end to end solution in a practically scalable manner.

We have implemented (§4.4) and evaluated (§4.5) a prototype of Pantheon. Our implementation includes parallelizing part of the state-of-the-art homomorphic encryption library Microsoft SEAL [187], thus enabling the Pantheon server to distribute its computation over a cluster of machines and multiple cores in a single machine. When the Pantheon server is deployed on a single AWS instance containing 48 vCPU, the latency for performing a GET query from a key-value store containing 64K tuples is 1.15 seconds (§4.5.2), 93× better than the to-date best system built on Constant-weight Keyword PIR [144]. We also deploy the Pantheon server over a cluster of AWS instances, and the latency for a private GET query from a key-value store containing 2M tuples is 0.99 seconds (§4.5.3). Indeed, the latency is substantially higher than a non-private system. We deem this increase in latency as the cost of privacy. However, to our knowledge, Pantheon is the first system to support private retrieval from a million-scale public key-value store with sub-second latency. Moreover, Pantheon is vertically (§4.4.1) and horizontally (§4.4.2) scalable. One can reduce the latency by adding computational resources and leveraging Pantheon's parallelization capability.

4.2 Problem Overview

Pantheon addresses a setting where an untrusted server owns a key-value store and provides a query service to its clients. The server performs the write operations– PUT, UPDATE, and DELETE. The clients issue GET queries for any key. Since write operations are performed by the server, client-side privacy only concerns read operations, i.e., GET queries. The goal is to hide the access pattern, i.e., which key or value a client is interested in, from the server or any adversary. In this section, we formalize the problem, state solution goals, and provide an overview of possible solution approaches.

4.2.1 Problem formulation

A server has a set S of key-value pairs $\{(k_1, v_1), (k_2, v_2), ..., (k_n, v_n)\}$ where each key in $\{k_1, k_2, ..., k_n\}$ is unique, i.e., the keys are primary keys. The server stores the content of S in plaintext and can insert, update, or delete key-value tuples from S. A client holds a key k and wants to know the corresponding value v if $k \in \{k_1, k_2, ..., k_n\}$ such that $(k, v) \in S$, or an empty value otherwise. During this retrieval, the server or any other network eavesdropper must not be able to know anything about k and also not distinguish between returning an empty value and some other value in $\{v_1, v_2, ..., v_n\}$. The server should be able to serve queries from multiple independent clients who may not trust each other.

4.2.2 Threat Model

Pantheon assumes a *passive-adversary* threat model to guarantee query privacy and result integrity. The adversary may see the content of the key-value store, monitor and store all the queries and responses associated with the clients, and perform any analysis on them. It may monitor and analyze any operation performed by the server and perform

side-channel attacks on the server. The adversary may also monitor, log, and analyze any network traffic. We assume the adversary is not actively malicious, i.e., it does not corrupt the key-value store, tamper with any server-side computation, or modify network traffic. Any such adversary may generate incorrect response and thus violate integrity. However, Pantheon must guarantee that no information about the client's query is leaked even in the presence of an *active-adversary* that can arbitrarily modify any data or computation.

We assume the adversary cannot compromise or perform side-channel attacks on the client, because in that case knowing the client query key becomes trivial. We also consider the adversary cannot break standard cryptographic assumptions, such as the semantic security of an encryption scheme.

4.2.3 Goals

Query privacy

Pantheon must guarantee *query privacy* to its clients. An adversary must not be able to learn any information about a client's query key k. It also implies the adversary must not learn any information about the value retrieved.

Consistency

Any response received by a client should be from a stable version of the key-value store that incorporates a write in its entirety or not at all, i.e., it should not involve *dirty* data. We materialize this goal by making Pantheon a single-round protocol.

Performance and scalability

The primary performance metric for a system like Pantheon is its server-side latency for serving a GET query. In addition, the system should be scalable with three parameters related to the size of the key-value store, namely, the size of each key, the size of each value, and the number of tuples in the key-value store (n). For instance, Pantheon should support at least 256-bit keys, so that any arbitrary size key-object can be mapped to a collision-resistant hash digest of the object using commonly known hash functions such as SHA-256 [196]. Further, Pantheon should support arbitrarily large values, because hashing a value does not serve the purpose in most cases. The Pantheon server should also be able to serve a GET query with a reasonable latency, say a few seconds, when the key-value store contains several millions of tuples.

Pantheon's client-side protocol for performing a GET query should be independent of the total number of tuples (n) in the key-value store. Otherwise, it creates a substantial overhead for dynamic key-value stores where the value of n changes frequently.

4.2.4 Possible solution approaches

Before going into the details of Pantheon's architecture, we discuss some possible solution approaches to develop the intuition. We also discuss the limitations of these approaches to rationalize the need for a system like Pantheon.

Strawman 1: Download the entire key-value store

The client may download the entire key-value store S and search for (k, v) locally. This approach may be suitable if the size of S is small, but not otherwise.

Strawman 2: Client downloads the key-set, then performs Private Information Retrieval

Usually, the size of a key is smaller than the value. In that case, the client can first download the entire set of keys, and find the index of k in the key-set locally. Then, the client may privately retrieve the value at that index from the server using Private Information Retrieval (PIR) [44, 46, 126]. There are two major problems with this approach. First, it is a multi-round protocol and therefore does not guarantee consistency (§4.2.3). Second, the client-side download increases with the size of the key-set, making the approach difficult to scale. For example, for a key-value store containing 2M tuples where each key is 256-bits, a client has to download 64MiB of data just for the first round of the protocol.

Keyword-PIR

Chor et al. [45] propose a protocol to retrieve the value corresponding to a key from an untrusted server using multiple sequential rounds of PIR. Their protocol, known as keyword-PIR, proceeds in two phases. In the first phase, the client performs $\lceil \log_2(n+1) \rceil$ round-trip PIR interactions with the server to find out the index of the desired key, where n is the total number of keys. Then, in the second phase, the client uses this index to retrieve the value using another round of PIR. This protocol may be preferable to the previous strawman approach if the number of keys is too large to download in its entirety. However, the number of rounds in this protocol increases with the number of keys, making it a scalability bottleneck. In addition, the protocol cannot guarantee the desirable consistency property (§4.2.3). Furthermore, the keyword-PIR protocol requires the client to know the total number of tuples (n) in the key-value store to initiate a query, which either adds one more round to the protocol for fetching the latest value of n or makes the server broadcast n after every write operation.

Homomorphic encryption

The primary challenge to building a single-round solution to the private key-value retrieval problem is obliviously checking equality between the client query key and the keys in the key-value store. After that, the result of the equality check can be utilized to retrieve the value using Private Information Retrieval (PIR) [44, 46, 126]. Conceptually, it is possible to check the equality obliviously using Fully Homomorphic Encryption (FHE) [81, 34]. We discuss the multiple approaches in the literature using homomorphic encryption as follows.

Evaluating fully homomorphic boolean function. In theory, FHE can evaluate any boolean function over an encrypted data and generate an encrypted output of the function. There are a number of approaches [8, 9, 69, 82, 144] that express the equality operator as a boolean function and evaluates that function homomorphically. However, these techniques are limited in terms of performance and scalability mainly because the boolean function operates over each bit of the operands individually and therefore involve prohibitively large number of homomorphic operations.

The state-of-the-art single-round solution to the private key-value retrieval problem is a protocol named Constant-weight Keyword PIR (CKP) [144]. This protocol proposes a new boolean operator named constant-weight equality operator for homomorphic equality check. This equality operator requires all the keys to have the same number of 1's in their binary representations, i.e., the hamming weight of the keys need to be the same (hence, constant-weight). Accordingly, CKP applies a transformation to map each key to a constant-weight binary string of larger size. For instance, when the key-size is 32-bit, CKP converts each key to a 2955-bit string having hamming-weight (h) as 3. As a result, the protocol requires an expensive initial step where the client's query is expanded to a large number of ciphertexts (2955 ciphertexts for 32-bit key-size), each

encrypting one bit (§4.5.1). Afterwards, the equality check requires $n(h-1)$ homomorphic multiplication operations, where n is the number of tuples in the key-value store. This huge number of operations make the performance impractical, since homomorphic operations are generally computationally expensive. For example, using an AWS EC2 instance of type `c5.12xlarge` (48 vCPU, 96 GiB of RAM) as the server and populating the key-value store with 64K tuples where each key is 32 bits and each value is 256 bytes long, the latency for privately retrieving a value is 107.8 seconds (§4.5.2). Another major limitation with CKP is, it does not scale with the size of the key or the number of tuples in the key-value store. For instance, the current implementation of the work [49] does not support keys larger than 60 bits and is not horizontally scalable. Lastly, the Constant-weight Keyword protocol requires the client to know the total number of tuples (n) in the key-value store for constructing a query. As a result, it introduces performance overhead for dynamic key-value stores.

Using number theoretic technique. Another way of performing a homomorphic equality check is to use technique from number theory such as Fermat's little theorem [177]. This approach considers keys as integers and does not require bitwise encryption of the key. However, this technique involves homomorphic exponentiation, which is also computationally expensive. Therefore, a straightforward implementation using this approach yields impractical performance as well. An example developed by HElib [107] is the best available solution that uses Fermat's little theorem for equality check. Nevertheless, this example implementation takes more than 10 seconds in a single machine to perform a query over a key-value store containing (country name, capital) tuples for 47 European countries, which is worse than the strawman solution of downloading the entire key-value store (§4.2.4). Overcoming the performance bottlenecks and scaling this technique to support millions of key-value tuples remains an open problem. Pan-

theon takes inspiration from this approach and uses Fermat's little theorem for oblivious equality check (§4.3.4). However, Pantheon drastically improves the performance to support queries over a key-value store containing millions of tuples with two key contributions. First, it carefully refines the cryptographic components to reduce the number of expensive homomorphic multiplications by three orders of magnitudes (§4.5.1). Then, Pantheon parallelizes its workload to make the system both vertically and horizontally scalable (§4.5.3).

4.3 Pantheon Design

In this section, we discuss Pantheon's design in detail. First, we discuss the architecture of Pantheon's protocol (§4.3.1) and the cryptographic constructs Pantheon relies on (§4.3.2). Next, we explain how a new client registers itself with the Pantheon server (§4.3.3). After that, we elaborate on the steps for privately querying with a key (§4.3.4). Then, we devise a query compression technique (§4.3.5) that optimizes Pantheon's network overhead. Finally, we provide a security analysis of Pantheon's protocol (§4.3.6).

4.3.1 Basic Architecture

Figure 4.1 shows Pantheon's high-level architecture. Pantheon consists of an untrusted server and its clients. The untrusted server maintains a key-value store S, where $S = \{(k_1, v_1), (k_2, v_2), (k_3, v_3), ..., (k_n, v_n)\}$. For operational flexibility, the server stores the keys $\{k_1, k_2, k_3, ..., k_n\}$ into an array K and the values $\{v_1, v_2, v_3, ..., v_n\}$ into an array V such that for any $(k_i, v_i) \in S$, $K[i] = k_i$ and $V[i] = v_i$. Pantheon's server-side operations to serve a query require all the keys in K to be of the same size and all the values in V to be of the same size. The server applies appropriate padding to the keys and the values, if required, to satisfy this condition.

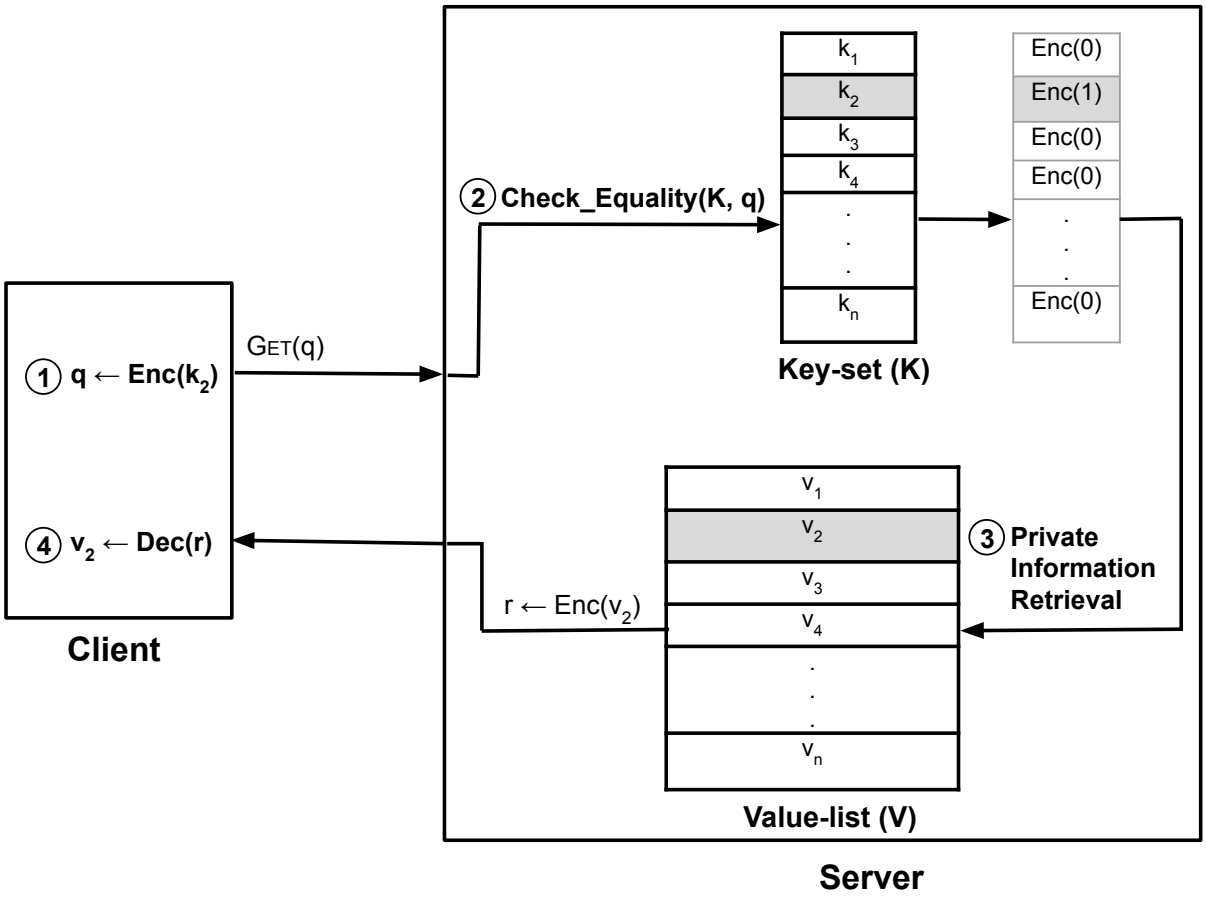


Figure 4.1: High-level architecture of Pantheon.

A Pantheon client can retrieve the value corresponding to a key of interest k from the server in a single round-trip between the client and the server. It takes place in four steps (as shown in Figure 4.1). In step 1, client encodes the query key k into q using Pantheon's encoding procedure that ensures q does not reveal any information about k to an adversary. Then, the client calls the Pantheon server's GET API with q. Steps 2 and 3 take place at the server end. In step 2, the Pantheon server runs an oblivious equality check with each key in K and the client's encoded query q. For each key in K, the equality check outputs an encryption of 1 if it equals the client's query key k or an encryption of 0 otherwise. Step 3 uses the output of step 2 to perform Private Information

Retrieval (PIR) on the value-list V. PIR outputs an encryption of the desired value if $k \in K$, or the encryption of an empty value otherwise. This encrypted value is then sent back to the client. In step 4, the client uses Pantheon's decode procedure to convert the encrypted value to its plaintext form.

Pantheon relies on homomorphic encryption to provide its privacy guarantees. More specifically, we use the BFV scheme [33, 75] of homomorphic encryption because it is standardized [10], resilient to quantum attacks, and has an actively maintained open-source implementation [187]. In this section, we first give a brief introduction to the BFV homomorphic encryption scheme, and then a detailed discussion of Pantheon's functionalities.

4.3.2 Basics of BFV homomorphic encryption

Pantheon uses the more efficient vectorized variant of BFV that allows operating over a vector of data simultaneously and takes advantage of the Single Instruction Multiple Data (SIMD) programming model. In this variant of BFV, a plaintext is a vector of dimension N, where N can be any value in $\{2^{10}, 2^{11}, ..., 2^{15}\}$ [10]. Each element in the plaintext vector is from a set of integers modulo a prime p, i.e., from the set $\{0, 1, ..., p-1\}$. This vector of dimension N is the smallest granularity with which a BFV plaintext can exist. BFV supports an Encrypt procedure that converts the plaintext vector into a ciphertext with the help of the encryptor's secret key. The ciphertext consists of 2 polynomials, each having N coefficients. The ciphertext coefficients are from a set of integers modulo a composite number $p\prime$ such that $p\prime \gg p$. A ciphertext can be decrypted to get the hidden plaintext vector by using BFV's Decrypt method and the secret key that was used for encrypting it in the first place.

BFV encryption supports a number of operations on its ciphertext that eventually

modify its underlying plaintext vector. One important point is that all such operations keep the plaintext elements modulo p. Pantheon uses the following homomorphic operations supported by the BFV scheme:

- **BFV.Add**(c_0, c_1) takes two ciphertexts c_0 and c_1 as input which are encryptions of plaintext vectors v_0 and $v_1$respectively, and outputs an encryption of $(v_0 + v_1)$ (component-wise addition).
- **BFV.Subtract**(c_0, c_1) takes two ciphertexts c_0 and c_1 as input which are encryptions of plaintext vectors v_0 and v_1 respectively, and outputs an encryption of $(v_0 - v_1)$ (component-wise subtraction).
- **BFV.SubtractPlain**(c_0, v_1) takes a ciphertext c_0 (encryption of plaintext vector v_0) and a plaintext vector v_1 as input, and outputs an encryption of $(v_0 - v_1)$ (component-wise subtraction).
- **BFV.Rotate**(c, i) takes as input a ciphertext c (encryption of plaintext vector v), an integer $0 < i < N$, and produces a ciphertext c_{out} such that c_{out} is an encryption of v rotated left cyclically by i positions. For example, if $N = 4$ and c encrypts the plaintext (w, x, y, z), then a rotation by $i = 3$ produces a ciphertext that is an encryption of (z, w, x, y).
- **BFV.Multiply**(c_0, c_1) takes two ciphertexts c_0, c_1 as input which are encryptions of plaintext vectors v_0 and v_1 respectively, and outputs an encryption of $(v_0 * v_1)$ (component-wise multiplication).
- **BFV.MultiplyPlain**(c_0, v_1) takes a ciphertext c_0 (encryption of plaintext vector v_0) and a plaintext vector v_1 as input, and outputs an encryption of $(v_0 * v_1)$ (component-wise multiplication).
- **BFV.Exponentiate**(c_0, i) takes a ciphertext c_0 (encryption of plaintext vector v_0), an integer i as input, and outputs an encryption of v_{out} such that, for $0 \leq j < N$, $v_{out}[j] = (v_0[j])^i$. BFV.Exponentiate may use BFV.Multiply as a subroutine.

4.3.3 One-time Registration phase

A new client joining Pantheon needs to go through an initial registration phase. During the registration phase, the client and server exchange some one-time information required for serving future queries. First, the server shares three cryptographic parameters: N, p, $p\prime$ (§4.3.2) and two system parameters: the size of a key and the size of a value with the client. Then, the client constructs its secret key according to the cryptographic parameters. The system parameters are useful for encoding queries (§4.3.4) and decoding responses (§4.3.4). In addition, the client constructs a number of public keys required for performing homomorphic rotation and multiplication operations. Note that, the secret and public keys are cryptographic keys, not to be confused with the ones in the key-value store. The client also encrypts a vector of length N containing all 1's using its secret key. We will refer to this encryption of all 1's as the *one-ciphertext* of the client. The client then shares its public keys and the *one-ciphertext* with the server. The server stores this information corresponding to the client and confirms registration.

4.3.4 Value Retrieval

The value retrieval protocol in Pantheon takes place in four steps (as shown in Figure 4.1). We now discuss these steps in detail.

Step 1: Encode client query key

This section discusses an unoptimized version of Pantheon's query encoding method that takes the client query key k as input and, depending on the size of k, outputs one or more BFV ciphertexts as q. Later, we will present an optimization technique (§4.3.5) to compress the output query q into a single ciphertext independent of the size of k. Pantheon's query encoding method uses the ENCRYPT procedure of the BFV encryption

scheme(§4.3.2) to hide k. First, let us consider the simple case where each key is of size t bits, where $t = \lceil \log_2 p \rceil - 1$. Therefore, a key can be represented as an integer in the set $\{0, 1, 2, ..., p-1\}$. To encode the query key k, the client will first construct a plaintext vector of size N with all elements as k and then encrypt this plaintext using the client's secret key. As a toy example, suppose $N = 4$, $p = 17$. Then the length of a key may be at most $t = 4$ bits and so the integer representation of a key will always be smaller than 17. Let us assume that client's query key k is the 4-bit binary string 1100, equivalent to integer value 12. Then the client constructs a plaintext vector $(12, 12, 12, 12)$ and encrypts it using BFV's ENCRYPT method. The output of the encode method q then consists of this single ciphertext.

Now we extend the query encoding procedure to the case where the size of each key is larger than t bits. Let each key be αt bits, where α is a positive integer. The keys can be padded accordingly if the size is not an integer multiple of t bits. Then, the client can split the query key k into α chunks, each of size t bits. The client then constructs α different ciphertexts with each of these chunks and outputs q as an array of these α ciphertexts. Continuing with the previous example, let $\alpha = 2$ and client's query key is the 8-bit binary string 11001110. The client will then split k into 2 chunks $\{1100, 1110\}$, also represented as $\{12, 14\}$ in integer forms. The client will then encrypt two plaintext vectors $(12, 12, 12, 12)$ and $(14, 14, 14, 14)$, and output the two ciphertexts as q. Note that, this construction is independent of the number of tuples in the key-value store.

Step 2: Oblivious equality check

After receiving the client's encoded query q, the Pantheon server performs an equality check between q (which is an encryption of client's desired key k) and all the keys in key-set K. Pantheon takes advantage of Fermat's little theorem [177] to perform this equality check obliviously. Fermat's little theorem implies that if p is a prime number and a is a

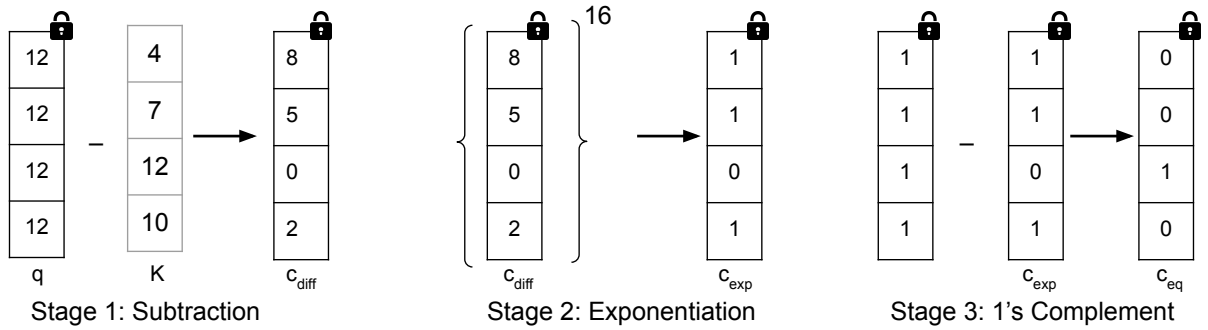


Figure 4.2: Three stages of Pantheon's oblivious equality check (assuming $\mathbf{N = 4}$ and $\mathbf{p = 17}$). In stage 1, the server homomorphically subtracts the key-array $\mathbf{K}$ form client's query $\mathbf{q}$ to obtain $\mathbf{c_{diff}}$. In stage 2, $\mathbf{c_{diff}}$ is exponentiated by $(\mathbf{p - 1})$ to obtain $\mathbf{c_{exp}}$ according to Fermat's little theorem. In stage 3, $\mathbf{c_{exp}}$ is subtracted from *one-ciphertext* to obtain the output $\mathbf{c_{eq}}$.

number not divisible by p, then $a^{(p-1)} \equiv 1 \pmod{p}$. For example, if $p = 17$, then for any $0 < a < p$, according to Fermat's little theorem, $(a^{16})\%p = 1$. In contrast, if $a = 0$, a^{16} still equals 0; a property to distinguish between a zero and a non-zero value.

For the equality check, let us first consider the simplest case where the key-set K contains N keys, each of size $t = (\lceil \log_2 p \rceil - 1)$ bits. So, each key can be represented as an integer smaller than p, and the client query q consists of a single ciphertext. We will later extend the formulation to support a larger key size and more keys. The goal of the equality operator is to obtain a ciphertext c_{eq} that is an encryption of a plaintext vector v_{eq} of size N such that,

$$v_{eq}[i] = \begin{cases} 1, & \text{if } K[i] == k \\ 0, & \text{otherwise} \end{cases}$$

The oblivious equality check proceeds in three stages as demonstrated in Figure 4.2.

In stage 1, the server performs $c_{diff} \leftarrow \text{BFV.SubtractPlain}(q, K)$. As a result, c_{diff} becomes the encryption of a plaintext v_{diff} which contains a zero at index i if $K[i] == k$ and a non-zero value otherwise. In stage 2, Fermat's little theorem is used

to distinguish between the encrypted zero and non-zero values. Server computes $c_{exp} \leftarrow$ BFV.Exponentiate$(c_{diff}, p-1)$. The resultant c_{exp} will be the encryption of a plaintext vector v_{exp} such that,

$$v_{exp}[i] = \begin{cases} 0, & \text{if } K[i] == k \\ 1, & \text{otherwise} \end{cases}$$

In stage 3, the one's complement of the values in v_{exp} is calculated by homomorphically subtracting c_{exp} from the *one-ciphertext* of the client (§4.3.3), which is an encryption of a vector containing N 1's. The resultant of the subtraction operation is the desired equality ciphertext c_{eq} as shown in Figure 4.2.

Now, let us consider the case where each key is larger than $t = (\lceil \log_2 p \rceil - 1)$ bits. Suppose, each key is αt bits long, where α is a positive integer. We can pad each key with the required number of zeros for making the key length an integer multiple of t bits. In this case, the server will first split K into α columns $K_1, K_2, ..., K_\alpha$ such that each K_i contains t bits of each of the keys in K. The client query q also contains α ciphertexts $\{q_1, q_2, ..., q_\alpha\}$ (§4.3.4), each of which encrypts t bits of the client's query key k. Then the server will check equality between each q_i and K_i following the procedure discussed above (3 stages of Figure 4.2) to get a corresponding resultant ciphertext c_{eq_i}. In order to realize the overall equality between k and the keys in K, a logical AND operation among these chunk-wise equality results is required. So, the server computes c_{eq} such that, $c_{eq} = \prod_{i=1}^{\alpha} c_{eq_i}$, where, $\prod$ denotes homomorphic multiplication using BFV.Multiply (§4.3.2).

Now, let us consider the case where the number of keys in K is βN, where β is a positive integer and N is the length of each BFV plaintext (§4.3.2). If necessary, we can pad K with dummy keys to make the size an integer multiple of N. The resultant of equality operation c_{eq} should now consist of β ciphertexts $\{c_{eq}^1, c_{eq}^2, c_{eq}^3, ..., c_{eq}^\beta\}$. The server horizontally partitions K into β parts $\{K^1, K^2, K^3, ..., K^\beta\}$, each partition containing N

keys. Then, for each partition K^j, c^j_{eq} is calculated to contain the equality between q and the partition K^j. Thus, the oblivious equality operator can be extended to support an arbitrary number of keys in K.

Step 3: Private Information Retrieval

Step 3 uses Private Information Retrieval (PIR) [44, 46, 126] to retrieve the desired value from V. A PIR protocol runs between a PIR server and a PIR client where the PIR server holds an array of n elements and the PIR client is interested in the element at index i. PIR allows the PIR client to retrieve the element at index i from the PIR server without revealing any information about i. A PIR protocol has three procedures: PIR.GenQuery, PIR.Answer, and PIR.Decode. The PIR client first constructs a PIR query using the PIR.GenQuery method and sends it to the PIR server. This query is typically the encryption of a one-hot vector of length n, with a 1 at index i and 0 at all other places. It can also be the encryption of n 0's if no element is desired. The PIR server then runs PIR.Answer to generate a PIR response and send it back to the PIR client. The PIR client runs PIR.Decode to decode the response and retrieve the element at index i, or an empty value if no element was desired.

Pantheon uses a slightly modified version of the usual PIR protocol. The PIR query does not directly come from the Pantheon client. Recall that, if $k \in K$, the output of Pantheon's step 2 (equality check §4.3.4) is the encryption of a one-hot vector, with a 1 at index i such that $K[i] == k$, and 0 at all other places. In the case where $k \notin K$, step 2 outputs an encryption of all 0's. As a result, the output of step 2 can be readily used as the PIR query for step 3. Then, the Pantheon server runs the PIR.Answer method on the value-array V and generates a PIR response. The PIR response is then sent back to the Pantheon client. The specific PIR library and other details are discussed in the implementation section (§4.4).

Step 4: Decode server response

After receiving a response from the Pantheon server, the client decodes it to retrieve the value corresponding to k. Since the server's response is the output of a PIR.Answer method, the client can use PIR.Decode to retrieve the corresponding value. Obtaining an empty value from PIR.Decode implies that k is not present in the key-value store.

4.3.5 Query Compression Optimization

The output q of the query encoding procedure discussed in step 1 (§4.3.4) grows linearly in size with client's query key k. More specifically, if k is αt bits long, then q will consist of α ciphertexts. This linear growth increases the network bandwidth cost linearly, creating a scalability bottleneck. Therefore, we propose a query compression method that always reduces the size of q to one ciphertext. The Pantheon server upon receiving this compressed query, needs to perform an expansion, so that it can obtain the α ciphertexts required to perform subsequent operations.

Compression

Suppose the size of each key is αt bits, where α is a factor of N and $t = (\lceil \log_2 p \rceil - 1)$. The client first splits k into α equal chunks, each of length t bits. The client also splits a plaintext vector into α equal parts, each consisting of N/α elements. Each part of this plaintext vector is then filled with the corresponding chunk of k. For instance, let $N = 4$, $p = 17$, $\alpha = 2$. The client query key is an 8-bit binary string 11001110. So, the client can split it into 2 chunks $\{1100, 1110\}$, also represented as integers $\{12, 14\}$. Now, instead of constructing two different ciphertexts for 12 and 14, the client can split the plaintext into two parts and fill them as $(12, 12, 14, 14)$ and encrypt it. This principle can be extended for any α that is not a factor of N by splitting the plaintext vector into $\alpha\prime$ equal parts,

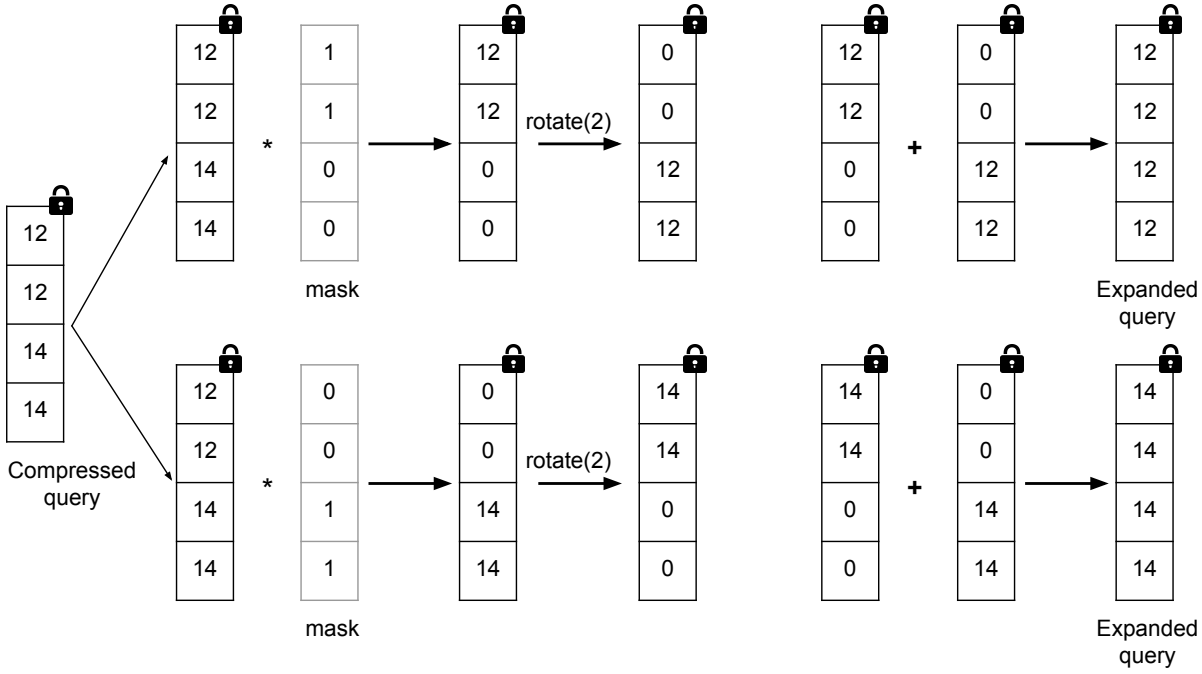


Figure 4.3: Query expansion procedure at Pantheon server (assuming $\mathbf{N = 4}$ and $\mathbf{p = 17}$).

where $\alpha\prime$ is the smallest factor of N such that $\alpha\prime > \alpha$. The client then uses the first α parts of the plaintext to fill with α chunks of k and the remaining parts with 0. As a result, the client query q will always consist of a single ciphertext.

Expansion

The Pantheon server needs to expand the single query ciphertext into α different ciphertexts to proceed with the oblivious equality check. From the previous example, the server receives a ciphertext encrypting $(12, 12, 14, 14)$ and wants to expand it to two ciphertexts encrypting $(12, 12, 12, 12)$ and $(14, 14, 14, 14)$. The expansion procedure is demonstrated in Figure 4.3. First, the server constructs α plaintext vectors to mask out each of the unique values from the query ciphertext obliviously. If α is a factor of N, each mask is split into α equal parts each containing N/α slots of the plaintext vector. Otherwise, each mask is split into $\alpha\prime$ parts as used for compression. For the i^{th} mask

plaintext, the i^{th} part is filled with 1 and the remaining slots with 0. Considering the previous example, the server constructs $\alpha = 2$ masks $(1, 1, 0, 0)$ and $(0, 0, 1, 1)$. The server then performs BFV.MultiplyPlain with the compressed query q and the mask plaintext $(1, 1, 0, 0)$ as input to obtain a ciphertext u that is the encryption of $(12, 12, 0, 0)$. As a result, this mask plaintext helps to filter out the value 12 from the query. The server then computes $u\prime \leftarrow$ BFV.Rotate$(u, 2)$. So, $u\prime$ will be the encryption of $(0, 0, 12, 12)$. The server then computes $c_{add} \leftarrow$ BFV.Add$(u, u\prime)$. This rotation and addition can be repeated for $\log_2 \alpha$ times to obtain the desired expanded query ciphertext encrypting $(12, 12, 12, 12)$. A similar process can be repeated with mask $(0, 0, 1, 1)$ to get the other expanded query encrypting $(14, 14, 14, 14)$.

4.3.6 Security analysis.

Pantheon provides *query privacy* (§4.2.3) to its clients. In this section, we sketch a formal proof of Pantheon's privacy guarantee. Let us define a security game $\mathcal{G}_0$ between a challenger and an adversary as defined in (§4.2.2). The adversary supplies two keys k^0 and k^1 of the same size and the challenger randomly selects one of them as k^b, where $b \in \{0, 1\}$. The challenger performs a query with k^b using Pantheon's protocol. Then, the adversary outputs its guess $b' \in \{0, 1\}$. The adversary wins the game if its guess is correct, i.e., $b' = b$. Let S_0 be the event that $b' = b$ in $\mathcal{G}_0$. Let us also consider another game $\mathcal{G}_1$ where the challenger simulates Pantheon's protocol with a key selected uniformly randomly from the key space, and let S_1 be the event that $b' = b$ in $\mathcal{G}_1$. Evidently, $\Pr[S_1] = 1/2$, since the challenger's query key is independent of those provided by the adversary, and the adversary can only perform random guesses. Now, in $\mathcal{G}_0$, according to Pantheon's protocol, the challenger encrypts k^b using Fully Homomorphic Encryption. Therefore, the advantage of the adversary being able to distinguish it from a random string is,

$|\Pr[S_0] - \Pr[S_1]| \leq \epsilon_{FHE}$. Substituting the value of $\Pr[S_1]$ we get, $|\Pr[S_0] - 1/2| \leq \epsilon_{FHE}$. So, the adversary cannot win the game $\mathcal{G}_0$ with non-negligible advantage.

4.4 Implementation details

Our prototype of Pantheon consists of ≈3,000 lines of C++ code. It uses the homomorphic encryption functionalities provided by the state-of-the-art Microsoft SEAL [187] library. We modify the open-source implementation of Microsoft SEAL, which is a single-threaded library, to provide parallel execution over multiple cores of a machine. For the Private Information Retrieval step (§4.3.4) of Pantheon, we parallelize the open-source implementation of the FastPIR [76] library. We use FastPIR because it requires lower processing time to generate a PIR response [7] and it also uses the vectorized variant of BFV scheme, resulting in a smoother interface with the other components of Pantheon. The details of our parallel implementation of these two libraries are discussed later in this section (§4.4.1).

Parameter selection. Recall that the BFV scheme has three parameters: the size of the plaintext vector N, the upper bound of each plaintext element p, and the upper bound of each ciphertext element $p\prime$. N must be of the form 2^j, where j is an integer such that $10 \leq j \leq 15$ [10]. We choose N to be 2^{15}. p must be a prime number such that $p \equiv 1 \pmod{2N}$. Selection of the particular value for p needs two considerations. Since the size of each plaintext element is $t = \lceil \log_2 p \rceil - 1$ bits, larger size of p in number of bits allows a larger plaintext element, thus accommodating a bigger chunk of client's query key in a single BFV ciphertext. On the other hand, larger p increases the computational cost of raising a ciphertext to the power $(p-1)$, required in the equality checking step (§4.3.4). Our implementation of BFV.Exponentiate uses the repeated squaring method and the

number of calls to BFV.Multiply increases with the number of 1 bits in the binary representation of $(p-1)$. For the same bit-length of p, the cost of BFV.Exponentiate is minimized if p is of the form $2^j + 1$. Therefore, we choose $p = (2^{16} + 1)$, which is a prime number congruent to 1 (mod $2N$). Another noteworthy point is, SEAL considers a pair of adjacent slots in a ciphertext together for its rotation operations. Therefore, it is convenient to consider two BFV plaintext slots together to represent a chunk of the client's query key. So, each chunk of client's query key in Pantheon is $2t$ (32-bits). As a result, number of such chunks in a ciphertext becomes $N/2$ or 2^{14}. We choose $p\prime$ as a 780-bit composite number, which is generated by the Microsoft SEAL library as a product of 13 default prime numbers, each of length 60 bits. These parameters guarantee 128 bit security [10], and ensure $p\prime \gg p$ which is essential for the correct execution of Pantheon's server-side operations.

4.4.1 Parallelization

We parallelize the server-side operations of Pantheon to make it vertically scalable by utilizing all the cores in a multi-core machine. We divide the parallelization into two levels: 1) coarse-grain parallelization, and 2) fine-grain parallelization. In coarse-grain parallelization, we implement data parallelization by operating on different parts of the key-value store in parallel. In fine-grain parallelization, we write a wrapper on the Microsoft SEAL [187] library to parallelize the homomorphic operations it provides. We discuss both of these parallelization levels in detail.

Coarse-grain parallelization

The Pantheon server performs three procedures while serving a client: query expansion, equality check, and private information retrieval (PIR). In coarse-grain paralleliza-

tion, we parallelize each of these three procedures. Brief details of the parallelization techniques are discussed below:

Query expansion. The server receives a compressed query ciphertext and expands it to α ciphertexts using α different masks (§4.3.5). The operations involving different masks are independent and can be executed in parallel. We thus parallelize the query expansion procedure by a factor of α by operating on the different masks in parallel, as shown in Figure 4.3.

Equality check. The server checks equality between the array of keys K and the expanded query ciphertexts. Suppose, the size of each key is αt bits, where t is the size of each element in a BFV plaintext. Then, the expanded query will consist of α ciphertexts. Also, let there be βN tuples in the key-value store, where N is the size of each BFV plaintext vector. Following the equality check step (§4.3.4), we first horizontally partition the array of keys K into β parts, each containing N keys. Then, the output of the equality check will consist of β ciphertexts, each denoting the equality between the expanded query and one of the β partitions. The Pantheon server can process each of these β partitions of K in parallel for equality check.

Now, let us focus on one such horizontal partition consisting of N keys. Each key can be split into α equal slices, each slice consisting of t bits. So, each horizontal partition is further vertically partitioned into α blocks. The i^{th} block needs to be checked for equality as shown in Figure 4.2 with the i^{th} query ciphertext obtained after query expansion. The Pantheon server can execute these α equality checks in parallel. However, after obtaining the results of these equality checks, they need to be multiplied together to obtain a logical AND of the intermediate results. So, this multiplication needs to wait for all the α intermediate equality checks to be completed.

Private Information Retrieval. The Private Information retrieval (PIR) step uses the output of the equality check and the value-array V to output an encryption of the client's desired value (§4.3.4). We parallelize the state-of-the-art FastPIR [76] library to implement this step of the Pantheon server. We vertically partition the value array into multiple parts and perform PIR on each part in parallel. Then, we conduct the "rotate and add" merging policy discussed in FastPIR [76] to merge them and obtain a consolidated PIR response.

Fine-grain parallelization

The coarse-grain parallelization discussed above reduces the server-side latency by processing different parts of the key-value store in parallel. However, this advantage becomes insignificant when the size of the key-value store is small and there are not enough partitions to process simultaneously. Moreover, the latency is dictated by the single threaded execution of the BFV.Exponentiate method, which in turn makes multiple sequential calls to the computationally expensive BFV.Multiply method. So, the Pantheon server may run into a situation where it has idle CPU's but the latency cannot be reduced just by applying coarse parallelization. To address this issue, we modify the BFV homomorphic operations provided by the Microsoft SEAL library to make them use multiple CPU cores in parallel. However, we cannot obtain perfect parallelization as the operations consist of parts that depend on the outcome of prior computations and must be executed in sequential order. We carefully examine the computation graphs of the relevant homomorphic operations and identify the tasks that can be executed in parallel. We primarily adopted two principles in fine grain parallelization. First, a BFV ciphertext consists of two polynomials and we execute the same operation on the two polynomials in parallel. Second, each element of a BFV ciphertext can be decomposed into a number of factors using the Chinese Reminder Theorem, and each factor can be

processed in parallel for the same homomorphic operation. We use the OpenMP [160] library to implement the fine grain parallelization of Microsoft SEAL [187].

4.4.2 Coordinator-worker architecture

Parallelizing the server-side operations in Pantheon improves performance over a single machine. However, to retain low latency for a large key-value store containing millions of tuples, we horizontally scale Pantheon by distributing the server-side workload over a cluster of machines. The cluster deployment of the Pantheon server follows a coordinator-worker architecture. A coordinator node receives the query from the client and then distributes the value retrieval workload among a number of worker nodes. Each worker node performs its part of the computation and sends the result to the coordinator. The coordinator then aggregates the partial results produced by the workers to generate the final response and sends that back to the client.

A high-level description of the coordinator-worker arrangement is as follows. Initially, there is a cluster setup phase where the key-value store is partitioned, and each worker stores a partition in its memory. Suppose there are w workers and a total of n tuples in the key-value store. Let $n = \beta N$, where N is the size of a BFV plaintext vector (§4.3.2), and β is a positive integer. Then, the key-value store can be partitioned into β parts, each consisting of N tuples. A partition size cannot be smaller than N because that is the smallest granularity upon which a BFV plaintext and ciphertext can operate. Then, the β partitions can be distributed among the w workers, each worker containing at most $\lceil \beta/w \rceil$ such partitions, i.e., a disjoint subset of $\lceil (\beta N/w) \rceil$ tuples from the key-value store.

After the cluster setup is complete, the Pantheon server can serve queries from its clients. A client sends its compressed query q to the coordinator. The coordinator broadcasts q to all w workers. Each worker performs the query expansion operation (§4.3.5)

to obtain the expanded query. Then each worker executes the equality check (§4.3.4) and Private Information Retrieval (PIR) (§4.3.4) steps on its partition of the key and value arrays, respectively. Since at most one of the workers can have the desired key in its key array, that worker will generate the client's desired PIR response. Each of the other workers will generate a PIR response of an empty value. All workers send their partial responses back to the coordinator. PIR responses have an additive homomorphic property such that adding empty responses to a response encrypting the desired value yields a new response that also encrypts the same desired value. Similarly, if all the w workers generate PIR responses of empty value, adding them together yields a new PIR response of empty value. The coordinator performs this aggregation by adding the PIR responses from all w workers and sends the aggregated response back to the client.

4.5 Evaluation

Our evaluation focuses on Pantheon's server-side latency for serving private GET queries. We compare Pantheon's performance with Constant-weight Keyword PIR (CKP) [144], the state-of-the-art system for private key-value retrieval. First, we provide a microbenchmark of Pantheon's and CKP's different components and analytically explain the reasons for Pantheon's performance superiority over CKP. Then, we compare the performance of the parallelized implementation of Pantheon (§4.4.1) with the open-source multi-threaded implementation [49] of CKP over all the cores of a single AWS instance. After that, we demonstrate the effect of Pantheon's parallelization techniques (§4.4.1) by comparing the final version with two other intermediate versions: Pantheon (S) - a single-threaded implementation with no parallelization, and Pantheon (C) - a version containing only the coarse-grain parallelization. We configure all the variants of Pantheon and the Constant-weight Keyword PIR system to provide 128-bits of security [10].

	32-bit key		64-bit key	
	CKP	Pantheon	CKP	Pantheon
Total server time (sec)	**463.38**	**3.64**	**921.2**	**6.90**
Query expansion (sec)	12.92	0	39.98	0.15
Equality check (sec)	438.36	3.08	869.12	6.19
PIR (sec)	12.10	0.56	12.10	0.56
Number of operations				
Substitution/ Rotation	4095	0	12286	2
BFV.Multiply	32768	16	65536	33

Table 4.1: Microbenchmarks for different operations to serve a Get query by both Pantheon and Constant-weight Keyword PIR (CKP). All the configurations use 16,384 tuples in the key-value store with each value being 256 bytes.

Experiment setup. We run all our experiments in AWS EC2 US East region (Ohio). The single-machine experiments use one instance of type `c5.12xlarge` (48 vCPU, 96 GiB of RAM, and 12 Gbps of network bandwidth) as the server. For Pantheon's cluster deployment, we use one instance of type `c5.24xlarge` (96 vCPU, 192 GiB of RAM, and 25 Gbps of network bandwidth) as the coordinator and 128 instances of type `c5.12xlarge` as workers. For each experimental configuration, we generate the keys by taking the hash digest of integers in $\{1, 2, ..., n\}$ to ensure unique keys. The values are filled in with random bit-strings. We repeat each experiment 10 times, discard the minimum and maximum values to avoid outliers, and then take the average of the remaining values.

4.5.1 Microbenchmarks

We benchmark both Pantheon and Constant-weight Keyword PIR (CKP) [144] on a single core of an AWS `c5.12xlarge` instance with two different key sizes. We run Pantheon with 32-bit and 64-bit keys. Since CKP does not support key size larger than 60 bits, we run it with 32-bit and 60-bit keys. CKP has a configurable parameter named hamming-weight of the keys (denoted as h). We consider $h = 3$ for 32-bit key configuration and $h = 5$ for 60-bit key configuration. These values of h yield the minimum

latency for the respective key sizes. For all the configurations, we take the number of tuples in the key-value store (n) as 16,384 and the size of each value as 256 bytes. Table 4.1 shows the server-side cost for each of the three steps in the execution of a GET query: 1) Query expansion, 2) Equality check, and 3) Private Information Retrieval (PIR). We present an analytical discussion of the component-wise cost as follows.

Query expansion. CKP has substantially higher query expansion time compared to Pantheon. This is because, in CKP, a single query ciphertext needs to be expanded to m ciphertexts such that, $\binom{m}{h} \geq 2^{keysize}$. The expansion procedure makes repeated calls to an expensive homomorphic substitution operation. On the other hand, in Pantheon, one query ciphertext is expanded to α ciphertexts where $\alpha = \lceil \frac{keysize}{2 \log_2 p} \rceil$. This involves $\alpha \lceil \log_2 \alpha \rceil$ calls to expensive rotation operations which is analogous to the substitution operation used in CKP. For example, with 60-bit keys, a query ciphertext in CKP is expanded to 10,673 ciphertexts with a total size of ≈5.21 GiB, involving 12,286 substitution operations. The total time for this expansion step is 39.98 seconds. On the other hand, a query in Pantheon for 64-bit keys expands to 2 ciphertexts with a total size of 6 MiB, requiring 2 rotation operations and a total time of 0.15 seconds. A single rotation operation in Pantheon takes more time than a substitution in CKP due to its larger parameters, but the total time for all the operations is significantly lower.

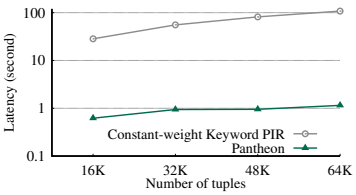
Equality check. The Equality check is the most expensive step for both Pantheon and CKP. The cost of the equality check is dominated by the BFV.MULTIPLY operation to multiply two ciphertexts together. CKP requires $n(h-1)$ total number of calls to BFV.MULTIPLY, where n is the total number of tuples in the key-value store and h is the hamming-weight of each key, a tunable parameter for CKP. On the other hand, Pantheon requires $(\alpha \cdot \frac{n}{N/2} \cdot \log_2 p + (\alpha - 1))$ multiplications, where $\alpha = \lceil \frac{keysize}{2 \log_2 p} \rceil$. Even

though the asymptotic relation with number of elements n is the same, the $\frac{1}{N}$ factor in Pantheon's cost significantly reduces the number of ciphertext multiplications. For instance, as shown in Table 4.1, for 32-bit keys the number of multiplications in CKP is $32,768$, whereas that in Pantheon is only 16. The corresponding times required for equality check are 438.36 and 3.08 seconds respectively. Note that, due to the larger parameter size, a single BFV.Multiply operation in Pantheon takes $\approx$14$\times$ more time than that in CKP. Even then, due to the significantly smaller number of multiplications, the overall time for equality check is smaller in Pantheon.

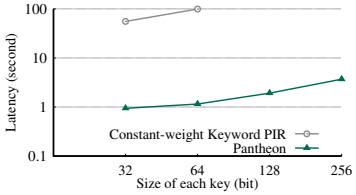
Private information retrieval. Pantheon also takes less time than CKP for the PIR step. The performance gain is mainly due to the performance difference between FastPIR [7] (used by Pantheon) and SealPIR [13] (used by CKP). It is worth mentioning that the design of the equality operator in Pantheon allows it to take advantage of FastPIR, which performs significantly faster for smaller objects. On the other hand, the output of CKP's equality operator does not allow it to utilize the plaintext packing optimization provided by SealPIR, thus requiring more computation in the PIR step. However, as objects grow bigger, the difference between the PIR cost of Pantheon and CKP gets smaller (§4.5.2). A more comprehensive comparison between FastPIR and SealPIR is available in [7].

4.5.2 Single-machine latency

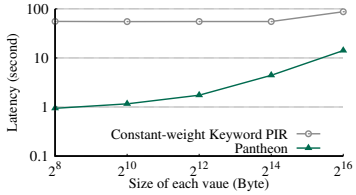
Factors affecting Pantheon's latency. We conduct experiments varying three size-related parameters: the number of tuples in the key-value store (n), the size of each key, and the size of each value. Pantheon's server-side latency for a Get query comprises the processing time for three operations: i) query expansion (§4.3.5), ii) equality check (§4.3.4), and iii) private information retrieval (§4.3.4). We briefly discuss how the three



(a) Variation of latency with number of tuples. The size of each key is 32 bits, and each value is 256 bytes.



(b) Variation of latency with key-size. CKP does not support keys larger than 60 bits, and so is shown for 32-bit and 60-bit keys. The key-value store contains 32,768 tuples, where each value is 256 bytes.



(c) Variation of latency with value-size. The key-value store contains 32,768 tuples, where each key is 32 bits.

Figure 4.4: Latency incurred by a single-machine server to perform a GET query with the variation of three size related parameters.

parameters affect the processing time for each operation. The processing time for query expansion depends on the number of ciphertexts obtained after expansion. Therefore, this time increases linearly with the size of each key but is independent of the number of tuples (n) and the size of each value. The processing time for the equality check depends on the total size of the array of keys K. Therefore, this time increases linearly both with the size of each key and the number of keys in K, but is independent of the size of each value. Finally, the processing time for PIR depends on the size of the value-array V and is independent of the size of each key. As explained in FastPIR [7], Pantheon's PIR time increases linearly with the size of each value, and linearly with a slope smaller than 1 with the number of tuples. These relationships will be reflected in our experimental results discussed later.

Varying the number of tuples. Figure 4.4a shows how the server-side latency of both Pantheon and Constant-weight Keyword PIR (CKP) depend on the number of tuples (n) in the key-value store. We vary the number of tuples from $16,384$ to $65,536$ while keeping the size of each key 32 bits and each value 256 bytes. In general, Pantheon's latency is much lower than that of CKP. For instance, when $n = 65,536$, CKP's server-

side latency for serving one GET query is 107.8 seconds, whereas the latency for Pantheon is 1.15 seconds, a 93× improvement. Also, the baseline latency increases almost linearly from 28.4 seconds for $n = 16,384$ to 107.8 seconds for $n = 65,536$. On the other hand, Pantheon's latency grows from 0.62 seconds for $n = 16,384$ to 1.15 seconds for $n = 65,536$, an increase of only 1.85×, whereas the number of tuples (n) increases 4×. This slower growth of latency is because the time for query expansion remains constant as the number of tuples increases, and the time for FastPIR increases with a slope smaller than 1.

Varying the key size. Figure 4.4b shows how the latency for Pantheon and the baseline change with the size of each key. For Pantheon, we vary the size of each key from 32-bit to 256-bit. However, the Constant-weight Keyword PIR implementation does not support key-size larger than 60-bit. So, we run CKP for 32-bit and 60-bit keys. We take 32,768 tuples in the key-value store for both systems, with each value being 256 bytes. For 32-bit keys, the latency for Pantheon is 0.94 seconds, 59× better than that of the baseline latency of 55.48 seconds. Pantheon's latency increases to 3.69 seconds for 256-bit keys.

Varying the value size. We populate the key-value store with $n = 32,768$ tuples while keeping the size of each key 32-bits. We vary the size of each value from 256 bytes to 65,536 bytes. Figure 4.4c shows how the latency of Pantheon and CKP vary with the size of each value. Pantheon's latency for 256-byte values is 0.94 seconds and increases to 1.74 seconds for 4,096-byte values, and then to 13.28 seconds for 65,536-byte values. Only the private information retrieval (PIR) step at the Pantheon server is affected by the size of values. The PIR processing time increases linearly while the processing times for query expansion and equality check remain unchanged with the size of values. Therefore,

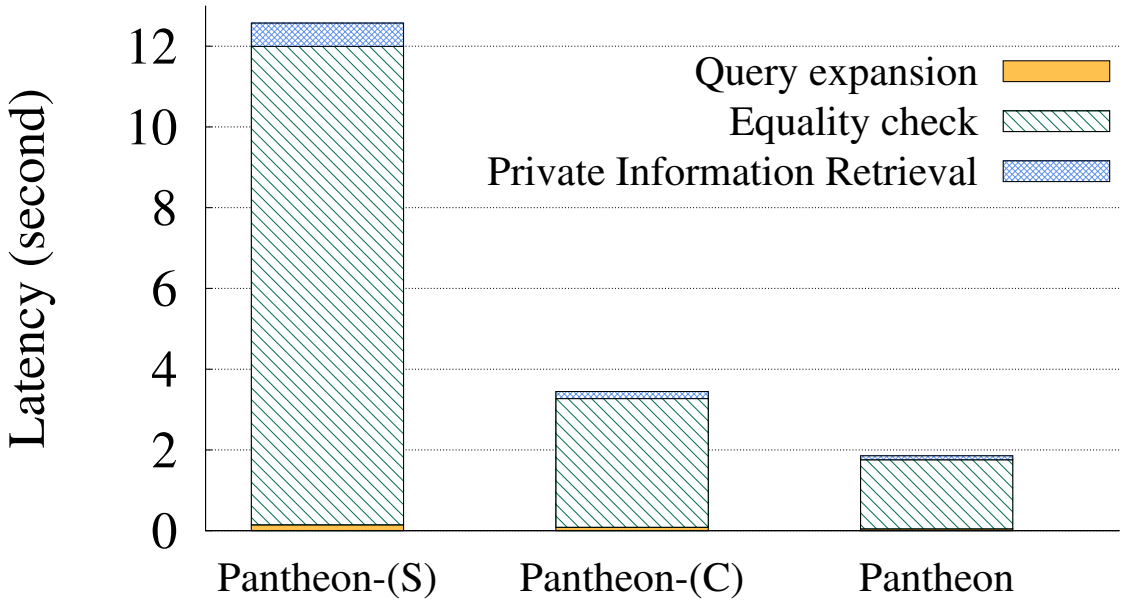
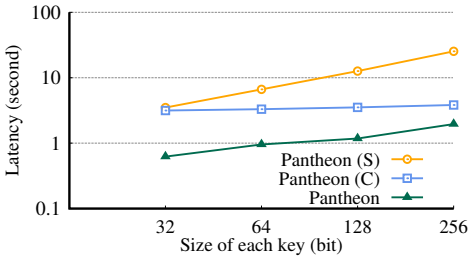


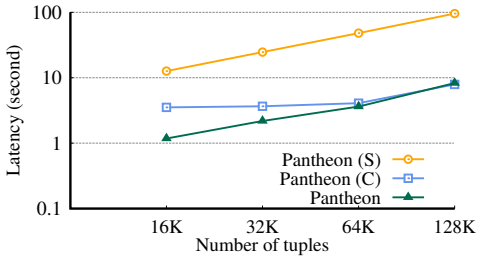
Figure 4.5: Impact of the two levels of parallelization on Pantheon's three server-side operations. The key-value store has 32,768 tuples, with 64 bit keys and 256 byte values. Pantheon-(S) is the single-threaded implementation with no parallelization, Pantheon-(C) includes only coarse-grain parallelization, and Pantheon is the final version containing both coarse-grain and fine-grain parallelization.

for Pantheon, variation in the size of value primarily reflects the performance of FastPIR. The baseline latency with the variation of the value-size remains constant at 55.4 seconds for values up to $16,384$ bytes and then increases to 87.47 seconds when the values are $65,536$ bytes. This is because CKP's PIR computation increases following a step function with every 20KB increase in value-size.

Effect of parallelization. We apply two levels of parallelization on the single-threaded implementation Pantheon-(S). First, we apply coarse-grain parallelization on Pantheon-(S) to obtain Pantheon-(C). Then, we apply fine-grain parallelization on top of Pantheon-(C) to get the final version of Pantheon. Figure 4.5 shows the impact of these two levels of parallelization on a key-value store containing $32,768$ tuples with 64-bit keys and 256-byte values. Evidently, the equality check takes the bulk of the server-side processing time. For coarse-grain parallelization, we horizontally partition the key array into two



(a) Variation of latency with key-size, number of tuples fixed at 16,384.



(b) Variation of latency with number of tuples, key-size fixed at 128 bits.

Figure 4.6: Impact of parallelization on single machine server-side latency. Pantheon-(S) is the single-threaded implementation with no parallelization, Pantheon-(C) includes only coarse-grain parallelization, and Pantheon is the final version containing both coarse-grain and fine-grain parallelization. In all cases, the value-size is 256 bytes.

parts, each containing $16,384$ tuples. We further split each key into two slices, each of 32-bits. This is the smallest partition size ($16,384$-tuples by 32-bits) allowed by Pantheon's BFV parameters (§4.4). Coarse-grain parallelization reduces the equality check time from 11.85 seconds to 3.19 seconds, an improvement of 3.7×. Our fine-grain parallelization of the Microsoft SEAL library further reduces the equality check time by a factor of 1.9× to 1.71 seconds. Recall that only a subset of the computations in Microsoft SEAL [187] can be executed in parallel for fine-grain parallelization. Hence, the sequential parts are the bottleneck for reducing the latency further.

Figure 4.6a shows the performance gains due to parallelization when the number of tuples in the key-value store equals the minimum horizontal partition size of $16,384$. We vary the size of each key from 32 bits to 256 bits. The performance gain of Pantheon-(C) over Pantheon-(S) depends on the number of partitions that the key-value store can be divided into. For 32-bit keys, Pantheon-(C) cannot partition the key-array, so the latency is 3.15 seconds, which is very close to the Pantheon-(S) latency of 3.5 seconds. The parallelization of the PIR operation reduces the latency by 0.35 seconds. The total latency, in this case, is dominated by the 2.98 seconds required for the equality check

operation, which cannot be reduced further by coarse-grain parallelization. We use fine-grain parallelization of the Microsoft SEAL library to reduce the processing time of the equality check operation to 0.55 seconds. The total latency obtained by fine-grain parallelization for 32-bit keys is 0.62 seconds, an improvement of 5× over Pantheon-(C). As the key-size increases, Pantheon-(C) can partition the key-array into multiple parts and use multiple cores in parallel for the equality check operation. So, the latency of Pantheon-(C) grows slowly to 3.83 seconds for 256-bit keys, and its difference with the final version of Pantheon reduces gradually.

Figure 4.6b shows the impact of parallelization on a heavier workload. We keep the key-size at 128-bit, value-size at 256-byte, and vary the number of tuples from $16,384$ to $131,072$. Pantheon-(C)'s latency for $16,384$ tuples is 3.53 seconds and increases slowly to 4.08 seconds for $65,536$ tuples. After that, the latency increases to 7.91 seconds for $131,072$ tuples. We speculate that the machine gets saturated at this point. On the other hand, Pantheon's implementation with both the coarse-grain and fine-grain parallelization has a latency of 1.18 seconds for $16,384$ tuples and 3.63 seconds for $65,536$ tuples, lower than the corresponding latencies for Pantheon-(C). However, Pantheon's latency for $131,072$ tuples is 8.29 seconds, marginally higher than Pantheon-(C). So, once the machine gets saturated for Pantheon-(C), the overhead of fine-grain parallelization makes the performance of Pantheon worse. This result imposes an interesting design decision. If low latency is the primary goal, then the final version of Pantheon is preferable by distributing Pantheon's workload over a large number of machines while keeping the partition size at each machine smaller. On the other hand, if reducing CPU cost is of primary interest, Pantheon-(C) may yield a better result with larger partitions over fewer machines.

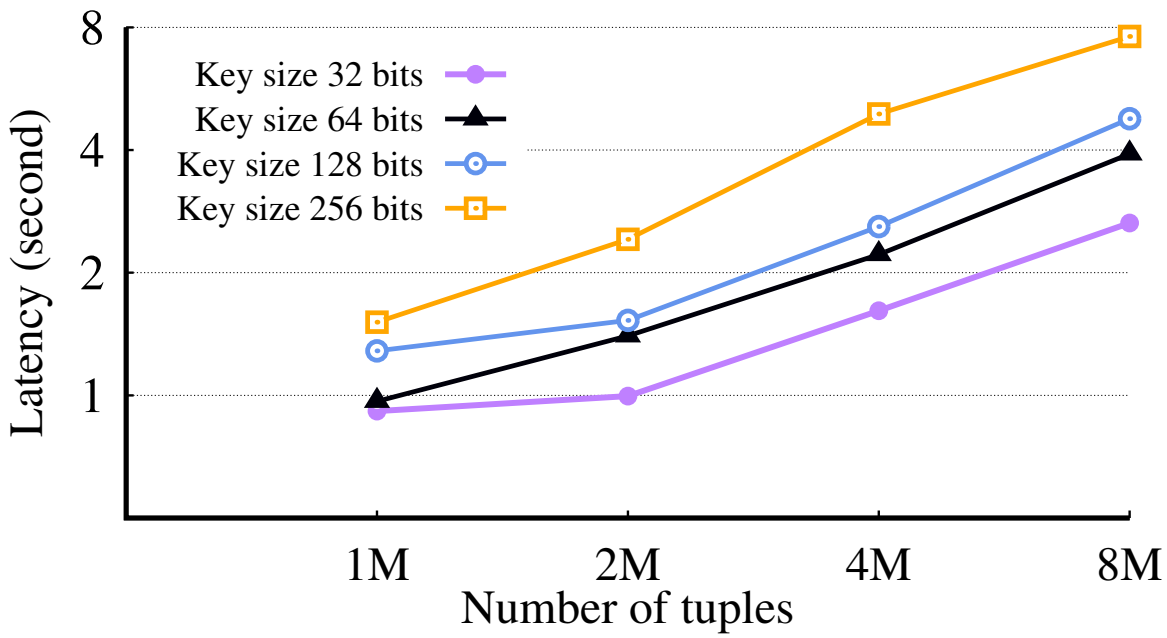


Figure 4.7: Server-side latency with variations in the number of tuples when the Pantheon server is deployed over a cluster of 128 worker machines. Each curve shows the latency trend for a different key-size, while the value-size is fixed at 256 bytes.

4.5.3 Cluster latency

Figure 4.7 shows Pantheon's latency trend when its server is distributed over a cluster of 128 worker instances. We vary the number of tuples from $1M$ to $8M$, the size of each key from 32-bit to 256-bit, and keep the size of each value at 256 bytes. Pantheon's final version containing both coarse-grain and fine-grain parallelization gives the lowest latency for all these experimental configurations. For the cluster deployment, the server-side latency consists of two components: i) processing time at the worker and ii) coordination overhead. The processing time depends on the size of the partition processed by each worker. The coordination overhead comprises the time required to broadcast the compressed query ciphertext to all the workers and the time to aggregate the PIR responses from each worker. The coordination overhead is fixed for a particular cluster configuration because the query size and the PIR response size from each worker

remain the same irrespective of the size of the key-value store. Pantheon's server-side latency increases with both the number of tuples and the size of each key. For example, for $2M$ tuples, the latency for 32-bit keys is 0.99 seconds and increases to 2.42 seconds for 256-bit keys. The corresponding processing times are 0.69 seconds and 2.12 seconds, respectively, while the coordination overhead for both is 0.3 seconds. If the number of tuples is increased to $8M$ while keeping the key-size at 256-bit, the latency rises to 7.6 seconds. Out of this, 7.3 seconds are for processing, and the coordination overhead is 0.3 seconds. The processing time increases by 3.4×, slightly slower than the 4× increase in the number of tuples, mainly because of a slower growth of PIR time (§4.5.2).

4.5.4 Resource overheads of Pantheon

This section discusses the overheads Pantheon imposes on its clients and estimates the dollar cost of a private GET query.

Client-side overhead. The client-side overhead of Pantheon stays fixed irrespective of the size of the key-value store. A Pantheon client incurs a CPU time of 0.07 seconds for retrieving a value privately from the Pantheon server. This CPU time comprises the time for encrypting a compressed query and decoding the PIR response from the server. A Pantheon client uploads 3MiB for each query and downloads 1.5MiB for each response.

Dollar cost. We convert the server-side CPU time and network usage for performing a private GET query to a dollar amount. Amazon EC2 charges $0.744 per hour for each worker instance (`c5.12xlarge`) and $1.488 per hour for the coordinator instance (`c5.24xlarge`) [191]. We use these unit costs to calculate the total cost for 128 workers and a coordinator for the duration of serving a query. For network usage, we use Amazon's pricing model of $0.05 per GiB download (Amazon does not charge for uploads) [190]. For

Pantheon, the cost for performing a private GET query over a key-value store containing $1M$ tuples, where each key is 256-bit, and each value is 256-byte, sums up to 4 cents. This cost rises to 20 cents when there are $8M$ tuples in the key-value store.

4.6 Related work

Querying over private data. There is a large body of work in the literature that allows a client to outsource its private data and perform queries on it. These works require the client to encrypt their data using their secret keys and store the encrypted data in the cloud server. The server then uses different techniques to serve the client's query while preserving privacy. CryptDB [171] allows a client to encrypt its data using multiple layers of encryption (onion encryption) and supports a subset of SQL queries over the encrypted data. Other works use techniques such as order preserving encryption [1], homomorphic encryption [223, 182], and searchable encryption [39] to hide clients' data. Several research works [172, 211, 124, 22, 237] and industry deployments [19, 20, 21, 202] provide encrypted database service using Trusted Execution Environment (TEE), such as Intel SGX [53]. In this setup, the client stores the encrypted data in the server and shares the secret key with trusted hardware located at the server. The trusted hardware decrypts the data while executing a client query. However, these works do not hide the access pattern of the client's query and are vulnerable to inference attacks that can retrieve the data in its plaintext form [30, 155, 96, 95, 131]. Arx [170] uses a combination of homomorphic encryption and garbled circuit to hide query access pattern over encrypted data. Oblidb [74] and Oblix [148] hide query access pattern over private data using TEE and ORAM [89]. PANCAKE [94] uses frequency smoothing to hide query access pattern on a private key-value store. Pantheon's problem domain is different from these works since Pantheon deals with *public* data, and the client cannot encrypt the data

as required in these solutions. Conceptually, one may develop a solution to hide access pattern over a public key-value store using TEE [145], where a client encrypts the query and the server decrypts it inside a trusted enclave. However, this approach guarantees a weaker privacy since TEEs are susceptible to side-channel attacks [137, 186, 208, 207], cache attacks [92, 35, 56], and fault injection attacks [153, 43] that can reveal the client's secret key and thus break the privacy.

Querying over public data The problem of searching privately over public data falls in the domain of Private Information Retrieval (PIR) [44, 46, 126]. In its basic form, PIR allows a client to obliviously retrieve the element at a particular index of a data array located at an untrusted server. Existing applications of PIR [3, 93, 7, 16, 6, 99, 150, 142] work in a setting where the client knows the index of the desired element in advance. Pantheon works in a different setting. A Pantheon client does not know whether the desired key exists in the key-value store, let alone its index. An extension of PIR, known as keyword PIR [45], allows a client to retrieve an element corresponding to a keyword using $\lceil \log_2(n+1) \rceil + 1$ round-trip interactions between the client and the server [16]. However, this technique deviates from Pantheon's goal of single round value retrieval (§4.2.3). The primary challenge for retrieving value in a single round is checking the equality between two keys. A body of work [8, 9, 69, 82] uses Fully Homomorphic Encryption (FHE) [81, 33] to implement an oblivious equality check operator between keys and can perform value retrieval in a single round. However, they involve prohibitively expensive operations and thus incur very high latency. A recent work by Mahdavi et al. [144] improves over the existing FHE-based equality checking by devising a new equality operator named constant-weight equality operator. This equality operator requires mapping each key to a constant-weight codeword, i.e., binary strings containing the same number of 1's. They use this equality operator and the

SealPIR [13] library to develop a single-round solution for the private key-value retrieval problem. Pantheon's approach aligns with this work. Pantheon uses a faster equality check operator that relies on Fermat's little theorem [177] and FastPIR [76] for better performance.

4.7 Conclusion

Providing privacy for clients querying a *public* key-value store is challenging because clients have no control over the data. Therefore, the access pattern of a client's query must be hidden from the server to guarantee privacy. Prior work that provides query privacy to a client either requires multiple rounds for each retrieval or has performance and scalability bottlenecks. This paper presents Pantheon, a round-optimal solution for private retrieval from a public key-value store, that scales to millions of tuples. When the Pantheon server is deployed over a cluster of 128 machines with a key-value store containing 2M tuples, where each key is 32 bits and each value is 256 bytes, the server-side latency for serving a client's query privately is under one second. Pantheon, for the first time, shows that it is possible to provide strong privacy for querying over a practically large public key-value store with reasonable latency.

Chapter 5

Concluding remarks

Privacy has become a major concern as the widespread use of internet services exposes users to escalating risks. The increasing reliance on digital platforms means that personal information is constantly at risk of being compromised. While significant advancements in privacy-enhancing technologies and cryptographic techniques have been developed to address these concerns, their integration into large-scale internet services has remained limited. One of the main obstacles is that the direct application of these privacy solutions often results in significant performance trade-offs, such as increased computational overhead and delays. These drawbacks create a tension between achieving strong privacy protections and maintaining the scalability needed for real-world systems.

This dissertation delves into this challenge, aiming to find ways to reduce the tension between *privacy* and *scalability*. By exploring new design principles and approaches, it seeks to develop systems that not only provide robust privacy guarantees but also perform efficiently at scale, making them suitable for widespread use in everyday internet applications. This dissertation demonstrates that substantial improvements in performance—by orders of magnitude—can be achieved through thoughtful system redesign and creative use of cryptographic techniques. It does so by describing three key projects:

Addra (§2), Coeus (§3), and Pantheon (§4). These projects illustrate how optimizing system architecture and tailoring the application of cryptographic methods can drastically enhance efficiency, overcoming the traditional trade-offs between privacy and scalability.

Addra (§2) is a voice communication system that can hide voice call metadata (such as the knowledge of who is talking to whom, and the time and duration of a call) even in the presence of a strong adversary that can compromise the service provider and the entire communication infrastructure. Notably, Addra supports metadata-private voice calls for 32K users, a substantial improvement over previous solutions limited to a few tens of users. Addra achieves its performance and scalability through an innovative system design that leverages a privacy construct known as Private Information Retrieval (PIR). Additionally, Addra incorporates a new and improved PIR solution called FastPIR [76], which significantly improves its performance. FastPIR has been utilized in various other privacy preserving solutions, including storage systems [142], and private querying on graph neural networks (GNNs) from untrusted servers [140].

Coeus (§3) addresses another widely used application domain of ranking documents based on keywords and retrieving them, all while ensuring strong privacy for the client. Ranking query results is critical in many applications where the clients do not know which element to retrieve a priori. For example, a user may search the Wikipedia repository with certain keywords, and then read an article from the top-ranked matches. Coeus addresses this problem by ensuring privacy throughout the entire process of ranking documents based on search keywords and subsequent retrieval. First, Coeus introduces a novel three round protocol in contrast to traditional two round protocols, decoupling the ranking operation from the retrieval and thus significantly reducing computation overhead. Then, through a series of refinements, it scales secure matrix-vector multiplication, which is needed for the ranking part, to matrices with hundreds of billions of elements, in contrast to the state-of-the-art prior work that scales to matrices with millions of elements. Thus,

Coeus enables users to privately query Wikipedia's 5 million documents in 3.9 seconds, a significant improvement compared to 93.9 seconds without its optimizations, bringing the solution to a practical realm.

Pantheon (§4) extends the Private Information Retrieval (PIR) interface to support key-value retrieval. While index-based PIR allows for private retrieval from public data, it necessitates that the client knows the index of the desired element on the server, which is often impractical in real-world scenarios. To overcome this limitation, Pantheon enables clients to perform private retrieval from an untrusted server's key-value store using a keyword of interest, thereby enhancing usability and privacy. The privacy guarantee is that the server or any other adversary does not learn anything about the key or the value retrieved. This generic solution is useful to guarantee client privacy in many real life online services such as querying price of a stock from a stock exchange database and streaming a movie by its title from a service such as Netflix. Pantheon overcomes limitations of previous solutions by supporting dynamic databases and achieves a 93× latency improvement over prior work. Pantheon is designed for efficient parallel processing, allowing it to be implemented in an embarrassingly parallelizable manner. This parallelized implementation makes Pantheon the first system to support private retrieval from a million-row key-value database with a sub-second latency.

In conclusion, this dissertation highlights the critical need to balance privacy and scalability in the design of modern internet services. As digital platforms continue to evolve, addressing the challenges of protecting personal information while ensuring high performance is essential. By emphasizing innovative approaches and strategic system redesign, it demonstrates that significant advancements can be made without compromising user privacy. The findings not only contribute to the academic discourse on privacy-enhancing technologies but also offer practical solutions that can be integrated into real-world applications, ultimately fostering a safer digital environment for users.

Chapter 6

Future directions

This chapter explores potential future research directions in scalable privacy-preserving systems, building on the projects presented in the previous chapters. The discussion is organized into two main areas: 1) Further extending the PIR (Private Information Retrieval) query interface. 2) Supporting privacy-preserving queries across data stored in separate, siloed environments. The following sections discuss them in detail.

6.1 Further extending the PIR query interface.

The current scope of the Private Information Retrieval (PIR) domain is primarily limited to point queries. In index-based PIR [44, 126, 3, 13, 7], users can retrieve a single element from a specified index in the database, while keyword-based PIR [46, 162, 144, 5] allows the retrieval of values associated with a specific keyword. However, there are several open problems and opportunities for extending the capabilities of PIR.

One potential direction for future research is to enable more expressive queries, such as private aggregation and range queries over public data. As an example, consider the following SQL query:

```
SELECT AGGREGATE_FUNCTION(column_name)
FROM table_name
WHERE column_name1 COMPARISON_OPERATOR value1
AND/OR column_name2 COMPARISON_OPERATOR value2
AND/OR ...
```

Here, `AGGREGATE_FUNCTION` refers to functions such as `COUNT`, `SUM`, and `AVG`. In the `WHERE` clause of the SQL query, comparison operators such as $<, \leq, >, \geq, =, \neq$ can be used to compare a query parameter with the values in a column.

Aggregation functions can exploit the homomorphic properties of encryption, allowing operations such as summing or counting elements to extend seamlessly from existing PIR computations. Additionally, the optimized secure matrix-vector multiplication construct discussed in Coeus (§3) can be applied to these aggregation tasks. However, range queries introduce more complexity. The size of the response in a range query typically depends on the number of rows that satisfy the predicates in the `WHERE` clause, resulting in a response size that varies between queries. This variability can unintentionally expose information about the query itself, necessitating techniques to obfuscate the response size to prevent sensitive data leakage. Another major challenge in supporting range queries is performing oblivious inequality checks. While recent studies [175, 29, 236] have proposed algorithms using homomorphic encryption for inequality comparisons, these methods often depend on bitwise encryption of operands, which introduces significant performance and scalability challenges, rendering them impractical for large-scale systems. Efficiently implementing private range queries over public data remains an open problem and a promising area for future research. Advancements in cryptography, combined with more refined system designs, could lead to practical and scalable solutions to this problem. Progress in this domain could greatly enhance the capabilities of PIR systems, enabling a broader range of secure and scalable query types.

6.2 Privacy-preserving federated analytics.

A promising direction for future research based on the content of this dissertation is enabling private querying across data from *multiple* data owners. In many real-world scenarios, related datasets are distributed across multiple institutions [168]. For example, a patient's medical records may reside in a hospital's electronic medical record (EMR) system, while laboratory test results are stored in external medical labs. A clinician or researcher might wish to query these combined datasets to gain comprehensive insights. However, strict privacy policies at both institutions often prevent data from being shared or transferred outside their respective premises. This challenge extends beyond simple querying and also applies to more complex tasks like training machine learning models on siloed datasets. For instance, training a model to detect medical conditions using both clinical notes and X-ray images stored across different institutions requires secure and private collaboration. A potential approach is federated analytics [228, 234], where the analysis or model training is distributed across multiple data owners without requiring the data to leave their premises. Although federated analytics has gained significant momentum in recent years, it still faces challenges related to privacy, scalability, and data heterogeneity [122].

Privacy concerns in this domain are multifaceted. Recent attacks have highlighted vulnerabilities not only in the data itself but also in model parameters and analysis results. Ensuring strong privacy requires advanced techniques like differential privacy, homomorphic encryption [105, 42], and secure multiparty computation. While these methods can offer robust privacy protections, they often suffer from inefficiencies and scalability limitations, leaving room for further research into developing solutions that are both scalable and efficient without compromising privacy. Data heterogeneity poses another significant challenge in federated settings. Data can be partitioned in various

ways across institutions—horizontally, vertically, or even in hybrid formats—and each type of partition requires distinct strategies for secure collaboration [227, 97, 98, 139, 224]. Addressing the complexities introduced by different types of data partitioning will be critical to building flexible and effective federated systems. These challenges provide a rich avenue for future research, offering opportunities to create scalable, privacy-preserving solutions that work across diverse and distributed datasets.

Appendix A

Security Analysis

A.1 Addra Security proof

In this appendix, we show that Addra's protocol (§2.3.2) meets the relationship unobservability property (§2.2.1). Our proof follows the proof technique of Pung [17, Appendix C], whose protocol also works over completely untrusted infrastructure and meets relationship unobservability (although with higher message latencies and overheads). We first define an abstract protocol for metadata private voice communication, then describe a cryptographic security game that captures the relationship unobservability property, and finally show why an adversary cannot win the game with non-negligible probability when the abstract protocol is instantiated with the Addra protocol.

A.1.1 An abstract protocol

We define an abstract protocol for metadata private voice communication using three algorithms: $\textsc{Init}(1^\lambda)$, $\textsc{RetrAbstract}(i, j)$, and $\textsc{SendAbstract}(i, j, m_{i \to j})$.

$\textsc{Init}(1^\lambda)$ takes as input a security parameter and initializes the state of the protocol for each participant of the protocol. In particular, it establishes the content encryption

key between pairs of user devices that communicate via the system.

RetrAbstract(i, j) takes as input a user identifier i for the caller and a user identifier j for the callee, and generates a retrieval request for messages sent to i by j.

SendAbstract$(i, j, m_{i \to j})$ takes as input a user identifier i for the caller, a recipient identifier j for the callee, and a message (which may be $\perp$ if sender is idle) that the caller wants to send to the callee, and outputs a tuple that is sent to the server.

The protocol proceeds in rounds, each of which consists of $t \geq 1$ subrounds. At the beginning of a round, each participant calls the Init algorithm. Then, each participant calls the RetrAbstract algorithm to generate a request to indicate that it wants to get messages from its peer. Finally, each participant calls the SendAbstract algorithm t times to send t messages to its peer.

A.1.2 The security game

We define a security game that a challenger and an adversary play that captures the relationship unobservability property. We denote this game as $\mathcal{G}^b_{\mathcal{A},\pi,K,t}(1^\lambda)$, where $\mathcal{A}$ is the adversary, π is a round-based protocol consisting of the Init, RetrAbstract, and SendAbstract algorithms, K is the number of correct users (not controlled by the adversary), t is the number of subrounds (messages exchanged per round of the protocol π), and λ is a security parameter. The game has three phases: setup, simulation, and guess. At a high level, during setup the adversary creates two scenarios for the challenger. The challenger then during the simulation phase runs the protocol π for one of the scenarios selected randomly (that is, the b-th scenario) and sends the transcript of the protocol messages due to π to the adversary. Finally, the adversary in the guess phase guesses which scenario the challenger simulated. The adversary wins the game if the guess is correct. We expand on these three phases next.

Setup phase. During setup, the adversary supplies two scenarios M^0 and M^1 to the challenger. Each scenario has K tuples corresponding to the actions of K correct (non-compromised) users.

Each tuple has an entry for the send and the retrieve part of the protocol. For the k-th ($k \leq K$) tuple, that is, for the k-th correct user, the send part of the b-th scenario, $M^b[k].send = (i, j^b, \{m^b_{i\to j}\}_t)$ specifies that the user device with id i should send the set of t messages $\{m^b_{i\to j}\}_t$ to user with id j^b. The messages could be dummy ($\perp$). Similarly, for the k-th tuple, the retrieve part of the b-th scenario $M^b[k].retr = (i, \ell^b)$ specifies that the user with id i should retrieve messages from the user with id ℓ^b.

The adversary constructs the two scenarios M^0 and M^1 and supplies them to the challenger. However, the adversary has three restrictions on how it can construct the scenarios. First, both scenarios must have the same number of entries describing the actions of each user in every round. Second, both scenarios should describe the send and retrieve actions of correct users only. This is required because the challenger can simulate the actions of correct users only. Third, if j^b or ℓ^b is a compromised user, then that *send* or *retr* entry should be identical across the two scenarios. Recall that relationship unobservability gives guarantees only for correct pairs of users, so such a restriction is necessary.

Simulation phase. During the simulation phase, the challenger simulates a protocol on one of the two scenarios that it picks randomly using a coin flip. The challenger uses the Simulate function described in Figure A.1. The challenger then sends the output of the algorithm to the adversary.

Note that in the simulation algorithm, the challenger calls the functions exposed by the adversary. For instance, it calls GetMailboxIDs to learn the mailbox IDs the adversary assigns to the users. Similarly, the challenger calls GetMailboxAccessTokens

GetNumMailboxes to learn mailbox access tokens and the total number of mailboxes. Finally, the challenger calls the GetResponse function to learn the output of a sub-round, if any, for a correct user. These functions give the adversary an opportunity to set these untrusted parts of the protocol. For instance, GetResponse allows adversary to drop requests or reorder them or compute the responses to requests incorrectly.

Guess phase. In the guess phase, the adversary $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ whether the challenger simulated the M^0 or M^1 scenario. The adversary wins the game if its guess is correct, that is, $b = b'$.

A.1.3 Proof

We want to show that the adversary's advantage in winning the game described above is negligible in the security parameter λ when the abstract protocol is instantiated with the Addra protocol. We use a series of hybrid games to calculate the adversary's advantage.

Game 0. This game is the original game as described above with π instantiated with the Addra protocol. That is, the Init algorithm establishes a shared secret between pairs of users. Denote the secret between users with id i and j as $key_{i,j}$. The RetrAbstract abstract algorithm is instantiated with the generation of the CPIR query in Figure 2.2. Specifically, RetrAbstract(i, j) calls $q \leftarrow$ Query$(\mathcal{M}_j, \mathcal{N}_i)$, where $\mathcal{M}_i$ is a mailbox ID provided by the adversary for user j, and $\mathcal{N}_i$ is the total number of mailboxes advertised by the adversary to user i. In case, $\mathcal{N}_i < \mathcal{M}_j$, then the challenger generates CPIR query for index zero. The SendAbstract abstract algorithm is instantiated using the Send function in Figure 2.2. Specifically, SendAbstract$(i, j, m_{i\to j})$ calls Send$(\mathcal{M}_i, \mathcal{T}_i, m_{i\to j}, key_{i,j})$, where $\mathcal{M}_i$ and $\mathcal{T}_i$ are the mailbox ID and access token pro-

vided by the adversary for user i, and $key_{i,j}$ is the shared secret established between users i and j.

Game 1. This game is same as game 0 except that the SendAbstract invokes Send over random messages rather than the ones specified in the scenario.

Game 2. This game is same as game 1 except that the Retr procedure generates query for a random mailbox ID. That is, it outputs a CPIR query for a random index (mailbox id) rather than $\mathcal{M}_j$ returned by GetMailboxIDs.

Let S_0 be the event that $b = b'$ in game 0, where M_b is the scenario chosen by the challenger, and b' is the guess made by the adversary. Similarly, let S_1 be the event that $b = b'$ in game 1, and S_2 be the event that $b = b'$ in game 2.

Lemma A.1.1 $Pr[S_2] = 1/2$.

Observe that in game 2 none of the requests or responses sent to the adversary depend on the information supplied in a scenario. Specifically, the SendAbstract function generates encryptions of random messages, and similarly RetrAbstract generates a CPIR request for a random index. Therefore, an adversary participating in game 2 cannot distinguish between the two scenarios.

Lemma A.1.2 $|Pr[S_1] - Pr[S_2]| \leq \epsilon_{CPIR}$

The difference between game 1 and game 2 is the input to the RetrAbstract algorithm. Specifically, in game 2, the index to CPIR query is random while it is the one supplied in the scenario in game 1. However, given the security of CPIR, an adversary cannot tell with non-negligible probability the index encoded in a CPIR query.

Lemma A.1.3 $|Pr[S_0] - Pr[S_1]| \leq \epsilon_{Enc}$

The difference between game 0 and game 1 is the input to the SendAbstract algorithm. Particularly, in game 1, the challenger inputs a random message while in game 0 it inputs the messages supplied in the scenario. Here, there are two sub-cases: the recipient is honest or the recipient is under the control of an adversary. If the recipient is honest, then given that the adversary does not have content encryption keys, it cannot distinguish between ciphertexts for the two scenarios (follows from the indistinguishability of ciphertexts). If the recipient is compromised, then it has the content encryption keys, but the recipient is the same across both scenarios and receives the same messages (see the restriction on scenario creation above). Again, the adversary cannot distinguish between the two scenarios.

Combining the three lemmas, we get the proof that $|\Pr[S_0] - 1/2]| \leq \epsilon_{Enc} + \epsilon_{CPIR}$. That is, the adversary does not win the security game with non-negligible probability.

A.2 Coeus Security proof

This appendix is included to show that Coeus's protocol (§3.3.3, §3.4) for oblivious document ranking and retrieval satisfies the notion of query privacy (§3.2.2).

We first define an abstract protocol for oblivious document ranking and retrieval, then describe a cryptographic security game that captures the notion of query privacy, and finally show why an adversary cannot win this game with non-negligible probability when the abstract protocol is instantiated with Coeus's protocol (§3.3.3).

A.2.1 An abstract description of protocol

A protocol for oblivious document ranking and retrieval runs between a server and a client. The server begins with a data structure for scoring document relevance given a client query, a metadata library containing n metadata objects for n documents, and

a document library that contains the n documents packed into n_{pkd} equal-sized objects, where a document does not span more than one object. Meanwhile, the client begins with a multi-keyword search query q. At the end of the protocol, the client receives one object in the document library.

This protocol consists of three algorithms: SQuery, MQuery, and DQuery.

SQuery$(1^\lambda, \text{Dict}, q)$ takes as inputs a security parameter 1^λ, a dictionary of keywords Dict, and a multi-keyword search query q, and outputs a query q_s for scoring relevance of q against all n documents in the server's document library.

MQuery$(1^\lambda, n, \{idx_1, \ldots, idx_K\})$ takes as input a security parameter 1^λ, an integer n that represents the number of objects in the server's metadata library, and a set of K indices whose values are between (and inclusive of) 1 and n, and outputs a query q_m that is suitable for retrieving objects in the metadata library whose indices are those in the set $\{idx_1, \ldots, idx_K\}$.

DQuery$(1^\lambda, n_{pkd}, idx)$ takes as input a security parameter 1^λ, an integer n_{pkd} representing the number of objects in a suitable encoding of the document library, and an index whose value is between 1 and n_{pkd}, and outputs a query q_d for retrieving one of the objects from the document library.

The protocol proceeds in three rounds. In the first round, the client runs the SQuery algorithm and sends q_s to the server. The server responds with an answer which consists of relevance scores for n documents in the server's document library. The client processes these scores to extract the value n and a set of K indices corresponding to the highest scoring documents. In the second round, the client feeds the outputs from the first round as inputs to MQuery, and sends the output q_m to the server. The server processes this query and returns metadata for K documents. The server also returns the number of objects n_{pkd} in the encoded document library. The client postprocesses the metadata returned by the server to obtain an integer idx whose value is between 1 and n_{pkd}. Finally,

in the third round, the client runs DQuery by feeding the outputs of the second step as inputs to DQuery. The client sends the output of DQuery to the server, who processes it against the document library to return one object from this library.

A.2.2 The security game for query privacy

We define a security game that a challenger and an adversary play that captures the notion of query privacy. We denote this game as $\mathcal{G}_{\mathcal{A},\pi,\text{Dict},K}(1^\lambda)$, where $\mathcal{A}$ is an probabilistic polynomial time adversary, π is a protocol for oblivious document ranking and retrieval consisting of the three algorithms of SQuery, MQuery, and DQuery, Dict is a set of keywords, K is an integer greater than or equal to 1 that represents the number of metadata objects a client wants to get, and λ is a security parameter. The game has three phases: setup, simulation, and guess.

During setup, the adversary chooses two multi-keyword queries q_0 and q_1 and sends them to the challenger. These queries may or may not contain keywords in the Dict.

During simulation, the challenger simulates the protocol π for one of the queries. The challenger first flips a coin and picks either q_0 or q_1 depending on the outcome of coin flip ($b \in \{0, 1\}$). It then simulates π for q_b using the Simulate2 function described in Figure A.2. Finally, the challenger shares the output of simulate with the adversary.

During the final guess phase, the adversary takes the output of Simulate2 corresponding to the scenario which the challenger simulated and outputs b', which is the adversary's guess for whether the challenger simulated the protocol for q_0 or q_1. The adversary wins the game if $b' = b$.

We note that the Simulate2 algorithm calls several functions exposed by the adversary. For instance, it calls the GetScores function to learn the relevance scores for q_s. Similarly, it calls the GetMetadata and GetDocument functions to retrieve K meta-

data objects and one object from the document library. The adversary may arbitrarily misbehave when responding to these calls. For instance, when replying to GetScores, it may or may not send the actual scores for the scoring query. It may even send back less or more number of scores than n, which is the number of documents in the server library. Similarly, GetMetadata may return an incorrect value of n_{pkd}, and incorrect number or incorrect content of metadata objects.

A.2.3 Proof of query privacy

We want to show that the adversary's advantage in winning the game is negligible when π is instantiated with Coeus's protocol. We use a series of hybrid games to calculate the adversary's advantage.

Game 0: This game is the original game as described above with π instantiated with Coeus's protocol. The SQuery converts q to a binary vector using the dictionary of keywords Dict and then encrypts this binary vector using a secure-matrix vector product primitive (§3.3.2, §3.4). The MQuery algorithm calls the query generation function of a multi-retrieval PIR query (e.g., [13]) with n as the total number of objects in the library and the K indices as the positions of the metadata objects client wants to retrieve. The DQuery algorithm calls the query generation function of a single-retrieval PIR with n_{pkd} as the number of objects in the library and idx as the position of the object that is retrieved.

Game 1: This game is the same as game 0 except that the DQuery algorithm calls the query generation function of a single-retrieval PIR with n_{pkd} and an index sampled uniformly at random from the range 1 to n_{pkd}.

Game 2: This game is the same as game 1 except that the MQuery algorithm calls the query generation function of a multi-retrieval PIR with K indices sampled uniformly

at random from the set $\{1, \ldots, n\}$.

Game 3: This game is the same as game 2 except that the SQuery algorithm calls the request generation algorithm of secure matrix-vector product with a binary vector of length |Dict| whose each element is sampled uniformly at random from $\{0, 1\}$.

Let S_0 be the event that $b = b'$ in Game 0, where q_b is the query chosen by the challenger, and b' is the adversary's guess. Similarly, let S_1 be the event $b = b'$ in Game 1, S_2 be the event $b = b'$ in Game 2, and S_3 be the event that $b = b'$ in Game 3.

Lemma A.2.1 $Pr[S_3] = 1/2$.

Observe that in game 3 none of the requests to the adversary depend on the query picked by the challenger. Specifically, DQuery and MQuery generate PIR queries for uniformly sampled indices, and SQuery generates a query for a uniformly sampled binary vector. Therefore, an adversary participating in game 3 cannot distinguish between the two scenarios.

Lemma A.2.2 $|Pr[S_2] - Pr[S_3]| \leq \epsilon_{Secure-matrix-vec}$

The difference between game 3 and game 2 is the input to secure matrix-vector product. Specifically, in game 3, the input is dependent on the actual query selected by the challenger, while in game 2, it is a uniformly sampled Boolean vector. But given the security of secure matrix-vector product (which in turn depends on the semantic security of an underlying encryption scheme), the adversary cannot differentiate the two cases with non-negligible probability.

Lemma A.2.3 $|Pr[S_1] - Pr[S_2]| \leq \epsilon_{Multi-retrieval-PIR}$

The difference between game 2 and game 1 are the indices input to multi-retrieval CPIR: in game 2, the indices are sampled uniformly at random while in game 1 they are dependent on the scores returned by the query-scoring round. However, given the security of

multi-retrieval CPIR that hides the value of these indices, the adversary cannot distinguish between the two games with non-negligible probability.

Lemma A.2.4 $|Pr[S_0] - Pr[S_1]| \leq \epsilon_{Single-retrieval-PIR}$

The difference between game 1 and game 0 is the index input to single-retrieval PIR: in game 1, the index is sampled uniformly at random while in game 0 it is dependent on the metadata returned by metadata-retrieval round. Again, given the security of single-retrieval CPIR that hides the value of the index, the adversary cannot distinguish between the two games with non-negligible probability.

Combining the four lemmas, we get the proof that $|Pr[S0]-1/2| <= \epsilon_{Secure-matrix-vec}+ \epsilon_{Multi-retrieval-PIR} + \epsilon_{single-retrieval-PIR}$. Therefore, an adversary cannot win the security game with non-negligible probability.

```
1: function SIMULATE(A, π, K, t, M^b)
2:     requests ← {}
3:     responses ← {}
4:     // Initialize content encryption key across parties
5:     // K contains the shared secrets (keys)
6:     K ← π.INIT(1^λ)

7:     // Obtain mailbox IDs, access tokens, number of mailboxes
8:     // M contains the assigned mailbox IDs
9:     M ← A.GETMAILBOXIDS()
10:    // T contains the assigned tokens
11:    T ← A.GETMAILBOXACCESSTOKENS()
12:    // N contains the number of advertised mailboxes
13:    N ← A.GETNUMMAILBOXES()

14:    // Run retrieve part of protocol
15:    for k = 0 to K − 1
16:        (i, j) ← M^b[k].retr
17:        req ← π.RETRABSTRACT(i, j)
18:        requests ←(insert)— req

19:    // Run send part of protocol
20:    for r = 0 to t − 1
21:        for k = 0 to K − 1
22:            (i, j, {m_{i→j}}_t) ← M^b[k].send
23:            req ← π.SENDABSTRACT(i, j, {m_{i→j}}_r)
24:            requests ←(insert)— req
25:            resp ← A.GETRESPONSE()
26:            responses ←(insert)— resp
27:    return (requests, responses)
```

Figure A.1: Pseudocode for the challenger to simulate a scenario supplied by the adversary in Addra's protocol.

1: **function** SIMULATE2($(\mathcal{A}, \pi, \text{DICT}, K, q_b)$)
2: // Run query-scoring round of protocol
3: $q_s \leftarrow \pi.\text{SQUERY}(1^\lambda, \text{DICT}, q_b)$
4: $scores \leftarrow \mathcal{A}.\text{GETSCORES}(q_s)$
5: $n \leftarrow |scores|$ // n is the number of documents
6: // Obtain indices for the highest scoring documents
7: // if $K > n$, TOP-K fills missing values randomly
8: $\{idx_1, \ldots, idx_K\} \leftarrow \text{TOP-K}(scores)$

9: // Run the metadata-retrieval round of the protocol
10: $q_m \leftarrow \pi.\text{MQUERY}(1^\lambda, n, \{idx_1, \ldots, idx_K\})$
11: $(n_{pkd}, \{\mathcal{M}_{idx_1}, \ldots, \mathcal{M}_{idx_K}\}) \leftarrow \mathcal{A}.\text{GETMETADATA}(q_m)$
12: // Process metadata to get an integer between 1 and n_{pkd}
13: $idx \leftarrow \text{SELECTDOCUMENT}(n_{pkd}, \{\mathcal{M}_{idx_1}, \ldots, \mathcal{M}_{idx_K}\})$

14: // Run the document-retrieval round of the protocol
15: $q_d \leftarrow \pi.\text{DQUERY}(1^\lambda, n_{pkd}, idx)$
16: $obj \leftarrow \mathcal{A}.\text{GETDOCUMENT}(q_d)$

17: return all messages sent to or received from $\mathcal{A}$

Figure A.2: Pseudocode for the challenger to simulate the protocol π for one of the queries supplied by the adversary in Coeus's protocol.

Bibliography

[1] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. "Order preserving encryption for numeric data". In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 2004, pp. 563–574.

[2] Shashank Agrawal, Saikrishna Badrinarayanan, Pratyay Mukherjee, and Peter Rindal. "Game-Set-MATCH: Using Mobile Devices for Seamless External-Facing Biometric Matching". In: *ACM Conference on Computer and Communications Security (CCS)*. 2020, pp. 1351–1370.

[3] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. "XPIR: Private information retrieval for everyone". In: *Privacy Enhancing Technologies Symposium (PETS)* 2016.2 (2016), pp. 155–174.

[4] Daniel Agun, Jinjin Shao, Shiyu Ji, Stefano Tessaro, and Tao Yang. "Privacy and efficiency tradeoffs for multiword top k search with linear additive rank scoring". In: *International World Wide Web Conference (WWW)*. 2018, pp. 1725–1734.

[5] Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. "Pantheon: Private Retrieval from Public Key-Value Store". In: *Proceedings of the VLDB Endowment* 16.4 (2022), pp. 643–656.

[6] Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. "Coeus: A System for Oblivious Document Ranking and Retrieval". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2021, pp. 672–690.

[7] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. "Addra: Metadata-private voice communication over fully untrusted infrastructure". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2021.

[8] Adi Akavia, Dan Feldman, and Hayim Shaul. "Secure data retrieval on the cloud: Homomorphic encryption meets coresets". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), pp. 80–106.

[9] Adi Akavia, Craig Gentry, Shai Halevi, and Max Leibovich. "Setup-Free Secure Search on Encrypted Data: Faster and Post-Processing Free". In: *Proceedings on Privacy Enhancing Technologies* 3 (2019), pp. 87–107.

[10] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. "Homomorphic encryption standard". In: *Protecting Privacy through Homomorphic Encryption*. Springer, 2021, pp. 31–62.

[11] Josh Allen. *Recent Cyber Attacks & Data Breaches In 2021*. 2022-12-14. 2021. URL: https://purplesec.us/recent-cyber-security-attacks/.

[12] Artak Amirbekyan and Vladimir Estivill-Castro. "A new efficient privacy-preserving scalar product protocol". In: *The Australasian Data Mining Conference (AusDM)*. 2007.

[13] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. "PIR with compressed queries and amortized query processing". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2018, pp. 962–979.

[14] Sebastian Angel, Sampath Kannan, and Zachary Ratliff. "Private resource allocators and their applications". In: *IEEE Symposium on Security and Privacy (S&P)*. 2020.

[15] Sebastian Angel, David Lazar, and Ioanna Tzialla. "What's a little leakage between friends?" In: *Workshop on Privacy in the Electronic Society (WPES)*. 2018.

[16] Sebastian Angel and Srinath Setty. "Unobservable communication over fully untrusted infrastructure". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2016, pp. 551–569.

[17] Sebastian Gomez Angel. "Unobservable communication over untrusted infrastructure". PhD thesis. The University of Texas at Austin, 2018.

[18] Julia Angwin, Charlie Savage, Jeff Larson, Henrik Moltke, Laura Poitras, and James Risen. "AT&T helped US spy on Internet on a vast scale". In: *The New York Times* (2015).

[19] Panagiotis Antonopoulos, Arvind Arasu, Kunal D Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, et al. "Azure SQL database always encrypted". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1511–1525.

[20] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. "Orthogonal Security with Cipherbase". In: *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 2013.

[21] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. "Transaction processing on confidential data using cipherbase". In: *2015 IEEE 31st International Conference on Data Engineering*. IEEE. 2015, pp. 435–446.

[22] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. "SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution". In: *USENIX Conference on File and Storage Technologies (FAST)*. 2019, pp. 173–190.

[23] Ero Balsa, Carmela Troncoso, and Claudia Diaz. "OB-PWS: Obfuscation-based private web search". In: *IEEE Symposium on Security and Privacy (S&P)*. 2012, pp. 491–505.

[24] Michael Barbaro and Tom Zeller Jr. "A Face Is Exposed for AOL Searcher No. 4417749". In: *The New York Times* (Aug. 2006).

[25] Ludovic Barman, Italo Dacosta, Mahdi Zamani, Ennan Zhai, Apostolos Pyrgelis, Bryan Ford, Joan Feigenbaum, and Jean-Pierre Hubaux. "PriFi: Low-Latency Anonymity for Organizational Networks". In: *Privacy Enhancing Technologies Symposium (PETS)* 2020.4 (2020), pp. 24–47.

[26] Brian Barrett. "Security news this week: Russia's SolarWinds hack is a historic mess". In: *Wired* (Dec. 2020). https://www.wired.com/story/russia-solarwinds-hack-roundup/.

[27] Amos Beimel, Yuval Ishai, and Tal Malkin. "Reducing the servers computation in private information retrieval: PIR with preprocessing". In: *Advances in Cryptology—CRYPTO*. 2000, pp. 55–73.

[28] John Bethencourt, Dawn Song, and Brent Waters. "New techniques for private stream searching". In: *ACM Transactions on Information and System Security (TISSEC)* 12.3 (2009), pp. 1–32.

[29] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. "HE3DB: An Efficient and Elastic Encrypted Database Via Arithmetic-And-Logic Fully Homomorphic Encryption". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023, pp. 2930–2944.

[30] Vincent Bindschaedler, Paul Grubbs, Cornell Tech, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. "The Tao of Inference in Privacy-Protected Databases". In: *Proceedings of the VLDB Endowment* 11.5 (2018), pp. 1715–1728.

[31] Nikita Borisov, George Danezis, and Ian Goldberg. "DP5: A private presence service". In: *Privacy Enhancing Technologies Symposium (PETS)* (2015), pp. 4–24.

[32] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. "A survey of provably secure searchable encryption". In: *ACM Computing Surveys (CSUR)* 47.2 (2014), pp. 1–51.

[33] Zvika Brakerski. "Fully homomorphic encryption without modulus switching from classical GapSVP". In: *Advances in Cryptology—CRYPTO*. Springer. 2012, pp. 868–886.

[34] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014), pp. 1–36.

[35] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. "Software grand exposure: SGX cache attacks are practical". In: *USENIX Workshop on Offensive Technologies*. 2017.

[36] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. "Verifying computations with state". In: *ACM Symposium on Operating Systems Principles (SOSP)*. Nov. 2013.

[37] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. "Touching from a distance: Website fingerprinting attacks and defenses". In: *ACM Conference on Computer and Communications Security (CCS)*. 2012, pp. 605–616.

[38] *Cambridge Analytica and Facebook: The Scandal and the Fallout So Far (Published 2018) — nytimes.com*. https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html. [Accessed 19-09-2024].

[39] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. "Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation". In: *Network and Distributed System Security Symposium (NDSS)*. 2014, pp. 1–16.

[40] David Chaum. "The dining cryptographers problem: Unconditional sender and recipient untraceability". In: *Journal of cryptology* 1.1 (1988), pp. 65–75.

[41] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[42] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 395–412.

[43] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D Garcia. "VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface". In: *USENIX Security Symposium (SEC)*. 2021, pp. 699–716.

[44] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. "Private Information Retrieval". In: *Symposium on Foundations of Computer Science (FOCS)*. 1995, pp. 41–50.

[45] Benny Chor, Niv Gilboa, and Moni Naor. *Private information retrieval by keywords*. Cryptology ePrint Archive, Report 1998/003. https://eprint.iacr.org/1998/003. 1998.

[46] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private information retrieval". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 965–981.

[47] David Cole. “Is Privacy Obsolete?” In: *The Nation* (Mar. 2015).

[48] David Cole. “We Kill People Based on Metadata”. In: *The New York Review* (May 2014).

[49] *Constant-weight PIR v0.1*. https://github.com/RasoulAM/constant-weight-pir. 2022.

[50] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. “Riposte: An anonymous messaging system handling millions of users”. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2015, pp. 321–338.

[51] Henry Corrigan-Gibbs and Bryan Ford. “Dissent: Accountable anonymous group messaging”. In: *ACM Conference on Computer and Communications Security (CCS)*. 2010, pp. 340–350.

[52] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. “Proactively accountable anonymous messaging in Verdict”. In: *USENIX Security Symposium (SEC)*. 2013, pp. 147–162.

[53] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Report 86. 2016. URL: https://eprint.iacr.org/2016/086.pdf.

[54] Ronald Cramer and Victor Shoup. “Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack”. In: *SIAM Journal on Computing* 33.1 (2003), pp. 167–226.

[55] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. “Searchable symmetric encryption: improved definitions and efficient constructions”. In: *Journal of Computer Security* 19.5 (2011), pp. 895–934.

[56] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. “Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2 (2018), pp. 171–191.

[57] George Danezis and Claudia Diaz. “Space-efficient private search with applications to rateless codes”. In: *International Conference on Financial Cryptography and Data Security (FC)*. Springer. 2007, pp. 148–162.

[58] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. “DORY: An Encrypted Search System with Distributed Trust”. In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2020, pp. 1101–1119.

[59] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. “Unique in the crowd: The privacy bounds of human mobility”. In: *Scientific reports* 3.1 (2013), pp. 1–5.

[60] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. "SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage". In: *USENIX Security Symposium (SEC)*. 2020, pp. 2433–2450.

[61] Daniel Demmler, Amir Herzberg, and Thomas Schneider. "RAID-PIR: Practical multi-server PIR". In: *ACM Workshop on Cloud Computing Security (CCSW)*. 2014, pp. 45–56.

[62] Casey Devet. *Evaluating private information retrieval on the cloud*. Tech. rep. University of Waterloo, 2013.

[63] Casey Devet and Ian Goldberg. "The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency". In: *Privacy Enhancing Technologies Symposium (PETS)*. Springer. 2014, pp. 63–82.

[64] Casey Devet, Ian Goldberg, and Nadia Heninger. "Optimally robust private information retrieval". In: *USENIX Security Symposium (SEC)*. 2012, pp. 269–283.

[65] Xuhua Ding, HweeHwa Pang, and Junzuo Lai. "Verifiable and private top-k monitoring". In: *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 2013, pp. 553–558.

[66] Zak Doffman. *China's Hackers Accused Of 'Mass-Scale Espionage' Attack On Global Cellular Networks — forbes.com*. https://www.forbes.com/sites/zakdoffman/2019/06/25/chinese-government-suspected-of-major-hack-on-10-global-phone-companies-reports/. [Accessed 19-09-2024].

[67] Josep Domingo-Ferrer, Agusti Solanas, and Jordi Castellà-Roca. "h(k)-private information retrieval from privacy-uncooperative queryable databases". In: *Online Information Review* (2009).

[68] Changyu Dong and Liqun Chen. "A fast secure dot product protocol with application to privacy preserving association rule mining". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. 2014.

[69] Yarkın Doröz, Berk Sunar, and Ghaith Hammouri. "Bandwidth efficient PIR from NTRU". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 195–207.

[70] *Driver's Licenses, Addresses, Photos: Inside How TikTok Shares User Data (Published 2023) — nytimes.com*. https://www.nytimes.com/2023/05/24/technology/inside-how-tiktok-shares-user-data-lark.html. [Accessed 19-09-2024].

[71] Cynthia Dwork. "Differential privacy". In: *International Colloquium on Automata, Languages, and Programming*. 2006, pp. 1–12.

[72] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. "End-to-end encrypted messaging protocols: An overview". In: *Internet Science: Third International Conference, INSCI 2016, Florence, Italy, September 12-14, 2016, Proceedings 3*. Springer. 2016, pp. 244–254.

[73] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. "Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy". In: *USENIX Security Symposium (SEC)*. 2021.

[74] Saba Eskandarian and Matei Zaharia. "ObliDB: Oblivious Query Processing for Secure Databases". In: *Proceedings of the VLDB Endowment* 13.2 (2019), pp. 169–183.

[75] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. 2012. URL: https://eprint.iacr.org/2012/144.pdf.

[76] *FastPIR: An efficient computational private information retrieval (CPIR) library*. https://github.com/ishtiyaque/FastPIR. 2021.

[77] Russell A Fink, David R Zaret, Rachel B Stonehirsch, Robert M Seng, and Samantha M Tyson. "Streaming, Plaintext Private Information Retrieval Using Regular Expressions on Arbitrary Length Search Strings". In: *IEEE Symposium on Privacy-Aware Computing (PAC)*. IEEE. 2017, pp. 107–118.

[78] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. "Keyword search and oblivious pseudorandom functions". In: *Theory of Cryptography Conference*. Springer. 2005, pp. 303–324.

[79] Brian Fung. *Analysis: There is now some public evidence that China viewed TikTok data — CNN Business — cnn.com*. https://www.cnn.com/2023/06/08/tech/tiktok-data-china/index.html. [Accessed 19-09-2024].

[80] *gensim – Topic Modelling in Python*. https://github.com/RaRe-Technologies/gensim/.

[81] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *ACM Symposium on Theory of Computing (STOC)*. 2009, pp. 169–178.

[82] Craig Gentry and Shai Halevi. "Compressible FHE with applications to PIR". In: *Theory of Cryptography Conference*. Springer. 2019, pp. 438–464.

[83] Craig Gentry, Shai Halevi, and Nigel P Smart. "Homomorphic evaluation of the AES circuit". In: *Advances in Cryptology—CRYPTO*. 2012.

[84] Torsten George. "Takeaways From the Shopify Hack". In: *SecurityWeek* (Sept. 2020). https://www.securityweek.com/takeaways-shopify-hack.

[85] Allen Gersho. "Principles of quantization". In: *IEEE Transactions on circuits and systems* 25.7 (1978), pp. 427–436.

[86] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. *Herbivore: A scalable and efficient protocol for anonymous communication*. Tech. rep. Cornell University, 2003.

[87] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. "On private scalar product computation for privacy-preserving data mining". In: *International Conference on Information Security and Cryptology (ICISC)*. 2004.

[88] Oded Goldreich. "Secure multi-party computation". In: *Manuscript. Preliminary version* 78.110 (1998), pp. 1–108.

[89] Oded Goldreich and Rafail Ostrovsky. "Software protection and simulation on oblivious RAMs". In: *Journal of the ACM (JACM)* 43.3 (1996), pp. 431–473.

[90] David Goldschlag, Michael Reed, and Paul Syverson. "Onion routing". In: *Communications of the ACM* 42.2 (1999), pp. 39–41.

[91] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[92] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. "Cache attacks on Intel SGX". In: *Proceedings of the 10th European Workshop on Systems Security*. 2017, pp. 1–6.

[93] Matthew Green, Watson Ladd, and Ian Miers. "A protocol for privately reporting ad impressions at scale". In: *ACM Conference on Computer and Communications Security (CCS)*. 2016, pp. 1591–1601.

[94] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. "PANCAKE: Frequency smoothing for encrypted data stores". In: *USENIX Security Symposium (SEC)*. 2020, pp. 2451–2468.

[95] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. "Learning to reconstruct: Statistical learning theory and encrypted database attacks". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2019, pp. 1067–1083.

[96] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. "Leakage-abuse attacks against order-revealing encryption". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2017, pp. 655–672.

[97] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. "Privacy-Preserving Asynchronous Federated Learning Algorithms for Multi-Party Vertically Collaborative Learning. CoRR abs/2008.06233 (2020)". In: *arXiv preprint arXiv:2008.06233* (2020).

[98] Hanlin Gu, Jiahuan Luo, Yan Kang, Lixin Fan, and Qiang Yang. "FedPass: privacy-preserving vertical federated deep learning with adaptive obfuscation". In: *arXiv preprint arXiv:2301.12623* (2023).

[99] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. "Scalable and private media consumption with Popcorn". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2016, pp. 91–107.

[100] Trinabh Gupta, Henrique Fingler, Lorenzo Alvisi, and Michael Walfish. "Pretzel: Email encryption and provider-supplied functions are compatible". In: *ACM SIGCOMM Conference*. 2017.

[101] Shai Halevi and Victor Shoup. "Algorithms in HElib". In: *Advances in Cryptology—CRYPTO*. Springer. 2014, pp. 554–571.

[102] Shai Halevi and Victor Shoup. "Faster homomorphic linear transformations in HElib". In: *Advances in Cryptology—CRYPTO*. Springer. 2018, pp. 93–120.

[103] Rohit Handa, C Rama Krishna, and Naveen Aggarwal. "Document clustering for efficient and secure information retrieval from cloud". In: *Concurrency and Computation: Practice and Experience* 31.15 (2019), e5127.

[104] Rohit Handa, C Rama Krishna, and Naveen Aggarwal. "Searchable encryption: A survey on privacy-preserving search schemes on encrypted outsourced data". In: *Concurrency and Computation: Practice and Experience* 31.17 (2019), e5201.

[105] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption". In: *arXiv preprint arXiv:1711.10677* (2017).

[106] Todd Haselton. *Credit reporting firm Equifax says data breach could potentially affect 143 million US consumers — cnbc.com*. https://www.cnbc.com/2017/09/07/credit-reporting-firm-equifax-says-cybersecurity-incident-could-potentially-affect-143-million-us-consumers.html. [Accessed 19-09-2024].

[107] *HElib v2.0.0 Release*. https://github.com/homenc/HElib/tree/master/examples/BGV_country_db_lookup/. 2021.

[108] Ryan Henry, Yizhou Huang, and Ian Goldberg. "One (Block) Size Fits All: PIR and SPIR with Variable-Length Records via Multi-Block Queries." In: *Network and Distributed System Security Symposium (NDSS)*. 2013.

[109] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A Yavuz. "Hardware-supported ORAM in effect: Practical oblivious search and update on very large dataset". In: *Privacy Enhancing Technologies Symposium (PETS)* 2019.1 (2019).

[110] Aaron Holmes. "533 million Facebook users' phone numbers and personal data have been leaked online". In: *Business Insider* (Apr. 2021). https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4.

[111] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. "How much anonymity does network latency leak?" In: *ACM Transactions on Information and System Security (TISSEC)* 13.2 (2010), pp. 1–28.

[112] http://www.nytimes.com/by/james-risen. *AT&T Helped U.S. Spy on Internet on a Vast Scale (Published 2015) — nytimes.com*. https://www.nytimes.com/2015/08/16/us/politics/att-helped-nsa-spy-on-an-array-of-internet-traffic.html. [Accessed 19-09-2024].

[113] Sal Humphreys and Melissa De Zwart. "Data retention, journalist freedoms and whistleblowers". In: *Media International Australia* 165.1 (2017), pp. 103–116.

[114] Ayad Ibrahim, Hai Jin, Ali A Yassin, and Deqing Zou. "Secure rank-ordered search of multi-keyword trapdoor over encrypted cloud data". In: *2012 IEEE Asia-Pacific Services Computing Conference*. IEEE. 2012, pp. 263–270.

[115] Ilia Iliashenko and Vincent Zucca. "Faster homomorphic comparison operations for BGV and BFV". In: *Privacy Enhancing Technologies Symposium (PETS)* 3 (2021), pp. 246–264.

[116] *India's Biggest Data Leak So Far? Covid-19 Test Info of 81.5Cr Citizens With ICMR Up for Sale — Exclusive - News18 — news18.com*. https://www.news18.com/india/indias-biggest-data-leak-so-far-covid-19-test-info-of-81-5cr-citizens-with-icmr-up-for-sale-exclusive-8637743.html. [Accessed 19-09-2024].

[117] Intel. *Intel Software Guard Extensions*. https://software.intel.com/en-us/sgx.

[118] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. "A secure protocol for computing dot-products in clustered and distributed environments". In: *International Conference on Parallel Processing (ICPP)*. 2002.

[119] ITU-T. *G.114 : One-way transmission time*. https://www.itu.int/rec/T-REC-G.114-200305-I/en. 2003.

[120] Shiyu Ji, Jinjin Shao, Daniel Agun, and Tao Yang. "Privacy-aware ranking with tree ensembles on the cloud". In: *International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 315–324.

[121] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. "Gazelle: A low latency framework for secure neural network inference". In: *USENIX Security Symposium (SEC)*. 2018, pp. 1651–1669.

[122] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. "Advances and open problems in federated learning". In: *Foundations and trends® in machine learning* 14.1–2 (2021), pp. 1–210.

[123] Murizah Kassim, Ruhani Ab Rahman, Mohamad Azrai A Aziz, Azlina Idris, and Mat Ikram Yusof. "Performance analysis of VoIP over 3G and 4G LTE network". In: *2017 International Conference on Electrical, Electronics and System Engineering (ICEESE)*. IEEE. 2017, pp. 37–41.

[124] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. "Shieldstore: Shielded in-memory key-value storage with SGX". In: *ACM European Conference on Computer Systems (EuroSys)*. 2019, pp. 1–15.

[125] Lea Kissner, Alina Oprea, Michael K Reiter, Dawn Song, and Ke Yang. "Private keyword-based push and pull with applications to anonymous communication". In: *International Conference on Applied Cryptography and Network Security (ACNS)*. 2004, pp. 16–30.

[126] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is not needed: Single database, computationally-private information retrieval". In: *Symposium on Foundations of Computer Science (FOCS)*. 1997, pp. 364–373.

[127] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. "Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services". In: *USENIX Security Symposium (SEC)*. 2015, pp. 287–302.

[128] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. "Atom: Horizontally scaling strong anonymity". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2017, pp. 406–422.

[129] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. "Riffle: An efficient communication system with strong anonymity". In: *Privacy Enhancing Technologies Symposium (PETS)* 2016.2 (2016), pp. 115–134.

[130] Albert Kwon, David Lu, and Srinivas Devadas. "XRD: Scalable Messaging System with Cryptographic Privacy". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2020, pp. 759–776.

[131] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. "Improved reconstruction attacks on encrypted data using range query leakage". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2018, pp. 297–314.

[132] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Karaoke: Distributed private messaging immune to passive traffic analysis". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2018, pp. 711–725.

[133] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Yodel: Strong metadata security for voice calls". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2019, pp. 211–224.

[134] David Lazar and Nickolai Zeldovich. "Alpenhorn: Bootstrapping secure communication without leaking metadata". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2016.

[135] Stevens Le Blond, David Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. "Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems". In: *ACM SIGCOMM Conference*. 2015, pp. 639–652.

[136] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. "Towards efficient traffic-analysis resistant anonymity networks". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 303–314.

[137] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. "Hacking in darkness: Return-oriented programming against secure enclaves". In: *USENIX Security Symposium (SEC)*. 2017, pp. 523–539.

[138] Robert Lenzner. "ATT, Verizon, Sprint are paid cash by NSA for your private communications". In: *Forbes* (2013).

[139] Songze Li, Duanyi Yao, and Jin Liu. "Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 20296–20311.

[140] Ling Liang, Jilan Lin, Zheng Qu, Ishtiyaque Ahmad, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. "SPG: Structure-Private Graph Database via SqueezePIR". In: *Proceedings of the VLDB Endowment* 16.7 (2023), pp. 1615–1628.

[141] *Libscapi - The Secure Computation API*. https://github.com/cryptobiu/libscapi.

[142] Jilan Lin, Ling Liang, Zheng Qu, Ishtiyaque Ahmad, Liu Liu, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. "INSPIRE: In-Storage Private Information Retrieval via Protocol and Architecture Co-design". In: *International Conference on Computer Architecture (ISCA)*. 2022, pp. 102–115.

[143] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On ideal lattices and learning with errors over rings". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2010, pp. 1–23.

[144] Rasoul Akhavan Mahdavi and Florian Kerschbaum. "Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators". In: *USENIX Security Symposium (SEC)*. 2022, pp. 1723–1740.

[145] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. "BITE: Bitcoin lightweight client privacy using trusted execution". In: *USENIX Security Symposium (SEC)*. 2019, pp. 783–800.

[146] Jonathan Mayer, Patrick Mutchler, and John C Mitchell. "Evaluating the privacy properties of telephone metadata". In: *Proceedings of the National Academy of Sciences* 113.20 (2016), pp. 5536–5541.

[147] Lindsay McKenzie. "Secure File Sharing Compromises University Security". In: *Inside Higher Ed* (Apr. 2021). https://www.insidehighered.com/news/2021/04/07/accellion-data-security-breach-latest-hit-universities.

[148] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. "Oblix: An efficient oblivious search index". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2018, pp. 279–296.

[149] Alan Mislove, Bimal Viswanath, Krishna P Gummadi, and Peter Druschel. "You are who you know: Inferring user profiles in online social networks". In: *International conference on Web search and data mining*. 2010, pp. 251–260.

[150] Prateek Mittal, Femi G Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval." In: *USENIX Security Symposium (SEC)*. 2011, p. 31.

[151] Mozilla. *LPCNet: Efficient neural speech synthesis*. https://github.com/mozilla/LPCNet.

[152] Steven J Murdoch and George Danezis. "Low-cost traffic analysis of Tor". In: *IEEE Symposium on Security and Privacy (S&P)*. 2005, pp. 183–195.

[153] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. "Plundervolt: Software-based fault injection attacks against Intel SGX". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2020, pp. 1466–1482.

[154] Condé Nast. *Ex-Googler Allegedly Spied on User E-Mails, Chats — wired.com*. https://www.wired.com/2010/09/google-spy/. [Accessed 19-09-2024].

[155] Muhammad Naveed, Seny Kamara, and Charles V Wright. "Inference attacks on property-preserving encrypted databases". In: *ACM Conference on Computer and Communications Security (CCS)*. 2015, pp. 644–655.

[156] Michael Oehler and Dhananjay S Phatak. "A Conjunction for Private Stream Searching". In: *International Conference on Social Computing*. IEEE. 2013, pp. 441–447.

[157] Office of the Director of National Intelligence. *Statistical Transparency Report Regarding the Use of National Security Authorities*. https://www.dni.gov/files/CLPT/documents/2020_ASTR_for_CY2019_FINAL.pdf. 2020.

[158] Femi Olumofin and Ian Goldberg. "Privacy-preserving queries over relational databases". In: *Privacy Enhancing Technologies Symposium (PETS)*. Springer. 2010, pp. 75–92.

[159] Femi Olumofin and Ian Goldberg. "Revisiting the computational practicality of private information retrieval". In: *International Conference on Financial Cryptography and Data Security (FC)*. 2011, pp. 158–172.

[160] *OpenMP API 5.2*. https://github.com/OpenMP. 2021.

[161] Cengiz Örencik and Erkay Savaş. "An efficient privacy-preserving multi-keyword search over encrypted cloud data with ranking". In: *Distributed and Parallel Databases* 32.1 (2014), pp. 119–160.

[162] Rafail Ostrovsky and William E Skeith. "Private searching on streaming data". In: *Journal of Cryptology* 20.4 (2007), pp. 397–430.

[163] Pascal Paillier. "Public-key cryptosystems based on composite degree residuosity classes". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 1999, pp. 223–238.

[164] *PALISADE Homomorphic Encryption Software Library*. https://palisade-crypto.org/.

[165] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. "Website fingerprinting in onion routing based anonymization networks". In: *Workshop on Privacy in the Electronic Society (WPES)*. 2011, pp. 103–114.

[166] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: Nearly practical verifiable computation". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2013, pp. 238–252.

[167] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management*. http://www.maroki.de/pub/dphistory/2010_Anon_Terminology_v0.34.pdf. 2010.

[168] Bjarne Pfitzner, Nico Steckhan, and Bert Arnrich. "Federated learning in a medical context: a systematic literature review". In: *ACM Transactions on Internet Technology (TOIT)* 21.2 (2021), pp. 1–31.

[169] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. "The Loopix anonymity system". In: *USENIX Security Symposium (SEC)*. 2017, pp. 1199–1216.

[170] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. "Arx: an encrypted database using semantically secure encryption". In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1664–1678.

[171] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. "CryptDB: protecting confidentiality with encrypted query processing". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2011, pp. 85–100.

[172] Christian Priebe, Kapil Vaswani, and Manuel Costa. "EnclaveDB: A secure database using SGX". In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE. 2018, pp. 264–278.

[173] Michael G Reed, Paul F Syverson, and David M Goldschlag. "Anonymous connections and onion routing". In: *IEEE Journal on Selected areas in Communications* 16.4 (1998), pp. 482–494.

[174] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". In: *LREC Workshop on New Challenges for NLP Frameworks*. May 2010, pp. 45–50.

[175] Xuanle Ren, Le Su, Zhen Gu, Sheng Wang, Feifei Li, Yuan Xie, Song Bian, Chao Li, and Fan Zhang. "HEDA: Multi-Attribute Unbounded Aggregation over Homomorphically Encrypted Database". In: *Proceedings of the VLDB Endowment* 16.4 (2022), pp. 601–614.

[176] Reuters. *TikTok admits using its app to spy on reporters in effort to track leaks — theguardian.com*. https://www.theguardian.com/technology/2022/dec/22/tiktok-bytedance-workers-fired-data-access-journalists. [Accessed 19-09-2024].

[177] Kenneth H Rosen. *Elementary number theory*. 2011.

[178] *rpclib - modern msgpack-rpc for C++*. http://rpclib.net/.

[179] Alan Rusbridger. "The Snowden Leaks and the Public". In: *The New York Review* (Nov. 2013).

[180] Dominic Rushe. "Yahoo $250,000 daily fine over NSA data refusal was set to double "every week"". In: *The Guardian* (2014).

[181] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[182] Stephen Tu M Frans Kaashoek Samuel and Madden Nickolai Zeldovich. "Processing Analytical Queries over Encrypted Data". In: *Proceedings of the VLDB Endowment* 6.5 (2013), pp. 289–300.

[183] Len Sassaman, Bram Cohen, and Nick Mathewson. "The Pynchon Gate: A secure method of pseudonymous mail retrieval". In: *Workshop on Privacy in the Electronic Society (WPES)*. 2005, pp. 1–9.

[184] Martin D Schatz, Robert A Van de Geijn, and Jack Poulson. "Parallel matrix multiplication: A systematic journey". In: *SIAM Journal on Scientific Computing* 38.6 (2016), pp. C748–C781.

[185] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge, 2008.

[186] Michael Schwarz and Daniel Gruss. "How trusted execution environments fuel research on microarchitectural attacks". In: *IEEE Security & Privacy* 18.5 (2020), pp. 18–27.

[187] *Microsoft SEAL (release 3.5)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Apr. 2020.

[188] *SealPIR: A computational PIR library that achieves low communication costs and high performance*. https://github.com/microsoft/SealPIR. Microsoft Research, Redmond, WA.

[189] Amazon Web Services. *Amazon EC2 Instance Savings Plans*. https://aws.amazon.com/savingsplans/compute-pricing/.

[190] Amazon Web Services. *Amazon EC2 On-Demand Pricing (Data transfer)*. https://aws.amazon.com/ec2/pricing/on-demand/. 2022.

[191] Amazon Web Services. *Amazon EC2 Reserved Instances Pricing*. https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/. 2022.

[192] Jinjin Shao, Shiyu Ji, Alvin Oliver Glova, Yifan Qiao, Tao Yang, and Tim Sherwood. "Index Obfuscation for Oblivious Document Retrieval in a Trusted Execution Environment". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 1345–1354.

[193] Jinjin Shao, Shiyu Ji, and Tao Yang. "Privacy-aware document ranking with neural signals". In: *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019, pp. 305–314.

[194] Emin Gün Sirer, Sharad Goel, Mark Robson, and Doğan Engin. "Eluding carnivores: File sharing with strong anonymity". In: *Proceedings of the ACM SIGOPS European workshop*. 2004.

[195] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data". In: *IEEE Symposium on Security and Privacy (S&P)*. 2000, pp. 44–55.

[196] National Institute of Standards and Technology (NIST). "Secure Hash Standard (SHS)". In: (Aug. 2015). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[197] Emil Stefanov, Elaine Shi, and Dawn Song. "Towards practical oblivious RAM". In: *Network and Distributed System Security Symposium (NDSS)*. 2012.

[198] Julien P Stern. "A new and efficient all-or-nothing disclosure of secrets protocol". In: *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. 1998, pp. 357–371.

[199] Mikhail Strizhov and Indrajit Ray. "Multi-keyword similarity search over encrypted cloud data". In: *IFIP international information security conference*. Springer. 2014, pp. 52–65.

[200] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking". In: *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 2013, pp. 71–82.

[201] Wenhai Sun, Ruide Zhang, Wenjing Lou, and Y Thomas Hou. "REARGUARD: Secure keyword search using trusted hardware". In: *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE. 2018, pp. 801–809.

[202] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. "Building enclave-native storage engines for practical encrypted databases". In: *Proceedings of the VLDB Endowment* 14.6 (2021), pp. 1019–1032.

[203] Paul Syverson, Roger Dingledine, and Nick Mathewson. "Tor: The second-generation onion router". In: *USENIX Security Symposium (SEC)*. 2004, pp. 303–320.

[204] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. "Stadium: A distributed metadata-private messaging system". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2017, pp. 423–440.

[205] Jean-Marc Valin and Jan Skoglund. "A real-time wideband neural vocoder at 1.6 kb/s using LPCNet". In: *arXiv preprint arXiv:1903.12087* (2019).

[206] Jean-Marc Valin and Jan Skoglund. "LPCNet: Improving neural speech synthesis through linear prediction". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019.

[207] Jo Van Bulck, Frank Piessens, and Raoul Strackx. "SGX-Step: A practical attack framework for precise enclave execution control". In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 2017, pp. 1–6.

[208] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution". In: *USENIX Security Symposium (SEC)*. 2017, pp. 1041–1056.

[209] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. "Vuvuzela: Scalable private messaging resistant to traffic analysis". In: *ACM Symposium on Operating Systems Principles (SOSP)*. 2015, pp. 137–152.

[210] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "Fully homomorphic encryption over the integers". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer. 2010, pp. 24–43.

[211] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. "StealthDB: a scalable encrypted database with full SQL query support". In: *Proceedings on Privacy Enhancing Technologies* (2019).

[212] Ivana Vojinovic. *Data Breach Statistics That Will Make You Think Twice Before Filling Out an Online Form*. 2022. URL: https://dataprot.net/statistics/data-breach-statistics/.

[213] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. "Secure ranked keyword search over encrypted cloud data". In: *International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2010, pp. 253–262.

[214] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. "Splinter: Practical private queries on public data". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2017, pp. 299–313.

[215] Yujue Wang, HweeHwa Pang, Yanjiang Yang, and Xuhua Ding. "Secure server-aided top-k monitoring". In: *Information Sciences* 420 (2017), pp. 345–363.

[216] Yunling Wang, Jianfeng Wang, and Xiaofeng Chen. "Secure searchable encryption: a survey". In: *Journal of communications and information networks* 1.4 (2016), pp. 52–65.

[217] Chengkun Wei, Qinchen Gu, Shouling Ji, Wenzhi Chen, Zonghui Wang, and Raheem Beyah. "OB-WSPES: A Uniform Evaluation System for Obfuscation-based Web Search Privacy". In: *IEEE Transactions on Dependable and Secure Computing* (2019).

[218] R Clint Whaley, Antoine Petitet, and Jack J Dongarra. "Automated empirical optimizations of software and the ATLAS project". In: *Parallel computing* 27.1-2 (2001), pp. 3–35.

[219] *Wikimedia Downloads: enwiki dump progress on 20210201*. https://dumps.wikimedia.org/enwiki/20210201/enwiki-20210201-pages-articles-multistream.xml.bz2/.

[220] *Wikipedia:Short description*. https://en.wikipedia.org/wiki/Wikipedia:Short_description#Formatting/.

[221] *Wikipedia:Wikipedia_records*. https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_records#Title_length/.

[222] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. "Dissent in numbers: Making strong anonymity scale". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2012, pp. 179–182.

[223] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. "Secure query processing with data interoperability in a cloud database environment". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014, pp. 1395–1406.

[224] Yuncheng Wu, Naili Xing, Gang Chen, Tien Tuan Anh Dinh, Zhaojing Luo, Beng Chin Ooi, Xiaokui Xiao, and Meihui Zhang. "Falcon: A privacy-preserving and interpretable vertical federated learning system". In: *Proceedings of the VLDB Endowment* 16.10 (2023), pp. 2471–2484.

[225] Karl Wüst, Sinisa Matetic, Moritz Schneider, Ian Miers, Kari Kostiainen, and Srdjan Čapkun. "Zlite: Lightweight clients for shielded zcash transactions using trusted execution". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2019, pp. 179–198.

[226] Zhihua Xia, Xinhui Wang, Xingming Sun, and Qian Wang. "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data". In: *IEEE transactions on parallel and distributed systems* 27.2 (2015), pp. 340–352.

[227] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, James Joshi, and Heiko Ludwig. "Fedv: Privacy-preserving federated learning over vertically partitioned data". In: *Proceedings of the 14th ACM workshop on artificial intelligence and security*. 2021, pp. 181–192.

[228] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[229] Xun Yi and Elisa Bertino. "Private searching for single and conjunctive keywords on streaming data". In: *Workshop on Privacy in the Electronic Society (WPES)*. 2011, pp. 153–158.

[230] Xun Yi, Elisa Bertino, Jaideep Vaidya, and Chaoping Xing. "Private searching on streaming data based on keyword frequency". In: *IEEE Transactions on Dependable and Secure Computing* 11.2 (2013), pp. 155–167.

[231] Xun Yi, Russell Paulet, and Elisa Bertino. "Private searching on streaming data". In: *Homomorphic Encryption and Applications*. Springer, 2014, pp. 101–126.

[232] Xun Yi and Chaoping Xing. "Private (t, n) threshold searching on streaming data". In: *International Conference on Privacy, Security, Risk and Trust and International Conference on Social Computing*. IEEE. 2012, pp. 676–683.

[233] Jiadi Yu, Peng Lu, Yanmin Zhu, Guangtao Xue, and Minglu Li. "Toward secure multikeyword top-k retrieval over encrypted cloud data". In: *IEEE transactions on dependable and secure computing* 10.4 (2013), pp. 239–250.

[234] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. "A survey on federated learning". In: *Knowledge-Based Systems* 216 (2021), p. 106775.

[235] Peng Zhang, Yan Li, Qingyun Liu, and Hailun Lin. "A Scalable Distributed Private Stream Search System". In: *International Conference on Distributed Computing Systems Workshops*. IEEE. 2015, pp. 128–135.

[236] Zhou Zhang, Song Bian, Zian Zhao, Ran Mao, Haoyi Zhou, Jiafeng Hua, Yier Jin, and Zhenyu Guan. "ArcEDB: An Arbitrary-Precision Encrypted Database via (Amortized) Modular Homomorphic Encryption". In: *Cryptology ePrint Archive* (2024).

[237] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. "Opaque: An oblivious and encrypted distributed analytics platform". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2017, pp. 283–298.

[238] Justin Zobel and Alistair Moffat. "Exploring the similarity space". In: *ACM SIGIR Forum*. Vol. 32. 1. 1998, pp. 18–34.