PRIVATE INFORMATION RETRIEVAL IN PRACTICE

Syed Mahbub Hafiz

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the School of Informatics, Computing, and Engineering,

Indiana University

January 2021

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Ryan Henry, PhD, Chair

Apu Kapadia, PhD

Funda Ergun, PhD

David Crandall, PhD

January 5, 2021

To my beloved wife, *Hafsa Akter*, whose countless sacrifices made this Ph.D. possible!

To my dear parents, *Syed Hafijul Islam* and *Siam Nesa*, who raised me and made me literate!

ACKNOWLEDGMENTS

Earning a Ph.D. degree is one of the most fascinating (and challenging) tasks in my life—I consider this is perfect opportunity to thank all who helped me by one measure or another.

First of all, I am incredibly indebted to my Ph.D. supervisor, Prof. Ryan Henry, for his effective advising, constant support, and helpful guidance on almost everything. I remember the day I first met him—in the interview of discussing Ph.D. admission opportunity—in 2015. From that day till today, and I know in the future, he was, is, and will be a mentor who is always ready to help in different capacities. Since Fall 2015, it was the most incredible opportunity to work with Ryan, whose outstanding intelligence always captivates me. He superintended me to learn how to conduct scientific research, write successful publications, and instruct courses. I borrow the mentality of being meticulous, pedant, and academic integrity from him and hope to meet the benchmark for advising he set. Also, I fell in love with cryptography and privacy-preserving technology due to him. Furthermore, his presentation skill and expertise in LaTex typesetting are unparalleled—I often feel very inspired to go beyond my comfort zone.

I am very thankful to my other Ph.D. research committee members: Prof. Apu Kapadia, Prof. Funda Ergun, and Prof. David Crandall, for their time, continued support, and helpful comments to ameliorate this thesis. Besides, I want to use this opportunity to thank my collaborators, Dr. Sadia Afroz and Prof. Damon McCoy (NYU), during my research internship at ICSI on censorship circumvention. Besides, I teamed up with Laia Amoros (Aalto University, Finland), Keewoo Lee (Seoul National University, Republic of Korea), and Caner Tol (Worcester Polytechnic Institute, USA) to work on privacy-preserving machine learning. These two collaborations are not related

to this thesis, but I highly enjoyed working on those diverse topics. Moreover, I am grateful to Apu and Sadia as they wrote reference letters for me (alongside Ryan) when needed.

I appreciate the Ph.D. admission opportunity and resources available at Indiana University (IU)—Bloomington. I recognize the fantastic administrative support I received from the Computer Science staff, especially Lynne Mikolon and Laura Reed. I thank Prof. Charles Pope and Prof. Yuzhen Ye for allowing me to teach CSCI C231/INFO-I231, Introduction to the Mathematics of Cybersecurity course, for three consecutive semesters since Fall 2019—I consider this as an outstanding opportunity to sharpen my teaching skills. Must mention, Ryan helped me tremendously such that I succeed in this excellent "professorship" opportunity.

I enjoyed enormous chatting and discussion on various things at Luddy Hall with Tousif Ahmed, Rakibul Hasan, Swaminathan Ramesh, Adithya Vadapalli in the coffee breaks. Hafsa and I are very fond of socialization. We shared so many joyful moments with some Bangladeshi families at Bloomington. I admire the affection—and tens of parties hosted by them—I received from them, especially from Prof. Khalid Khan, Prof. Rasheda Sultana, Prof. Hillol Bala, Arpita Bala, and so on. During our stay in Bloomington, we had some unfortunate medical conditions—I had four surgeries since 2019—their support in those difficult times are unforgettable. Besides, I recall my continuous online chatting with the so-called "Jajabor Buddies," e.g., Tousif, Taukir, Noman, Irfan, Sharif, Sabid, and others, on varying topics. Moreover, I have some awesome friends from high schools and college, e.g., Rubab, Sahib, Abir, Rifat, Limon, Sunny, Redoy, Khalid, Sagar, to name a few, who always encouraged me to overcome challenges in different phases of life.

I feel honored to serve as the Bangladesh Students Association (BDSA) president at IU in the 2017-18 academic year. Also, I thank the voting members who elected me two times to be an executive committee member—though I passed the second time to give a chance to a new member. Organizing multicultural events under the BDSA banner honed my leadership and management skills, undoubtedly.

I am perpetually grateful to my parents for their endless sacrifices and unconditional love—in

every way possible. Their hardwork to ensure me (and my siblings) a smoother life had no limit. I am very thankful to my relatives, including my in-laws' families, for all of their love and support and their role in my upbringing and education journey. I am lucky to have them in my life. I remember the memorable days I spent at my maternal grandparents' home in Muksudpur, Bangladesh, after class V-X's final exams as a vacation. Besides, I am undoubtedly grateful to my pre-Ph.D. teachers— from primary school to college—who shared their knowledge with me and certainly improved my life. I like to thank the Bangladesh government for letting me get (almost) free education from primary school to college. The education I received from my earlier institutions made me today.

Lastly, after Ryan, I am indebted to the most for this Ph.D. is my beautiful wife, my best friend, Hafsa. Our first child, a daughter, Hilwana, was born in 2018, and the second child, a son, Warith, was born—five days before my final defense—in 2020. It is quite impossible to explain how she helped me, directly and indirectly, to enable me to complete this journey. My medical conditions made the situation more difficult for her. Her smartness in managing everything is magnificent. She was supposed to be a successful fashion designer in Dhaka, Bangladesh, after her bachelor's degree in Fashion Design and Technology. Nevertheless, she paused her dream career to support me in the USA—keeping all of our relatives 8,000 miles away in Dhaka, Bangladesh—to facilitate me to pursue this Ph.D. program. Her caring, love, unwavering support, and sacrifices make it evident that I have dedicated this accomplishment to her entirely.

Funding sources: Ryan's following grants funded me through a research assistantship at IU. This dissertation's materials are based upon work supported in part by the National Science Foundation under Grant No. 1718475 and 1718595, by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute, and by the Indiana METACyt Initiative. The Indiana METACyt Initiative at IU is also supported in part by Lilly Endowment, Inc.

Syed Mahbub Hafiz

PRIVATE INFORMATION RETRIEVAL IN PRACTICE

Private Information Retrieval (PIR) solves the ubiquitous problem of safeguarding the privacy of users' (reading) access patterns to remote, untrusted databases. PIR is an extremely powerful cryptographic primitive, yet existing PIR techniques mainly implement inconvenient database abstractions and impose exorbitant performance overheads; thus, no PIR-based system has ever been deployed at scale in the wild. Modern *non-private* database systems expose many different "views" of the underlying data to their users. Thus, to make PIR appealing to the users of such database systems, PIR systems should themselves support a variety of expressive and realistic query types. Likewise, the underlying cryptographic protocols must be sufficiently practical to let the benefits of supporting private queries overshadow the associated costs.

The proposed thesis focuses on three novel contributions to the PIR literature, each of which represents an important step toward transitioning PIR from a theoretical construct to a useful tool in the privacy practitioners toolkit. (In addition to these three contributions, it offers a new set of formal definitions that unifies the existing set of ad hoc definitions used throughout the existing PIR literature.) First, we introduce a novel technique called *indexes of queries* that allows users to query PIR databases using "contextual," "expressive," and aggregate queries. Second, we present new protocols that exhibit "essentially optimal" efficiency—both in practice and in theory—for the class of so-called "1-private" many-server PIR protocols. Third, we demonstrate PIR's applicability to address a real-world privacy issue; namely, that of allowing Tor users to privately fetch hidden

service (aka. onion service) descriptors and propose some redesign suggestions to improve the privacy and security of Tor onion services. Each of these thrusts releases open-source libraries to examine the efficiency of our constructions, setting the parameters at scale and evaluating the performance in microbenchmark settings as well as in comparison with other libraries recognized as benchmarks in the PIR paradigm.

Ryan Henry, PhD, Chair

Apu Kapadia, PhD

Funda Ergun, PhD

David Crandall, PhD

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF RELATED PUBLICATIONS

The following list of publications is the outcome of this dissertation—once again, I thank Prof. Ryan Henry for the excellent supervision. Chapter 3 and Chapter 5 presented the key results of [68] and Chapter 4 illustrated the upshot of [67]. The repercussion of Chapter 6 is slated for submission shortly.

(ACM CCS '17 [67]) Syed Mahbub Hafiz and Ryan Henry. Querying for Queries: Indexes of Queries for Efficient and Expressive IT-PIR. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (*CCS '17*). Dallas, TX, USA. 1361–1373.
DOI: https://doi.org/10.1145/3133956.3134008

(PoPETS '19 [68]) Syed Mahbub Hafiz and Ryan Henry. A Bit More Than a Bit Is More Than a Bit Better: Faster (essentially) optimal-rate many-server PIR. In *Proceedings on Privacy Enhancing Technologies 2019* (*PETS/PoPETS '19*). Stockholm, Sweden. 2019(4), 112-131.
DOI: https://doi.org/10.2478/popets-2019-0061

BIBTEX entry for this dissertation:

```
@phdthesis{thesis:Hafiz21,
    author = {Syed Mahbub Hafiz},
    title = {Private Information Retrieval in Practice},
    school = {Indiana University},
    address = {Bloomington, IN, USA},
    month = {January},
    year = {2021}
}
```

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

Privacy is one of the fundamental human rights [9]. In the modern era of online media, people are at more risk of maintaining their privacy [102]. The digital economy mostly depends on collecting sensitive information (of users interacting with the underlying services), which can be used to infer personally identifiable information [103]. Online users should control the dissemination and use of such data, including complete freedom of expressing personal choices. Tracking a user's media consumption diet over a public (or private) database, the malicious database holder can deduce her political affiliation, sexual orientation, and more [102]. The problem of providing privacy to this kind of sensitive search over databases has drawn significant attention over time. States and private companies can play a substantial role in respecting an individual's query to public data-stores. Even in cases where the database holders are trustworthy, server compromise and subpoenas can leak the sensitive user information to a third party [133]. Private Information Retrieval (PIR) can help solve these problems by safeguarding users' privacy of reading access patterns to the database.

Chor et al. introduced the first solution to the PIR problem in their seminal paper [32]. Over the past two-and-a-half decades, research communities of cryptography, privacy, and information theory advanced PIR in many respects. Most of the PIR protocols can be classified into two

major groups corresponding to the security guarantees they provide to hide the user query. The perfect privacy or information-theoretic (IT-PIR) class [32, 33, 44, 57] requires $\ell \geq 2$ servers, each holding a complete replica of the original database, together with a non-collusion assumption up to a threshold $1 \leq t < \ell$. Such non-collusion assumptions are common to other well-known and in-the-wild privacy-enhancing technologies (PETs) like anonymous network Tor [47], secret sharing schemes [119], electronic voting protocols [28, 115], and mix networks [38]. Especially the assumption is more realistic when the replica servers are under different administrative domains. The user query's privacy to such database server is provably guaranteed even though the adversary is computationally unbounded and until the threshold number of servers collude with each other. The other class, computationally private (CPIR) systems [1, 4, 25, 86, 87, 109] involve a single server and rely on well-established computationally intractable (cryptographically hard) problems. Moreover, some "hybrid" protocols depend on both non-collusion and computational assumptions, providing a spectrum of security guarantees if one or more assumptions fail.

A vast majority of PIR protocols in the literature follow what we call the *vector-matrix paradigm*. In this paradigm, users employ some linearly homomorphic secret sharing (for IT-PIR) or encryption (for CPIR) scheme to construct a query that hides the item of interest's numerical index. Upon receiving a query, each server generates a response using some computations over the database, leveraging the aforementioned linear homomorphism. Lastly, the user reconstructs the desired item from the responses. Results from the information theory research community mainly focus on optimizing the download cost only without other costs. On the other hand, privacy practitioners are typically more concerned with lower computation costs and place less emphasis on achieving low communication cost.

A trivial solution to the PIR problem is downloading the entire database. Any practical PIR should have a non-trivial (sub-linear) communication cost compared to the whole database's size. Additionally, modern PIR constructions come equipped with many useful features. For example, some ensure that queries succeed even if some servers are non-responsive (robust PIR [14]) or given

incorrect responses (byzantine-robust PIR [44]); some prevent users from retrieving more items than explicitly requested (symmetric PIR [53, 54]); and some hide the contents of the database from the participating servers (independent PIR [53]). Moreover, some PIR approaches enable users to retrieve multiple blocks [4, 42, 70, 74, 79] at a time and allow clients to retrieve desired item(s) using more expressive and complex queries like keyword-based [21, 31, 56] and SQL-based [105, 133] searches. Our expressive PIR techniques (Chapter 4) bear a superficial resemblance to these prior efforts; however, the precise problem we solve and the technical machinery we use to solve it are fundamentally new. Indeed, our approach offers several distinct advantages (and a few limitations) compared with existing techniques. Therefore, we view indexes of queries as complementary to—as opposed to an alternative to—existing techniques for expressive PIR. In contrast to existing approaches, our techniques are inherently non-interactive, requiring just one communication round.

In recent work, Shah, Rashmi, and Ramchandran [118] proposed construction that achieved a remarkable property—to fetch a b-bit record from a database comprising r such records, the client need only download $b+1$ bits total. Inspired by Shah et al.'s result, the information-theory research community has produced a steady succession of papers characterizing the download capacity of multiserver PIR protocols under a variety of constraints [10, 128–130]. Nevertheless, this work has had nearly zero impact on PIR research within the cryptography and applied privacy research communities. Indeed, so aggressively optimizing download cost at the expense of all else imposes rather exorbitant costs for the other metrics of interest—so exorbitant that the constructions in each of the papers mentioned above end up falling short of satisfying any reasonable definition of "non-triviality" for a PIR protocol (see Definition 3.1.3 from Chapter 3). We turn this impractical protocol into a practical one and constructed other families of optimal-rate PIR schemes in Chapter 5.

1.2 THESIS STATEMENT

We study novel techniques to advance PIR from a theoretical primitive to a more practical privacy-enhancing technology and deploy it in a real-world unsolved problem. Most PIR works consider an elementary setting that allows a user to retrieve a single bit or a fixed-size block from the database by indexing the item, which we call positional query. Nevertheless, modern applications interact with the underlying database by various abstractions; many of them are by key-value pair and SQL-based queries. For instance, a user who wants to know the most popular Bangladeshi restaurant near her location from Google Map is unlikely to know (or care to know) the precise table and row within Google's POI database. Instead, the user is more interested in an abstraction that can facilitate the search "5 most popular Indian restaurants near me on Google Map" without concerning herself with the underlying database layout. Few works addressed the concern and proposed expressive PIR equipped with keyword-searching [21, 31, 56] capability, and SQL-based [105,133] queries. We introduce a new *indexes of queries* technique [67] to support expressive query which offers some advantages compared to the prior works.

Furthermore, we present a novel family of **1**-private, many-server PIR protocols [68] exhibiting unprecedented performance concerning *every* cost metric—download, upload, computation, and round complexity—typically considered in the PIR literature. With two servers, our protocols match the performance of the fastest known IT-PIR [32] and CPIR [21] protocols; with more servers, our constructions are much faster relative to them. Lastly, we present *Tunica* protocols which employ PIR to prevent frustrating attempts to generate popularity histograms over *onion service descriptors* and thereby develop an intersection-style attack [15] to deanonymize Tor clients so that the *hidden service directories (HSDirs)* cannot distinguish among requests from Tor clients for different *onion service descriptors*.

1.3 RESEARCH CONTRIBUTIONS

1. **Formal definition of PIR properties.** In Chapter 3, we introduce new formal definitions of PIR properties which improve and unite the existing definitions in the PIR literature. We rigorously define the correctness, non-triviality, and computational and perfect (also known as information-theoretic) t-privacy of PIR constructions.

2. **Most efficient expressive queries on PIR databases.** In Chapter 4, we propose *indexes of queries*, a novel mechanism for supporting efficient, expressive, and information-theoretically private single-round queries over multi-server PIR databases. Our approach decouples the way that users construct their requests for data from the physical layout of the remote data store, thereby enabling users to fetch data using "contextual" queries that specify *which* data they seek, as opposed to "positional" queries that specify *where* those data happen to reside. Our basic approach is compatible with any PIR protocol in the ubiquitous "vector-matrix" model for PIR. However, the most sophisticated and useful of our constructions rely on some excellent algebraic properties of Goldberg's IT-PIR protocol (Oakland 2007). We have implemented our techniques as an extension to Percy++, an open-source implementation of Goldberg's IT-PIR protocol. Our experiments indicate that the new techniques can significantly improve the utility for private information retrievers and efficiency for private information retrievers and servers alike.

3. **Most optimal 1-private PIR protocols.** In Chapter 5, we study both the practical and theoretical efficiency of PIR protocols in a model wherein several untrusted servers work to service remote clients' requests for data obliviously, and yet no pair of servers collude in a bid to violate said obliviousness. In exchange for such a strong security assumption, we obtain new PIR protocols exhibiting remarkable efficiency concerning every cost metric—download, upload, computation, and round complexity—typically considered in the PIR literature. The new constructions extend a multi-server PIR protocol of Shah, Rashmi, and

Ramchandran (ISIT 2014), which exhibits a remarkable property of its own: to fetch a b-bit record from a collection of r such records, the client need only download $b + 1$ bits total. We find that allowing "a bit more" download (and optionally introducing computational assumptions) yields a family of protocols offering very attractive tradeoffs. In addition to Shah et al.'s protocol, this family includes as special cases (2-server instances of) the seminal protocol of Chor, Goldreich, Kushilevitz, and Sudan (FOCS 1995) and the recent DPF-based protocol of Boyle, Gilboa, and Ishai (CCS 2016). An implicit "folklore" axiom that dogmatically permeates the research literature on multi-server PIR posits that the latter protocols are the "most efficient" protocols possible in the perfectly and computationally private settings, respectively. Nevertheless, our findings soundly refute this supposed axiom: These special cases are (by far) the *least* performant representatives of our family, with essentially *all other* parameter settings yielding instances that are *significantly* faster.

4. **More private and secure Tor onion (aka hidden) services.** In Chapter 6, we propose Tunica, a family of protocols that leverage *PIR* to mitigate the risks posed by most known attacks on Tor's *onion services* (aka *hidden services*) architecture. Tunica protocols prevent so-called *hidden service directories (HSDirs)* from distinguishing among requests for different *onion descriptors*, thereby frustrating attempts to construct popularity histograms over those descriptors and outright thwarting certain intersection-style attacks against the sender anonymity of users that access onion services. We present two concrete Tunica constructions, based respectively on 2-server PIR from distributed point functions and 1-server PIR from lattice assumptions.

1.4 PROPOSED WORK

Our expressive and optimal-rate PIR constructions both work with the *linear PIR* where the database is represented as an $r \times s$ matrix $\mathbf{D}$ over a finite field $\mathbb{F}$. Each of the r rows (blocks) is a

retrievable unit of data. For the expressive PIR, users process requests for the block(s) as vectors from the so-called standard basis $\vec{e}_i \in \mathbb{F}^r$. The response to the request is computed at the $\ell \geq 1$ number of server(s) as the vector-matrix product of $\vec{e}_i \cdot \mathbf{D}$. We refer to such vector-based requests as *positional queries* and mark this kind of PIR as *vector-matrix model*. Here, the user "shares" its query vector $\vec{e}_i$ component-wise, and then it sends each of the shares to a different server. Upon receiving a share vector, each server computes the product with $\mathbf{D}$. To recover its requested block, the user performs a component-wise secret reconstruction over the responses. The vanilla *indexes of queries* works with any protocol in the "vector-matrix" model for PIR. However, the most useful of our constructions rely on some basic algebraic properties of Goldberg's IT-PIR protocol [57], which obtains perfect privacy by employing Shamir's $(t+1, \ell)$-threshold linear secret sharing scheme.

1.4.1 EXPRESSIVE PIR

Our very first and straightforward construction *simple indexes of queries*, a pseudo-permutation matrix with full rank, yields a t-private IT-PIR query to another t-private IT-PIR query. Here, we trade off some information leakage in exchange for a more efficient and expressive PIR protocol. Our other non-trivial construction *batch indexes of queries* reduces the information leakage of *simple indexes of queries* by applying the "u-ary family of codes" [70]. Our last and most exciting construction is *indexes of batch queries* which ameliorates previous construction by allowing users to fetch several related blocks like the best k matches for some search term by a single request. Our *indexes of batch queries* technique can retrieve a bunch of related blocks using a single round of interaction and a single query as contrary to the multi-round earlier expressive PIR protocols [31, 105]. However, we view indexes of queries as complementary to—as opposed to an alternative to—existing techniques for expressive PIR.

Querying through any of the three variants of the *indexes of queries* constructions is an instance of sparse matrix-vector multiplication, an extremely parallelizable workload much fit to run on

GPU devices. To assess the practicality of our *indexes-of-queries* approaches, we implemented finite-field Sparse-Matrix-Vector multiplication and ran a range of experiments both on GPU and CPU. Our microbenchmark experiments showed that the new constructions incur overhead that is insignificant compared to the whole PIR process in the worst case. However, in most cases, our techniques can reduce the upload cost and server-side computation cost compared to the positional query-based standard PIR. We further evaluated the feasibility of deploying our technologies over a real-world dataset *IACR Cryptology ePrint archive*. We constructed four different indexes of 4-batch queries; namely, we created indexes of 4-batch queries supporting requests for the "4 most highly cited" and the "4 most recently posted" ePrint papers for each keyword as well as for each author. The measurements suggest that *indexes of queries* are a useful building block in constructing practical PIR-based systems for large-modern datasets.

1.4.2 OPTIMAL-RATE PIR

We convert one-extra-bit construction of Shah et al. [118], which enables a user to obtain a b-bit record from a database by downloading only $b+1$ bits, from an impractical and theoretical result into what we consider to be the most efficient 1-private PIR protocol. We present the novel family of "*one-extra-word*" protocols [68], of which the one-extra-bit construction is a particular case. Moreover, the family also realizes the 2-server perfectly 1-private protocol of Chor et al. [32] and the computationally 1-private protocol of Boyle et al. [23] as exceptional cases, which are conjectured to be the "most practical protocol possible" in the perfectly and computationally private settings, respectively. Our findings refute this conjecture by showing the exceptional cases are the *least* practical representatives of our family, with essentially *all other* parameter settings, yield instances that are *significantly* faster.

In our "*one-extra-word*" family of protocols, user uniformly at random chooses $s+1$ number of matrices in $\mathbb{F}^{r\times(s+1)}$ whose rows are taken from the standard basis vectors. The matrices constitute an equivalence class that differs at the ith row as the user wants to retrieve the ith record from

the database. The database is augmented with an additional column whose elements are evaluated after finding the linear combination of its first s columns governed by a particular vector $v \in \mathbb{F}^s$. The user sends the matrices to $s + 1$ number of servers chosen according to a mapping function. Each server stores the *Frobenius inner product* between the received matrix and the augmented database. After downloading the $s + 1$ number of words from the servers, the user solves a s linear equation system to reconstruct the desired record.

From our "*one-extra-word*" family of protocols, the one-extra-bit construction can be actualized by setting the vector $v = {}_1$ and the mapping function in a particular way. The one-extra-bit protocol needs a super-exponential number of servers increases with the number of blocks in the database, which is impractical. Our "bit-more-than-a-bit" construction, a subfamily of the perfectly 1-private one-extra-word protocols parametrized by the number of servers $\ell \geq 2$ and the number of words per block s as $\ell = s + 1$, solves this impractical issue having the vector $v = {}_0$ and a distinct mapping function.

We further analyze our most efficient construction, a *computationally* 1-private instance of the bit-more-than-a-bit subfamily with 2^L number of servers, which replaces the *uniform random* query strings with *pseudorandom* query strings generated by an L-tuple of $(2, 2)$-*distributed point functions (DPF)*. The DPF-based bit-more-than-a-bit construction imposes some extra computation overhead that comprises the DPF key pairs' sampling and the cost for each server to expand its DPF keys using a full-domain evaluation and a final component-wise concatenation. However, our experimental evidence from the prototype implementation suggests that the extra computation overhead is insignificant relative to the time required to generate the PIR response. We also extend the computationally 1-private bit-more-than-a-bit protocol to support the arbitrary number of servers at the expense of some other upload and computation overhead governed by $poly(\lambda)$ to overcome the possible *modulo bias*.

We have implemented open-source libraries to examine the efficiency of our constructions setting the parameters at scale and evaluate the performance of the head-to-head comparison

with the optimal protocols as well as Percy++ v1.0 [58] and RAID-PIR v0.9.5 [39] multi-server PIR libraries considered as benchmarks in the PIR literature. We found that the slowest member of our family is faster than the fastest mode of Percy++ v1.0 [58] and RAID-PIR v0.9.5 [39] and folklore protocols of Chor et al. [32] and Boyle et al. [23].

1.4.3 PIR IN THE WILD

Tor is a low-latency anonymous communication system that provides anonymity for both user and service provider through onion routing mechanisms [47]. Tor network establishes a channel (i.e., circuit) between client and service, which contains intermediate nodes to relay data. Each circuit between two Tor network entities consists of three-node relays, e.g., guard node, middle node, and exit node. Tor offers a protocol to hide the IP address of a web service from its clients and calls it hidden (now onion [63]) services.

After the deployment of Tor's onion service protocol, several attacks have been investigated that finally deanonymized the onion services [15, 81, 88]. Biryukov et al. [15] ranked all onion services to build a popularity histogram after observing the client's access frequency to onion services. The information leakage made it easy to target a particular onion service of interest and carried a denial of service (DoS) attack on the responsible hidden service directories (HSDirs) by making the associated descriptors unavailable for the clients to fetch. The author of the Tor project blogpost [84] envisioned that tracking for onion service popularity can be made harder by operating a Private Information Retrieval (PIR) protocol. Despite this suggestion, there were no proposals that studied this idea any further. We inspect how the descriptor retrieval pattern can be made private by applying feasible PIR schemes.

We propose two Tunica constructions that leverage PIR to mitigate the said information leakage: the first variant, $\mathsf{Tunica}_{\mathsf{DPF}}$, is based on 2-server PIR from distributed point functions [56], while the second variant, $\mathsf{Tunica}_{\mathsf{LWE}}$, is based on single-server XPIR [1]. As onion services upload descriptors via HTTP Post announcements to HSDirs, there is a risk that the common partition of

the engaged HSDirs might not be synced at a given time as expected. To hold the synchronization among the HSDirs, we suggest Tor clients employ *Bloom Filters* [18] to identify the intersection set of the common partitions of the related HSDirs. Otherwise, the Tor client must follow the second variant of the single-server $\mathsf{Tunica}_{\mathsf{LWE}}$ protocol.

1.5 THESIS ORGANIZATION

Chapter 2 discusses the large amount of work conducted in the PIR paradigm. Initially, Section 2.1 classifies the existing PIR constructions in four categories concerning security guarantees they provide, such as information-theoretic PIR (§2.1.1), computational PIR (§2.1.2), trusted hardware-based PIR (§2.1.3), and hybrid PIR (§2.1.4). Next, we enlist various useful features that are equipped with PIR schemes in Section 2.2. Section 2.3 demonstrates optimal-rate (§2.3.1) and capacity-achieveing (§2.3.2) PIR schemes. There are close relationship of PIR with other cryptographic building blocks that are outlined in Section 2.4. Finally, this chapter ends with describing several promising applications of PIR in Section 2.5.

Chapter 3 gives formal definitions for private information retrieval describing the correctness (§3.1.2), non-triviality (§3.1.3), and t-privacy (§3.1.4) properties.

Chapter 4 begins with an introduction Section 4.1 and afterward, Section 4.2 describes the abstract PIR framework in which all of the indexes-of-queries constructions reside. Section 4.3 introduces *simple indexes of queries*, the most basic (and least interesting) form of our construction, while Section 4.4 describes a more sophisticated construction for *batch indexes of queries*, which leverage ideas from coding theory to reduce costs and improve privacy compared to simple indexes of queries. Section 4.5 further extends this idea to construct *indexes of batch queries*, which allow users to fetch a batch of several related blocks using a single, fixed-length query. In Section 4.8, we briefly discuss a generalization called *indexes of aggregate queries*, which allow queries for certain aggregates

(i.e., linear combinations) of raw (but structured) numeric data. Section 4.6 reviews prior work on expressive PIR queries—SQL- and keyword-based PIR queries and PIR from function secret sharing—and comments on the synergistic relationship between our work and those techniques. We present some findings from our proof-of-concept implementation in Section 4.7 before concluding in Section 4.9.

Chapter 5 starts with an introduction Section 5.1 and then Section 5.2 presents the one-extra-word family of protocols including its "one-extra-bit" instantiation in Section 5.3 and a detailed security analysis for the family as a whole in Section 5.4. Section 5.5 introduces the bit-more-than-a-bit construction, a subfamily of one-extra-word protocols that improves on one-extra-bit potentially in several respects. Section 5.6 describes an efficient, computationally 1-private bit-more-than-a-bit construction, first for 2^L servers (§5.6.2) and then for arbitrarily many servers (§5.6.3). Section 5.7 briefly discusses of our results in the context of related work. Section 5.8 presents findings from our prototype implementation, including head-to-head comparisons with the "folklore" protocols of Chor et al. and Boyle et al. in addition to a selection of multiserver protocols from the open-source Percy++ (§5.8.4) and RAID-PIR (§5.8.5) libraries. Section 5.9 wraps up with some concluding remarks and directions for future work.

Chapter 6 inaugurates with an introduction Section 6.1 and later on Section 6.2 covers preliminaries, including our basic notation and definitions for the cryptographic primitives upon which our Tunica constructions rely (§6.2.1). That is followed by an overview of Tor and its rendezvous (onion services) specification in Section 6.3—both version 2 (§6.3.1) and version 3 (§6.3.2)—and known attacks against onion services (§6.3.3). Section 6.4 formulates a threat model with the express goal of sealing the very leakage of information that makes the attacks outlined in Section 6.3.3 possible. Next, we present and analyze two Tunica constructions, respectively based on 2-server PIR from distributed point functions (§6.5) and

on single-server PIR from the Ring-LWE assumption (§6.7). We rigorously discuss the security and privacy analysis—attacks against arbitrary and targeted onion services as well as expected coverage of an attacker—of $\text{Tunica}_{\text{DPF}}$ construction in Section 6.6. We have a brief related work discussion in Section 6.8. We illustrate results from some microbenchmarks in Section 6.9. Finally, Section 6.10 concludes with our recommendations for deployment of tunicate onion descriptors in real Tor.

Chapter 7 addresses future directions to move towards building practical privacy-preserving technologies.

CHAPTER 2

LITERATURE REVIEW

Private information retrieval (PIR) is a seemingly impossible problem that enables users to fetch data from remote database servers without letting the service provider know which data they are interested in. For the last 25 years, a diverse pool of research communities, e.g., cryptography, privacy engineering, information theory, and so on, has been advancing PIR literature in multiple aspects. In this chapter, we try to discuss some of the most exciting contributions of them to the PIR paradigm and demonstrate other cryptographic primitives' connections with it.

2.1 PIR WITH VARYING PRIVACY GUARANTEE

Solutions to the seemingly impossible problem of allowing users to access a database obliviously do not come without assumptions. Depending on various assumptions, we achieve different types of privacy guarantees—we categorize them in four major groups.

2.1.1 INFORMATION-THEORETICALLY SECURE PIR

Information-theoretic (aka perfectly private) PIR (IT-PIR) involves more than one server and has a non-collusion assumption between participating servers. Each engaging server has a replica of the original database and thus, incurs a synchronization requirement. The PIR queries from the user are distributed between servers using a linearly (homomorphic) sharing mechanism.

PIR from secret sharing. The user can use a linear secret sharing scheme to share the query vector with each participating server. The first-ever PIR construction was proposed by Chor et al. [32, 33] that employs an XOR-based ℓ-out-of-ℓ additive sharing scheme to share the user query to multiple engaging servers. The XOR-based CGKS protocol performs very fast in terms of server-side computation. Another obvious choice is Shamir's secret sharing scheme [119] which was first investigated by Goldberg in 2007 [57]. Likewise, Shamir's secret sharing scheme can support threshold security, an IT-PIR protocol that utilizes Shamir's secret sharing scheme can also enable threshold privacy. That means any coalition of a certain number of arbitrary participating servers can collude with each other, but still, the PIR construction holds perfect privacy against a computationally unbounded adversary. Goldberg's IT-PIR protocols use Shamir's $t+1$-out-of-ℓ secret sharing scheme. There is a substantial amount of work [44, 67, 70, 74, 75] performed in this direction to achieve IT-PIR functionality, including advanced features applying linear secret sharing schemes.

2.1.2 COMPUTATIONALLY SECURE PIR

PIR constructions can be built utilizing various computationally intractable problems. Usually most of the CPIR constructions involve a single server, thus having no non-collusion assumption and synchronization issue. There are a few many-server CPIR protocols that have both non-collusion assumption as well as computational assumption. CPIR schemes are secure against probabilistic polynomial time (PPT), i.e., computationally bounded adversary only.

PIR from cryptographically hard problems. Kushilevitz et al. [87] first introduced the single-server CPIR protocol that assumes the quadratic residuosity is a computationally intractable problem [61] against a PPT attacker. Cachin et al. [25] named the intractability of deciding whether a small prime is a divisor of $\phi(m)$, where m is a composite integer of unknown prime factorization, as ϕ-hiding assumption (ϕHA). They constructed a CPIR scheme that achieves

communication as well as computation efficiency. Homomorphic encryption is an ideal choice for developing single-server CPIR schemes. Kiayias et al. [86] proposed a new complexity class of functions implementable by polynomial-size (i.e., *leveled*) large-output branching programs (LBP). They built a leveled LBP-homomorphic encryption-based CPIR protocol with the assumption of the decisional composite residuosity hard problem. Aguilar-Melchor et al. [1–3] employed lattice-based cryptographic hard problems such as Ring learning with error (Ring-LWE) to construct homomorphic encryption-based single-server CPIR.

PIR from function secret sharing. Boyle et al. [21] introduced an additive secret sharing like multi-party *function secret sharing* (FSS) which generalizes and improves two-party distributed point function (DPF) [56]. The security lies in the existence of one way function, aka the pseudo-random number generator (PRG). They deployed FSS schemes to develop many-server CPIR protocols which have computational as well as non-collusion assumption. There are further improvements and extensions to FSS inspected in [22, 23].

PIR from anonymity. Ishai et al. [80] used a two-way anonymous communication channel to build a multi-server CPIR scheme—the security of the protocol relies on the computational hardness of reconstructing noisy low-degree curves in a low-dimensional space. Toledo et al. [131] provided differential privacy-based PIR definitions and constructed efficient PIR schemes incorporating an anonymity system. Trostle et al. [132] built a family of single-server CPIR protocols from the assumption of a hidden modular group order (HMGO) and sender anonymity. However, the PIR based on HMGO was broken later by Lepoint et al. [91].

2.1.3 TRUSTED HARDWARE-BASED PIR

Secure coprocessors aim to provide tamperproofing to physical attacks and can run cryptographic operations. Smith et al. [123] first proposed a PIR scheme utilizing a secure coprocessor with the assumption of trustworthy hardware. In this protocol, the user encrypts the block number and

sends that to the coprocessor. Coprocessor decrypts that and reads all blocks from the database, and returns the requested block after encrypting it by the user's public key. Upon receiving, the user decrypts the block. Afterward, there is a line of works [8, 77, 135, 136] that tried to reduce the computation complexity at the server-side by cleverly employing temper-resistant general-purpose secure coprocessor (such as the IBM 4764) to build such communication- and computation- efficient PIR protocols that are not achievable by IT-PIR and CPIR variants. Wang et al. [135] encrypt the database that gets permuted (aka reshuffled) after a fixed number of queries. The secure coprocessor reads an encrypted block from the database and stores it in the cache in each query. If the required block is already available on the cache, the coprocessor stores a randomly selected block from the encrypted database. Several hardware manufacturing companies include a trusted execution environment (TEE) on their regular computing architecture. Intel's SGX, ARM's TrustZone, AMD's Secure Encrypted Virtualization (SEV) are such TEEs that create isolated environments to guarantee protected code and data loading concerning confidentiality and integrity. Mokhtar et al. [98] demonstrated private web search using Intel SGX technology.

2.1.4 HYBRID PIR

There are a few initiatives [43, 65] to combine both IT-PIR and CPIR settings towards utilizing the best features and reducing some demerits of both. One unique advantage of integrating both categories is that when any of the security assumptions are broken, the hybrid system holds some partial privacy. For example, if a subset of servers colludes with each other, the queries still have computational privacy. Besides, the hybrid PIR set up shows less communication cost compared to the employment of the constituent protocols individually. Gupta et al. [65] utilized the potentiality of such hybrid PIR constructions to build *Popcorn*—a private video streaming site like *Netflix*.

2.2 PIR WITH ADVANCED FEATURES

Basic PIR schemes realize a very elementary setting of the positional query, simple database schema, and concern about communication complexity and computation complexity only. In modern internet services, users interact with the database in sophisticated views and expect various usabilities. Also, advanced server-side functionalities are essential to enable with PIR settings. In this regard, many works tried to cope with modern internet service usabilities satisfying PIR requirements.

PIR with expressive queries. Chor et al. [31] first supported the expressive PIR query by augmenting the database with an auxiliary data structure; e.g., a binary search tree, a trie, or a perfect hash function, which redirects the keyword-based requests into positional queries, which are served by the underlying PIR protocol. The user also employs positional queries to obliviously traverse the intermediary data structure to learn the record of interest's final location after multi-round interactions. Recently, Boyle et al. developed the *function secret sharing* (FSS) [21–23] schemes, which, theoretically, permit users to fetch records obliviously using any search criteria. Alas, practical constructions are limited to a few options, for example, comparisons, interval queries, and point functions. Their approach is single-round and does not need any auxiliary data structure but depends on a stronger security assumption and a considerable computation cost, especially when more than two servers are involved.

Olumofin and Goldberg [105] expanded Chor et al.'s keyword-based PIR method to facilitate users to retrieve blocks using simple *SQL queries* with WHERE or HAVING clauses only. They accomplished this by preparing inverted indexes at the server that links the search term to the physical locations of associated blocks in the database. The approach may leak some information about which blocks a user seeks, as it hides the sensitive search terms that appear in a query but not the category of that query. When a single SQL query involves several records to fetch, the technique needs the user to run multiple positional PIR queries and thus turns into a multi-round

protocol. A recent paper by Wang et al. [133] proposed *Splinter*, a system that employs *function secret sharing* to support a range of queries comparable to those supported by the previous SQL-based approach. Indeed, Splinter is a single round but (somewhat) information-leaking protocol.

PIR with multi-block queries. A user sometimes wants to fetch more than one block from a remote database at once in a real application. There are some works [42, 67, 70, 74, 79, 93] that supported a user to retrieve multiple blocks from PIR databases in a single round. Some of them [79, 93] spun the support of multi-block queries to amortize the per-query communication and computation cost. Some of them [67, 70, 74] showed constant communication cost regardless the user fetches a single block or multiple blocks in a single round.

PIR with multi-client. A real database system receives requests from multiple clients at a time. To make PIR more usable, we need eligible PIR construction that can serve multiple clients simultaneously. Ishai et al. [80] proposed batch codes to permit one or many clients to fetch one or many blocks obliviously in a single round. Lueks et al. [93] drew a relationship between batch codes and matrix multiplication methods to build a PIR database that scales sublinearly in the number of records queried by multiple clients.

PIR with (byzantine) robustness. In a multi-server system, it is quite possible that a few of the servers may not respond to a user query due to various reasons such as they are inactive, out of date, etc. Besides, servers can send back erroneous responses being malicious or compromised. Therefore, it is convenient that multi-server PIR schemes show byzantine robustness against inactive or compromised servers. Duvet et al. [44] improved the byzantine robustness of Beimel et al. [14] and Goldberg [57] by applying Reed-Solomon codes (error-correcting codes)-based decoding algorithms.

PIR with symmetricity There are application databases that may hold valuable digital goods and may restrict a user to fetch a certain number of arbitrary blocks from them at a time. For

instance, a movie database service may allow a client to download a single movie at once, i.e., the database wants the user query to comply with its policy. The PIR setup needs to permit the user to retrieve only a movie. To avail privacy for both the database and for the user query, the symmetric PIR has been proposed by Gertner et al. [54]. Henry et al. [74] achieved such symmetricity by integrating zero-knowledge proof to verify the correctness of the user PIR query.

PIR with independence In a multi-server setting, each server holds a replica of the original database. Sometimes the database can be sensitive, and the service provider may not like to outsource the database to other remote servers. Gertner et al. [53] first introduced τ-independent PIR scheme where each of ℓ servers holds a secret share of the plain database and until and otherwise $\tau + 1$ number of servers collude, each of the servers cannot get the plain data. There is a trade-off between byzantine robustness, independence, the limit of multi-block queries, and the threshold privacy for Goldberg-style IT-PIR schemes. Note that Jia and Jafar name this feature as τ-secure PIR in their work [82].

2.3 PIR BOUNDARIES

Research communities of information theory, cryptography or mathematics, and privacy practitioners focus on from one to many PIR metrics to optimize—sometimes in such an extreme way that remaining metric(s) may lose practicality. We discuss such initiatives next in two categories, such as optimal-rate performances and capacity-achieving constructions.

2.3.1 OPTIMAL-RATE PIR

In the IT-PIR setting, Shah et al. [118] proves the optimal download is one extra bit than the block size and achieves this optimal download with an exponential number of servers and inefficient communication cost. Recently a single-server CPIR protocol has been proposed by Kiayias et al. [86] has asymptotically optimal download cost. The construction depends on number-theoretic

assumptions, which make computation costs prohibiting for a large-scale deployment [122]. Additionally, the download overhead becomes small only when the requested data approaches several GiBs, either because records are that large or because the client is simultaneously requesting multiple records and amortizing the overhead over all those requests. In this direction of multi-server IT-PIR works, Henry et al. [74] proposed *batch queries* atop of Goldberg's IT-PIR protocol [57]. Afterward, Wang et al. [134] and Demmler et al. [42] extended the batch queries technique to Chor et al.'s protocol [32]. Utilizing the $(2, 2)$-Distributed Point Function [56] based approaches, logarithmic upload cost has been achieved by several recent works [23, 37, 56]. Whereas, Angel et al. [4] investigated a new technique to lower the upload cost for a single-server fully homomorphic encryption-based CPIR protocol using so-called "plaintext slot permutations."

2.3.2 CAPACITY-ACHIEVING PIR

A multi-server PIR scheme's capacity is the reciprocal of minimum download cost for the client from the server, i.e., the maximum number of bits of desired data that can be obliviously fetched per bit of downloaded data. Sun and Jafar contributed to measure and achieve a capacity of PIR in multiple settings. They evaluated the capacity (aka rate) of basic PIR and proposed construction that attains the rate in [129]. In successive works, they computed and realized the capacity of t-private IT-PIR with robustness in [130], with supporting a query in multiple rounds of interaction in [126], with symmetricity in [127], when the user stores previous downloaded blocks as side information in [29], and with τ-independence data privacy in [82, 83]. In exchange for meeting the (download) capacity, sometimes this work line does not concern the sublinear communication complexity required to design a non-trivial PIR.

2.4 CONNENTIONS TO OTHER CRYPTOGRAPHIC PRIMITIVES

Some cryptographic primitives share common goals with PIR. Hence, there are PIR schemes that can be constructed leveraging those cryptographic primitives. This section discusses the basic functionalities of such primitives and demonstrates how recent works realized PIR properties from them.

Locally decodable code. An ℓ-query Locally Decodable (error-correcting) Code (LDC) encodes a message of n bits to a codeword of N bits such that any bit of the original message can be retrieved by ℓ queries even if some responses are corrupted. Yekhanin [139] transformed 3-query LDCs to 3-server PIR with efficient communication complexity assuming there are infinitely many Mersenne primes.[1] There are subsequent works [12, 49, 69] that proposed better LDCs and built more communication-efficient PIR constructions.

Oblivious transfer. Oblivious transfer (OT) is a cryptographic primitive that enables a database server to send a message to a receiver so that the receiver does not know about other messages the server holds and the server does not know which message has been sent. Rabin first introduced this problem and a solution in [112]. Later on, there are extensive amount of works [99–101, 108] that developed 1-out-of-2 and 1-out-of-r OT protocols. OT does not have the requirement of sublinear communication, whereas PIR protocols do. Note that symmetric PIR and 1-out-of-r OT with sublinear communication complexity is identical. Indeed, we can build a PIR scheme generalizing 1-out-of-r OT though it requires additional computation cost. In contrast, Naor and Pinkas [99] showed how to convert any single-server PIR to a 1-out-of-r OT scheme.

Oblivious RAM. Goldreich and Ostrovsky [60] introduced the concept of oblivious RAM (ORAM) with the motivation of hiding memory access pattern by a computing program against an adversary. ORAM converts an algorithm keeping the original behavior unchanged to a modified one such that the distribution of the memory access pattern of the new program is independent of

[1] Any prime that can be represented as $2^p - 1$, where p is smaller prime than the Mersenne prime.

the memory access pattern of the previous program. From the definition, we note that ORAM is a more potent privacy-preserving technology that hides reading-access pattern and conceals writing-access pattern. Mayberry et al. [95] proposed *path PIR* that leverages the best and overcomes the shortcomings of both worlds of ORAM and PIR. Their hybrid ORAM scheme modifies the tree-based ORAM of Shi et al. [120] utilizing PIR strategies. There is an intensive amount of works [6, 52, 96, 124] aimed to optimize the ORAM schemes.

Secure multiparty computation. It is a strong cryptographic technique that enables multiple mutually-distrusted parties to compute an efficient function on input data—keeping each input data private to each party. Yao [138] used garbled circuit, Goldreich et al. [59] employed secret sharing, Gilboa [55] and Demmler et al. [41] deployed arithmetic sharing methods to build MPC constructions. Ishai et al. [78] illustrated a mechanism to convert from PIR to MPC and vice versa. They obtained perfect constant-round MPC from single-round sublinearly communication-efficient multi-server PIR through their proposed transformation. On the other hand, they demonstrated MPC protocols with linear privacy threshold can be turned into efficient PIR. Recently, Laud et al. [90] achieved PIR functionalities on top of an MPC construction, which showed efficient computation complexity utilizing a preprocessing phase.

2.5 PIR APPLICATIONS

There are numerous use cases for which PIR has been applied. For instance, privately querying to location-based services [107], patent and pharmaceutical data-stores [7], domain-name registers [105], software update repositories [26], stock market database [137, 140], and online census information sites [137] has been suggested as exciting applications. Pynchon Gate [116] is a prototype of a secure and fault-tolerant pseudo-anonymous mail retrieval system that employs PIR to hide pseudonymous users from real users. Boneh et al. utilized PIR to support private queries on outsourced and encrypted mailboxes [19]. To cope with the growth of (and traffic

analysis attacks against) the anonymous communication network Tor [47], Riffle [89] and PIR-Tor [97] have been proposed to provide the required functionalities. There are other characterized prototypes designed on PIR as a building block; to name a few, privacy-preserving e-commerce site [75], private presence service (aka friend discovery) [20], unobservable private communication system [5], anonymous messaging service [37], private ad delivery [64], online movie streaming site [66], and private contact discovery protocol [40].

PIR in Tor Tor, the most famous anonymous communication system, faces scalability issues mainly due to requiring the maintenance of the global view of its relays' network consensus document. Unlike peer-to-peer network-based complicated solutions, Mittal et al. [97] applied PIR techniques to find relays privately and build circuits with better performances and features.

CHAPTER 3

FORMAL TREATMENT OF PIR

3.1 PRIVATE INFORMATION RETRIEVAL

Let $\mathbb{F}$ be a finite field and let $\boldsymbol{D} \in \mathbb{F}^{r \times s}$ be a database consisting of r many s-element records (called *blocks*), each of which is indexed by a positive integer less than or equal to r. Intuitively, PIR is a cryptographic technique that allows a client to fetch one or more blocks of its choosing from $\boldsymbol{D}$ without disclosing to the remote server (or servers) holding $\boldsymbol{D}$ any information about which blocks it fetches. We formalize this intuitive notion of privacy using an *indistinguishability*-based definition, as described below.

Consider the interaction that occurs between a client who seeks the block $\vec{\boldsymbol{D}}_i \in \mathbb{F}^s$ indexed by i and a remote database server (or servers) who holds $\boldsymbol{D}$. The client initiates the interaction by sending a *query* string $q \in \{0,1\}^*$ to the server(s), who return a *response* string $z \in \{0,1\}^*$ from which the client extracts its desired block. Let I, Q, R, and E respectively denote the random variables that describe (i) the index $i \in [1..r]$ of the block the client seeks, (ii) the query string $q \in \{0,1\}^*$ the client sends to the server, (iii) the response string $z \in \{0,1\}^*$ the server(s) returns to the client, and (iv) the string $d \in \mathbb{F}^s$ the client ultimately outputs. From the client's perspective, the 4-tuple (I, Q, R, E) completely characterizes the interaction that occurs in a given PIR protocol run. Figure 3.1 demonstrates the interaction between the client and database servers in a basic PIR setting.

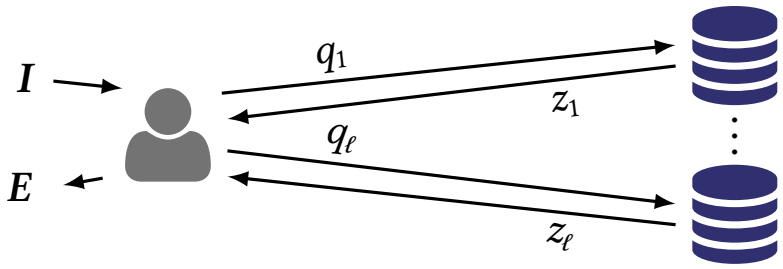


Figure 3.1: Information flow in a vanilla PIR.

3.1.1 "IMPOSSIBLY PERFECT" PRIVACY

Ideally, a PIR protocol would guarantee that an adversary who controls the database server(s)—no matter how powerful and well-positioned to launch an attack—cannot possibly deduce *anything* from a client's requests (beyond the trivial fact that the client is looking for *something*). One could formalize this requirement by insisting that the distribution of I be statistically independent of that of Q. This requirement leads to a notion that we refer to as *"impossibly perfect" privacy*, so-called because Chor et al. observed in their seminal paper [33, §5.1] that this level of privacy is impossible for any "reasonable" PIR construction to provide. (We elaborate on what we mean by "reasonable" shortly.) More precisely, while it *is* possible—indeed, easy—to provide impossibly perfect privacy (Definition 3.1.1), it is *not* possible to simultaneously provide correctness (Definition 3.1.2) and non-triviality (Definition 3.1.3). It is nevertheless instructive to state a formal definition for impossibly perfect privacy—no matter how unsatisfiable—as it will serve as a useful starting point for our actual, satisfiable privacy definition (Definition 3.1.4) below.

Definition 3.1.1. *The interaction described by* (I, Q, R, E) *provides* impossibly perfect privacy *if, for all block indices* $i, i^* \in [1..r]$ *and for all query strings* $q \in \{0,1\}^*$*, we have*

$$\Pr\big[Q = q \;\big|\; I = i\big] = \Pr\big[Q = q \;\big|\; I = i^*\big],$$

where the probability is over all the random coin tosses made by the client.

Definition 3.1.1 insists that the conditional distribution of Q given $I = i$ be *identical in its entirety* to that of Q given $I = i^*$ (as opposed to merely insisting that these distributions be *ε-close* or *polynomial-time indistinguishable*, or insisting that certain *substrings* of Q be identically distributed); that is, it insists that the client's query string q contains "no information"—in a strict, information-theoretic (i.e., perfect) sense—about which block the client is using that query string to fetch. This suggests two natural relaxations that might lead to "possible" notions of privacy: (i) insist that the above distributions be merely computationally indistinguishable or (ii) restrict the attacker to observing only part of any given sample from Q.

In fact, the privacy definition we use in Chapter 5—called *computational* 1*-privacy*—adopts both relaxations. Before stating that definition and computations t-privacy, we define two additional important requirements (*correctness* and *non-triviality*) for a PIR protocol, which collectively describe what we meant above when we spoke of "reasonable" PIR protocols with respect to which Definition 3.1.1 is impossible to satisfy.

3.1.2 CORRECTNESS

One trivial way to satisfy Definition 3.1.1 is to have the client choose $q \in \{0,1\}^*$ arbitrarily and without regard for i. Upon receiving q from the client, the server(s) likewise responds with arbitrary $z \in \{0,1\}^*$, unrelated to q or $\boldsymbol{D}$. This interaction offers perfect privacy, but it is not useful: the response z provides nothing to help the client correctly extract its desired block. The following *correctness* criterion disqualifies such "useless" protocols, requiring that the client learns its desired block.

Definition 3.1.2. *The interaction described by* (I, Q, R, E) *provides* (perfect) correctness *if*

$$\Pr\left[E = \vec{\boldsymbol{D}}_i \,\middle|\, I = i\right] = 1,$$

where $\vec{\boldsymbol{D}}_i \in \mathbb{F}^s$ *denotes the block indexed by* i *within* $\boldsymbol{D}$.

Notice that the distribution of E is dependent not only on those of I and Q but also on that of R; thus, correctness is necessarily contingent on the faithful execution of the protocol by the server(s).

3.1.3 NON-TRIVIALITY

Another trivial way to satisfy Definition 3.1.1, while simultaneously satisfying Definition 3.1.2, is to again have the client choose $q \in \{0,1\}^*$ arbitrarily and without regard for i, but then to have the server(s) respond with *the entire database*; that is, with $z = \boldsymbol{D}$. Upon receiving $\boldsymbol{D}$, the client would then extract the block $\vec{\boldsymbol{D}}_i$ from $\boldsymbol{D}$ on her local machine, ensuring perfect privacy in exchange for considerable download overhead. The following *non-triviality* criterion disqualifies such "trivially correct" protocols by insisting that the (bidirectional) communication cost of the interaction be asymptotically smaller than the size of the database $\boldsymbol{D}$ itself.

Definition 3.1.3. *The interaction described by* (I, Q, R, E) *provides* non-trivial communication *if*

$$\sum_{n \in \mathbb{N}} n \cdot \Pr\big[|Q| + |R| = n\big] \in o(|\boldsymbol{D}|);$$

that is, if the expected (combined) bitlength *of the query and response strings scale sublinearly with the bitlength* $|\boldsymbol{D}|$ *of* $\boldsymbol{D}$.[1]

We remark that the various capacity-achieving PIR constructions described in the body of work [10, 118, 128–130] cited in Section 5.1 satisfy a weaker definition of non-triviality wherein only *download* (i.e., $|Q|$) is required to scale sublinearly with $|\boldsymbol{D}|$. In fact, in several of those constructions—e.g., the capacity-achieving constructions of Sun and Jafar [128–130]—the upload cost $|R|$ scales *(super-)exponentially* in $|\boldsymbol{D}|$.

[1] One could also consider a variant of Definition 3.1.3 that considers the *maximum* possible—rather than *expected*—communication cost of an interaction.

3.1.4 COMPUTATIONAL AND PERFECT C-PRIVACY (t-PRIVACY)

The confluence of Definitions 3.1.2 and 3.1.3 rules out the "useless" and "trivial" PIR candidates we have considered thus far; indeed, as noted previously, Chor et al. proved in their seminal paper on PIR that it would be *impossible* to construct any protocol—no matter how clever—that satisfies Definitions 3.1.1–3.1.3 simultaneously. Nevertheless, in the very same paper, they proposed a (simultaneously *correct* and *non-trivial*) protocol involving several (say, $\ell > 1$) non-colluding but otherwise untrusted servers that *does* perfectly hide the client's request from each of the ℓ servers in isolation. In such a multiserver interaction, the query and response are both ℓ-tuples, say $q = (q_1, \ldots, q_\ell)$ and $z = (z_1, \ldots, z_\ell)$, from which the client sends $q_j \in \{0,1\}^*$ to and receives $z_j \in \{0,1\}^*$ from the jth server.

A *grand coalition* comprising all ℓ servers in such a multiserver interaction wields privacy-infringing powers tantamount to those of the lone server in a single-server interaction, in which case Chor et al.'s impossibility result applies.[2] Consequently, no multiserver protocol can hope to provide perfect privacy unless certain coalitions among the servers do not form. To capture this limitation, it is necessary to both extend and relax Definition 3.1.1. Let $C \subseteq [1..\ell]$ be an arbitrary coalition among the ℓ servers and let Q_C denote the random variable that describes the subsequence of query strings $(q_j)_{j \in C}$ that the client sends to the servers in C.

Definition 3.1.4. *The interaction described by* (I, Q, R, E) *provides* perfect privacy with respect to C *(or* perfect C-privacy*) if, for all block indices* $i, i^* \in [1..r]$,

$$\Pr\left[Q_C = (q_j)_{j \in C} \;\middle|\; I = i\right] = \Pr\left[Q_C = (q_j)_{j \in C} \;\middle|\; I = i^*\right],$$

where the probability is over all the random coin tosses made by the client.

Notice that the conditional probabilities on Q_C depend only on the distributions of I and Q:

[2] Indeed, given an ℓ-server protocol that maintains privacy against such a grand coalition, one could easily construct a single-server protocol in which the lone server takes on the roles of all ℓ servers. The resulting protocol would contradict Chor et al.'s impossibility result.

Definition 3.1.4 permits no restrictions on the computational abilities of nor strategies employed by the servers in C—it merely assumes that they learn nothing of the q_j for $j \in [1..\ell] \setminus C$. Many of the PIR constructions considered herein are provably private in the sense of Definition 3.1.4; however, proving that our most efficient construction in Chapter 5 (see §5.6) is private requires—in addition to a non-collusion assumption—a *computational* assumption (namely, the existence of fixed-expansion pseudorandom generators). Such a "computational" notion of privacy for multiserver PIR protocols was first considered by Chor and Gilboa [30] shortly after the introduction of perfectly private PIR.

Definition 3.1.5. *The interaction described by* (I, Q, R, E) *provides* computational privacy with respect to C *(or* computational C-privacy*) if, for all block indices* $i, i^* \in [1..r]$*, the distribution ensembles* $\{Q_C \mid I = i\}_{\lambda \in \mathbb{N}}$ *and* $\{Q_C \mid I = i^*\}_{\lambda \in \mathbb{N}}$ *are computationally indistinguishable (with security parameter* λ*).*

We stress that the hardness of distinguishing distribution ensembles depends not on $\boldsymbol{D}$ but on some underlying security parameter (the size of which will invariably impact the client's computational burden and honest servers). Looking ahead, in Chapter 5, the security parameter in our computationally 1-private bit-more-than-a-bit constructions will dictate the seed-length for some fixed-expansion pseudorandom generator (which we concretely realize using AES).

An interaction that satisfies either Definition 3.1.4 or 3.1.5 with respect to *all* coalitions in the family of subsets $\Gamma = \{C \subseteq [1..\ell] \mid |C| \leq t\}$ for some positive-integer threshold $t < \ell$ is colloquially said to be (*perfectly* or *computationally*) t*-private*. Given Definitions 3.1.2–3.1.5, we can now provide a formal definition for t-private PIR protocols.

Definition 3.1.6. *The interaction described by* (I, Q, R, E) *is a (*perfectly *or* computationally*)* t-private information retrieval protocol *if it provides (i) (*perfect*)* correctness*, (ii)* non-trivial communication*, and (iii) (*perfect *or* computational*)* t-privacy *(in the senses of Definitions 3.1.2–3.1.5, respectively).*

If a given PIR construction is *t-private*, i.e., until and otherwise $t+1$ or more number of servers collude, the scheme assures a secure system that protects the privacy of the query of the client. We call a PIR protocol has the strongest non-collusion assumption when it provides $(t = 1)$-privacy and has the weakest non-collusion assumption when it offers $(t = \ell - 1)$-privacy. Which states that, $1 \leq t \leq \ell - 1$. Our expressive PIR constructions (in Chapter 4) support arbitrary t-privacy and our bit-more-than-a-bit constructions (in Chapter 5) present 1-privacy. We can easily define 1-privacy from Definition 3.1.6 as follows—

Definition 3.1.7. *The interaction described by* (I, Q, R, E) *is a (*perfectly *or* computationally*)* 1-private information retrieval protocol *if it provides (i) (*perfect*)* correctness*, (ii)* non-trivial communication*, and (iii) (*perfect *or* computational*)* 1-privacy *(in the senses of Definitions 3.1.2–3.1.5, respectively).*

CHAPTER 4

QUERYING FOR QUERIES: EXPRESSIVE PIR

4.1 INTRODUCTION

Private information retrieval (PIR) is a cryptographic technique that enables users to fetch records from untrusted and remote database servers without revealing to those servers which particular records are being fetched. This chapter proposes a new technique for conducting *efficient*, *expressive*, and *information-theoretically or perfectly private* PIR queries over structured or semistructured (i.e., tagged) data. Conceptually, the new approach involves building a layer of indirection (realized using a special kind of sparse "database" we call *index of queries*) atop existing PIR protocols.

With only a few exceptions, existing PIR constructions require users to indicate which records they wish to fetch via the indices of those records—that is, via the *physical locations* of those records relative to others in the data store. Our indexes of queries decouple the way that users construct their requests for data from the physical layout of the remote data store, thereby enabling users to fetch data using "contextual" PIR queries that specify *which* data they seek, as opposed to "positional" PIR queries that specify *where* in the database those data happen to reside.

Database operators can construct many different indexes of queries for a given data set, thus providing many distinct views through which users can interact with the underlying data. Abstractly, each index of queries facilitates requests for "best k matches for z", where the precise meaning of 'best,' an upper bound on the number of matches to return k, and a domain of possible

search criteria z are all determined by the database operator and fixed for the particular index of queries under consideration. Queries of the above form arise naturally in a plethora of online and mobile applications. In many such applications, the query term z reveals a great deal of identifiable and potentially sensitive information about the habits, interests, and affiliations of the querier [104]. The index-of-queries approach we propose herein provides significant improvements to both the *efficiency* and *expressiveness* of the most performant and well studied PIR techniques in the literature, exposing intuitive APIs through which applications can safely, quickly, and efficiently interact with the underlying PIR. Therefore, we believe (and certainly hope) that indexes of queries will prove to be a useful building block in the construction of efficient, privacy-preserving alternatives to many widely deployed products and services.

4.1.1 RELATIONSHIP WITH PRIOR WORK

The research literature on PIR is vast; for over two decades, the cryptography, privacy, and theory research communities have studied PIR intensively and from various perspectives. However, this considerable attention notwithstanding, apart from a few notable exceptions, the current work focuses exclusively on an oversimplified model in which users request fixed-length blocks—or even *individual bits!*—of data by specifying the physical locations of those data within the database.

A small body of existing work constructs PIR queries whose expressiveness extends beyond the ability to fetch records by index, including techniques that enable keyword-based [21–23, 31] and simple SQL-based [105, 113, 133] PIR queries. Although our techniques bear a superficial resemblance to these prior efforts, the precise problem we solve and the technical machinery we use to solve it are fundamentally new. Indeed, our approach offers several distinct advantages (and a few limitations) compared with existing techniques. Therefore, we view indexes of queries as complementary to—as opposed to an alternative to—existing techniques for expressive PIR. A later Section 4.6 compares and contrasts indexes of queries with competing approaches. We

defer a detailed comparison to the end of the chapter, for now noting only a handful of practical benefits of our approach. In contrast to existing approaches, the indexes of queries are inherently non-interactive, always requiring just one communication round. They do not require the servers to dynamically replicate data (in whole or part) to answer queries. This not only implies lower round-trip communication latencies, but it also implies less obvious performance advantages related to caching and memory utilization.

4.1.2 MOTIVATION

This chapter's primary objective is to introduce and analyze the indexes of queries as a new PIR technique in the cryptographic engineers' toolkit, rather than to explore the nuances of any particular system that one might use the indexes of queries to build. Nevertheless, to both motivate and ground our proposal, we briefly consider three natural use cases that showcase the immediate applicability of indexes of queries to privacy-respecting technologies' construction. We reiterate that these use cases are merely intended to illustrate a high-level idea; indeed, it is beyond this thesis's scope to present the full architecture of—let alone to treat the minutiae of implementing full, workable systems from—any of these motivating examples.

Use case 1: Maps and location-based recommendation systems. A mapping service like Google Maps[1] or a recommendation service like Yelp[2] could instantiate the indexes of queries over a *Points of Interest* (POI) database to satisfy PIR requests such as for "the 10 cafés nearest my current location" or "the 5 highest rated Bangladeshi restaurants in Manhattan".

Use case 2: Social networks and microblogging platforms. A Twitter[3]-like microblogging service could instantiate indexes of queries over its database of tweets to satisfy PIR requests such as for "the 10 most recent tweets by @realDonaldTrump" or "the 15 top trending tweets for hashtag

[1] https://maps.google.com/
[2] https://www.yelp.com/
[3] https://www.twitter.com/

#ccs17".

Use case 3: Streaming audio and video services. Streaming media services like Youtube[4] or Spotify[5] could instantiate indexes of queries over their respective media catalogs to satisfy PIR requests such as for "the most recent episode of Last Week Tonight with John Oliver" or "the 10 songs topping the latest Billboard Hot 100".

Countless use cases beyond those just listed are possible; e.g., we will use the running example of privately fetching emails from a remote inbox throughout our technical discussions. One could use this idea too, say, hide users' email access patterns from a webmail service like Gmail[6] or, more interestingly, to build a next-generation Pynchon Gate [116] for pseudonymous mail retrieval.

4.2 THE "VECTOR-MATRIX" PIR MODEL

Our constructions are in the ubiquitous *vector-matrix model* for PIR. Vector-matrix PIR is a special case of *linear PIR* where the database is represented as an $r \times s$ matrix $\mathbf{D}$ over a finite field $\mathbb{F}$ in which each of the r rows is a fetchable unit of data (called a *block* in typical PIR parlance). Users encode requests for blocks as vectors on the so-called standard basis for $\mathbb{F}^r$: a user desiring the ith block (i.e., the ith row of $\mathbf{D}$) represents its request with the length-r vector $\vec{e}_i$ having a 1 in its ith coordinate and 0s elsewhere. The response to request $\vec{e}_i$ is defined as the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$, which is easily seen to equal the desired ith row of $\mathbf{D}$. We refer to such vector-based requests as *positional queries* to highlight the fact that they require queriers to know the physical positions (i.e., row numbers) within $\mathbf{D}$ of whatever blocks they seek to fetch.

PIR protocols in the literature obtain privacy in the vector-matrix model through a variety of different means. Of particular interest to us is the information-theoretically private (IT-PIR)

[4] https://www.youtube.com/
[5] https://www.spotify.com/
[6] https://www.gmail.com/

approach based on linear secret sharing. Here, the user "shares" its query vector $\vec{e}_i$ component-wise using a linear secret sharing scheme, and then it sends each of the resulting vectors of shares to a different server from a pool of (non-colluding, but otherwise untrusted) servers whom each hold a replica of $\mathbf{D}$. Upon receiving a share vector from the user, each server independently computes and returns the product with $\mathbf{D}$ of the share vector it just received. As an immediate consequence of linearity, the servers' responses are each component-wise secret sharing of the vector-matrix product $\vec{e}_i \cdot \mathbf{D}$. Thus, to recover its requested block, the user performs a component-wise secret reconstruction over the responses it collects from the various servers.

4.2.1 GOLDBERG'S IT-PIR PROTOCOL

One natural and attractive choice for the secret sharing scheme, the use of which for vector-matrix IT-PIR was first advocated by Goldberg [57], is Shamir's $(t+1, \ell)$-threshold scheme [119]. To share a basis vector $\vec{e}_i$ with Shamir's $(t+1, \ell)$-threshold scheme, the user selects pairwise distinct scalars $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0\}$ and a uniform random vector of polynomials $\vec{\mathbf{F}} \in (\mathbb{F}[x])^r$, subject to the conditions that (i) each polynomial in $\vec{\mathbf{F}}$ has degree at most t, and (ii) a component-wise evaluation of $\vec{\mathbf{F}}$ at $x = 0$ gives $\vec{e}_i$. The jth server receives $(x_j, \vec{\mathbf{Q}}_j)$, where $\vec{\mathbf{Q}}_j = \vec{\mathbf{F}}(x_j)$ is a component-wise evaluation of $\vec{\mathbf{F}}$ at $x_j = j$. We refer to a sequence $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ of $\ell > t$ such ordered pairs (computed from a common $\vec{\mathbf{F}}$ and pairwise distinct x_i) as a *component-wise $(t+1, \ell)$-threshold sharing of $\vec{e}_i$*.

Shamir's threshold scheme provides the requisite linearity and a useful Byzantine robustness property, owing to its relationship with (indeed, equivalence to) Reed-Solomon codes [114] and related multiple-polynomial error-correcting codes [34]. The protocol obtained by using Shamir's $(t+1, \ell)$-threshold scheme in the vector-matrix model realizes *t-private (m, ℓ)-server IT-PIR* for any $m \geq t+1$: the user retrieves its desired block provided $m \geq t+1$ out of ℓ servers respond, yet no coalition of t or fewer malicious servers can use the share vectors its members receive to learn any information about which blocks the user has requested. (It is also *v-Byzantine robust* for any

$v \leq m - t - 2$: the user retrieves its desired block even if up to $m - t - 2$ servers return incorrect responses [44].)

The above intuitive notion of *t-privacy* is formalized by requiring statistical independence between the pair (I, Q) of random variables respectively describing (i) which particular block the querier is requesting, and (ii) the joint view of any coalition of up to t servers involved in the request. We refer the reader to Henry [70, end of §2] for a detailed formal definition for *t-private k-batch (m, ℓ)-server IT-PIR* in the vector-matrix model. Our discussion of the Shamir-based PIR protocol up to this point corresponds to Henry's definition with a fixed batching parameter of $k = 1$; looking forward, the construction we present in Section 4.5 returns several related blocks for each query and, therefore, corresponds to Henry's definition for some $k > 1$.

Our indexes of queries are compatible with any PIR protocol in the vector-matrix model in their most basic form. However, our exposition assumes—and our more sophisticated and useful indexes-of-queries constructions rely on some excellent algebraic properties of—(a scheme [70] that builds upon a scheme [74] that builds upon) Goldberg's Shamir-based IT-PIR; thus, although we have attempted to make our exposition of the new constructions as self-contained as possible, readers unfamiliar with the various building blocks may wish to peruse Goldberg's paper [57]—and the follow-up papers by Henry [70] and by Henry, Huang, and Goldberg [74]—for an initial 'lay of the land.'

4.3 QUERYING FOR QUERIES

At the heart of our approach is a simple observation regarding the use of $(0, 1)$-matrices as PIR database. We begin with the most simplistic possible version of our idea, restricting our attention to $r \times r$ permutation matrices and building up to more general and interesting cases as the chapter progresses.

Recall that an *$r \times r$ permutation matrix* is just an $r \times r$ matrix having exactly one 1 in each row

and each column, and 0s elsewhere (equivalently, it is a matrix obtained by permuting the rows of an $r \times r$ identity matrix). Each such matrix represents a specific permutation with r elements: given a length-r vector $\vec{\boldsymbol{v}}$ and an $r \times r$ permutation matrix Π, the vector-matrix product $\vec{\boldsymbol{v}} \cdot \Pi$ yields a length-r vector with the same components as $\vec{\boldsymbol{v}}$, but in a permuted order.

For example, given $\vec{\boldsymbol{v}} = \langle a \;\; b \;\; c \rangle$ and a permutation matrix

$$\Pi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

it is easy to check that $\vec{\boldsymbol{v}} \cdot \Pi = \langle a \;\; c \;\; b \rangle$; i.e., Π permutes $\vec{\boldsymbol{v}}$ by transposing its second and third components.

The following observation is exceedingly obvious, and yet it is sufficiently central to our approach as to warrant formal explication nonetheless.

Observation 4.3.1. *If $\vec{\boldsymbol{e}} \in \mathbb{F}^r$ is a standard basis vector and $\Pi \in \mathbb{F}^{r \times r}$ is a permutation matrix, then $\vec{\boldsymbol{e}} \cdot \Pi$ is a (possibly different) standard basis vector.*

For example, given the above-defined 3×3 permutation matrix Π and the standard basis $\left\{\vec{\boldsymbol{e}}_1, \vec{\boldsymbol{e}}_2, \vec{\boldsymbol{e}}_3\right\}$ for $\mathbb{F}^3$, we have that $\vec{\boldsymbol{e}}_1 \cdot \Pi = \vec{\boldsymbol{e}}_1$, that $\vec{\boldsymbol{e}}_2 \cdot \Pi = \vec{\boldsymbol{e}}_3$, and that $\vec{\boldsymbol{e}}_3 \cdot \Pi = \vec{\boldsymbol{e}}_2$. In the context of IT-PIR, we are interested in the following immediate corollary to Observation 4.3.1.

Corollary 4.3.2. *Let $\vec{\boldsymbol{e}} \in \mathbb{F}^r$ be a standard basic vector and let $\Pi \in \mathbb{F}^{r \times r}$ be a permutation matrix. If $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ is a component-wise $(t+1, \ell)$-threshold sharing of $\vec{\boldsymbol{e}}$, then $(x_1, \vec{\mathbf{Q}}_1 \cdot \Pi), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell \cdot \Pi)$ is a component-wise $(t+1, \ell)$-threshold sharing of a (possibly different) standard basis vector $\vec{\boldsymbol{e}}' \in \mathbb{F}^r$; namely, of $\vec{\boldsymbol{e}}' = \vec{\boldsymbol{e}} \cdot \Pi$.*

Colloquially, one can think of Corollary 4.3.2 as stating that a t-private IT-PIR query issued against a permutation matrix yields another t-private IT-PIR query (possibly for some other block) or, put another way, that a permutation matrix is, in a sense, just a "database of positional PIR

$$\mathbf{D} \coloneqq \begin{pmatrix} \texttt{subject} & \texttt{sender} & \texttt{date} & \texttt{size} & \texttt{body} \\ \text{Re: ccs2017 submission} & \text{Bob} & \text{2017-02-17} & \text{7.7KB} & \texttt{0x2ff1e1a9...} \\ \text{definitely not a virus} & \text{Dave} & \text{2017-02-1f1} & \text{13.0KB} & \texttt{0xb05fd7a1...} \\ \text{UK-LOTTO sweepstake!} & \text{Alice} & \text{2017-02-04} & \text{336KB} & \texttt{0x0365ce00...} \\ \text{Fwd: Re: Fwd: roflmao} & \text{Carol} & \text{2017-01-07} & \text{2.5KB} & \texttt{0x7e7a36b7...} \\ \text{cash4gold!!!1} & \text{Edward} & \text{2016-12-23} & \text{4.0KB} & \texttt{0xd96dfaff...} \end{pmatrix}$$

Figure 4.1: Toy example of an email inbox database comprising five emails and associated metadata.

queries". Despite the naïvety of our discussion up to this point, we are already well-positioned to demonstrate a novel application of permutation matrices to PIR queries.

4.3.1 EXAMPLE APPLICATION: PRIVATE QUERIES OVER A REMOTE EMAIL INBOX

Consider the toy example of an email inbox depicted in Figure 4.1. The inbox **D** in the figure contains five emails, which are physically stored naturally, in the same order they were received. Each row of **D** represents one email and is structured around a schema that includes—in addition to the `body` of the email—fields for metadata about the email including its `subject`, its `sender`, `date` it was received, and its `size`. Of course, the schema for a real email inbox would include several additional fields.

Suppose we wish to set up a PIR protocol to facilitate the retrieval of emails from the inbox **D**. In a typical PIR setting, the user would fetch an email from **D** using a positional PIR query. Doing so would require the user to know (or, at least, to learn) quite a lot about the *physical layout* of **D**, as the row number of the desired email corresponds to its chronological order among all other emails. By contrast, a typical nonprivate email client would provide a convenient interface to help the user locate emails of interest, for instance, by imposing a user-selected *logical order* on the

emails and allowing the user to browse them in this sorted order. As a concrete example, the email client might present the user with a view of the inbox in which emails are sorted numerically by `size`, or lexicographically by `subject` or `sender`, among other possibilities.

Observation 4.3.3. *Each of the above-mentioned views of* $\mathbf{D}$ *(i.e., sorted by* `size`*, by* `subject`*, or by* `sender`*) corresponds to a particular* 5×5 *permutation matrix.*

For example, referring back to $\mathbf{D}$, we observe that the permutation matrices

$$\Pi_{\texttt{sender}} \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \Pi_{\texttt{size}} \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

respectively, map a query encoding the ith standard basis vector of $\mathbb{F}^5$ to a query for the ith email in a lexicographic ordering of the inbox by `sender` or a numerical (decreasing) ordering of the inbox by `size`. Thus, the user could request, say, the *largest* email in $\mathbf{D}$ by sending a vector of shares of the basis vector $\vec{\boldsymbol{e}}_1 \in \mathbb{F}^5$, along with the hint "`size`", to each of the PIR servers hosting $\mathbf{D}$. We emphasize that the user can construct this query knowing only the total number of rows in $\Pi_{\texttt{size}}$; in particular, the user need not know anything about which emails occupy which rows of $\mathbf{D}$.

Upon receiving the share vector $\vec{\mathbf{Q}}_j$ and hint "`size`" from the user, server j first permutes the components of $\vec{\mathbf{Q}}_j$ via multiplication with $\Pi_{\texttt{size}}$ to get $\vec{\mathbf{Q}}_j^{\texttt{size}} \coloneqq \vec{\mathbf{Q}}_j \cdot \Pi_{\texttt{size}}$, after which it computes and returns the response $\vec{\mathbf{R}}_j \coloneqq \vec{\mathbf{Q}}_j^{\texttt{size}} \cdot \mathbf{D}$ as usual. It is easy to verify—and the reader should take a moment to do so, since going forward we will repeatedly use this simple idea, but in increasingly sophisticated ways—that, upon reconstructing the servers' responses, the user indeed learns the largest email in $\mathbf{D}$, just as it sought to do. To see why, simply note that $\vec{\boldsymbol{e}}_1 \cdot \Pi_{\texttt{size}} = \vec{\boldsymbol{e}}_3$, and that $\vec{\boldsymbol{e}}_3 \cdot \mathbf{D}$

yields the 336 KB email, which has the largest `size` among emails in the inbox.

Before we move on, a few remarks about this simple example are in order. First, we note that the example highlights a potential application of permutations in PIR. However, it reveals no obvious advantage to thinking about such permutations in multiplication by a permutation matrix (as opposed to using some other representation of a permutation). Nevertheless, in the sequel, we will see increasingly sophisticated variations of this idea which *do* rely inherently on the idea of "permuting queries" by way of matrix multiplication.[7] Second, we reiterate that the user in this example *does not require any specific knowledge* about $\mathbf{D}$ or $\Pi_{\texttt{size}}$, beyond the height and semantic meaning of $\Pi_{\texttt{size}}$. Even upon reconstructing the servers' responses, the user still learns nothing about the physical layout of $\mathbf{D}$—not even the row number of the email it just fetched! Finally, although the permuted query vectors arising in our example provide the same t-privacy guarantee as regular queries in the underlying PIR protocol,[8] the additional hint "`size`" does reveal some meta-information about which emails the user is after. This meta-information can have privacy implications; for instance, in our email fetching example, the servers may infer that a user requesting emails by `size` is interested in emails residing in the tail of the `size` distribution (i.e., very small or large emails) as opposed to those near the middle. Thus, in applications that wish to leverage permutations in this way, special care must be taken to identify and quantify if and how such leakage might betray the users' privacy. We emphasize that (i) such leakage is application-dependent and cannot be meaningfully quantified outside the context of a specific application, and (ii) in many (if not most) applications, business logic already betrays similar meta-information.

[7] We point out, moreover, that even for this elementary example, representing the requisite permutations as matrix multiplication is not unreasonable, as the special structure of permutation matrices (specifically, their sparsity and the restriction of their components to $\{0, 1\}$) allows the servers to store and compute with them very efficiently, a fact that will later prove crucial.

[8] Indeed, Shamir's $(t + 1, \ell)$-threshold scheme perfectly hides the secret from coalitions of up to t shareholders; thus, no amount of post-processing—including, of course, multiplication by a permutation matrix—will allow coalition members to extract any information about the user's query.

4.3.2 FROM PERMUTATION MATRICES TO "SIMPLE INDEXES OF QUERIES"

One can think of the permutation matrices Π_{sender} and Π_{size} from the preceding subsection as two "indexes" through which users can fetch the blocks comprising $\mathbf{D}$. Indeed, both indexes are themselves just particular databases whose blocks are all (nonprivate) positional *queries* for blocks in $\mathbf{D}$; in other words, Π_{sender} and Π_{size} are two straightforward examples of what we call "indexes of queries".

Note that such indexes of queries need not take the form of permutation matrices—permutation matrices merely capture the special case in which the index of queries presents a sorted view of all blocks in $\mathbf{D}$. Indeed, one could just as well consider an index of queries Π such that (i) is not square, (ii) has some columns containing no 1s (meaning that certain blocks from $\mathbf{D}$ are not accessible via Π), and/or (iii) has some columns containing multiple 1s (meaning that certain blocks from $\mathbf{D}$ are accessible in multiple ways via Π). One even consider indexes of *aggregate* queries, in which some *rows* may contain several arbitrary non-zero entries. Such indexes of queries would map standard basis vectors to requests for *linear combinations* of blocks from $\mathbf{D}$ and will be useful for solving simple statistical queries. However, we leave the development of this idea to the Section 4.8 and, for the time being, cast the following definition for "simple indexes of queries", which captures all but the last possibility just mentioned.

Definition 4.3.4. *A simple index of queries for a database* $\mathbf{D} \in \mathbb{F}^{r \times s}$ *is a* $(0,1)$*-matrix* $\Pi \in \mathbb{F}^{p \times r}$ *in which each row contains exactly one* 1 *entry.*

An equivalent definition states that $\Pi \in \mathbb{F}^{p \times r}$ is a *simple index of queries* for $\mathbf{D} \in \mathbb{F}^{r \times s}$ if it maps each standard basis vector from $\mathbb{F}^p$ to a standard basis vector from $\mathbb{F}^r$.

4.3.3 LEAKAGE: IT'S NOT A BUG, *IT'S A FEATURE*

In the epilogue to Section 4.3.1, we remarked that the mere act of fetching blocks through a given index of queries could implicitly leak meta-information about which blocks the user seeks. Furthermore, Definition 4.3.4 explicitly permits indexes of queries through which (owing to the presence of all-0 columns) it is impossible to access individual blocks from $\mathbf{D}$, thus potentially making the information leakage *explicit*. In this subsection, we briefly revisit this information leakage—seemingly a weakness of simple indexes of queries—and spin it as a potentially useful feature.

In particular, in use cases where unavoidable implicit leakage is tolerable (or even inevitable), it is possible to reduce the cost of PIR queries by *explicitly leaking the same information*. Trading off some (limited and controlled) information leakage in exchange for more efficient and expressive PIR queries is not without precedence; for example, both of Olumofin and Goldberg's [105] and Wang, Yun, Goldwasser, Vaikuntanathan, and Zaharia's [133] SQL-based PIR queries leak the "shape" of a query while hiding its sensitive constants. Learning the shape of an SQL query may betray information about the possible (and likely) constants in a way analogous to indexes of queries; however, as is the case with our indexes of queries, quantifying precisely how much information is leaked (and how troubling this leakage is) remains highly application-dependent. To make our explicit-leakage proposal more concrete, we return to the earlier example of requesting emails by `size` and suppose that, due to the context in which indexes of queries are being employed, the servers can immediately deduce that any email requested by `size` resides in the "large" tail of the `size` distribution (and yet, for the sake of the example, that such leakage is deemed acceptable). In this case, it is possible to support queries by `size` much more efficiently if we replace $\Pi_{\texttt{size}}$ with a matrix through which only emails in the "large" tail are accessible.

This involves deleting each row of $\Pi_{\texttt{size}}$ that corresponds to an email not in the "large" tail of the `size` distribution, resulting in a rectangular *pseudo-permutation matrix*; that is, in a $p \times r$ matrix that has *at most* one 1 in each row and each column and 0s elsewhere. Thus, we end up

with a short-and-fat $(0,1)$-matrix having *full rank* (i.e., rank p). For instance, the three `largest` emails in $\mathbf{D}$ could be accessed via

$$\Pi_{\texttt{largest}} \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The following analog of Corollary 4.3.2 applies.

Observation 4.3.5. *Let $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ be a standard basic vector and let $\Pi \in \mathbb{F}^{p \times r}$ be a pseudo-permutation matrix with rank p. If $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ is a component-wise $(t+1, \ell)$-threshold sharing of $\vec{\boldsymbol{e}}$, then $(x_1, \vec{\mathbf{Q}}_1 \cdot \Pi), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell \cdot \Pi)$ is a component-wise $(t+1, \ell)$- and $(1, \ell)$-threshold sharing of a standard basis vector $\vec{\boldsymbol{e}}' \in \mathbb{F}^r$; namely, of $\vec{\boldsymbol{e}}' = \vec{\boldsymbol{e}} \cdot \Pi$.*[9]

Intuitively, Observation 4.3.5 implies that a t-private IT-PIR query through a short-and-fat pseudo-permutation matrix yields a t-private IT-PIR query *over a non-hidden subset of a larger database*. Specifically, such a matrix $\Pi \in \mathbb{F}^{p \times r}$ necessarily contains $r - p$ all-0 columns; consequently, every pseudo-permuted share vector $(x_j, \vec{\mathbf{Q}}_j \cdot \Pi)$ has $r - p$ corresponding 0 entries, which means queries through Π cannot fetch blocks corresponding to the all-0 columns in Π. Note that anyone can deduce the set of unfetchable blocks by inspecting Π (or a pseudo-permuted query vector $\vec{\mathbf{Q}}_j \cdot \Pi$), and it is in this sense that Π explicitly leaks information: it explicitly leaks that the request is for a block "indexed by some query" in Π.

The upshot of explicitly leaking this information is twofold. First, the query vectors become shorter (their lengths correspond to the number of queries p in Π, rather than to the number of blocks r in $\mathbf{D}$); thus, each request incurs strictly lower upstream communication cost (p group elements) compared to a positional query over $\mathbf{D}$ (r group elements). Besides, the client does not need to know how many records the database $\mathbf{D}$ holds. Second, because each pseudo-permuted

[9] Specifically, the $r - p$ entries corresponding to all-0 columns in Π are $(1, \ell)$-threshold shares of 0; the remaining p entries are each $(t+1, \ell)$-threshold shares of either 0 or 1.

query vector $\vec{\mathbf{Q}}'_j \coloneqq \vec{\mathbf{Q}}_j \cdot \Pi$ has support of size p, the vector-matrix product $\vec{\mathbf{Q}}'_j \cdot \mathbf{D}$ incurs strictly lower computation cost ($\approx 2ps$ field operations) compared to a positional query over $\mathbf{D}$ ($\approx 2rs$ field operations). We also stress that whatever information does leak is known *a priori* to the user; i.e., although queries leak some information, they do so transparently—there are no surprises.

4.3.4 PRIVACY IN THE FACE OF IMPLICIT AND EXPLICIT INFORMATION LEAKAGE

In the preceding subsection, we claimed that a t-private query through a simple index of queries Π is, at least in some sense, still t-private. Formally proving that this is indeed the case necessitates a slight (though natural) modification to the standard definition of t-privacy. In the light of Section 3.1.4, a direct application of the standard definition would require, for every coalition $C \subseteq [1..\ell]$ of at most t servers and for every record index $i \in [1..r]$, that

$$\Pr\left[I = i \;\middle|\; Q_C = (\Pi;\, \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right] = \Pr\left[I = i\right], \tag{4.1}$$

where I and Q_C denote the random variables respectively describing the block index the user requests and the joint distribution of share vectors it sends to servers in C (including the "hint" that the query should go through Π).

However, it is evident that Equation (4.1) need not hold, for example, when the block in row i of $\mathbf{D}$ is not accessible through Π. It would not suffice to merely restrict the quantifier so that I ranges only over the subset of block indices accessible through Π; indeed, there may be several different indexes of queries, each inducing its *conditional distribution* for I. In other words, a correct definition must account for the fact that curious PIR servers will inevitably—upon learning that a given request is through a particular index of queries Π—leverage this information to update their priors. The following modified t-privacy definition captures this idea.

Definition 4.3.6. *Let* $\mathbf{D} \in \mathbb{F}^{r \times s}$ *and let each* $\Pi_1, \ldots, \Pi_n$ *be an index of queries*[10] *for* $\mathbf{D}$*. Requests are* t*-private with respect to* $\Pi_1, \ldots, \Pi_n$ *if, for every coalition* $C \subseteq [1..\ell]$ *of at most* t *servers, for every record index* $i \in [1..r]$*, and for every index of queries* $\Pi \in \{\Pi_1, \ldots, \Pi_n\}$*,*

$$\Pr\left[I = i \;\middle|\; Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right] = \Pr\left[I = i \;\middle|\; E_\Pi\right],$$

where I *and* Q_C *denote the random variables respectively describing the block index the user requests and the joint distribution of query vectors it sends to servers in* C *(including the "hint" that the query should go through* Π*), and where* E_Π *is the event that the request is through* Π*.*

Observe that a t-private query through a simple index of queries Π is functionally equivalent to—ergo, provides the exact same privacy guarantee as—a t-private positional query over the database $\mathbf{D}_\Pi := \Pi \cdot \mathbf{D}$. Consequently, Definition 4.3.6 reduces to the usual t-privacy definition when $\Pi \in \mathbb{F}^{r \times r}$ is the identity matrix. The next theorem follows from the above observation and the t-privacy of Goldberg's IT-PIR [57, 70].

Theorem 4.3.7. *Let* $\mathbf{D} \in \mathbb{F}^{r \times s}$ *and let each* $\Pi_1, \ldots, \Pi_n$ *be a simple index of queries for* $\mathbf{D}$*. If* $\Pi \in \{\Pi_1, \ldots, \Pi_n\}$ *with* $\Pi \in \mathbb{F}^{p \times r}$ *and if* $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ *is a component-wise* $(t+1, \ell)$*-threshold sharing of a standard basis vector* $\vec{\boldsymbol{e}} \in \mathbb{F}^p$*, then* $(\Pi, x_1, \vec{\mathbf{Q}}_1), \ldots, (\Pi, x_\ell, \vec{\mathbf{Q}}_\ell)$ *is* t*-private with respect to* $\Pi_1, \ldots, \Pi_n$*.*

Proof. Consider a coalition C comprising t servers. Fix $i \in [1..r]$ and $\Pi \in \{\Pi_1, \ldots, \Pi_n\}$ with $\Pi \in \mathbb{F}^{p \times r}$, and let I, J, and Q_C respectively denote the random variables describing the index (within $\mathbf{D}$) of the block the user requests, the index of the standard basis vector the user actually encodes in its query, and the joint distribution of share vectors it sends to servers in C (including the "hint" that the query should go through the index Π).

[10] Our omission of the word "simple" here is intentional: each Π_j can either be simple indexes of queries or one of the more sophisticated types we introduce in the sequel. In particular, by allowing some or all of the Π_j to be different kinds of indexes of queries, we can use Definition 4.3.6 to define privacy for all constructions in this chapter.

As per Definition 4.3.6, we need to show that $\Pr[I = i \mid Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i \mid E_\Pi]$, where E_Π denotes the event that the user's request is through Π. The key observation underlying the proof is that $\Pr[I = i \mid E_\Pi] = \Pr[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid E_\Pi]$ and $\Pr[I = i \mid Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] = \Pr[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})]$.

Hence, we have

$$\begin{aligned}
\Pr\left[I = i \,\middle|\, E_\Pi\right] &= \Pr\left[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \mid E_\Pi\right] \\
&= \Pr\left[\vec{\boldsymbol{e}}_J \cdot \Pi = \vec{\boldsymbol{e}}_i \,\middle|\, Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right] \\
&= \Pr\left[I = i \,\middle|\, Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right],
\end{aligned}$$

as desired. Note that the second line of the above derivation follows immediately from the t-privacy of $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$. □

4.4 BATCH INDEXES OF QUERIES

In the preceding section, we discussed how requests through a simple index of queries could leak meta-information about which blocks the user seeks. We now turn our attention to our first non-trivial indexes of queries, called *batch indexes of queries*, which improve on simple indexes of queries by leveraging ideas from coding theory to decrease this information leakage (and perhaps also improve efficiency).

Suppose we wish to leverage simple indexes of queries for an application. The servers will hold multiple indexes of queries intended to facilitate different kinds of requests, yet all requests will always go through one of these indexes of queries. In this case, knowing only that a given request passed through *some* index of queries yields no information for an attacker: for information about a request to leak, the attacker would have to learn through *which* index of queries that request passed. Thus, concealing through which index of queries each request passes would effectively eliminate this information leakage source while maintaining the utility that indexes of queries

provide.

For example, suppose that a server believes *a priori* that a given request will pass through the index Π_1 with probability p and that it will pass through the index Π_2 with probability $1 - p$, so that

$$\Pr\left[I = i\right] = \Pr\left[I = i \;\middle|\; E_{\Pi_1}\right] \cdot p + \Pr\left[I = i \;\middle|\; E_{\Pi_2}\right] \cdot (1 - p).$$

In this case, if the server receives the hint "1", then it can immediately update its priors to conclude that $\Pr\left[I = i\right] = \Pr\left[I = i \;\middle|\; E_{\Pi_1}\right]$; thus, the hint "1" in this example is leaking information about which block the client is fetching. On the other hand, if the client were somehow able to route its request through Π_1 without revealing through which of Π_1 or Π_2 its request is passing, then the server would be unable to update its priors and the request would leak no new information.

Even in cases where, say, some queries through an index and others are positional, and hence bypass the indexes of queries altogether, hiding through which index a given non-positional request passes would still reduce the quantity of information that leaks. Batch indexes of queries provide one such way to hide through several simple indexes of queries a given request passes.

The batch-indexes-of-queries construction we present here is specific to Goldberg's IT-PIR, leveraging the so-called "u-ary family of codes" [70, §3]. Of course, one could consider analogous instantiations for other PIR protocols or based on other codes; however, we leave the exploration of this idea to future work. Before proceeding, we briefly review u-ary codes and how they are used to construct efficient IT-PIR protocols.

4.4.1 IT-PIR FROM *U*-ARY CODES

Recall that, in the vector-matrix model for PIR (as expounded in Section 4.2), and each server typically holds a complete, plaintext replica of the database $\mathbf{D}$. Several recent IT-PIR constructions [16, 27, 51, 70] have instead considered a generalization of the vector-matrix model wherein

each server holds an encoded *bucket* that is merely derived from—and typically much smaller than—the actual database $\mathbf{D}$. This bucketized vector-matrix approach echoes the benefits of explicit leakage described in the previous section: smaller buckets directly translate into lower upstream communication, lower per-server computation costs, and lower per-server storage costs.

One recently proposed construction in the bucketized vector-matrix model modifies Goldberg's IT-PIR protocol to utilize what is called the *u-ary family of codes* [70, §3 and §5]. In this scheme, each bucket is a matrix of (0-private) "shares" obtained using a "rampified" variant of Shamir's $(1, \ell)$-threshold scheme: given $u \in \mathbb{N}$, the u-ary code encodes $\mathbf{D} \in \mathbb{F}^{r \times s}$ by (i) partitioning the r blocks comprising $\mathbf{D}$ into r/u many u-tuples,[11] (ii) interpolating component-wise through each u-tuple at some predefined x-coordinates to obtain r/u many length-s vectors of degree-$(u-1)$ polynomials from $\mathbb{F}[x]$, and then (iii) placing a single component-wise evaluation of each vector of polynomials into each of the $\ell > u$ buckets. Thus, the bucket held by each of the ℓ servers resides in $\mathbb{F}^{(r/u) \times s}$ and, in particular, is a factor u smaller than $\mathbf{D}$.

Despite no server holding $\mathbf{D}$, users can still fetch blocks of $\mathbf{D}$ using slightly modified t-private positional queries over the buckets. Specifically, a user desiring the ith block from $\mathbf{D}$ needs to determine (i) which of the r/u bucket rows holds evaluations of the polynomial vector passing through the desired block, and (ii) at which x-coordinate that polynomial vector passes through the desired block. It then constructs a length-(r/u) vector of $(t+1, \ell)$-threshold shares encoding a positional query for the above bucket row *at the above x-coordinate* (in contrast to always encoding the positional query at $x = 0$, as it would typically do in Goldberg's protocol). The rest of the protocol is exactly as in the standard vector-matrix model, except that, in the secret reconstruction step, the user interpolates the servers' responses to the same x-coordinate is used to encode its request. The result is a t-private v-Byzantine-robust (m, ℓ)-server IT-PIR protocol

[11] For ease of exposition, here and throughout we make the simplifying assumption that $u \mid r$. Eliminating this assumption is trivial, but doing so would only serve to introduce unnecessary clutter and exceptional cases to our notation and the descriptions of our constructions.

for any $m \geq t + u$ [70, Theorem 1] and $v \leq m - t - u - 1$ [70, Theorem 2]. In particular, note that the number of servers, the privacy threshold, and the downstream communication cost is identical to Goldberg's protocol. In contrast, the upstream communication cost, the storage cost, and the server-side computation cost are all a factor u lower. (The tradeoff for the latter improvements is a reduction by u in the protocol's robustness to non-responding and Byzantine servers.) For additional details and proofs, we refer the reader to the original paper [70].

There are at least two ways to improve our simple indexes of queries using u-ary codes. The most obvious way is to encode a simple index of queries into u-ary buckets, thereby reducing the upstream communication, and possibly the computation cost, associated with queries through that index. This slightly improves efficiency (though, as we will see in Section 4.7, indexes of queries are already plenty fast), but it does nothing to address information leakage. The remainder of this section deals with a more interesting approach that combines multiple disparate indexes of queries into a single batch index, thereby reducing information leakage by letting each user query $\mathbf{D}$ through the index of its choice *without revealing which particular index of queries it uses*. The idea is to merge all the indexes of queries into a single matrix of polynomials using component-wise polynomial interpolation (á la the above u-ary codes) so that each server holds only a single bucket obtained via component-wise evaluation of the resulting polynomial matrix. Users can then formulate requests through any constituent indexes of queries using appropriately crafted queries over the buckets while concealing through which of the simple underlying indexes of queries their requests pass. Before formalizing this idea in Section 4.4.3, we walk through the process of merging the simple indexes of queries $\Pi_{\texttt{sender}}$ and $\Pi_{\texttt{size}}$ from Section 4.3.1.

4.4.2 BATCHING TWO INDEXES OF QUERIES

Recall that $\Pi_{\texttt{sender}}$ and $\Pi_{\texttt{size}}$ are the permutation matrices that respectively map a request encoding the ith standard basis vector of $\mathbb{F}^5$ to a positional query for the ith email in a lexicographic ordering of the email inbox $\mathbf{D}$ depicted in Figure 4.1 by `sender` or a numerical (decreasing) ordering of $\mathbf{D}$

by `size`. They are defined as

$$\Pi_{\text{sender}} \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \Pi_{\text{size}} \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

The merging process is simple. We first fix some x-coordinates, say $x = 0$ and $x = 1$, which serve as identifiers for Π_{sender} and Π_{size}. Then, for each entry in Π_{sender}, we interpolate through that entry (at $x = 0$) and the corresponding entry of Π_{size} (at $x = 1$) to obtain a linear polynomial in $\mathbb{F}[x]$. As both Π_{sender} and Π_{size} are $(0,1)$-matrices, only four polynomials can arise in this step (corresponding to the pairs $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$); i.e., every interpolation yields one of $f_{00}(x) = 0$, $f_{01}(x) = x$, $f_{10}(x) = 1 - x$, or $f_{11}(x) = 1$.

Carrying out this process for Π_{sender} and Π_{size} yields

$$\Pi_{\text{sender,size}}(x) \coloneqq \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1-x & x & 0 & 0 & 0 \\ x & 0 & 0 & 1-x & 0 \\ 0 & 1-x & 0 & 0 & x \\ 0 & 0 & 0 & x & 1-x \end{pmatrix} \in \left(\mathbb{F}[x]\right)^{5\times 5}.$$

One can verify that evaluating $\Pi_{\text{sender,size}}(x)$ component-wise at $x = 0$ and $x = 1$ recovers Π_{sender} and Π_{size}, respectively; indeed, computing the vector-matrix product of $\Pi_{\text{sender,size}}(x)$ with $\vec{e}_i \in \mathbb{F}^5$ and then evaluating the result at $x = 0$ and $x = 1$ yields the ith rows from Π_{size} and Π_{sender}, respectively. For example, $\vec{e}_3 \cdot \Pi_{\text{sender,size}}(x) = \begin{pmatrix} x & 0 & 0 & 1-x & 0 \end{pmatrix}$, which evaluates to $\vec{e}_3 \in \mathbb{F}^5$ and $\vec{e}_1 \in \mathbb{F}^5$ at $x = 0$ and $x = 1$.

Let $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, 1\}$ be arbitrary, pairwise distinct scalars. The bucket held by each server j will be obtained via component-wise evaluation of $\Pi_{\texttt{sender,size}}(x)$ at $x = x_j$. Thus, to fetch the ith email in a lexicographic ordering of $\mathbf{D}$ by `sender`, the user will "encode at $x = 0$" the standard basis vector $\vec{e}_i \in \mathbb{F}^5$ into ℓ vectors, $(x_1, \vec{Q}_1^{(0)}), \ldots, (x_\ell, \vec{Q}_\ell^{(0)})$, of $(t+1, \ell)$-threshold shares; specifically, it will select a length-5 vector of degree-t polynomials from $\mathbb{F}[x]$ uniformly at random, subject only to the requirement that this vector passes component-wise through $\vec{e}_i$ at $x = 0$, and then it will send to each server j the component-wise evaluation $\vec{Q}_j^{(0)}$ of this vector at $x = x_j$. Likewise, to fetch the ith email in a numerical (decreasing) ordering of $\mathbf{D}$ by `size`, the user will "encode at $x = 1$" the same standard basis vector $\vec{e}_i \in \mathbb{F}^5$ into ℓ vectors $(x_1, \vec{Q}_1^{(1)}), \ldots, (x_\ell, \vec{Q}_\ell^{(1)})$ of $(t+1, \ell)$-threshold shares.

Notice that the only difference between how the user constructs the above two requests is the x-coordinate at which it encodes the standard basis vector $\vec{e}_i$; thus, the x-coordinate here serves the same purpose that the hint, "`sender`" or "`size`", served back in Section 4.3.1, allowing the user to specify through which of the two indexes of queries its request is intended to pass. However, in contrast to with the hints that the user explicitly revealed in Section 4.3.1, from the perspective of any coalition of up to t servers, requests encoded at $x = 0$ are perfectly indistinguishable from those encoded at $x = 1$; that is, such a coalition cannot infer (based on the shares its members receive) through which of the two indexes of queries a given request passes [73].

Before we move on, a few brief remarks about this simple example are in order. First, we note that the resulting buckets are still quite sparse, having at most 2 non-zero entries in each row and each column. Second, we observe that the t-privacy of requests through the buckets is still an immediate consequence of [70, Theorem 1]; indeed, the buckets are nothing more than 2-ary buckets of a database obtained by appropriately splicing together the indexes of queries $\Pi_{\texttt{sender}}$ and $\Pi_{\texttt{size}}$. Finally, we point out that, since each entry of $\Pi_{\texttt{sender,size}}$ is an (at-most-)linear polynomial, reconstructing the servers' responses now requires one additional response. Hence, the protocol just described implements t-private (m, ℓ)-server IT-PIR for any $m > t + 1$.

4.4.3 BATCHING u INDEXES OF QUERIES

Definition 4.4.1 formalizes a generalization of the construction from Section 4.4.2, which allows combining for arbitrarily many simple indexes of queries.

Definition 4.4.1. *Fix $u > 1$ and let $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, u-1\}$ be pairwise distinct scalars. A sequence $\Pi_1, \ldots, \Pi_\ell \in \mathbb{F}^{p \times r}$ of matrices is a u-batch index of queries for Goldberg's IT-PIR with bucket coordinates $x_1, \ldots, x_\ell$ if (i) $\Pi_{i_1} \neq \Pi_{i_2}$ for some $i_1, i_2 \in [1..\ell]$, and (ii) for each $j = 0, \ldots, u-1$,*

$$\pi_j \coloneqq \sum_{i=1}^{\ell} \Pi_i \cdot \left(\frac{j - x_1}{x_i - x_1}\right) \cdots \left(\frac{j - x_{i-1}}{x_i - x_{i-1}}\right)\left(\frac{j - x_{i+1}}{x_i - x_{i+1}}\right) \cdots \left(\frac{j - x_\ell}{x_i - x_\ell}\right)$$

is a simple index of queries.

The first requirement of Definition 4.4.1, which insists that $\Pi_{i_1} \neq \Pi_{i_2}$ for some $i_1, i_2 \in [1..\ell]$, is a *non-triviality* requirement included merely to prevent simple indexes of queries from qualifying.[12] The second requirement is what captures the key property we intuitively desire from batch indexes of queries. The expression arising in that second requirement is just the familiar Lagrange interpolation formula. Intuitively, the definition says that the sequence of buckets $\Pi_1, \ldots, \Pi_\ell$ is a u-batch index of queries if interpolating component-wise through the Π_i at each $x = 0, \ldots, u-1$ yields a length-u sequence of simple indexes of queries. The restriction that $x_1, \ldots, x_\ell$ be elements of $\mathbb{F} \setminus \{0, \ldots, u-1\}$ is necessary to guarantee that users can request blocks through the constituent simple indexes of queries without betraying the privacy of their requests (see [70, proof of Theorem 1]).

The next theorem follows quickly from [70, Theorems 1&2].

Theorem 4.4.2. *Fix $u > 1$ and $j \in [0..u-1]$, and let $\Pi = (\Pi_1, \ldots, \Pi_\ell) \in (\mathbb{F}^{p \times r})^\ell$ be buckets of a u-batch index of queries with bucket coordinates $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, u-1\}$. If $(x_1, \vec{\mathbf{Q}}_{j_1}), \ldots, (x_\ell, \vec{\mathbf{Q}}_{j_\ell})$*

[12] Omitting the non-triviality requirement would mean that whenever $\Pi \in \mathbb{F}^{p \times r}$ is a simple index of queries, the sequence of buckets $\Pi, \Pi, \ldots, \Pi \in \mathbb{F}^{p \times r}$ is a u-batch index of queries for every $u \geq 1$. This fails to jibe with what we intuitively mean by the "u-batch" index of queries.

is a sequence of component-wise $(t+1, \ell)$*-threshold shares of a standard basis vector* $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ *encoded at* $x = j$*, then* $(\Pi, x_1, \vec{\mathbf{Q}}_{j_1}), \ldots, (\Pi, x_\ell, \vec{\mathbf{Q}}_{j_\ell})$ *is* t*-private with respect to* Π*.*

Proof. The proof of this theorem is nearly identical to that of Theorem 4.3.7. Consider a coalition C comprising t servers. Fix $i \in [1..r]$ and let I, J, K, and Q_C respectively denote the random variables describing the index (within $\mathbf{D}$) of the block the user requests, the index of the standard basis vector the user actually encodes in its query, the x-coordinate at which it encodes that standard basis vector, and the joint distribution of share vectors it sends to servers in C (including the "hint" that the query should go through the u-batch index of queries Π).

As per Definition 4.3.6, we need to show that $\Pr[I = i \mid Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] = \Pr[I = i \mid E_\Pi]$, where E_Π denotes the event that the user's request is through Π. The key observation is that $\Pr[I = i \mid E_\Pi] = \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, E_\Pi] \cdot \Pr[K = k \mid E_\Pi]$ and $\Pr[I = i \mid Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] = \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] \cdot \Pr[K = k \mid E_\Pi]$. Hence, we have

$$\begin{aligned}
\Pr\left[I = i \,\middle|\, E_\Pi\right] &= \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, E_\Pi] \cdot \Pr[K = k \mid E_\Pi] \\
&= \sum_{k=0}^{u-1} \Pr[\vec{\boldsymbol{e}}_J \cdot \pi_k = \vec{\boldsymbol{e}}_i \mid K = k, Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})] \\
&\qquad\qquad \cdot \Pr[K = k \mid E_\Pi] \\
&= \Pr\left[I = i \,\middle|\, Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right],
\end{aligned}$$

as desired. Note that the second line of the above derivation follows immediately from the t-privacy of $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$. □

Corollary 4.4.3. *The construction just described implements* t*-private* v*-Byzantine-robust* (m, ℓ)*-server IT-PIR for any* $m \geq t + u$*.*

In each of the following results, when we speak of a "u-batch" index of queries, we are implicitly

assuming that u is the largest value for which Definition 4.4.1 is satisfied—i.e., that

$$\pi_u \coloneqq \sum_{i=1}^{\ell} \Pi_i \cdot \left(\frac{u - x_1}{x_i - x_1}\right) \cdots \left(\frac{u - x_{i-1}}{x_i - x_{i-1}}\right)\left(\frac{u - x_{i+1}}{x_i - x_{i+1}}\right) \cdots \left(\frac{u - x_\ell}{x_i - x_\ell}\right)$$

is not another simple index of queries—and that the buckets have minimal degree in this regard. More precisely, we assume that interpolating through the buckets (at the indeterminate x) yields a matrix of polynomials each having degree at most $u - 1$. We also point out that the results all hold for $u = 1$, provided we treat "1-batch index of queries" as synonymous with "simple index of queries". The first observation regards the sparsity of u-batch indexes of queries, while the second regards the possible values that their non-zero entries can take on.

Observation 4.4.4. *Fix $u > 1$. If $\Pi_i \in \mathbb{F}^{p \times r}$ is a bucket of a u-batch index of queries, then the rows and columns of Π_i each contain at most u non-zero entries; hence, the total number of non-zero entries in Π_i is at most $\min(p, r) \cdot u$.*

Observation 4.4.5. *Fix $u > 1$. If $\Pi_i \in \mathbb{F}^{p \times r}$ is a bucket of a u-batch index of queries, then there exists a set C comprising at most $2^u - 1$ scalars from $\mathbb{F}$ such that every non-zero element in Π_i is an element of C.*

Both observations are trivial to prove: it suffices to note that all entries in a bucket are y-coordinates of points on polynomials obtained via interpolating through the u values that reside in corresponding coordinates of the u constituent pseudo-permutation matrices. When all u components are 0, interpolation yields the zero polynomial (Observation 4.4.4); in all cases, every polynomial corresponds to a particular non-zero u-bit binary string.

4.5 INDEXES OF BATCH QUERIES

In the previous section, we proposed batch indexes of queries to obtain all the benefits of simple indexes of queries but with improved privacy guarantees. We now turn our attention to a special

kind of batch indexes of queries, called *indexes of batch queries*, which improve on the earlier batch indexes of queries by enabling users to fetch several related blocks (i.e., a batch of related blocks) with a single request.

Suppose we wish to leverage indexes of queries for an application in which typical requests seek the best k matches for some search term z. An obvious straw man construction would involve creating, for each possible search term z, a simple index of queries $\Pi_z \in \mathbb{F}^{k \times r}$ whose k rows are positional queries for the best k matches for that z. Unfortunately, this trivial solution offers little privacy: knowing which simple index of queries a user's requests go through immediately reveals precisely which blocks those requests are for. In theory, merging all of the simple indexes of queries into a batch index of queries would eliminate this leakage, but this approach does not scale; indeed, several of the motivating use cases from Section 4.1 require best k queries involving *millions of possible search terms*, which would require millions of buckets held by millions of non-colluding servers! Indexes of batch queries provide an alternative construction that facilitates such requests supporting many—perhaps *millions* of—search terms much more efficiently and without requiring a large number of servers.

4.5.1 IT-PIR WITH k-BATCH QUERIES

Recall that in the vector-matrix model for PIR, a typical request takes the form of a positional query represented by a standard basis vector. In the case of Goldberg's IT-PIR, the querier encodes this vector component-wise into ℓ vectors of $(t+1, \ell)$-threshold shares, and then it sends one such vector of shares to each of ℓ servers; thus, a user seeking the blocks referenced by the k rows of one of the simple indexes of queries $\Pi_z \in \mathbb{F}^{k \times r}$ from our straw man construction would need to make k separate requests, respectively encoding the standard basis vectors $\vec{e}_1, \ldots, \vec{e}_k \in \mathbb{F}^k$. Of course, as we already noted, such a user should not expect any privacy.

Henry, Huang, and Goldberg [73] proposed *k-batch queries* as a more efficient way to request k blocks at once. Their k-batch queries are based on the same idea as u-ary codes: instead of

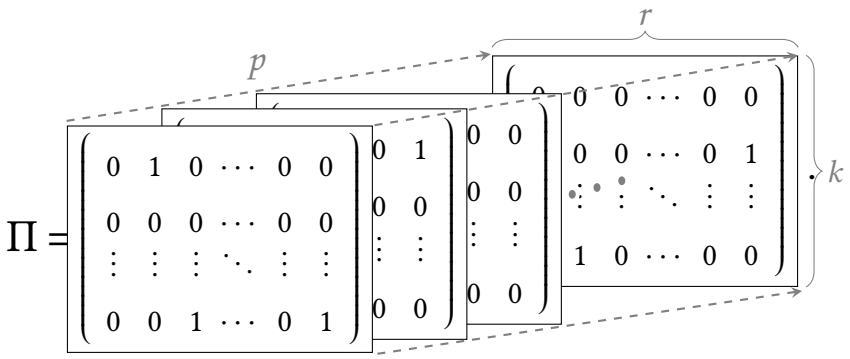
encoding each basis vector $\vec{e}_1, \ldots, \vec{e}_k$ in a separate request, a k-batch query encodes them all in a single request using $(t+1, \ell)$-threshold ramp shares, much like we saw in Section 4.4. Specifically, the user selects a length-k vector of degree-$(t+k-1)$ polynomials uniformly at random, subject to the requirement that, for each $i = 1, \ldots, k$, the vector passes component-wise through $\vec{e}_i$ at $x = i - 1$. Nothing changes from the perspective of the servers[13] and yet a little algebra establishes that, if such a request passes through the simple index of queries $\Pi_z \in \mathbb{F}^{k \times r}$ to a database $\mathbf{D} \in \mathbb{F}^{r \times s}$, then the servers' responses reconstruct to $\vec{e}_1 \cdot \Pi_z \cdot \mathbf{D}$ at $x = 0$, to $\vec{e}_2 \cdot \Pi_z \cdot \mathbf{D}$ at $x = 1$, and so on up to $\vec{e}_k \cdot \Pi_z \cdot \mathbf{D}$ at $x = k - 1$. Of course, the user should still not expect any privacy; we have only succeeded in making the non-private solution more efficient.

Whereas k-batch queries commingle effortlessly with simple indexes of queries, some technicalities interfere when one attempts to naïvely perform k-batch queries through batch indexes of queries (cf. [70, §5]), due to the way batch indexes of queries associate their constituent simple indexes of queries with specific x-coordinates.

4.5.2 *K*-BATCH QUERYABLE BATCH INDEXES OF QUERIES

Our indexes of batch queries are essentially just k-batch indexes of queries that have been constructed to map specific k-batch queries into other, meaningful k-batch queries over $\mathbf{D}$. Conceptually, we "transpose" the impractical straw man construction that began this section in a way that makes the best k queries for each search term z occupy a single row of a k-batch index of queries, at k pairwise distinct x-coordinates. To see how this works, it is helpful to think of the buckets comprising a k-batch index of queries as 2-dimensional projections of a particular 3-dimensional matrix; for instance, if there are p possible search terms z, then the p-batch index of queries arising from the straw man construction would be projections of a matrix Π residing in $\mathbb{F}^{k \times r \times p}$, say

[13] In fact, coalitions of up to t servers cannot distinguish k-batch from non-batch queries [74].



Viewed in this way, it becomes apparent that we should transpose Π concerning its height (k) and depth (p) axes. Doing so yields a matrix $\Pi' \in \mathbb{F}^{p \times r \times k}$ wherein, for each $i = 1, \ldots, p$ and $j = 1, \ldots, k$, the ith "plane" corresponds to a specific search term z_i in which the vector intersecting the "layer" at depth j holds a positional query for the jth-best matching block for z_i in $\mathbf{D}$. Each server will then hold one bucket from a "layer-wise" k-ary encoding of Π'. To fetch the best k matches for search term z_i, the user will simply encode k copies of the standard basis vector $\vec{\boldsymbol{e}}_i$ in a t-private k-batch query, at $x = 0$, at $x = 1$, and so on up to $x = k - 1$. We emphasize that the user here encodes *the same basis vector* at each of $x = 0, \ldots, k - 1$. In a typical k-batch query, multiple encoding copies of the same basis vector would provide no benefit to the user and would only unnecessarily reduce the query's Byzantine robustness. It is also worth noting that the user can choose to encode just $m < k$ copies of $\vec{\boldsymbol{e}}_i$ at $x = 0, \ldots, m - 1$ in order to fetch only the best m matches for search term z_i.

The following definition formalizes the above construction, while the theorem that proceeds it addresses IT-PIR protocol queries' parameters through such an index.

Definition 4.5.1. *Fix $k > 1$ and let $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, k - 1\}$ be pairwise distinct scalars. A sequence $\Pi_1, \ldots, \Pi_\ell \in \mathbb{F}^{p \times r}$ of matrices is an index of k-batch queries for Goldberg's IT-PIR with*

bucket coordinates $x_1, \ldots, x_\ell$ *if, for each* $i = 1, \ldots, p$*, the matrix*

$$\begin{pmatrix} \\ \pi_{i0} \\ \vdots \\ \pi_{i(k-1)} \end{pmatrix}$$

is a pseudo-permutation matrix, where, for each $j \in [0\,..\,k-1]$*,*

$$\pi_{ij} \coloneqq \vec{\boldsymbol{e}}_i \cdot \sum_{n=1}^{\ell} \Pi_n \cdot \left(\tfrac{j-x_1}{x_n-x_1}\right) \cdots \left(\tfrac{j-x_{n-1}}{x_n-x_{n-1}}\right)\left(\tfrac{j-x_{n+1}}{x_n-x_{n+1}}\right) \cdots \left(\tfrac{j-x_\ell}{x_n-x_\ell}\right).$$

We emphasize that indexes of k-batch queries are a particular case of k-batch indexes of queries; hence, Theorem 4.4.2 implies that t-private queries through an index of k-batch queries Π are t-private concerning Π. In light of this, Theorem 4.5.2 is just a restatement of Corollary 4.4.3.

Theorem 4.5.2. *The construction just described implements* t*-private* v*-Byzantine-robust* (m, ℓ)*-server IT-PIR for any* $m \geq t + 2k - 1$ *and* $v \leq m - t - 2k + 1$*.*

4.6 RELATED WORK

This section discusses the small body of existing literature on expressive PIR queries, describing how our new techniques relate to and differ from those prior works.

Keyword-based PIR queries. A technical report by Chor, Gilboa, and Naor [31] proposed a mechanism through which users can fetch blocks privately by specifying *keywords* of interest. Similar to our indexes of queries, they accomplish this by augmenting the database with an auxiliary data structure (a binary search tree, a trie, or a minimal perfect hash function) intended to help users translate keyword-based requests into positional PIR queries, which are ultimately handled by the underlying PIR protocol. Specifically, the user employs positional queries to

obliviously traverse the auxiliary data structure (which, for tree-based data structures, may require many iterative queries) in order to learn the physical location within the database of some record of interest, which it eventually fetches using a final positional PIR query over the actual data.

In contrast to keyword-based PIR, indexes of queries let users fetch data in a single round of interaction, and they do not reveal any information about the structure and layout of the underlying data set. Indeed, the communication and computation costs incurred while fetching records via an index of queries are decoupled from the number of blocks in the database and are, in fact, *upper bounded* by the cost of a positional PIR query over the database (such as the one occurring in the last step of Chor et al.'s scheme).

SQL-based PIR queries. Olumofin and Goldberg [105] extended Chor et al.'s approach to a scheme enabling users to fetch blocks privately using simple *SQL queries* filtered by WHERE or HAVING clauses. Like indexes of batch queries, Olumofin and Goldberg accomplish this by having the database operator prepare (perhaps several) inverted indexes that map sensitive search criteria to the physical locations of associated blocks in the database. Also similar to our approach, their technique may leak some information about which blocks a user seeks, as it hides the sensitive search terms that appear in a query but not the overall "shape" of the query.

Of course, because Olumofin and Goldberg's construction directly build on keyword-based PIR, the differences we highlighted above also differentiate our approach from theirs. Moreover, although a single SQL query in their model may return a batch consisting of several records, this comes at the cost of requiring the user to perform multiple positional queries against the underlying database (indeed, the user must always perform several queries corresponding to the maximum possible size of a response, to avoid leaking information about the actual size of the response); indexes of batch queries, by contrast, can return such batches in a single response using only a single query (and without leaking the size of the response).

PIR from function secret sharing. In terms of functionality, our proposal is most directly

comparable to the current PIR protocols based on Boyle, Gilboa, and Ishai's *function secret sharing* (FSS) [21–23]. FSS provides a way for clients to split certain functions f into pairs of "function shares", which are themselves functions that can be evaluated at an input x to produce additive shares of $f(x)$. This enables the construction of expressive 2-server protocols with which users can fetch records privately using any search criteria expressible as a polynomial sized *branching program*. (FSS constructions that split functions into ℓ-tuples for $\ell > 2$ have also been proposed, thus yielding analogous ℓ-server PIR protocols, but these constructions are dramatically less efficient and require stronger computational assumptions compared to the 2-party construction.)

In contrast to our index-of-queries approach, FSS permits keyword searches without the server's need to prepare auxiliary data structures. However, this added flexibility comes at the cost of stronger security assumptions and a (potentially) higher computation cost. Specifically, unlike the information-theoretic PIR underlying our approach, existing PIR protocols based on (2-party) FSS schemes (i) require a comparatively stronger non-collusion assumption (i.e., that there exists a pair of servers which may not collude), (ii) provide only computational security even when this maximally strong non-collusion assumption holds, and (iii) necessarily incur computational cost comparable to the *upper bound* on that of our index-of-queries approach.

SQL-based PIR queries from FSS. A recent paper by Wang, Yun, Goldwasser, Vaikuntanathan, and Zaharia [133] proposed *Splinter*. This system employs function secret sharing to support a range of queries comparable to those supported by Olumofin and Goldberg's SQL-based approach. Splinter provides both the best and worst of both worlds: on the one hand, Splinter supports a similar set of queries as SQL-based PIR with improved performance (by replacing many recursive PIR-by-keyword queries with single-round FSS queries); on the other hand, it leaks the shape of queries (á la SQL-based PIR) and requires both computational assumptions and rigid non-collusion assumptions (á la FSS-based PIR).[14]

[14]Wang et al. assert that Olumofin and Goldberg's SQL-based PIR "requires all the providers to be honest" [133, §8.2]; however, this claim is false. Indeed, Olumofin and Goldberg provide the same degree of flexibility as our indexes of

Despite the above benefits of indexes of queries over existing keyword-, and SQL-, and function secret sharing-based PIR approaches, there exist use cases in which the latter are more useful—each approach facilitates fundamentally different classes of interactions. Indeed, it is not obvious how to realize efficient keyword-based queries using indexes of queries alone, as this would require users to learn somehow which rows in the index correspond to which keywords. For instance, returning to our running private-inbox-queries example, we note that while many casual interactions with an email client leverage only the views naturally supportable with indexes of queries, users can and do frequently rely on keyword-based searches to locate emails of interest. Thus, an actual private email client would benefit from simultaneous support for both indexes of queries and keyword- or SQL-based queries. Fortunately, because none of the three approaches require any modification to the underlying database, no technical challenges prevent the servers from supporting all of them at the same time.

4.7 IMPLEMENTATION AND EVALUATION

All three variants of indexes of queries introduced in this chapter yield sparse matrices; thus, querying through an index of queries is an instance of sparse matrix-vector (SpMV) multiplication, an embarrassingly parallelizable workload that is particularly well suited to implementation on a massively parallel computing platform, such as a general-purpose GPU device.

In order to empirically gauge the practicality of our indexes-of-queries approach, we implemented finite-field SpMV multiplication. We ran a series of experiments both on an Nvidia Tesla K20 GPU Accelerator [36] and on an Intel Core i5-2500 CPU. Both implementations support SpMV

queries in choosing security assumptions: the default instantiation is unconditionally private provided at most t out of ℓ servers collude, for *any* choice of $t \leq \ell$ including, e.g., $\ell = t - 1$. (By contrast, existing FSS schemes, including those used by Splinter, *only* support $t = \ell - 1$ and, even then, only provide computational privacy against smaller coalitions.) Moreover, one can employ either computational or hybrid PIR in Olumofin and Golberg's framework, thus providing computational privacy when all servers collude and, indeed, even allowing the protocol to run with a single server. This can be accomplished under various computational assumptions, including Paillier's decisional composite residuosity assumption (DCRA) or standard lattice assumption. Finally, setting $t < \ell - 1$ allows the former scheme to provide some level of Byzantine-robustness, which equates to better liveness and potential mitigation of active attacks by small coalitions of servers.

multiplication in the binary fields $\mathbf{GF}(2^8)$ and $\mathbf{GF}(2^{16})$ and in $\mathbb{Z}_q$ for arbitrary multi-precision prime moduli q.[15] We use lookup tables and exclusive-ORs for fast binary field arithmetic; for prime-order field arithmetic, our GPU code uses a hand-optimized PTX implementation of "school-book" multiplication/addition together with Barrett reduction [11], while our CPU implementation outsources arithmetic to NTL [121] and GMP [50]. Our implementations are licensed under version 2 of the GNU General Public License (GPLv2). We are presently working to integrate both versions into Percy++ [58], an open-source implementation of Goldberg's IT-PIR protocol.

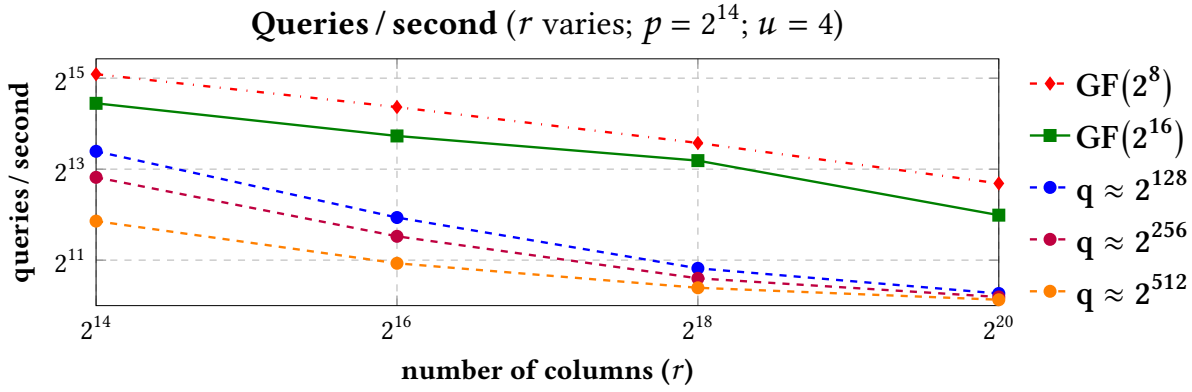
We conducted two sets of experiments. The first set of experiments consists of microbenchmarks designed to measure the latency imposed by routing PIR queries through an index of queries before conducting a positional query against the actual database. The second set of experiments evaluates the feasibility of deploying our techniques over a real-world dataset; specifically, we constructed several batch indexes of queries through which users can fetch academic articles posted to the *IACR Cryptology ePrint archive* [76].

4.7.1 SPMV MICROBENCHMARKS

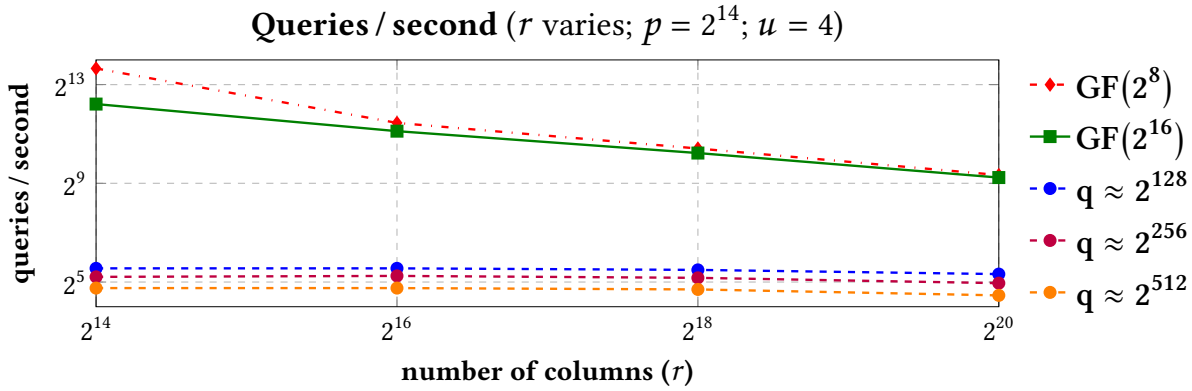
For the first set of experiments, we generated many random u-batch indexes of queries for various choices of u, index dimensions, and finite fields. Then we measured the number of SpMV operations we could evaluate per second, either as a massively parallel computation on our Nvidia K20 GPU Accelerator or as a single-threaded computation on our Intel Core i5-2500 CPU. This experiment's results are unsurprising—our SpMV multiplications consistently run extremely fast, even when the indexes of queries have quite large dimensions.

[15]Numerous efficient CUDA-based SpMV multiplication implementations already exist, yet nearly all implementations we found assume that the entries are floating-point numbers. Modifying any of these implementations to do integer arithmetic modulo a 32-bit word-size prime would be relatively straightforward; however, in order to obtain good PIR performance, we need support for SpMV multiplication over small binary fields and/or over prime-order fields with multiple-precision prime moduli. Indeed, benchmarks we ran on Percy++, an open-source implementation of Goldberg's PIR protocols, indicate that the PIR over small binary fields is fastest, followed by PIR over prime order fields with moduli $\geq$ 128 bits long. For instance, we observe a 3.5× speedup switching from a 32-bit modulus to a 1024-bit modulus.

In line with expectations, we observed that varying the height of the index (p) and the batching parameter (u) had minimal impact on throughput for our GPU implementation,[16] whereas the throughput decreased linearly with pu for our CPU implementation.



(a) 4-batch query throughput on an Nvidia Tesla K20 GPU Accelerator (massively parallel)



(b) 4-batch query throughput on an Intel Core i5-2500 CPU @ 3.30GHz (single-threaded)

Figure 4.2: Number of 4-batch index of queries requests our implementation can process per second. Figure 4.2(a) depicts the counts for a massively parallel implementation on an Nvidia Tesla K20 GPU Accelerator; Figure 4.2(b) depicts the same counts for a single-threaded implementation on an Intel Core i5-2500 CPU. Each experiment was repeated for 100 trials; we report here the mean number of requests per second. Error bars are omitted due to their small size (all standard deviations were below 2% of the mean).

[16] We ran experiments for various choices of $u \in [1..16]$ and power-of-two heights and widths, with dimensions ranging from extremely short-and-fat to perfectly square (but never tall-and-skinny); hence, our indexes were consistently too sparse (at most about 0.1% of entries were nonzero), causing most GPU threads to sit idle most of the time, regardless of how we set u and p.

Figure 4.2 plots the measurements we obtained from one arbitrary-yet-representative set of parameters; specifically, it shows the results for a sequence of indexes of 4-batch queries having $p = 2^{14}$ rows and mapping to databases $\mathbf{D}$ having between $r = 2^{14}$ and $r = 2^{20}$ blocks. In all cases, our GPU implementation was able to process well over a thousand requests per second (indeed, we found that memory bandwidth to and from the GPU was consistently the bottleneck); our CPU implementation was able to process between a few hundred (for $r = 2^{20}$) and a few thousand (for $r = 2^{14}$) requests per second in the binary fields and on the order of a few dozen requests per second (for all r) in large prime-order fields. In all cases, increasing r yielded a roughly linear decrease in throughput, with a slope inversely proportional to the cost of a single field operation.

For comparison, we found that it took just over 1.4 second per GiB of the database (using a single thread) to process a single positional query using fast arithmetic in $\mathbf{GF}(2^8)$, with every other field we measured taking notably longer. Thus, we conclude that, even in the worst conceivable cases, indexes of queries introduce no significant latency to PIR requests (and, when $p \ll r$, they may significantly speed up the subsequent PIR processing by producing positional queries with small support).

4.7.2 IACR CRYPTOLOGY EPRINT ARCHIVE

For the second set of experiments, we created a dataset by scraping the *IACR Cryptology ePrint Archive* [76], an open-access repository that provides rapid access to recent research in cryptology. In particular, we scraped metadata (paper id, paper title, author list, submission date, keywords, and file size) for 10,181 papers (which was the entire dataset as of midday on February 10, 2017, excluding 60 papers that our scraper skipped over because of inconsistently formatted metadata). We also scraped citation counts for each paper in the dataset from *Google Scholar* [62].

Using this data, we constructed a "synthetic ePrint" database. The ith row holds a random bitstring whose length equals the file size of the ith paper in the actual ePrint dataset (padded

with 0s to the length of the most extensive paper).[17] The most extensive paper in the dataset was 19.3 MiB, but only 56 out of the 10,181 papers exceeded 4.69 MiB; therefore, we pruned those 56 papers to obtain a dataset comprising 10,125 papers (the discarded manuscripts were predominantly older PostScript files). This resulted in a 46.35 GiB database (including the 0-padding) of chronologically sorted "synthetic ePrint papers" that users can fetch using IT-PIR queries.

We also constructed histograms to determine (i) the total number of papers associated with each keyword, and (ii) the total number of papers by each author. We identified 1,005 unique keywords associated with five or more different papers each, and 1,750 unique authors that were each associated with four or more different papers each within the pruned dataset. From here, we constructed four different indexes of 4-batch queries over $\mathbf{GF}(2^8)$; namely, we created indexes of 4-batch queries supporting requests for the "4 most highly cited" and the "4 most recently posted" ePrint papers for each keyword (associated with at least five papers) and for each author (associated with at least four papers).

Search criteria	**Sort criteria**	**Simple index generation**	**Bucket generation** (interp. + eval.)	**Bucket size**	**# of nonempty columns**	**GPU index throughput** (queries/sec)	**CPU index throughput** (queries/sec)	**PIR throughput** (secs/query)
Keyword ($p = 1005$)	Recency	1.5 ± 0.1 s	54 ± 9 ms	71.76 KiB	2692	49100 ± 100	32800 ± 400	19.1 ± 0.7 s
	Citations	1.3 ± 0.1 s	52 ± 10 ms	70.17 KiB	2645	49100 ± 100	30700 ± 600	18.8 ± 0.6 s
Author ($p = 1750$)	Recency	3.1 ± 0.1 s	63 ± 6 ms	92.38 KiB	4548	49100 ± 100	22700 ± 400	32.6 ± 0.8 s
	Citations	3.1 ± 0.2 s	63 ± 8 ms	91.69 KiB	4546	49000 ± 100	20100 ± 300	32.4 ± 0.9 s

Table 4.1: Experimental results obtained for the IACR *Cryptology ePrint Archive* [76] dataset. The dataset consists of 10,181 academic papers and associated metadata. All timing experiments were repeated for 100 trials to obtain a standard deviation to one significant figure, and are reported to that precision (± the standard deviation).

We performed two kinds of experiments for each of the four indexes of queries. Table 4.1 summarizes the results of these experiments, as well as some statistics about the time required

[17] We refrained from downloading all ePrint papers and instead opted for random data purely to avoid unduly burdening ePrint with a high volume of unnecessary downloads.

to generate and the storage requirements for each index of queries. First, we measured the total number of requests through each of the four indexes of queries that both our Nvidia Tesla K20 GPU Accelerator and our Intel Core i5-2550 CPU could process per second; given their small dimensions and the choice of working over $\mathbf{GF}(2^8)$, in all cases we managed a whopping 49,000+ queries per second on the GPU and over 20,000 queries per second on a single core of the CPU. Second, we measured the total time required to retrieve a random paper from the dataset using a positional query output by each of the four indexes of queries. Because each of these indexes of queries contains a relatively large number of all-0 columns, the last PIR step's cost was substantially lower than that of a standard positional query. In particular, queries by keyword took around 19 seconds, on average, whereas queries by author took around 33 seconds, on average; by contrast, positional PIR queries over the entire database took nearly 70 seconds, on average. These measurements suggest that indexes of queries can indeed be a useful building block in constructing practical PIR-based systems for datasets on the order of tens of GiB.

4.8 INDEXES OF AGGREGATE QUERIES

In this section, we extend our indexes of queries techniques to support aggregate queries (only when the *effective*[18] database has numerical attributes. We define the standard aggregate vector and simple index of aggregate queries next.

Definition 4.8.1. *Let $\mathcal{I}$ be a set of indices and $2 \leq |\mathcal{I}| \leq r$. A standard aggregate vector $\vec{\boldsymbol{a}}_{\mathcal{I}} \in \mathbb{F}^r$ is a $(0,1)$-vector with an 1 value at each index of the set $\mathcal{I}$.*

Note that when $|\mathcal{I}| = 1$, the standard aggregate vector turns into a standard basis vector. When we multiply a standard aggregate vector with a database, the following observation is straightforward according to the simple linear algebra.

[18]The database owner can create a snapshot of the original database with only numeric attributes and support aggregate queries them.

Observation 4.8.2. *If $\vec{\boldsymbol{a}}_I$ is a standard aggregate vector and I is a set of indices $\{i_1, \ldots, i_h\}$ where $h = \mathrm{wt}(\vec{\boldsymbol{a}}_I)$,*[19] *then $\vec{\boldsymbol{a}}_I \cdot \mathbf{D}$ is a linear combination of particular records (indexed by I) of the database.*

That means, $\vec{\boldsymbol{a}}_I \cdot \mathbf{D} = \vec{\mathbf{D}}_{i_1} + \ldots + \vec{\mathbf{D}}_{i_h} \in \mathbb{F}^s$. We can accumulate such standard aggregate vectors and build an index of aggregate queries. Next, we define the simple index of aggregate queries in light of Definition 4.3.4.

Definition 4.8.3. *An simple index of aggregate queries for a database $\mathbf{D} \in \mathbb{F}^{r \times s}$ is a $(0,1)$-matrix $\Pi \in \mathbb{F}^{p \times r}$ in which each row is a standard aggregate vector.*

Now, we can quickly write the following observations—

Observation 4.8.4. *If $\vec{\boldsymbol{e}} \in \mathbb{F}^r$ is a standard basis vector and $\Pi \in \mathbb{F}^{r \times r}$ is a simple index of aggregate queries, then $\vec{\boldsymbol{e}} \cdot \Pi$ is a standard aggregate vector $\vec{\boldsymbol{a}}_I \in \mathbb{F}^r$, namely, $\vec{\boldsymbol{a}}_I = \vec{\boldsymbol{e}} \cdot \Pi$.*

Observation 4.8.5. *Let $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ be a standard basic vector and let $\Pi \in \mathbb{F}^{p \times r}$ be a simple index of aggregate queries with rank ph. If $(x_1, \vec{\mathbf{Q}}_1), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell)$ is a component-wise $(t+1, \ell)$-threshold sharing of $\vec{\boldsymbol{e}}$, then $(x_1, \vec{\mathbf{Q}}_1 \cdot \Pi), \ldots, (x_\ell, \vec{\mathbf{Q}}_\ell \cdot \Pi)$ is a component-wise $(t+1, \ell)$- and $(1, \ell)$-threshold sharing of a standard aggregate vector $\vec{\boldsymbol{a}}_I \in \mathbb{F}^r$; namely, of $\vec{\boldsymbol{a}}_I = \vec{\boldsymbol{e}} \cdot \Pi$.*[20]

We define the t-privacy for indexes of aggregate queries as follows—

Definition 4.8.6. *Let $\mathbf{D} \in \mathbb{F}^{r \times s}$ and let each $\Pi_1, \ldots, \Pi_n$ be an index of aggregate queries*[21] *for $\mathbf{D}$. Requests are t-private with respect to $\Pi_1, \ldots, \Pi_n$ if, for every coalition $C \subseteq [1..\ell]$ of at most t servers, for every record index $i \in [1..p]$, and for every index of aggregate queries $\Pi \in \{\Pi_1, \ldots, \Pi_n\}$,*

$$\Pr\left[I = i \;\middle|\; Q_C = (\Pi; \vec{\mathbf{Q}}_{j_1}, \ldots, \vec{\mathbf{Q}}_{j_t})\right] = \Pr\left[I = i \;\middle|\; E_\Pi\right],$$

[19] i_h is the position where the last 1 is in the $\vec{\boldsymbol{a}}_I$, and h can be computed by the hamming weight, wt(), of $\vec{\boldsymbol{a}}_I$.

[20] Specifically, the $r - ph$ entries corresponding to all-0 columns in Π are $(1, \ell)$-threshold shares of 0; the remaining ph entries are each $(t+1, \ell)$-threshold shares of either 0 or 1.

[21] Our omission of the word "simple" here is intentional: each Π_j can either be simple index of aggregate queries or one of the more sophisticated types we introduce soon. In particular, by allowing some or all of the Π_j to be different kinds of indexes of aggregate queries, we can use Definition 4.8.6 to define privacy for all constructions in the sequel.

where I and Q_C denote the random variables respectively describing the "category" index the user requests and the joint distribution of query vectors it sends to servers in C (including the "hint" that the query should go through Π), and where E_Π is the event that the request is through Π.

From this definition, we understand that we can replace any "index of queries" matrices in any observations, definitions, and theories with the "index of aggregate queries" so far we studied in this chapter. For example, Theorem 4.4.2 turns into—

Theorem 4.8.7. *Fix $u > 1$ and $j \in [0..u-1]$, and let $\Pi = (\Pi_1, \ldots, \Pi_\ell) \in (\mathbb{F}^{p\times r})^\ell$ be buckets of a u-batch index of aggregate queries with bucket coordinates $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, u-1\}$. If $(x_1, \vec{\mathbf{Q}}_{j_1}), \ldots, (x_\ell, \vec{\mathbf{Q}}_{j_\ell})$ is a sequence of component-wise $(t+1, \ell)$-threshold shares of a standard basis vector $\vec{\boldsymbol{e}} \in \mathbb{F}^p$ encoded at $x = j$, then $(\Pi, x_1, \vec{\mathbf{Q}}_{j_1}), \ldots, (\Pi, x_\ell, \vec{\mathbf{Q}}_{j_\ell})$ is t-private with respect to Π.*

Where we define u-batch index of aggregate queries as follows—

Definition 4.8.8. *Fix $u > 1$ and let $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, u-1\}$ be pairwise distinct scalars. A sequence $\Pi_1, \ldots, \Pi_\ell \in \mathbb{F}^{p\times r}$ of matrices is a u-batch index of aggregate queries for Goldberg's IT-PIR with bucket coordinates $x_1, \ldots, x_\ell$ if (i) $\Pi_{i_1} \neq \Pi_{i_2}$ for some $i_1, i_2 \in [1..\ell]$, and (ii) for each $j = 0, \ldots, u-1$,*

$$\pi_j \coloneqq \sum_{i=1}^{\ell} \Pi_i \cdot \left(\frac{j-x_1}{x_i-x_1}\right) \cdots \left(\frac{j-x_{i-1}}{x_i-x_{i-1}}\right)\left(\frac{j-x_{i+1}}{x_i-x_{i+1}}\right) \cdots \left(\frac{j-x_\ell}{x_i-x_\ell}\right)$$

is a simple index of aggregate queries.

4.8.1 EXAMPLE APPLICATION: PRIVATE AGGREGATE QUERIES OVER AN EPRINT ARCHIVE

We realize how the indexes of aggregate queries do fascinating expressive queries privately by another toy example illustrated in Figure 4.3. The ePrint archive $\mathbf{D}$ holds six papers stored in

$$\mathbf{D} \coloneqq \begin{pmatrix} \texttt{author} & \texttt{keywords} & \texttt{year} & \texttt{\#cites} & \texttt{\#downloads} & \texttt{paper title} \\ \text{Syed Hafiz} & \text{PIR, MPC} & 2020 & 19 & 121 & \text{A bit more than a bit...} \\ \text{Syed Hafiz} & \text{MPC, DPF} & 2020 & 77 & 911 & \text{A scanning-free learn...} \\ \text{Ryan Henry} & \text{DPF} & 2020 & 31 & 642 & \text{Polynomical batch...} \\ \text{Syed Hafiz} & \text{PIR, DPF} & 2019 & 7 & 312 & \text{Querying for queries...} \\ \text{Ryan Henry} & \text{DPF, PIR} & 2019 & 53 & 59 & \text{There are 10 types...} \\ \text{Ryan Henry} & \text{MPC, PIR} & 2019 & 17 & 19 & \text{Pirsona: recommend...} \end{pmatrix}$$

Figure 4.3: Toy example of an ePrint database comprising six papers and associated metadata.

the order they are published on the site. Each record of $\mathbf{D}$ corresponds to one paper and its schema includes the attributes of `author` name, `keywords`, `year` of publication, citation counts `#cites`, number of downloads `#downloads`, and `paper title`. Note that this schema has only two numerical fields on which a PIR client can make the private aggregate query.

Suppose we target to build a PIR set up to support aggregate queries like "total citations and downloads of papers of a given author on a given keyword in the last two years." To facilitate such interesting (but complicated and) aggregate queries, we need two simple indexes of aggregate queries as follows:

$$\Pi_{\texttt{PIR}} \coloneqq \begin{pmatrix} \texttt{(@Syed Hafiz)} & 1 & 0 & 0 & 1 & 0 & 0 \\ \texttt{(@Ryan Henry)} & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \Pi_{\texttt{DPF}} \coloneqq \begin{pmatrix} \texttt{(@Syed Hafiz)} & 0 & 1 & 0 & 1 & 0 & 0 \\ \texttt{(@Ryan Henry)} & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Each row of $\Pi_{\texttt{PIR}}$ and $\Pi_{\texttt{DPF}}$ is a standard aggregate query that is pointing to the papers published in the last two years on a particular keyword. For instance, the second row of $\Pi_{\texttt{DPF}}$ adds the values of numeric attributes (e.g., `#cites` and `#downloads`) of two papers that are published in the last two years and have "DPF" in their `keywords` list. Now we can batch them by the process of

Section 4.4.2, i.e., the 2-ary polynomial batch codes, and produce:

$$\Pi_{\text{PIR,DPF}}(x) \coloneqq \begin{pmatrix} 1-x & x & 0 & 1 & 0 & 0 \\ 0 & 0 & x & 0 & 1 & 1-x \end{pmatrix} \in \left(\mathbb{F}[x]\right)^{2\times 6}.$$

We can easily check that evaluating $\Pi_{\text{PIR,DPF}}(x)$ component-wise at $x = 0$ and $x = 1$ recovers Π_{PIR} and Π_{DPF}, respectively. Each of ℓ servers holds an evaluation of $\Pi_{\text{PIR,DPF}}(x)$ at $x = x_j$, where $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, 1\}$ are arbitrary, pairwise distinct scalars. (Note that each such evaluation, a bucket held by a server, is a 2-batch index of aggregate queries per Definition 4.8.8.) If the client likes to know total citations and downloads of papers of *Ryan Henry* on *PIR* in the last two years, she encodes the standard basis vector $\vec{e}_1 \in \mathbb{F}^2$ at $x = 0$. Likewise, if she wants to fetch total citations and downloads of papers of *Ryan Henry* on *DPF* in the last two years, she encodes the standard basis vector $\vec{e}_1 \in \mathbb{F}^2$ at $x = 1$ in a different query. Moreover, according to Section 4.5.2, we can enable the client to fetch aggregate information of both of them only in a single round by the batch query mechanism. In that case, she has to encode the same standard basis vector $\vec{e}_1$ at $x = 0$ and $x = 1$ at once. This aggregate query brings the information of total citations and downloads of papers of *Ryan Henry* on *PIR* and *DPF* in the last two years, respectively.[22] Thus, we can define the indexes of k-batch aggregate queries as follows:

Definition 4.8.9. *Fix $k > 1$ and let $x_1, \ldots, x_\ell \in \mathbb{F} \setminus \{0, \ldots, k-1\}$ be pairwise distinct scalars. A sequence $\Pi_1, \ldots, \Pi_\ell \in \mathbb{F}^{p\times r}$ of matrices is an index of k-batch aggregate queries for Goldberg's IT-PIR*

[22]Note that to facilitate aggregate queries, the service provider projects on two columns (corresponds to numeric attributes `#cites` and `#downloads`) from the entire database and conceptualize a *thin* database on what the PIR computations occur at the server-side.

with bucket coordinates $x_1, \ldots, x_\ell$ *if, for each* $i = 1, \ldots, p$*, the matrix*

$$\begin{pmatrix} \\ \pi_{i0} \\ \vdots \\ \pi_{i(k-1)} \end{pmatrix}$$

is a simple index of aggregate queries matrix, where, for each $j \in [0..k-1]$*,*

$$\pi_{ij} \coloneqq \vec{e}_i \cdot \sum_{n=1}^{\ell} \Pi_n \cdot \left(\tfrac{j-x_1}{x_n-x_1}\right) \cdots \left(\tfrac{j-x_{n-1}}{x_n-x_{n-1}}\right)\left(\tfrac{j-x_{n+1}}{x_n-x_{n+1}}\right) \cdots \left(\tfrac{j-x_\ell}{x_n-x_\ell}\right).$$

4.9 CHAPTER SUMMARY

We proposed indexes of queries, a novel mechanism for supporting efficient and expressive, single-round queries over multi-server PIR databases. Our approach decouples the way users construct their queries from the physical layout of the database, thereby enabling users to retrieve information using contextual queries that specify *which* data they seek, as opposed to position-based queries that specify *where* in the database those data happen to reside. We demonstrated the feasibility of at least one promising application of our indexes-of-queries approach. We proposed several other compelling possibilities, which we believe present several exciting opportunities for future work.

Another potential avenue for future work is to explore the index of queries approach as it applies to other vector-matrix PIR protocols, leading to additional useful instantiations (e.g., eliminating non-collusion assumptions and compressing queries by settling for computational privacy). Likewise, it would be interesting to explore how other batch codes' families might yield alternative constructions for batch indexes of queries and indexes of batch queries, which may offer different tradeoffs or compatibility with a broader range of PIR protocols.

CHAPTER 5

A BIT MORE THAN A BIT: OPTIMAL-RATE MANY-SERVER PIR

5.1 INTRODUCTION

Private information retrieval (PIR) is a cryptographic primitive enabling clients to fetch records from a remote database without revealing to the database holder(s) which records they fetch. This chapter concerns both the practical and theoretical efficiency of so-called "1-private PIR protocols", wherein the database is hosted jointly by several, say $\ell > 1$, *computationally bounded* and *pairwise non-colluding* servers.

In recent work, Shah, Rashmi, and Ramchandran [118] proposed construction in this model[1] with the following remarkable property: to fetch a b-bit record from a database comprising r such records, the client need only download $b+1$ bits total. In other words, their construction imposes *just one bit* of download overhead compared with the most efficient conceivable non-private protocol, thereby establishing that the *(download) capacity* of 1-private multiserver PIR is effectively perfect. Inspired by Shah et al.'s result, the information-theory research community has produced a steady succession of papers characterizing the download capacity of multiserver PIR protocols under a variety of constraints—when the number of servers is fixed [129]; when a threshold number of servers can collude [128]; when a subset of the servers may be non-responsive [130] or Byzantine [10]; and so on. One can justifiably view Shah et al.'s result and the

[1] In fact, the servers in Shah et al.'s construction—much like the servers in all but the most efficient of our constructions—need not be computationally bounded for the privacy guarantee to hold.

body of work it inspired as a breakthrough in our understanding of multiserver PIR's theoretical underpinnings. Nevertheless, this work has had nearly zero impact on PIR research within the cryptography and applied privacy research communities. Indeed, so aggressively optimizing download cost at the expense of all else imposes rather exorbitant costs for the other metrics of interest—so exorbitant that the constructions in each of the papers mentioned above end up falling short of satisfying any reasonable definition of "non-triviality" for a PIR protocol (see Definition 3.1.3 from Chapter 3).

OUR CONTRIBUTIONS

In this chapter, we revisit the "one-extra-bit" construction of Shah et al. with an eye toward mathematical rigor and concrete practicality. Specifically, we present and analyze a novel family of what we call "*one-extra-word*" protocols, of which the one-extra-bit construction is an exceptional case. Interestingly, the new family also captures as exceptional cases the 2-server perfectly 1-private protocol of Chor, Goldreich, Kushilevitz, and Sudan [33, §2.1] and the computationally 1-private protocol of Boyle, Gilboa, and Ishai [23, §3.2]. In terms of practicality, the fact that our family captures the latter two protocols is significant; indeed, an implicit "folklore" conjecture positing that these protocols are the "most efficient" possible appears to underlie a good deal of existing PIR research [65, 106, 140]. However, our findings soundly refute this folklore conjecture: These exceptional cases are among the *least* practical representatives of our family, with essentially *all other* reasonable parameter settings yielding instances that are *significantly* more practical. Specifically, the performance improvements we obtain relative to these particular cases grows in proportion to the number of pairwise non-colluding servers that participate in each query, up to some inflection point after which further marginal improvements to download and computation costs are dwarfed by rapidly growing upload costs to an unrealistically large number of servers. (Continuing well beyond this inflection point eventually yields the one-extra-bit construction.) The new family sports detailed security and concrete efficiency analyses, which help to elucidate

(and improve on) some curious properties of the efficiency and privacy (or lack thereof) of the original one-extra-bit protocol.

As our main contribution, we exhibit, for any $L \geq 1$, a 2^L-server one-extra-word protocol instance that, although falling short of imposing only a single bit of download overhead, nonetheless (i) provides superior privacy guarantees compared to the one-extra-bit construction and (ii) is *shockingly* efficient (when $L > 1$) for every cost metric—download, upload, computation, and round complexity—typically considered in the PIR literature. For a database comprising r-many b-bit records, the client in our protocol uploads just $130L(\lceil \lg r \rceil - 7)$ bits and downloads just $b/(2^L - 1)$ bits of data per server, while each server performs $L\lceil r/64 \rceil$ AES-128 evaluations and an expected $rb/(2^L - 1)$ bit operations (at the 128-bit security level). These download and computation costs sharply match known lower bounds, suggesting that our protocols are among the most efficient possible. We also extend this protocol to arbitrarily many (non-power-of-2) servers, at the cost of increasing the upload cost and AES-128 evaluation count each by a (small) factor polynomial in the security parameter. This all compares *very* favorably with prior work; based on our analysis and empirical performance measurements from our prototype implementation, we hazard to claim that, all told, ours is the most practical multiserver PIR protocol proposed and implemented to date—by a significant margin.

In short: Our main contribution is to transform Shah et al.'s one-extra-bit construction from a (highly impractical) theoretical result into what we believe to be the most efficient PIR protocol currently in existence, albeit under the same extreme trust assumption employed by the original.

5.2 "ONE-EXTRA-WORD" PROTOCOLS

We now present our generalization of the "one-extra-bit" construction over an arbitrary finite field $\mathbb{F}$; we refer to this generalization as the "one-extra-word" family of PIR protocols. Members

of the one-extra-word family are parametrized by (i) the underlying field $\mathbb{F}$, (ii) the number s of $\mathbb{F}$ elements needed to represent each block, (iii) the number of participating servers $\ell > 1$, and (iv) a vector $v \in \mathbb{F}^s$ and mapping φ to be described below. We call such a tuple of parameters $(\mathbb{F}, s, \ell, \varphi, v)$ a *one-extra-word instance*. (To recover one-extra-bit as a one-extra-word instance, we simply set $\mathbb{F} = \mathbf{GF}(2)$ and $\ell = s + 1$, and then instantiate the v and φ in a particular way—more on this in Section 5.3.)

Notation and preliminaries: Recall that the database $\boldsymbol{D}$ is an $r \times s$ matrix over a finite field $\mathbb{F}$, in which each of the r rows is an s-word block of fetchable data. For each $j \in [1..s+1]$, let $\vec{\boldsymbol{e}}_j$ denote the jth standard basis vector of $\mathbb{F}^{s+1}$. (We also overload $\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_s$ to denote the standard basis vectors in $\mathbb{F}^s$; the dimension of the vector space in which a given $\vec{\boldsymbol{e}}_j$ resides will always be clear from context.)

Denote by $\mathbf{M}^{(r,s)} \subseteq \mathbb{F}^{r \times (s+1)}$ the set of all height-r matrices whose rows are vectors from the standard basis $\left\{\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_{s+1}\right\}$, and consider the family (indexed by $i \in [1..r]$) of equivalence relations $\equiv_i$ defined on the $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ as

$$\boldsymbol{A} \equiv_i \boldsymbol{B} \text{ iff } \mathrm{Row}_{i^*}(\boldsymbol{A} - \boldsymbol{B}) \neq \vec{\boldsymbol{0}} \text{ implies } i^* = i,$$

where $\mathrm{Row}_{i^*}(\boldsymbol{A} - \boldsymbol{B})$ denotes the i^*th row of $\boldsymbol{A} - \boldsymbol{B}$ and $\vec{\boldsymbol{0}}$ denotes the zero vector in $\mathbb{F}^{s+1}$. Read the expression $\boldsymbol{A} \equiv_i \boldsymbol{B}$ as "$\boldsymbol{A}$ is *i-equivalent* to $\boldsymbol{B}$".

The following two-part observation is an immediate consequence of the above definitions of $\mathbf{M}^{(r,s)}$ and the $\equiv_i$.

Observation 5.2.1. *Given any $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ and $i, i^* \in [1..r]$,*

(i) there exists an equivalence class $\mathrm{Eq}(i;\ \boldsymbol{A}) \subseteq \mathbf{M}^{(r,s)}$ comprising precisely $s+1$ matrices (including $\boldsymbol{A}$ itself) such that $\boldsymbol{A} \equiv_i \boldsymbol{B}$ iff $\boldsymbol{B} \in \mathrm{Eq}(i;\ \boldsymbol{A})$, and

(ii) if $\boldsymbol{A} \equiv_i \boldsymbol{B}$ and $\boldsymbol{A} \equiv_{i^*} \boldsymbol{B}$, then $i = i^*$ or $\boldsymbol{A} = \boldsymbol{B}$.

The $s+1$ matrices in $\mathrm{Eq}(i;\ \boldsymbol{A})$ whose existences are guaranteed by Part (i) of Observation 5.2.1

are precisely those matrices in $\mathbf{M}^{(r,s)}$ that are identical to $\boldsymbol{A}$ in all but (perhaps) their ith rows. Moreover, the ith rows of the matrices in $\mathrm{Eq}(i;\, \boldsymbol{A})$ collectively span the entire standard basis for $\mathbb{F}^{s+1}$; in other words, if $\mathrm{Eq}(i;\, \boldsymbol{A}) = \{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_{s+1}\}$, then $\{\mathrm{Row}_i(\boldsymbol{B}_1), \ldots, \mathrm{Row}_i(\boldsymbol{B}_{s+1})\} = \{\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_{s+1}\}$ up to ordering. By convention, we assume the subscripts indexing the $\boldsymbol{B}_j \in \mathrm{Eq}(i;\, \boldsymbol{A})$ are selected so as to induce a "sorted" view in which $\mathrm{Row}_i(\boldsymbol{B}_j) = \vec{\boldsymbol{e}}_j$ for each $j \in [1..s+1]$. Part (ii) of Observation 5.2.1 merely adds that a given pair of distinct matrices can be i-equivalent for *at most one* index $i \in [1..r]$. An easy counting argument establishes that a given pair $(\boldsymbol{A}, \boldsymbol{B})$ need not be i-equivalent for *any* of the $i \in [1..r]$.

Encoding the database: The database encoding algorithm takes as input the database $\boldsymbol{D} \in \mathbb{F}^{r \times s}$, a vector $v \in \mathbb{F}^s$, and a (surjective) function $\varphi\colon \mathbf{M}^{(r,s)} \to [1..\ell]$ mapping each matrix $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ to a positive integer; it outputs a collection of ℓ buckets—one per server—indexed by the $j \in [1..\ell]$. The algorithm first initializes the ℓ buckets to empty and "augments" the database $\boldsymbol{D} \in \mathbb{F}^{r \times s}$ by affixing an $(s+1)$th column, which it computes as the matrix-vector product $\boldsymbol{D}\, v^{\mathrm{T}} \in \mathbb{F}^{s \times 1}$. Denote the augmented database by $\boldsymbol{D}^* \coloneqq \boldsymbol{D} \| \left(\boldsymbol{D}\, v^{\mathrm{T}}\right) \in \mathbb{F}^{r \times (s+1)}$ and observe that, by construction, the final column of $\boldsymbol{D}^*$ is just a linear combination of its first s columns.

Next, to populate the buckets, the algorithm computes, for each of the $(s+1)^r$ matrices $\boldsymbol{A}$ in $\mathbf{M}^{(r,s)}$, the *Frobenius inner product*[2], $\langle \mathbf{D}*, \boldsymbol{A} \rangle_{\mathrm{F}}$, of $\boldsymbol{D}^*$ with $\boldsymbol{A}$, and then it places the result (a *scalar* from $\mathbb{F}$) in the bucket indexed by $\varphi(\boldsymbol{A})$. In the simplest possible instantiations, φ is a bijection so that each of the $(s+1)^r$ resulting scalars constitutes the sole contents of its own bucket (thus necessitating $\ell = (s+1)^r$ non-colluding servers). In the sequel, we discuss more efficient instantiations of φ (including the one employed by one-extra-bit), which allow the scheme to use significantly fewer servers.

Fetching a block: Recall that the database $\boldsymbol{D}$ comprises r blocks and that we defined r equivalence relations $\equiv_i$ over $\mathbf{M}^{(r,s)}$. A query for $\vec{\boldsymbol{D}}_i$, the block at index i within $\boldsymbol{D}$, corresponds to a random

[2]The *Frobenius inner product* of $\boldsymbol{D}^*$ and $\boldsymbol{A}$ is $\langle \mathbf{D}*, \boldsymbol{A} \rangle_{\mathrm{F}} \coloneqq \mathrm{tr}(\boldsymbol{D}^* \boldsymbol{A}^{\mathrm{T}})$ or, equivalently, the sum of the products of each pair of corresponding components in $\boldsymbol{D}^*$ and $\boldsymbol{A}$.

equivalence class of $\mathbf{M}^{(r,s)}$ under $\equiv_i$. Specifically, the client fetches $\vec{D}_i$ by first selecting a uniform random matrix $\boldsymbol{A} \in_{\mathrm{R}} \mathbf{M}^{(r,s)}$, and then retrieving $\langle \mathbf{D}*, \boldsymbol{B}_j \rangle_{\mathrm{F}}$ from bucket $\varphi(\boldsymbol{B}_j)$ for each $\boldsymbol{B}_j \in \mathrm{Eq}(i;\, \boldsymbol{A})$. Because $\equiv_i$ is symmetric (so that $\boldsymbol{B} \in \mathrm{Eq}(i;\, \boldsymbol{A})$ iff $\boldsymbol{A} \in \mathrm{Eq}(i;\, \boldsymbol{B})$), for any given index i, there exist precisely $(s+1)^r / (s+1) = (s+1)^{r-1}$ distinct queries with which the client can request $\vec{D}_i$. The next observation is a direct consequence of the remarks following Observation 5.2.1.

Observation 5.2.2. *If* $\mathrm{Eq}(i;\, \boldsymbol{A}) = \{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_{s+1}\}$, *then the* s *equations* $(\vec{\boldsymbol{e}}_j - v) \cdot \langle x_1, \ldots, x_s \rangle^{\mathrm{T}} = \langle \mathbf{D}*, \boldsymbol{B}_j \rangle_{\mathrm{F}} - \langle \mathbf{D}*, \boldsymbol{B}_{s+1} \rangle_{\mathrm{F}}$ *for* $j \in [1..s]$ *form a system of* s *linear equations in* s *unknowns. Moreover,* $\vec{D}_i = \langle x_1, \ldots, x_s \rangle$ *is in the solution set of this system.*

Note that Observation 5.2.2 implies that the client can compute $\vec{D}_i$ given v and the sequence of Frobenius products $\langle \mathbf{D}*, \boldsymbol{B}_1 \rangle_{\mathrm{F}}, \ldots, \langle \mathbf{D}*, \boldsymbol{B}_{s+1} \rangle_{\mathrm{F}}$ whenever the system of linear equations

$$\begin{pmatrix} \vec{\boldsymbol{e}}_1 - v \\ \vdots \\ \vec{\boldsymbol{e}}_s - v \end{pmatrix} \vec{D}_i^{\mathrm{T}} = \begin{pmatrix} \langle \mathbf{D}*, \boldsymbol{B}_1 \rangle_{\mathrm{F}} - \langle \mathbf{D}*, \boldsymbol{B}_{s+1} \rangle_{\mathrm{F}} \\ \vdots \\ \langle \mathbf{D}*, \boldsymbol{B}_s \rangle_{\mathrm{F}} - \langle \mathbf{D}*, \boldsymbol{B}_{s+1} \rangle_{\mathrm{F}} \end{pmatrix} \tag{5.2.2.1}$$

has full rank and, therefore, a unique solution.

Theorem 5.2.3. *A one-extra-word instance provides* (perfect) correctness *(in the sense of Definition 3.1.2) provided* $\|v\|_1 \neq 1$.[3]

Proof. Write $\sum_{j=1}^{s} c_j(\vec{\boldsymbol{e}}_j - v) = \vec{\mathbf{0}}$, so that $\sum_{j=1}^{s} c_j \vec{\boldsymbol{e}}_j = \sum_{j=1}^{s} c_j v$. If $\sum_{j=1}^{s} c_j = 0$, then $\sum_{j=1}^{s} c_j \vec{\boldsymbol{e}}_j = \vec{\mathbf{0}}$, which (due to the linear independence of $\vec{\boldsymbol{e}}_1, \ldots, \vec{\boldsymbol{e}}_s$) can only happen when $c_j = 0$ for every $j \in [1..s]$. Otherwise, if $\sum_{j=1}^{s} c_j \neq 0$, then $v = \left(\sum_{j=1}^{s} c_j \vec{\boldsymbol{e}}_j\right) / \left(\sum_{j=1}^{s} c_j\right)$, in which case $\|v\|_1 = \sum_{j=1}^{s} \left(c_j / \left(\sum_{j=1}^{s} c_j\right)\right) = \left(\sum_{j=1}^{s} c_j\right) / \left(\sum_{j=1}^{s} c_j\right) = 1$. □

We note that the choice $v = \vec{\mathbf{0}}$ yields an especially convenient (i.e., "already solved") system of equations in which the matrix on the left-hand side of Equation (5.2.2.1) is just the identity

[3] Here $\|v\|_1 \coloneqq \sum_{j=1}^{s} v_j$ denotes the L_1-norm of $v = \langle v_1, \ldots, v_s \rangle$.

matrix in $\mathbb{F}^{s\times s}$ so that $\langle \mathbf{D}*, \boldsymbol{B}_j\rangle_{\mathrm{F}} - \langle \mathbf{D}*, \boldsymbol{B}_{s+1}\rangle_{\mathrm{F}}$ is equal to the jth word of $\vec{\boldsymbol{D}}_i$ for each $j \in [1..s]$. We consider arbitrary v not because it is ever beneficial to use a $v \neq \vec{\mathbf{0}}$—indeed, it seems always best to set $v = \vec{\mathbf{0}}$—but to ensure that the one-extra-bit construction remains a special case of our own.

Communication cost analysis: By inspection, the download cost aggregated across all servers (i.e., $|R|$) is $s+1$ field elements—one element from each of $s+1$ many buckets. When $\mathbb{F} = \mathbf{GF}(2)$, this yields a download cost of just $b+1$ bits to fetch a b-bit block, which is the lowest possible download cost for any PIR protocol [118, Theorem 1].

For each $j \in [1..s+1]$, the client must specify to the server holding bucket $\varphi(\boldsymbol{B}_j)$ which of the $|\varphi^{-1}(\varphi(\boldsymbol{B}_j))|$ Frobenius products from that bucket it seeks; thus, the upload cost aggregated across all servers (i.e., $|Q|$) is

$$F(\boldsymbol{A}) \coloneqq \sum_{j=1}^{s+1} \max\left(\left\lceil \lg\left|\varphi^{-1}(\varphi(\boldsymbol{B}_j))\right|\right\rceil, 1\right) \text{ bits},$$

where the max is due to the fact that the client must still send at least one bit to the server holding bucket $\varphi(\boldsymbol{B}_j)$, as a "signal", even if $|\varphi^{-1}(\varphi(\boldsymbol{B}_j))| = 1$ so that $\lceil \lg|\varphi^{-1}(\varphi(\boldsymbol{B}_j))|\rceil = 0$.

Theorem 5.2.4. *A one-extra-word instance provides* non-trivial communication *(in the sense of Definition 3.1.3) provided the mapping φ satisfies*

$$\max_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}} \left\{\lg\left|\varphi^{-1}(\varphi(\boldsymbol{A}))\right|\right\} \in \mathrm{o}(rw),$$

where $w \in \Omega(\lg|\mathbb{F}|)$ is the bitlength of (the representation used to describe) each $\mathbb{F}$ element.

Proof. Since $|R| = (s+1)w < 2sw \in \mathrm{o}(|\boldsymbol{D}|)$, it will suffice to show that $|Q| \in \mathrm{o}(rws)$. Let $x = \max_{\boldsymbol{A}\in\mathbf{M}^{(r,s)}} \left\{\lg|\varphi^{-1}(\varphi(\boldsymbol{A}))|\right\}$. The size of a query $q = (q_1, \ldots, q_{s+1})$ involving the servers holding

buckets $\mathrm{Eq}(i;\, A) = \{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_{s+1}\}$ is

$$\begin{aligned}
\left|(q_1, \ldots, q_{s+1})\right| &= \textstyle\sum_{j=1}^{s+1} \max\left(\left\lceil \lg\left|\varphi^{-1}\left(\varphi(\boldsymbol{B}_j)\right)\right|\right\rceil, 1\right) \\
&\leq \textstyle\sum_{j=1}^{s+1} \max\left(\lceil x \rceil, 1\right) \\
&= (s+1)\max\left(\lceil x \rceil, 1\right) \\
&= (s+1)\,\mathrm{o}\left(rw\right) \\
&\in \mathrm{o}\left(rws\right).
\end{aligned}$$

□

The condition on $\max_{\boldsymbol{A} \in \mathbf{M}^{(r,s)}} \left\{\lg|\varphi^{-1}\left(\varphi(\boldsymbol{A})\right)|\right\}$ given by Theorem 5.2.4 is *sufficient but far from necessary* to obtain non-triviality. A necessary condition would require only that the function $F(\boldsymbol{A})$ mapping each choice of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ to the upload cost conditioned on that choice satisfies $\mathbf{Exp}\left[F(\boldsymbol{A})\right] \in \mathrm{o}\left(|\boldsymbol{D}|\right)$. However, the simpler (and stricter) condition in Theorem 5.2.4 immediately yields the following corollary.

Corollary 5.2.5. *A one-extra-word instance provides* non-trivial communication *(in the sense of Definition 3.1.3) if each bucket holds a number of Frobenius products in* $2^{\mathrm{o}(rw)}$.

Note that Corollary 5.2.5 implies that a one-extra-word instance provides non-trivial communication under the (perfectly reasonable, yet much stronger) assumption that the storage capacity of each server is polynomially bounded in $|\boldsymbol{D}|$.

Computation cost analysis: The (online) computation cost for each server is effectively nil. The computation cost for the client consists of the cost of setting up the system of equations—which is s subtractions in $\mathbb{F}$—plus the cost of solving that system—which is at most an additional $\mathrm{O}\left(s^{2+\epsilon}\right)$ field operations, for some $\epsilon \geq 0$. In the particular case where $\boldsymbol{v} = \vec{\mathbf{0}}$, the last step is "free" and the total client-side computation cost is just s subtractions in $\mathbb{F}$. (When $\mathbb{F}$ is a binary field, as in the one-extra-bit construction and our bit-more-than-a-bit constructions, each of the s subtractions becomes a bitwise exclusive-OR.)

Security analysis: Privacy of a one-extra-word instance depends on the choice of φ. Note that each server holds one and only one bucket; for convenience, we equate each server with the bucket it holds. The next theorem gives necessary and sufficient conditions for the interaction to provide perfect privacy against singleton coalitions (i.e., against *individual* servers).

Theorem 5.2.6. *A one-extra-word instance provides* perfect 1-privacy *(in the sense of Definition 3.1.4) if and only if the mapping* $\varphi\colon \mathbf{M}^{(r,s)} \to [1..\ell]$ *satisfies the following condition:*

$$\forall \boldsymbol{A} \in \mathbf{M}^{(r,s)} \text{ and } i \in [1..r],\ \left|\varphi\big(\mathrm{Eq}(i;\ \boldsymbol{A})\big)\right| = s+1.$$

Intuitively, Theorem 5.2.6 says that a one-extra-word instance is private (with respect to singleton coalitions) provided φ never maps two distinct matrices from the *same equivalence class* (under *any* of the $\equiv_i$) to the same bucket. We note that this condition is trivially satisfied when φ is a bijection; in the sequel, we describe some other choices for φ that also satisfy the condition outlined in Theorem 5.2.6.

The proof of Theorem 5.2.6 uses Lemma 5.2.7, whose own proof follows easily from Part (i) of Observation 5.2.1. Before stating and proving the lemma, it will be helpful to briefly recall the relevant portions of the process by which the client requests a block $\vec{\boldsymbol{D}}_i$: the client selects a matrix $\boldsymbol{A}$ *uniformly* at random from $\mathbf{M}^{(r,s)}$, and then it retrieves $\langle \mathbf{D}*, \boldsymbol{B}_j \rangle_{\mathrm{F}}$ for each $\boldsymbol{B}_j \in \mathrm{Eq}(i;\ \boldsymbol{A})$.

Lemma 5.2.7. *For any* $i \in [1..r]$ *and any* $\boldsymbol{B} \in \mathbf{M}^{(r,s)}$*, a client seeking to fetch* $\vec{\boldsymbol{D}}_i$ *retrieves the Frobenius product* $\langle \mathbf{D}*, \boldsymbol{B} \rangle_{\mathrm{F}}$ *from* $\varphi(\boldsymbol{B})$ *with probability* $1/(s+1)^{r-1}$.

Proof. Fix $\boldsymbol{B} \in \mathbf{M}^{(r,s)}$ and $i \in [1..r]$, and let X be the (uniform) random variable describing the client's initial selection of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$. We need to prove that $\Pr\big[B \in \mathrm{Eq}(i;\ X)\big] = 1/(s+1)^{r-1}$. From

the Law of Total Probabilities, we have

$$\begin{aligned}\Pr\left[\boldsymbol{B} \in \mathrm{Eq}(i;\, X)\right] &= \textstyle\sum_{\boldsymbol{A} \in \mathbf{M}^{(r,s)}} \Pr\left[\boldsymbol{B} \in \mathrm{Eq}(i;\, X) \;\middle|\; X = \boldsymbol{A}\right] \cdot \Pr\left[X = \boldsymbol{A}\right] \\ &= \textstyle\sum_{\boldsymbol{A} \in \mathbf{M}^{(r,s)}} \Pr\left[\boldsymbol{B} \in \mathrm{Eq}(i;\, X) \;\middle|\; X = \boldsymbol{A}\right] / (s+1)^r,\end{aligned}$$

which, since $\mathrm{Eq}(i;\, \boldsymbol{A}) = \mathrm{Eq}(i;\, \boldsymbol{B})$ iff $\boldsymbol{B} \in \mathrm{Eq}(i;\, \boldsymbol{A})$, equals

$$= \textstyle\sum_{\boldsymbol{A} \in \mathbf{M}^{(r,s)}} \Pr\left[X \in \mathrm{Eq}(i;\, \boldsymbol{B}) \;\middle|\; X = \boldsymbol{A}\right] / (s+1)^r.$$

Now, X is a uniform random variable on a sample space of size $(s+1)^r$ and we know, from Part (i) of Observation 5.2.1, that $|\mathrm{Eq}(i;\, \boldsymbol{B})| = s+1$; hence, it follows that X takes on one of the $s+1$ values in $\mathrm{Eq}(i;\, \boldsymbol{B})$ with probability $(s+1)/(s+1)^r = 1/(s+1)^{r-1}$. Therefore,

$$\begin{aligned}\Pr\left[\boldsymbol{B} \in \mathrm{Eq}(i;\, X)\right] &= \textstyle\sum_{\boldsymbol{A} \in \mathbf{M}^{(r,s)}} \left(1/(s+1)^{r-1}\right)/(s+1)^r \\ &= 1/(s+1)^{r-1}.\end{aligned}$$

□

We emphasize that the probability in Lemma 5.2.7 depends neither on the particular choice of $\boldsymbol{B} \in \mathbf{M}^{(r,s)}$ nor on $i \in [1..r]$. We are now ready to prove Theorem 5.2.6, which follows from Lemma 5.2.7 and Observation 5.2.1.

Proof of Theorem 5.2.6. Let I be the random variable describing the index $i \in [1..r]$ of the block the client seeks and, for each $j \in [1..\ell]$, let $Q_{\{j\}} \colon \mathbf{M}^{(r,s)} \to 2^{\varphi^{-1}(j)}$ be the random variable that associates the client's (uniform random) selection of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ with the subset of matrices $\boldsymbol{B} \in \varphi^{-1}(j)$ for which the client retrieves $\langle \mathbf{D}*, \boldsymbol{B} \rangle_{\mathrm{F}}$ from bucket j as part of its query.

"If": The "if" direction is an easy corollary to Lemma 5.2.7 and Part (i) of Observation 5.2.1. Fix a bucket index $j \in [1..\ell]$ and note that, by construction, $Q_{\{j\}}(\boldsymbol{A}) = \mathrm{Eq}(I;\, \boldsymbol{A}) \cap \varphi^{-1}(j)$.

By Lemma 5.2.7, for any given $i \in [1..r]$ and $\boldsymbol{B} \in \varphi^{-1}(j)$, we have $\Pr\left[\boldsymbol{B} \in Q_{\{j\}} \;\middle|\; I = i\right] =$

$1/(s+1)^{r-1}$; thus, it will suffice to show that $|Q_{\{j\}}(A)| \leq 1$. But this follows contrapositively from the condition $|\varphi\big(\mathrm{Eq}(i;\ \boldsymbol{A})\big)| = s+1$, since $|Q_{\{j\}}| \geq 2$ would imply the existence of $\boldsymbol{B}, \boldsymbol{B}' \in \mathrm{Eq}(i;\ X)$ with $\boldsymbol{B} \neq \boldsymbol{B}'$ and $\varphi(\boldsymbol{B}) = \varphi(\boldsymbol{B}')$ so that $|\varphi\big(\mathrm{Eq}(i;\ X)\big)| < |\mathrm{Eq}(i;\ X)| = s+1$.

"Only if": The "only if" direction follows from Part (ii) of Observation 5.2.1. Suppose there is an $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ with $|\varphi\big(\mathrm{Eq}(i;\ \boldsymbol{A})\big)| < s+1$. Then, by the Pigeonhole Principle, there must be a pair of distinct matrices $\boldsymbol{B}, \boldsymbol{B}' \in \mathrm{Eq}(i;\ \boldsymbol{A})$ such that $\varphi(\boldsymbol{B}) = \varphi(\boldsymbol{B}')$; so, set $j = \varphi(\boldsymbol{B})$ and let $q^* \coloneqq Q_{\{j\}}(\boldsymbol{A})$ be the query string for server j arising when the client selects $\boldsymbol{A}$ in a query for $\vec{\boldsymbol{D}}_i$. By Part (ii) of Observation 5.2.1, we know that $\boldsymbol{B} \notin \mathrm{Eq}(i^*;\ \boldsymbol{B}')$ for any $i^* \neq i$; hence, we have $\Pr\big[Q_{\{j\}} = q^* \ \big|\ I = i\big] > 0$ and $\Pr\big[Q_{\{j\}} = q^* \ \big|\ I = i^*\big] = 0$, which proves that the construction is not perfectly $\{j\}$-private. □

Taken together, Theorems 5.2.3–5.2.6 imply that, given a suitable vector v and mapping φ, the one-extra-word construction yields a perfectly 1-private PIR protocol. Moreover, as noted previously, setting $\mathbb{F} = \mathbf{GF}(2)$ yields a protocol having the lowest download cost possible. The next section discusses the possible choices for φ.

5.3 THE "ONE-EXTRA-BIT" CONSTRUCTION

Let $\vec{\mathbf{1}} \coloneqq \sum_{j=1}^{s} \vec{\boldsymbol{e}}_j$ be the all-1s vector in $\mathbb{F}^s$ and, for each $i \in [1..r]$, define $\mathrm{Ord}_i \colon \mathbf{M}^{(r,s)} \to [0..s]$ such that

$$\mathrm{Ord}_i(\boldsymbol{A}) = j \ \text{ iff } \ \mathrm{Row}_i(\boldsymbol{A}) = \vec{\boldsymbol{e}}_{j+1}.$$

The "one-extra-bit" construction, as originally proposed by Shah et al., is a special case of our one-extra-word construction. It uses the binary field $\mathbb{F} = \mathbf{GF}$, the all-1s vector $v = \vec{\mathbf{1}}$, and a special choice for the mapping $\varphi \colon \mathbf{M}^{(r,s)} \to [1..\ell]$ as defined in Equation (5.1) below.

THE "ONE-EXTRA-BIT" MAPPING Denote by $\mathbf{M}_0^{(r,s)} \coloneqq \left\{ \boldsymbol{A} \in \mathbf{M}^{(r,s)} \;\middle|\; \mathrm{Ord}_1(\boldsymbol{A}) = 0 \right\}$ the subset of matrices $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ having $\vec{\boldsymbol{e}}_1 \in \mathbb{F}^{s+1}$ as their first rows. The following observation is immediate.

Observation 5.3.1. *There are* $(s+1)^{r-1}$ *distinct matrices in* $\mathbf{M}_0^{(r,s)}$.

Indeed, a natural bijection associates with each matrix $\boldsymbol{A} \in \mathbf{M}_0^{(r,s)}$ a non-negative integer less than $(s+1)^{r-1}$ by treating the sequence $\mathrm{Ord}_2(\boldsymbol{A}), \ldots, \mathrm{Ord}_r(\boldsymbol{A})$ as the low- through high-order digits of an $(r-1)$-digit integer expressed in base $(s+1)$; that is, it associates with $\boldsymbol{A}$ the integer $\sum_{i=2}^{r} (s+1)^{i-2} \mathrm{Ord}_i(\boldsymbol{A})$.

The one-extra-bit mapping extends the aforementioned bijection between $\mathbf{M}_0^{(r,s)}$ and $[0\,..\,(s+1)^{r-1}-1]$ to an injection from all of $\mathbf{M}^{(r,s)}$ to $[0\,..\,(s+1)^{r-1}-1]$ as follows. For each $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, let $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbb{F}^{(s+1)\times(s+1)}$ denote the *cyclic permutation matrix* that acts on $\boldsymbol{A}$ (via right multiplication) by rotating it column-wise to the left by $\mathrm{Ord}_1(\boldsymbol{A})$ positions. (In other words, $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}}$ is the cyclic permutation matrix having the property that $\mathrm{Row}_1(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}) = \vec{\boldsymbol{e}}_1$, thereby ensuring that $\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbf{M}_0^{(r,s)}$ for every $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$.)

The one-extra-bit mapping is then given by

$$\varphi(\boldsymbol{A}) \coloneqq \sum_{i=2}^{r} (s+1)^{i-2} \mathrm{Ord}_i(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}). \tag{5.1}$$

The notation and definition in Equation (5.1) are quite unlike the ones proposed by Shah et al. [118, Algorithm 1]; however, it is straightforward, if tedious, exercise to verify that the two definitions are, in fact, equivalent.

ANALYSIS OF THE "ONE-EXTRA-BIT" CONSTRUCTION. Observe that $\varphi(\boldsymbol{A}) = \sum_{i=2}^{r} (s+1)^{i-2} \mathrm{Ord}_i(\boldsymbol{A})$ if and only if $\boldsymbol{A} \in \mathbf{M}_0^{(r,s)}$. More generally, we have the following observation.

Observation 5.3.2. *Let* $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$. *Then* $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ *iff* $\boldsymbol{B}$ *is among the* $s+1$ *column-wise cyclic permutation of* $\boldsymbol{A}$.

An immediate consequence of Observation 5.3.2 is that any two (distinct) matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ having $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ necessarily differ on *every single row*, a fact from which the next corollary directly follows.

Corollary 5.3.3. *If $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ such that $\boldsymbol{A} \neq \boldsymbol{B}$ but $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$, then $\boldsymbol{A} \notin \mathrm{Eq}(i; \boldsymbol{B})$ for any $i \in [1..r]$.*

Note, in particular, that Corollary 5.3.3 implies $|\varphi\big(\mathrm{Eq}(i; \boldsymbol{A})\big)| = s + 1$ for every $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$; hence, this choice of φ satisfies the necessary and sufficient conditions given by Theorem 5.2.6 and the next corollary is immediate.

Corollary 5.3.4. *The one-extra-bit construction provides* perfect 1-privacy *(in the sense of Definition 3.1.7).*

The (online) computation cost for each server is effectively nil;[4] the computation cost for the client to set up and solve the resulting system of equations is about s^2 bit operations. The client can uniquely reference a specific Frobenius product $\langle \mathbf{D}*, \boldsymbol{A} \rangle_{\mathrm{F}}$ via the ordered pair (x, j) in which $x = \mathrm{Ord}_1(\boldsymbol{A})$ and $j = \varphi(\boldsymbol{A})$; hence, for any $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, the client need only send $\lceil \lg(s+1) \rceil$ bits to each of the $s + 1$ servers in $\varphi\big(\mathrm{Eq}(i; \boldsymbol{A})\big)$. More precisely, $|\varphi^{-1}\big(\varphi(\boldsymbol{A})\big)| = s + 1$ for all $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ so that φ satisfies the condition given by Theorem 5.2.4 as long as s is polynomial in r. This proves the following.

Lemma 5.3.5. *The one-extra-bit construction provides* non-trivial communication *(in the sense of Definition 3.1.3), provided $s \in 2^{\mathrm{o}(r)}$.*

Combining Corollary 5.3.4 and Lemma 5.3.5 with the fact that $\|\vec{\mathbf{1}}\|_1 = s$ (cf. Theorem 5.2.3) yields the following theorem.

[4] Section 5.3.1 recalls a modest optimization due to Shah et al., which slightly reduces the per-server storage cost of any one-extra-word construction that uses $v = \vec{\mathbf{1}}$ together with the one-extra-bit mapping φ (as defined in Equation (5.1)) at the expense of requiring a small (amortized) computation cost per query.

Theorem 5.3.6. *If $s \in 2^{o(r)}$ with $s > 1$, then the one-extra-bit construction is a perfectly 1-private $(s+1)^{r-1}$-server PIR protocol.*

The preceding theorem establishes that the one-extra-bit construction provides perfect 1-privacy, which is the strongest privacy result possible for the construction. Indeed, as we argue below, there necessarily exist (many) doubleton coalitions $C \subseteq [1..\ell]$ with respect to which the construction is clearly *not* C-private. Yet it is apparent that the one-extra-bit construction *must* be private against *at least some* non-singleton coalitions. Section 5.4 analyzes the necessary and sufficient conditions on a coalition $C \subseteq [1..\ell]$ under which a one-extra-word instance is C-private.

5.3.1 TRADING A BIT OF WORK FOR A BIT LESS STORAGE

In this sub-section, we briefly recall and generalize the following simple observation of Shah et al.; regarding the one-extra-bit construction, which leads to a modest improvement in the per-server storage cost. Recall that one-extra-word uses $v = \vec{1}$

Observation 5.3.7. *Fix $j \in [1..\ell]$ and consider the set $\varphi^{-1}(j) = \{A_1, \ldots, A_{s+1}\}$ comprising the $s+1$ distinct matrices in $\mathbf{M}^{(r,s)}$ that map to bucket j. We have*

$$\sum_{i=1}^{s+1} \langle \mathbf{D}*, A_i \rangle_{\mathrm{F}} = 0, \tag{5.3.7.1}$$

or, equivalently,

$$\sum_{i=1}^{s} \langle \mathbf{D}*, A_i \rangle_{\mathrm{F}} = \langle \mathbf{D}*, A_{s+1} \rangle_{\mathrm{F}}. \tag{5.3.7.2}$$

Observation 5.3.7 follows from the choice of $v = \vec{1}$ and the fact that $\mathbb{F} = \mathbf{GF}$ is a field of characteristic 2. Shah et al. point out that each server can store just s bits (rather than $s+1$ bits), say, $\langle \mathbf{D}*, A_1 \rangle_{\mathrm{F}}, \ldots, \langle \mathbf{D}*, A_s \rangle_{\mathrm{F}}$, and then use Equation (5.3.7.2) to compute the discarded bit

$\langle \mathbf{D}*, \boldsymbol{A}_{s+1} \rangle_{\mathrm{F}}$ on the fly, as needed. The tradeoff for this modest optimization is an increase in the amortized server-side computation cost, which increases from nil to (just under) one **GF** addition per request (on average): Exactly one out of $s + 1$ servers must reconstruct its discarded bit for each request and, on average, each server must reconstruct its discarded bit for one out of every $s + 1$ requests in which it participates.

We remark (sans formal proof) that the same idea generalizes easily to an arbitrary field $\mathbb{F}$ by setting $v = (p - 1)\vec{\mathbf{1}}$, where $p = \mathrm{char}(\mathbb{F})$ is the characteristic of $\mathbb{F}$.[5]

5.4 NECESSARY AND SUFFICIENT CONDITIONS FOR C-PRIVACY

The privacy guarantees of a one-extra-word instance are completely determined by choice of φ. Intuitively, for any coalition C, a one-extra-word instance is C-private if (i) it provides 1-privacy, and (ii) no two servers in C ever participate in the same query regardless of which block the client is requesting, nor the outcomes of any coin toss the client makes. The next theorem formalizes this intuition.

Theorem 5.4.1. *Let $C \subseteq [1..\ell]$ be a coalition. A one-extra-word instance is* C-private *if and only if, for each server $j \in C$,*

1. *the construction is $\{j\}$-private (cf. Theorem 5.2.6), and*
2. *for all $\boldsymbol{A} \in \varphi^{-1}(j)$ and $i \in [1..r]$, $\varphi\big(\mathrm{Eq}(i; \boldsymbol{A})\big) \cap C = \{j\}$.*

Proof (sketch). Let $n = \sum_{j \in C} |\varphi^{-1}(j)|$. The first condition guarantees that no client can ever request two or more distinct Frobenius products from a single coalition member (as part of the same query), while the second condition extends the first to ensure that no client will ever request two or more distinct Frobenius products from two or more different coalition members. It follows that

[5] Intuitively, for any $\varphi^{-1}(j) = \{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{s+1}\}$, summing $\boldsymbol{A}_1 + \cdots + \boldsymbol{A}_{s+1}$ yields the all-1s matrix in $\mathbb{F}^{r \times (s+1)}$ so that $\sum_{i=1}^{s+1} \langle \mathbf{D}*, \boldsymbol{A}_i \rangle_{\mathrm{F}}$ is just a summation over all entries in $\mathbf{D}*$. Now, by construction, the last entry in each row of $\mathbf{D}*$ is a product of the sum of all earlier entries in the row with $p - 1$; thus, each row sums to p so that $\sum_{i=1}^{s+1} \langle \mathbf{D}*, \boldsymbol{A}_i \rangle_{\mathrm{F}} = rp$, which, since $\mathbb{F}$ has characteristic p, is equal to 0 in $\mathbb{F}$.

there are precisely $n+1$ possible "joint views" that can arise for coalition C in each query: namely, the client can request any one of the n Frobenius products that coalitions' members hold, or it can request nothing at all from any of the coalition members.

By Lemma 5.2.7, the first n events each happen with probability $1/(s+1)^{r-1}$, regardless of which block the client seeks. Moreover, because these $n+1$ events are pairwise disjoint and mutually exhaustive, it follows that the remaining ("nothing requested") event occurs with probability $1 - n/(s+1)^{r-1}$. As all probabilities are independent of the block being fetched, this establishes what we set out to prove. □

Theorem 5.4.1 and the remarks following Theorem 5.3.6 suggest that the privacy guarantees of the one-extra-bit construction may, in fact, be much *stronger* than Corollary 5.3.4 and Theorem 5.3.6 suggest; the reality, however, is rather nuanced.

PRIVACY OF THE "ONE-EXTRA-BIT" CONSTRUCTION, REVISITED We first note that if $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbf{M}^{(r,s)}$ are distinct matrices such that $\boldsymbol{A}_1 \in \mathrm{Eq}(i; \boldsymbol{A}_2)$ for some $i \in [1..r]$, then the construction cannot be $\{\varphi(\boldsymbol{A}_1), \varphi(\boldsymbol{A}_2)\}$-private; indeed, letting I denote the random variable that describes the index of the block the client seeks, and letting F_1 and F_2 respectively denote the events that the client requests $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ as part of its query, it follows from Lemma 5.2.7 that

$$\Pr\left[F_1 \cap F_2 \;\middle|\; I = i\right] = 1/(s+1)^{r-1},$$

whereas it follows from Part (ii) of Observation 5.2.1 that

$$\Pr\left[F_1 \cap F_2 \;\middle|\; I \neq i\right] = 0.$$

Hence, there exist *many* doubleton coalitions with respect to which privacy cannot be guaranteed. (Indeed, each $j \in [1..\ell]$ belongs to rs distinct doubleton coalitions C with respect to which the construction is not C-private.) Thus, Theorem 5.3.6 gives the strongest "threshold" notion of

privacy one can hope for.

What is more, the one-extra-bit mapping φ, as defined in Equation (5.1), results in a protocol with the peculiar property that is merely knowing that *any given pair* of servers holding a pair of i-equivalent Frobenius products is (or is not) involved in a query (without necessarily knowing what *query strings* those servers receive) is sufficient for a *passive observer*—such as the querier's Internet service provider—to compromise the querier's privacy! We are unaware of any other PIR protocol whose privacy falls to such *traffic analysis* attacks when the observer may be limited to seeing only encrypted query strings. This surprising and troubling property is due to the following observation and its corollary.

Observation 5.4.2. *Let φ be as defined in Equation (5.1). If $\boldsymbol{B}_1 \in \mathrm{Eq}(i;\, \boldsymbol{A}_1)$ and $\boldsymbol{B}_2 \in \mathrm{Eq}(i;\, \boldsymbol{A}_2)$, then $\varphi(\boldsymbol{A}_1) = \varphi(\boldsymbol{A}_2)$ if and only if $\varphi(\boldsymbol{B}_1) = \varphi(\boldsymbol{B}_2)$.*

Recalling that we defined $\boldsymbol{\mathcal{P}}_{\boldsymbol{A}} \in \mathbb{F}^{(s+1)\times(s+1)}$ as the permutation matrix such that $\mathrm{Row}_1(\boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}) = \vec{\boldsymbol{e}}_1$, an equivalent formulation of Observation 5.4.2 is:

$$\boldsymbol{B} \in \mathrm{Eq}(i;\, \boldsymbol{A}) \text{ iff } \boldsymbol{B} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{B}} \in \mathrm{Eq}(i;\, \boldsymbol{A} \cdot \boldsymbol{\mathcal{P}}_{\boldsymbol{A}}).$$

That is, if *some* Frobenius product in bucket j_1 has an i-equivalent partner in bucket j_2, then (i) *every* Frobenius product in j_1 has an i-equivalent partner in j_2, and (ii) no Frobenius product in j_1 has a $\star i$-equivalent partner in j_2 for any $\star i \neq i$ (see Part (ii) of Observation 5.2.1).

Corollary 5.4.3. *Let I denote the random variable that describes the index of the block the client seeks, and let F_1 and F_2 denote the events that a client requests a Frobenius product from bucket j_1 and from j_2, respectively. Then, we have that either (i) $\Pr\left[F_1 \cap F_2\right] = 0$, or (ii) there exists $i \in [1..r]$, such that*

$$\Pr\left[I = i \;\middle|\; F_1 \cap F_2\right] = 1 \tag{5.4.3.1}$$

and

$$\Pr\left[I = i \;\middle|\; F_1 \cap \bar{F}_2\right] = 0. \tag{5.4.3.2}$$

Colloquially, Equation (5.4.3.1) says that knowing *any pair* of servers involved in a given query is sufficient to determine the index of the block sought uniquely. Moreover, Equation (5.4.3.2) says that if $\boldsymbol{B} \in \mathrm{Eq}(i;\, \boldsymbol{A})$, then knowing that $\varphi(\boldsymbol{A})$ is involved in a query, while $\varphi(\boldsymbol{B})$ is not, is sufficient to deduce that the request is not for block i.

5.5 "BIT-MORE-THAN-A-BIT" PROTOCOLS

We now describe our new "bit-more-than-a-bit" construction, a family of perfectly 1-private one-extra-word protocols parametrized by the number of buckets $\ell \geq 2$ and the number of words per block s. (Recall that the number of buckets equals the number of servers.) Each member of the bit-more-than-a-bit family uses a binary field $\mathbb{F} = \mathbf{GF}(2^w)$ where $w = \lceil \frac{b}{s} \rceil$, the all-0s vector $v = \vec{\mathbf{0}}$, and the mapping $\varphi\colon \mathbf{M}^{(r,s)} \to [1..\ell]$ defined in Equation (5.2) below.

The "bit-more-than-a-bit" mapping: The new mapping is much simpler than the one-extra-bit mapping described in Section 5.3. It represents each matrix $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$ as an r-digit integer expressed in radix $(s+1)$ using a "natural" bijection, and then it assigns each integer to a bucket according to its congruence class modulo ℓ. The mapping is given by

$$\varphi(\boldsymbol{A}) \coloneqq \textstyle\sum_{i=1}^{r} (s+1)^{i-1}\, \mathrm{Ord}_i(\boldsymbol{A}) \bmod \ell, \tag{5.2}$$

where $\mathrm{Ord}_i(\boldsymbol{A}) = j$ iff $\mathrm{Row}_i(\boldsymbol{A}) = \vec{\boldsymbol{e}}_{j+1}$.

This mapping induces ℓ buckets, compared with $(s+1)^{r-1}$ buckets for the one-extra-bit mapping (cf. Equation (5.1)); hence, relative to the one-extra-bit construction, it reduces the number of servers required from $(b+1)^{r-1}$—which is super-exponential in $|\boldsymbol{D}|$—to an arbitrary constant

$\ell \geq 2$.

A consequence of this super-exponential reduction in the number of buckets is that each bucket now contains about $(s+1)^r/\ell$ distinct w-bit Frobenius products. Specifically, the buckets now have a size that grows (at least formally) exponentially with the number of blocks comprising $\boldsymbol{D}$. Fortunately, because each Frobenius product is easily computed as a sum of just r-many words from $\mathbf{GF}(2^w)$, it suffices for the servers to store a copy of $\boldsymbol{D}$ and then construct needed Frobenius products on the fly. We discuss this idea further in Section 5.5.1.

Communication cost analysis (or "choosing a field size"): As each bit-more-than-a-bit instance is just a special instance of the one-extra-word construction, the analysis in Section 5.2 implies that the download cost (i.e., $|R|$) to fetch a b-bit block is just $(s+1)w \approx (b+w)$ bits. This suggests that setting w to be very small—for example, setting $w = 1$, as in the one-extra-bit construction—gives an ideal download cost.

Yet small constants like $w = 1$ lead to enormous bucket sizes and, thereby, fail to yield protocols that are non-trivial (in the sense of Definition 3.1.3). Indeed, to query for block i, the client must uniquely reference a specific Frobenius product from among the $\approx (s+1)^r/\ell$ such products held by each of the $s+1$ buckets in an i-equivalence class. To reference the Frobenius product $\langle \mathbf{D}*, \boldsymbol{A} \rangle_{\mathrm{F}}$, the client uses an ordered pair (x, j) in which $x = \lfloor \frac{a}{\ell} \rfloor$ and $j = a \bmod \ell$ respectively denote the quotient and remainder obtained upon dividing $a \coloneqq \sum_{i=1}^{r} (s+1)^{i-1} \operatorname{Ord}_i(\boldsymbol{A})$ by ℓ; hence, for any choice of $\boldsymbol{A} \in \mathbf{M}^{(r,s)}$, the client sends $(r-1)\lceil \lg(s+1) \rceil$ bits to each of the $s+1$ servers in $\varphi\big(\mathrm{Eq}(i;\, \boldsymbol{A})\big)$. This gives an upload cost aggregated across all servers (i.e., $|Q|$) of $(s+1)(r-1)\lceil \lg(s+1) \rceil$ bits. Notably, if $w = 1$, then this is $(b+1)(r-1)\lceil \lg(b+1) \rceil \in \omega\big(|\boldsymbol{D}|\big)$ and the construction fails to provide non-trivial communication. To yield a non-trivial protocol it is therefore necessary to set w somewhat larger (thereby incurring higher download overhead). The next theorem characterizes the choices of w (and, consequently, s) that yield protocols providing non-trivial communication.

Theorem 5.5.1. *A bit-more-than-a-bit instance provides* non-trivial communication *(in the sense*

of Definition 3.1.3) if and only if $w \in \omega(1)$ *or, equivalently, if and only if* $s \in o(b)$.

In light of Theorem 5.5.1, we suggest choosing a desired (say, small constant) number of words per block $s \in \mathbb{N}$ and then selecting the field to be $\mathbf{GF}(2^w)$ for $w := \lceil \frac{b}{s} \rceil$. The next observation follows by inspection.

Observation 5.5.2. *Let* ℓ *and* s *be positive integers such that* $\ell \geq 2$ *and* $s < \min(\ell, b+1)$. *A bit-more-than-a-bit instance with* ℓ *buckets and* s *words per* b*-bit block has upload and download costs of* $(s+1)(r-1)\lceil \lg(s+1) \rceil$ *and* $(s+1)w \approx \frac{s+1}{s} b$ *bits, respectively.*

Notice that setting $s = 1$ yields a 2× download overhead (ignoring ceilings) with an upload cost linear in r—indeed, one can effectively view a protocol instance with $s = 1$ as a 2-server instance of the seminal PIR protocol for blocks due to Chor et al. [32, §6]. Increasing s decreases the download overhead *geometrically* while only increasing the upload cost *arithmetically* (ignoring the logarithmic term).

Computation cost analysis: As each Frobenius product is precomputed during the setup phase, the (online) computation cost for each server remains effectively nil. The client-side computation cost is s additions in $\mathbf{GF}(2^w)$, which equates to about b bit operations—one exclusive-OR for each bit in the requested block.

Security analysis: We establish sufficient conditions for the perfect 1-privacy of our bit-more-than-a-bit constructions in Theorem 5.5.4 below, which is an easy corollary to Theorem 5.2.6 and the following lemma.

Lemma 5.5.3. *Let* s *and* ℓ *be positive integers such that* $s < \ell$ *and* $\gcd(s+1, \ell) = 1$. *If* a *and* b *are distinct positive integers satisfying* $a \equiv b \bmod \ell$, *then the radix-*$(s+1)$ *representation of* a *and* b *differ in at least two digit positions.*

Proof. Assume without loss of generality that $a > b$. If a and b differ in just one digit position, then there must exist integers $c \in [1..s]$ and $d \geq 0$ such that $a - b = c(s+1)^d$. Now, since $a \equiv b \bmod \ell$,

it follows that $\ell \mid c(s+1)^d$. But $\gcd(s+1, \ell) = 1$, so by a generalization of Euclid's Lemma to arbitrary integers, we have $\ell \mid c$, which contradicts the restriction that $c \in [1..s]$ and the fact that $s < \ell$. □

Theorem 5.5.4. *Let s and ℓ be positive integers such that $s < \ell$ and $\gcd(s+1, \ell) = 1$. A bit-more-than-a-bit instance with ℓ buckets and s words per block provides* perfect 1-privacy *(in the sense of Definition 3.1.4).*

Proof. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$ be any pair of distinct matrices. From Lemma 5.5.3, $\varphi(\boldsymbol{A}) = \varphi(\boldsymbol{B})$ implies that $a \coloneqq \sum_{i=1}^{r} (s+1)^{i-1} \operatorname{Ord}_i(\boldsymbol{A})$ and $b \coloneqq \sum_{i=1}^{r} (s+1)^{i-1} \operatorname{Ord}_i(\boldsymbol{B})$ differ in at least two digits or, equivalently, that $\boldsymbol{A}$ and $\boldsymbol{B}$ differ in at least two rows. In turn, this implies that $\boldsymbol{B} \notin \operatorname{Eq}(i; \boldsymbol{A})$ for any $i \in [1..r]$. Since this is true for all distinct $\boldsymbol{A}, \boldsymbol{B} \in \mathbf{M}^{(r,s)}$, it follows that each element of $\operatorname{Eq}(i; \boldsymbol{A})$ resides in a different bucket; i.e., that $|\operatorname{Eq}(i; \boldsymbol{A})| = s + 1$. Hence, by Theorem 5.2.6, it follows that the construction provides perfect 1-privacy. □

Combining Corollary 5.5.4 and Theorem 5.5.1 with the fact that $\|\vec{\mathbf{0}}\|_1 = 0$ (cf. Theorem 5.2.3) yields the following theorem.

Theorem 5.5.5. *Let s and ℓ be positive integers such that $s < \ell$ and $\gcd(s+1, \ell) = 1$. A bit-more-than-a-bit instance with ℓ buckets and s words per block is a* perfectly 1-private ℓ-server PIR protocol. *Moreover, the protocol has download rate $\frac{s+1}{s} + o(1)$.*

We note that, in contrast to the one-extra-bit construction, our bit-more-than-a-bit constructions maintain perfect privacy in the face of "traffic analysis" attacks (cf. Section 5.4). Indeed, if we let X denote the random variable describing which of the ℓ servers receive no message in a given query, then for all $i \in [1..r]$ and $j \in [1..\ell]$, we find that $\Pr[j \in X \mid I = i] = 1 - |\varphi^{-1}(j)|/(s+1)^{r-1}$. The latter probability is a consequence of (i) the one-to-one correspondence between queries for block i and equivalence classes under $\equiv_i$, and (ii) the fact that server j holds exactly one element from each of $|\varphi^{-1}(j)|$ out of $(s+1)^{r-1}$ such equivalence classes.

5.5.1 VIRTUAL BUCKETS

Recall that each bucket in a bit-more-than-a-bit instance having s words per block contains about $(s+1)^r/\ell$ distinct w-bit Frobenius products. This quantity scales exponentially with the number of blocks in $\boldsymbol{D}$. Fortunately, because each Frobenius product is easily computed as a sum of just r-many words from $\mathbf{GF}(2^w)$—i.e., one word from each row of $\boldsymbol{D}^*$—it suffices for a server to store a copy of $\boldsymbol{D}$ and then construct needed Frobenius products on the fly. In fact, having all servers store $\boldsymbol{D}$ eliminates the need to associate each server with a fixed bucket, thereby eliminating the requirement from Theorems 5.5.4 and 5.5.5 that $\gcd(s+1,\ell)=1$. Indeed, with this approach, we are free to set $\ell = s+1$ so that the client involves all servers in all queries. For this reason, we say that a server that stores all of $\boldsymbol{D}$ and computes arbitrary Frobenius products on the fly holds a *virtual bucket* of $\boldsymbol{D}$.

Virtual buckets fix the per-server storage cost as $|\boldsymbol{D}|$ bits, while increasing the online computation cost to at most r additions in $\mathbf{GF}(2^w)$, each of which is implemented as a bitwise exclusive-OR of w-bit binary strings. However, on average a fraction $1/(s+1)$ of the rows of $\boldsymbol{A}$ contain $\vec{\boldsymbol{e}}_{s+1}$, which, because we set $v = \vec{\mathbf{0}}$, induces a no-op; hence, the *expected* computation cost per server is closer to $\frac{rs}{s+1}$ additions in $\mathbf{GF}(2^w)$. In particular, the expected computation cost aggregated across all servers is about rs additions in $\mathbf{GF}(2^w)$, or about one exclusive-OR per bit in $\boldsymbol{D}$, with each of $s+1$ participating servers shouldering a roughly equal computational burden, and the *worst-case* computation cost is only nominally higher. This compares favorably with other ℓ-server protocols in the literature; indeed, Beimel, Ishai, and Malkin [13, Theorem 6.4] proved that using *even one fewer bit operation* in expectation is impossible without increasing the storage cost by having each server store extra, pre-processed information about $\boldsymbol{D}$.

Observation 5.5.6. *Suppose $\ell - 1 \mid b$. A bit-more-than-a-bit construction with ℓ virtual buckets and $s = \ell - 1$ words per b-bit block has download cost of exactly $(1 + \frac{1}{\ell-1})b$ bits.*

The $(1+\frac{1}{\ell-1})b$-bit download cost from Observation 5.5.6 matches the best possible download

for any ℓ-server PIR protocol, provided that r is sufficiently large [17, Theorem 2.5].

5.6 COMPUTATIONALLY 1-PRIVATE "BIT-MORE-THAN-A-BIT" PROTOCOLS

This section presents our most efficient construction, a *computationally* 1-private variant of the bit-more-than-a-bit family of PIR protocols with virtual buckets. The simplest form of this computationally 1-private construction, which can be instantiated with $\ell = 2^L$ servers for any positive integer L, reduces the per-server upload cost from rL bits in the perfectly 1-private bit-more-than-a-bit construction to just $(\lambda + 2)\lceil \lg \frac{r}{\lambda} \rceil L$ bits, where λ is a security parameter (say, $\lambda = 128$ or 256). The download cost remains unchanged. The computation cost increases only modestly in practice (assuming both clients and servers are equipped with modern CPUs that support the AES-NI instruction set).

The computationally 1-private construction replaces the *uniform random* query strings from our perfectly 1-private construction with *pseudorandom* query strings generated by an L-tuple of *2-out-of-2 distributed point functions* [56], each mapping integers from $[1..r]$ to 1-bit outputs. A *point function* in this context refers to a function $p_i : [1..r] \to \{0, 1\}$ such that

$$p_i(i^*) = \begin{cases} 1 & \text{if } i^* = i, \text{ and} \\ 0 & \text{otherwise;} \end{cases}$$

while one can think of a *distributed* point function as an extremely compact, secret-shared representation of a point function.

We provide the following formal definition for a 2-out-of-2 distributed point function with 1-bit outputs, which we adapt from Definition 2.2 of Boyle, Gilboa, and Ishai [23, §2] and Definition 1 of Gilboa and Ishai [56, §2].

Definition 5.6.1. *Let $\mathbb{P}(r)$ denote the set of all point functions $p_i\colon [1..r] \to \{0,1\}$ with domain $[1..r]$ and 1-bit outputs. A pair of probabilistic polynomial-time algorithms* (Gen, Eval) *with syntax*

- $\mathsf{Gen}(1^\lambda;\, i)$*, taking $\lambda \in \mathbb{N}$ and $i \in [1..r]$, outputs a* DPF key pair*, $(k_0, k_1) \in \{0,1\}^* \times \{0,1\}^*$; and*
- $\mathsf{Eval}(k, i^*)$*, taking $k \in \{0,1\}^*$ and $i^* \in [1..r]$, outputs a bit,*

is a 2-out-of-2 distributed point function*, or* $(2,2)$-DPF*, for $\mathbb{P}(r)$ if it provides the following two guarantees:*

(i) ***(Perfect) correctness****: For all inputs $i, i^* \in [1..r]$, if $(k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda;\, i)$, then*

$$\mathsf{Eval}(k_0, i^*) \oplus \mathsf{Eval}(k_1, i^*) = p_i(i^*).$$

(ii) ***(Computational) secrecy****: There exists a PPT algorithm* Sim *such that the distribution ensembles*

$$\left\{\mathsf{Sim}(1^\lambda;\, b)\right\}_{b \in \{0,1\}}$$

and

$$\left\{k_b \;\middle|\; (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda;\, i)\right\}_{b \in \{0,1\}, i \in [1..r]}$$

are computationally indistinguishable.

The following (relatively trivial) lemma will be convenient for proving that our DPF-based query construction provides computational 1-privacy (à la Definition 3.1.5), which is an indistinguishability-based (rather than simulation-based) notion of privacy.

Lemma 5.6.2. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$*, then for all* $i, i^* \in [1..r]$ *and for all* $b \in \{0,1\}$*, the distribution ensembles*

$$\left\{k_b \;\middle|\; (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda;\, i)\right\}_{b\in\{0,1\}}$$

and

$$\left\{k_b \;\middle|\; (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda;\, i^*)\right\}_{b\in\{0,1\}}$$

are computationally indistinguishable.

Proof (sketch). This follows immediately from the definition of computational secrecy for a $(2,2)$-DPF (Part (ii) of Definition 5.6.1) and the triangle inequality. □

Our construction uses the efficient $(2,2)$-DPF construction of Boyle, Gilboa, and Ishai [23, Figure 1] with 1-bit outputs. The computational secrecy of that construction relies only on the existence of a fixed-expansion pseudorandom generator $G\colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda+2}$; a computationally efficient candidate for such a PRG is easily constructed from any block cipher. A typical choice for concrete instantiations is to use 128- or 256-bit AES (hence our reliance on CPUs with the AES-NI instruction set for efficiency in practice).

The efficiency of our construction also benefits from the fast *full-domain evaluation* algorithm of Boyle et al. [23, §3.2.1], which expands a $(2,2)$-DPF key k into the length-r bit vector

$$\mathsf{EvalFull}(k) := \left\langle \mathsf{Eval}(k,1), \mathsf{Eval}(k,2), \ldots, \mathsf{Eval}(k,r) \right\rangle$$

using just $\lceil r/\lambda \rceil$ PRG evaluations [22, Theorem 3.3].

We also introduce the '$\|$' operator to denote the component-wise concatenation of length-r

vectors over $\{0,1\}^*$; in particular, if $\vec{\mathbf{u}} = \langle u_0, \ldots, u_n \rangle$ and $\vec{v} = \langle v_0, \ldots, v_n \rangle$, then

$$\vec{\mathbf{u}} \,[\!]\, \vec{v} \coloneqq \langle u_0 \| v_0, \ldots, u_n \| v_n \rangle,$$

where $v_i \| u_i$ denotes concatenation of u_i and v_i.

5.6.1 DPF-BASED COMPUTATIONAL 1-PRIVACY

We are now ready to describe our computationally 1-private query construction. As previously explained, the construction's objective is to leverage $(2, 2)$-DPFs to provide extremely compact representations of the client's query strings in an otherwise standard bit-more-than-a-bit instantiation.

The main technical challenge to overcome is the impedance mismatch between the "1-out-of-2" secrecy of the $(2, 2)$-DPFs and the desired "1-out-of-ℓ" privacy of the PIR queries. We resolve this by using an L-tuple of $(2, 2)$-DPF key pairs (all realizing the same underlying point function), which the client samples independently and distributes to the servers using a strategy reminiscent of Beimel and Stahl's generic transformation for 2-out-of-ℓ "robust" PIR from any 2-server PIR protocol [14, §3.1]. We first examine the most straightforward and most efficient case in which $\ell = 2^L$ for some $L \geq 1$; an extension to the case of arbitrary ℓ follows in Section 5.6.3.

5.6.2 POWER-OF-2 NUMBER OF SERVERS

Query construction (aka "DPF key distribution"): The DPF-based query construction for $\ell = 2^L$ servers works as follows. Assign to each of the ℓ servers a numeric label j between 0 and $\ell - 1$ and then, for each $j \in [0\,..\,\ell - 1]$, consider the L-bit binary representation $(j_{L-1} \cdots j_1\, j_0)_2$ of j, where j_e denotes the eth-least-significant bit of j. To query for block $\vec{D}_i$, the client samples L

independent $(2, 2)$-DPF key pairs for the point function p_i, say

$$(k_0^{(L-1)}, k_1^{(L-1)}), \ldots, (k_0^{(0)}, k_1^{(0)}) \leftarrow \mathsf{Gen}(1^\lambda;\, i) \times \cdots \times \mathsf{Gen}(1^\lambda;\, i),$$

and then, to each server $j \in [0\,..\,\ell - 1]$, it sends the query string

$$q_j \coloneqq \left(k_{j_{L-1}}^{(L-1)}, \ldots, k_{j_0}^{(0)}\right),$$

where, as above, j_e is the eth-least-significant bit of j.

Because each server receives *either* the 0th or 1th key from each DPF key pair, and because *which* key a given server receives is determined by the corresponding bit in its label, it follows that (i) no server receives both keys from a single DPF key pair, and (ii) no two servers receive the same sequence of DPF keys.

Query expansion: Upon receiving the query string q_j from the client, server j parses it as a sequence of DPF keys

$$q_j \coloneqq \left(k^{(L-1)}, \ldots, k^{(0)}\right),$$

and then it performs a full-domain evaluation on each of the keys and concatenates the resulting bit vectors component-wise to obtain a length-r vector of L-bit integers; that is, it computes

$$\tilde{q}_j \coloneqq \mathsf{EvalFull}(k^{(L-1)}) \,[]\, \cdots \,[]\, \mathsf{EvalFull}(k^{(0)}).$$

From here, the server proceeds exactly as it would upon receiving the query string $\tilde{q}_j$ directly from the client in a perfectly 1-private bit-more-than-a-bit instance with virtual buckets.

Figure 5.1 illustrates the DPF key distribution and query expansion procedure for a protocol instance involving $\ell = 2^2$ servers. In that figure, the ordered pairs $(k_0^{(1)}, k_1^{(1)})$ and $(k_0^{(0)}, k_1^{(0)})$ are both

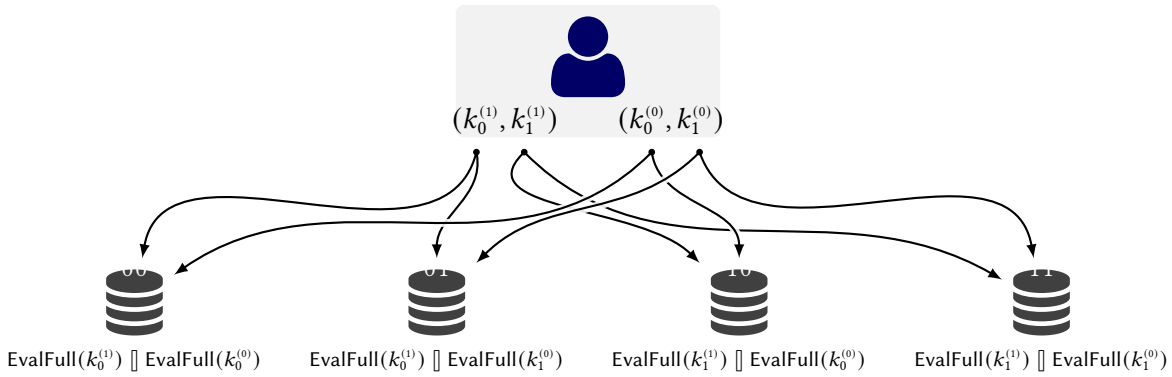


Figure 5.1: **(2, 2)**-DPF key distribution and query expansion procedure for $\ell = 2^2$ servers.

$(2, 2)$-DPF key pairs sampled independently using $\mathsf{Gen}(1^\lambda;\, i)$.

Correctness analysis: We now analyze the correctness of the computationally 1-private bit-more-than-a-bit construction. In particular, Theorem 5.6.3 establishes that the vectors $\tilde{q}_0, \ldots, \tilde{q}_{\ell-1}$ produced by the ℓ servers indeed correspond to an i-equivalence class of $\mathbf{M}^{(r,s)}$ where $s = \ell - 1$.

Theorem 5.6.3. *If* $\left(k_0^{(L-1)}, k_1^{(L-1)}\right), \ldots, \left(k_0^{(0)}, k_1^{(0)}\right) \leftarrow \left(\mathsf{Gen}(1^\lambda;\, i)\right)^L$ *is an L-fold sequence of* $(2, 2)$*-DPF key pairs for a point function* $p_i \colon [1..r] \longrightarrow \{0, 1\}$ *and if* $i^* \in [1..r]$*, then for any pair of distinct L-bit binary strings* $w = (w_{L-1}\, w_{L-2} \cdots w_0)_2$ *and* $x = (x_{L-1}\, x_{L-2} \cdots x_0)_2$*, the vectors*

$$\vec{w} \coloneqq \mathsf{EvalFull}(k_{w_{L-1}}^{(L-1)}) \,\|\, \cdots \,\|\, \mathsf{EvalFull}(k_{w_0}^{(0)})$$

and

$$\vec{x} \coloneqq \mathsf{EvalFull}(k_{x_{L-1}}^{(L-1)}) \,\|\, \cdots \,\|\, \mathsf{EvalFull}(k_{x_0}^{(0)})$$

differ in column i^* *if and only if* $i^* = i$*.*

Proof. We prove the "if" and "only if" directions separately.

"If": Let e be a bit position at which $w_e \neq x_e$. From the correctness criterion for a $(2, 2)$-DPF, we have $\mathsf{Eval}(k_{w_e}^{(e)}, i) \oplus \mathsf{Eval}(k_{x_e}^{(e)}, i) = 1$. Hence, $\mathsf{Eval}(k_{w_e}^{(e)}, i) \neq \mathsf{Eval}(k_{x_e}^{(e)}, i)$ and it follows that the ith

components of $\vec{w}$ and $\vec{x}$ differ in at least one bit.

"Only if": Suppose that $\vec{w}$ and $\vec{x}$ differ in column i^* and let $e \in [0\,..\,L-1]$ be the position of a bit at which they differ; i.e., let e be such that $\mathsf{Eval}(k_{w_e}^{(e)}, i^*) \oplus \mathsf{Eval}(k_{x_e}^{(e)}, i^*) = 1$. Then, from the correctness criterion of a $(2,2)$-DPF, $p_i(i^*) = 1$ so that $i^* = i$. □

The preceding theorem, together with a simple counting argument, yields the following corollary.

Corollary 5.6.4. *If* $\left(k_0^{(L-1)}, k_1^{(L-1)}\right), \ldots, \left(k_0^{(0)}, k_1^{(0)}\right) \leftarrow \left(\mathsf{Gen}(1^\lambda;\, i)\right)^L$ *is an* L*-fold sequence of* $(2,2)$*-DPF key pairs for a point function* $p_i\colon [1\,..\,r] \to \{0,1\}$*, then the* i*th components of the* 2^L *vectors*

$$\left\{\mathsf{EvalFull}(k_{j_{L-1}}^{(L-1)}) \,[\!]\, \cdots \,[\!]\, \mathsf{EvalFull}(k_{j_0}^{(0)}) \;\middle|\; j_{L-1}, \ldots, j_0 \in \{0,1\}^L\right\}$$

collectively span the interval $[0\,..\,2^L - 1]$.

The next theorem is an immediate consequence of Corollary 5.6.4 and the "only if" direction of Theorem 5.6.3.

Theorem 5.6.5. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$ *providing* (perfect) correctness *(in the sense of Part (i) of Definition 5.6.1), then the* 2^L*-server DPF-based bit-more-than-a-bit construction provides* (perfect) correctness *(in the sense of Definition 3.1.2).*

Communication cost analysis: The download cost of a DPF-based bit-more-than-a-bit protocol instance is identical to that of a perfectly 1-private instance with the same number of virtual buckets. In terms of upload cost, the client in a DPF-based bit-more-than-a-bit instance sends L distinct DPF keys to each server. Assuming that the DPFs are implemented with the optimized $(2,2)$-DPF construction of Boyle, Gilboa, and Ishai [23, Figure 1] with 1-bit outputs, the total upload cost is then $L(\lambda+2)(\lceil \lg \frac{r}{\lambda} \rceil)$ bits [22, Theorem 3.3]—in contrast to the $r\lceil \lg(s+1) \rceil = rL$

bits needed for the perfectly 1-private construction with $\ell = 2^L$ buckets and $s = \ell - 1$ words per block—and the following theorem is immediate.

Theorem 5.6.6. *The 2^L-server DPF-based bit-more-than-a-bit construction provides* non-trivial communication *(in the sense of Definition 3.1.3).*

Computation cost analysis: The DPF-based bit-more-than-a-bit construction imposes some (modest) computation overhead relative to a perfectly 1-private bit-more-than-a-bit instance. The overhead consists of (i) the cost for the client to sample the DPF key pairs, and (ii) the cost for each server to expand its DPF keys using a full-domain evaluation (plus nominal costs associated with component-wise concatenation). Sampling the DPF key pairs is an extremely lightweight operation: each sample takes about $2\lceil \lg \frac{r}{\lambda} \rceil$ evaluations of a PRG and about $2\lambda \lceil \lg \frac{r}{\lambda} \rceil$ additional bit operations [22, Figure 1]. The full-domain evaluations for query expansion are also quite efficient: each uses just $\lceil \frac{r}{\lambda} \rceil$ evaluations of the PRG [22, Theorem 3.3]. All told, when the PRG is implemented using AES-128, this equates to a total of about $4L(\lceil \lg r \rceil - 7)$ AES-128 evaluations and $256L(\lceil \lg r \rceil - 7)$ additional bit operations for the client, and about $L\lceil \frac{r}{64} \rceil$ AES-128 evaluations for each server.

In Section 5.8, we present experimental evidence from our prototype implementation. Our findings suggest that, for plausible parameter settings, the computation overhead associated with sampling DPF key pairs and query expansion are insignificant relative to the time required to compute the Frobenius products specified by the expanded query vectors. The DPF-associated computation times are low enough that we can reasonably surmise that reductions in upload time will vastly overshadow them for all but the most contrived of network settings and parameter choices.

Security analysis: A trivial reduction proves the computational 1-privacy of a DPF-based bit-more-than-a-bit construction follows from the perfect 1-privacy of regular bit-more-than-a-bit and the computational secrecy (Part (ii) of Definition 5.6.1) of the DPFs. In particular,

given blackbox oracle access to a PPT algorithm $\mathcal{A}$ distinguishing between the distribution ensembles $\{Q_{\{j\}} \mid I = i\}$ and $\{Q_{\{j\}} \mid I = i^*\}$ with advantage $\mu(\lambda)$, we obtain a PPT algorithm $\mathcal{B}$ distinguishing between the distribution ensembles $\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i)\}_{b\in\{0,1\}}$ and $\{k_b \mid (k_0, k_1) \leftarrow \mathsf{Gen}(1^\lambda; i^*)\}_{b\in\{0,1\}}$ with the same advantage (cf. Definition 3.1.5 and Lemma 5.6.2). The reduction $\mathcal{B}$ works by (i) sampling L DPF keys from its distribution, (ii) concatenating the L samples to form q_j, (iii) passing q_j to $\mathcal{A}$, and then (iv) returning whatever $\mathcal{A}$ returns. This reduction and Theorems 5.6.5 and 5.6.6 proves the following.

Theorem 5.6.7. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a* $(2, 2)$*-DPF for* $\mathbb{P}(r)$ *providing* computational secrecy *(in the sense of Part (ii) of Definition 5.6.1), then a* 2^L*-server DPF-based bit-more-than-a-bit instance is a* computationally 1-private 2^L-server PIR protocol.

5.6.3 ARBITRARY NUMBER OF SERVERS

We now describe how to extend the computationally 1-private bit-more-than-a-bit protocol to an arbitrary number of servers, at the expense of some other upload and computation overhead. The challenge to overcome is the existence of so-called *modulo bias* introduced when reducing integers distributed uniformly in $[1..2^L]$ modulo ℓ when $\ell \nmid 2^L$.

Query construction (aka "DPF key distribution"): Let $poly\colon \mathbb{N} \to \mathbb{N}$ be some positive integer-valued polynomial. The basic idea behind the extension is quite simple: Set $L = \lceil \lg \ell \rceil + poly(\lambda)$ and, as before, have the client sample an L-fold sequence of DPF key pairs and then send a query string q_j comprising one DPF key from each key pair to each server $j \in [0..\ell - 1]$. Note that although there are just ℓ servers, there are more than $\ell + 2^{poly(\lambda)}$ possible combinations of DPF keys from which the client may choose. The client chooses which sequences of DPF keys to send to each server *uniformly*, subject to the values in column i of the ℓ resulting vectors being pairwise incongruent modulo ℓ. The additional $poly(\lambda)$ DPF keys serve to "smooth out" the distribution, reducing the magnitude of the modulo bias to a negligible level; thus, we refer to $poly(\lambda)$ as

the *smoothing parameter*. We stress that the smoothing parameter need not be large: Formally proving computational privacy requires only that $poly(\lambda) \in \Omega(\lambda^{\epsilon})$ for some $\epsilon > 0$; in practice it is sufficient to regard $poly(\lambda)$ as a small(ish) constant, say $poly(\lambda) = 64$ or 80.

Query expansion: Upon receiving the query string q_j from the client, server j parses it as a sequence of DPF keys and then performs L full-domain evaluations and an L-ary component-wise concatenation to obtain a length-r vector of L-bit strings (which it regards as integers). From here, server j reduces the vector component-wise modulo ℓ to obtain the desired length-r vector of integers in $[0\,..\,\ell-1]$.[6]

Correctness analysis: The "if" direction of Theorem 5.6.3 guarantees that the servers generate vectors of L-bit integers containing identical values in all but the ith column; thus, after reduction modulo ℓ the vectors remain identical in all but (perhaps) the ith column. Furthermore, because the client selects DPF keys to ensure that the entries in column i at the servers are pairwise incongruent modulo ℓ, the values in column i remain pairwise distinct after reduction modulo ℓ. This proves the following theorem.

Theorem 5.6.8. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$ *providing* (perfect) correctness *(in the sense of Part (i) of Definition 5.6.1), then the* ℓ*-server DPF-based bit-more-than-a-bit construction provides* (perfect) correctness *(in the sense of Definition 3.1.2).*

Communication and computation cost analysis: Relative to the 2^L-server construction, the construction for arbitrary ℓ increases upload cost and computation cost associated with sampling DPF key pairs and performing full-domain evaluations each by a factor of approximately $\big(\ell + poly(\lambda)\big)/\ell$; the download cost is unaffected. Additionally, each server must perform a component-wise reduction modulo ℓ. However, in implementations, it is possible to interleave this operation with the component-wise concatenation to introduce essentially no overhead (at most one conditional subtraction) per DPF key.

[6]This is always possible as (i) Corollary 5.6.4 ensures the 2^L possible sequences of DPF keys collectively span $[0\,..\,2^L-1]$, and (ii) $[0\,..\,\ell-1] \subseteq [0\,..\,2^L-1]$.

Theorem 5.6.9. *The ℓ-server DPF-based bit-more-than-a-bit construction provides* non-trivial communication *(in the sense of Definition 3.1.3).*

Security analysis: In terms of privacy, the ℓ-server construction is nearly identical to the 2^L-server construction, with one notable exception: the expanded query vectors in the ℓ-server construction *exhibit a (small) statistical bias.* Specifically, writing $2^L = Q \cdot \ell + R$ with $0 \leq R < \ell$ using the Division Algorithm, each integer $j \in [0\,..\,R-1]$ is congruent to $Q+1$ integers from $[0\,..\,2^L-1]$, while each $j \in [R\,..\,\ell-1]$ is congruent to just Q integers from $[0\,..\,2^L-1]$; thus, the probability mass function describing each component of an expanded query vector assigns a probability of $(Q+1)/2^L$ to each value in $[0\,..\,R-1]$ and $Q/2^L$ to each value in $[R\,..\,2^L-1]$—except for column i, where the distribution is uniform (since clients choose the DPF key sequences uniformly). Fortunately, as the smoothing parameter $poly(\lambda)$ guarantees that the statistical distance between these two distributions is bounded above by $2^{-poly(\lambda)}$, we still obtain following theorem.

Theorem 5.6.10. *If* $(\mathsf{Gen}, \mathsf{Eval})$ *is a* $(2,2)$*-DPF for* $\mathbb{P}(r)$ *providing* computational secrecy *(in the sense of Definition 5.6.1) and if* $poly(\lambda) \in \Omega(\lambda^\epsilon)$ *for some* $\epsilon > 0$*, then the ℓ-server DPF-based bit-more-than-a-bit construction provides* computational 1-privacy *(in the sense of Definition 3.1.4).*

5.7 RELATED WORK

The literature on PIR is vast, though relatively few works focus specifically on 1-private multi-server PIR or (practical) protocols exhibiting provably optimal costs. Before concluding, we briefly highlight a few of the most closely related efforts in this space.

Kiayias, Leonardos, Lipmaa, Pavlyk, and Tang [86] propose a single-server PIR protocol with asymptotically optimal download cost. For sufficiently large records—beyond around 1 GiB each—the download overhead of their scheme is at most a few percent; however, their reliance on number-theoretic assumptions yields computation costs prohibitively high for large-scale deployment [122].

In the multi-server setting, Henry, Huang, and Goldberg [74] proposed *batch queries* for Goldberg's multi-server protocol, which allows amortizing the costs associated with fetching multiple blocks at a time. In follow-up work, both Wang, Kuppusamy, Liu, and Cappos [134] and Demmler, Herzberg, and Schneider [42] extend batch queries to Chor et al.'s folklore protocol. For maximally large batch sizes, this approach significantly reduces the efficiency gaps between our bit-more-than-a-bit protocols and those in Figures 5.7 and 5.8—provided the client wishes to make sufficiently many queries in parallel.

In the 2-server setting, several works have employed $(2, 2)$-DPFs to achieve low upload costs [23, 37, 56], while Angel, Chen, Laine, and Setty [4] proposed a novel technique to obtain low upload costs in single-server FHE-based PIR using so-called plaintext slot permutations. For $\ell > 2$ servers, there does not appear to be any prior work that provides upload costs scaling sublinearly in r.

5.8 IMPLEMENTATION & EVALUATION

To validate the efficiency of the constructions presented in Sections 5.5 and 5.6, we have implemented them as an open-source C++ library called libbitmore [72], which we built atop our own dpf++ library [71]. To facilitate a fair head-to-head comparison with the "folklore optimal" protocols, our codebase also incorporates hand-optimized implementations of both the 2-server variant of Chor et al.'s protocol [32] and the basic 2-server DPF-based protocol incorporating all optimizations suggested by Boyle et al. [22].[7] We also compare our implementation with Percy++ v1.0 [58] and RAID-PIR v0.9.5 [39], a pair of open-source multi-server PIR libraries that are widely used as benchmarks in the PIR research community.

The remainder of this section presents selected findings from a pool of experiments designed to measure the performance implications of various parameter choices and resulting speedups

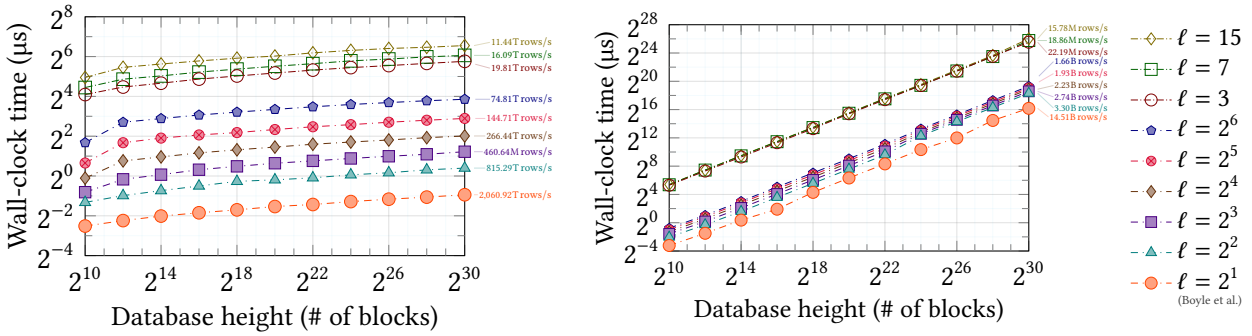
[7] Although the new constructions include the latter two protocols as special cases, our standalone implementations are slightly faster than fixing $\ell = 2$ in our implementation of the general constructions.

relative to prior work. We conducted all experiments on a workstation running Ubuntu 18.04.2 LTS on an AMD Ryzen 7 2700x eight-core CPU @ 4.30 GHz with 16 GiB of RAM and a 1 TB NVMe M.2 SSD. (We had this workstation custom-built for CA$1050 in April 2019.) All experiments measured the running times for a single client or server instance running in isolation on a single CPU core. We repeated all experiments for 100 trials and report herein the sample mean (wall-clock) timings across all trials. Except for RAID-PIR, all experiments exhibited sample standard deviations that were small relative to the sample means; thus, we omit error bars in our plots to reduce clutter and improve visual clarity. When discussing selected statistics in the text, we write $M \pm s$ to indicate a sample mean of M and a sample standard deviation of s. We report all such statistics to one digit of precision in the sample standard deviation.

5.8.1 DPF KEY SAMPLING AND EXPANSION

Our first set of experiments measures the time required for DPF key sampling (by the client) and expansion (by the server) for various numbers of servers (ℓ) and database heights (r) in our computationally 1-private protocols. We provide log-log plots of our findings in Figure 5.2. We emphasize that the costs for DPF key sampling and expansion do *not* depend on the bitlength (b) of the database blocks. As expected, both operations' costs are consistently low, even for databases with relatively large numbers (i.e., hundreds of millions) of blocks—notice that the y-axis on both plots measures wall-clock time in *microseconds*.

DPF key sampling: For $\ell = 2^L$, the running times reported in our DPF key sampling plot (Figure 5.2a) consist only of sampling L independent $(2, 2)$-DPF key pairs. For ℓ not a power of 2, they also include the time required for (i) the sampling of $poly(\lambda) = 80$ extra DPF key pairs for smoothing and (ii) the uniform random selection of key sequences to satisfy the requisite pairwise-incongruence property (see Section 5.6.3); consequently, the running times for these experiments are an order of magnitude higher than for $\ell = 2^L$. Nonetheless, all experiments complete in *under 100 microseconds*.



(a) DPF key sampling (client).

(b) DPF key expansion (server).

Figure 5.2: Wall-clock time for client-side DPF key sampling (a) and server-side DPF key expansion (b) as the database height grows. All experiments use $\lambda = 128$; for computationally 1-private protocols with ℓ not a power of 2, we set the smoothing parameter to $poly(\lambda) = 80$.

DPF key expansion: The DPF key expansion plot (Figure 5.2b) reports the time required for both the full-domain evaluation of each DPF key and the subsequent component-wise concatenation (including modular reductions).

We implement the component-wise concatenation using our algorithms based on 256-bit vectorized (AVX2) instructions. For $\ell = 2^L$, our algorithm iterates over the bits obtained via full-domain evaluation of each DPF key in 32-bit increments, using AVX2 intrinsics to expand each 32-bit segment into a 32-byte vector type. It then uses simple (vectorized) masking and bitwise logical ORs to concatenate 32 components in parallel effectively. This yields an order-of-magnitude speedup versus an "obvious" implementation of component-wise concatenation, helping to ensure that DPF key expansion contributes insignificantly to the servers' total computation time; however, it also limits our implementation to at most $L = 8$ keys per server, thus imposing a limit of $\ell = 256$ on the total number of servers.

Our algorithm for ℓ not a power of 2 is similar, except it periodically performs (parallel) partial modulo-ℓ reductions and uses lookup tables to simulate expanding each bit in the full-domain evaluations beyond the architecture-imposed 1-byte boundary. This approach significantly reduces the overhead from modular reductions, though it imposes an even stricter limit of $\ell = 127$ on the number of servers (to avoid overflowing bytes between partial reductions).

As expected, the costs for DPF key expansion (Figure 5.2b) are significantly higher than for sampling (Figure 5.2a); indeed, the cost for DPF key expansion scales with r, whereas DPF key sampling scales with $\lg r$. Nonetheless, all experiments with $\ell = 2^L$ complete in *under 1 second* and all experiments with ℓ not a power of 2 still complete within *about 1 minute*. Focusing on a "modestly" sized database with just $r = 2^{20}$ rows, we find that all experiments for $\ell = 2^L$ complete within *just over half a millisecond* and all experiments with ℓ not a power of 2 complete in *well under 50 milliseconds*.[8]

Looking ahead to Figure 5.3, we observe (perhaps surprisingly) that in all cases, DPF key expansion ends up being (slightly) faster than sampling a pseudorandom query of the same length using `arc4random`; thus, in addition to significantly reducing upload costs (e.g., from 0.5 GiB to just 31.66 KiB per server when $\ell = 15$ and $r = 2^{30}$), the computationally 1-private protocols have a modestly lower aggregate computation cost relative to their perfectly 1-private counterparts! Looking even further to Figure 5.4, we conclude that DPF key sampling and expansion contribute insignificantly to the total computation cost of a PIR query.

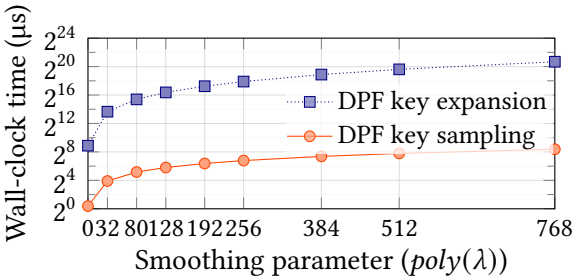


Figure 5.3: Wall-clock time for client-side DPF key sampling and server-side expansion as the smoothing parameter grows. All experiments fix $\ell = 3$ and $r = 2^{20}$.

Varying the "smoothing parameter": Note that the plots in Figure 5.2 all fix the smoothing parameter as $poly(\lambda) = 80$. We also conducted a series of experiments to study the implications of

[8]Astute readers may notice that, although the cases of $\ell = 3$, 7, and 15 respectively expand and concatenate 82, 83, and 84 DPF keys, the differences in their throughputs are nontrivial (ranging from 22.192 ± 0.006M rows/s for $\ell = 3$ to 15.78 ± 0.01M rows/s for $\ell = 15$). This larger-than-expected difference owes to the frequency with which our implementation must perform partial modular reductions to avoid overflowing bytes, which is dependent on the bit length of the modulus.

varying $poly(\lambda)$, from $poly(\lambda) = 0$ up to $poly(\lambda) = 6\lambda$ (i.e., from $poly(128) = 0$ up to $poly(128) = 768$) for a fixed number of servers ($\ell = 3$) and database height ($r = 2^{20}$). We provide a log-linear plot of the results in Figure 5.3. The results are unsurprising: increasing $poly(\lambda)$ effects a corresponding increase in DPF key sampling and expansion times, with key expansion times ranging from 0.470 ± 0.006 ms when $poly(\lambda) = 0$ up to 1681 ± 3 ms when $poly(\lambda) = 768$. In a practical deployment, we recommend setting $poly(\lambda)$ between 64 and 96 to strike a good balance between performance and security, which in these experiments yielded expansion times from 31.4 ± 0.1 ms up to 55.4 ± 0.2 ms.

In terms of upload cost, setting $poly(\lambda) = 0$ yields queries of size 0.41 KiB, while $poly(\lambda) = 80$ yields queries of size 16.92 KiB and setting $poly(\lambda) = 768$ yields 158.85 KiB. Thus, even for "unreasonably" large values of the smoothing parameter, DPF sampling times, DPF expansion times, and per-server upload costs remain quite modest for a database having on the order of a few millions of blocks.

5.8.2 COMPUTATIONAL VS. PERFECT PRIVACY

Our second set of experiments compares the cost of (client-side) query generation for DPF-based computationally 1-private protocol instances versus perfectly 1-private protocol instances as the database height (r) grows. We provide a log-log plot of our findings in Figure 5.4. For $\ell = 2^L$, perfectly 1-private query generation is implemented using a single call to `arc4random_buf`, relying on the Linux kernel to provide the required number of pseudorandom bits as efficiently as it knows how. For ℓ not a power of 2, it is implemented using r consecutive calls to `arc4random_uniform`, which involves a modular reduction and the occasional re-sampling (to account for modulo bias) for each query component; thus, similar to the computationally 1-private protocol, we observe a substantial performance penalty when ℓ not a power of 2.

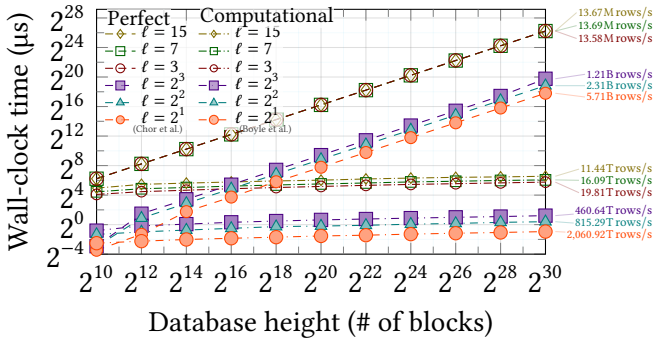


Figure 5.4: Wall-clock time for client-side query construction in both computationally and perfectly 1-private protocols. For computationally 1-private instances with ℓ not a power of 2, we set the smoothing parameter to $poly(\lambda) = 80$.

As expected, query generation times in the computationally 1-private protocols scale quite well (indeed, logarithmically) relative to query generation in the perfectly 1-private protocols. Indeed, as noted in SECTION 5.8.1, even the linear-cost DPF key expansion consistently outperforms query sampling (by a factor $2-3\times$ for $\ell = 2^L$ and by $1.15-1.63\times$ for ℓ not a power of 2). We conclude that our computationally 1-private protocol is strictly faster than its perfectly 1-private counterpart for any realistic parameter settings.

5.8.3 RESPONSE GENERATION

Our third set of experiments compares the time required for each of $\ell = 2^L$ servers to respond to a query, for various choices of L and a fixed number of blocks ($r = 256$), as the database width (b) grows. We measure here only the time required to respond to an "expanded" query; thus, we do not distinguish between perfectly and computationally 1-private protocols, which perform identically (indeed, run the same code on inputs from computationally indistinguishable distributions) for this step of the protocol. We emphasize that response generation is far and away the most computationally intensive step in all constructions we consider (notice that the y-axis on Figure 5.5 is in *milliseconds* versus *microseconds* in all the other plots); indeed, response-generation speedups from increasing ℓ dwarf the comparatively tiny query costs in Figures 5.2–5.4. When

$\ell = 2^1$ (i.e., in the "folklore" protocols), the servers process the database at a rate of 5.7 ± 0.5 GiB/s, whereas when $\ell = 2^6$ this throughput increases to a whopping 144 ± 3 GiB/s—more than a 25× speedup!

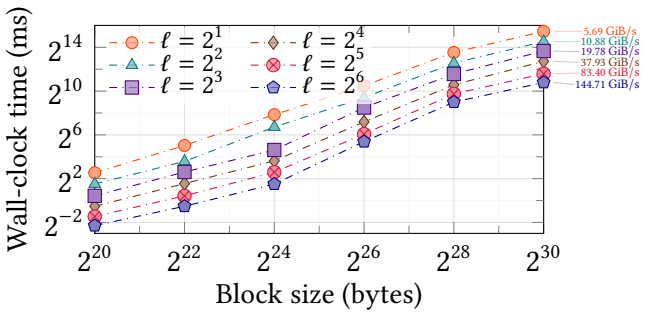


Figure 5.5: Wall-clock time for server-side response generation as block size grows. All experiments fix $r = 256$.

Transitioning from $b = 2^{24}$ to $b = 2^{26}$ bytes per block yields a database that exceeds our 16 GiB of physical memory, resulting in a steepening of each trendline. The trendlines gradually converge to a new slope—reflecting that the computation rate is now limited by disk-read throughput—with instances having fewer servers converging more quickly. The differing convergence rates stem from our using `posix_madvise` to help the Linux kernel prefetch memory pages we will soon need—with $\ell = 2^6$ servers, the (at most) 256 required words occupy about 8 GiB and still fit in physical memory.

The 25× speedup for $\ell = 2^6$ versus $\ell = 2^1$ servers is comparable to the increase in the total number of servers, in the total amount of parallelism introduced. This is not surprising, as all servers' total (expected) number of bit operations is independent of ℓ. It is worth noting, however, that one cannot simply match the throughput of our single-threaded $\ell = 2^6$ computation by merely 32 threads on each of $\ell = 2^1$ servers, since the response-generation computation is I/O bound; rather, to match the speed of the $\ell = 2^6$ protocol, one would need to divy instead work of each server among multiple compute nodes (à la MapRedudivvyand allow each "thread" to saturate its memory bus. We hasten to add that, although this would roughly match the *throughput* of our

new protocols with an equivalent hardware investment, the result would remain non-competitive in terms of communication and client-side reconstruction costs.

Client-side reconstruction: We also measured the time required for the client to reconstruct the responses. We provide log-log plots of our findings in Figure 5.6. As expected, the cost of reconstruction is essentially constant across all experiments, at one (vectorizable) bit operation for each bit in a block. When $b = 1$ GiB, took about 128.7 ± 0.7 ms.

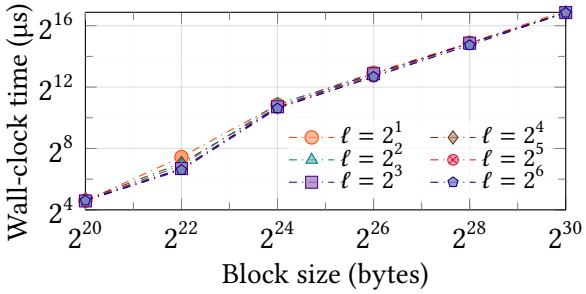


Figure 5.6: Wall-clock time for client-side response reconstruction.

5.8.4 HEAD-TO-HEAD COMPARISON WITH PERCY++

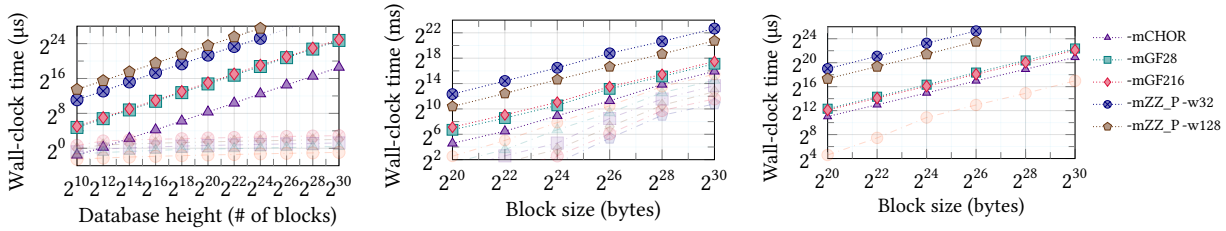
Our fourth set of experiments compares the performance of libbitmore with that of Percy++ v1.0 [58], an open-source C++ implementation of various PIR protocols. Percy++ supports numerous protocols with varying privacy thresholds [3, 32, 44, 57], Byzantine robustness guarantees [44], query batching [70, 74, 93], τ-independence [54], and more. Here we repeat a subset of our libbitmore experiments on selecting "basic" protocols, fixing the privacy threshold and the number of servers in all Percy++ experiments as $t = 1$ and $\ell = 2$, respectively. These parameter choices provide the best possible performance for query generation and response reconstruction (whereas server-side computation and per-server communication cost are both agnostic to these settings [70]).

The fastest protocol Percy++ supports is the folklore-optimal protocol of Chor et al. [33, 106]—by far the *slowest* protocol in libbitmore. Figure 5.7 shows that the implementation of that protocol

in libbitmore is noticeably faster than the one in Percy++. We also plot running times for Goldberg's protocol [57] with arithmetic in $\mathbf{GF}(2^8)$, $\mathbf{GF}(2^{16})$, and integers modulo 32- and 128-bit primes. We stress although libbitmore is much faster, many useful features in Percy++ cannot be realized in its simpler setting.

5.8.5 HEAD-TO-HEAD COMPARISON WITH RAID-PIR

Our fifth and final set of experiments compares the performance of libbitmore with that of RAID-PIR v0.9.5 [39]. RAID-PIR is highly configurable: ℓ-server instances can be configured with a tunable privacy level ranging from 1-privacy through to $(\ell - 1)$-privacy. Here we repeat a subset of our libbitmore experiments, fixing the privacy threshold (which they refer to as the *redundancy parameter* [42]) as $t = 1$ to yield instances with computational 1-privacy. Unlike with Percy++, a direct comparison between libbitmore and RAID-PIR is apt: in both cases, the cost of a (1-private) query decreases as the number of servers grows. We find that, given equivalent parameter choices, libbitmore consistently outperforms RAID-PIR *on every metric*, with improvements ranging from modest (1.3× faster when $\ell = 2^2$ and $b = 4\,\text{MiB}$) to extreme (44 888.9× less upload when $\ell = 2^2$ and $r = 2^{30}$ blocks).

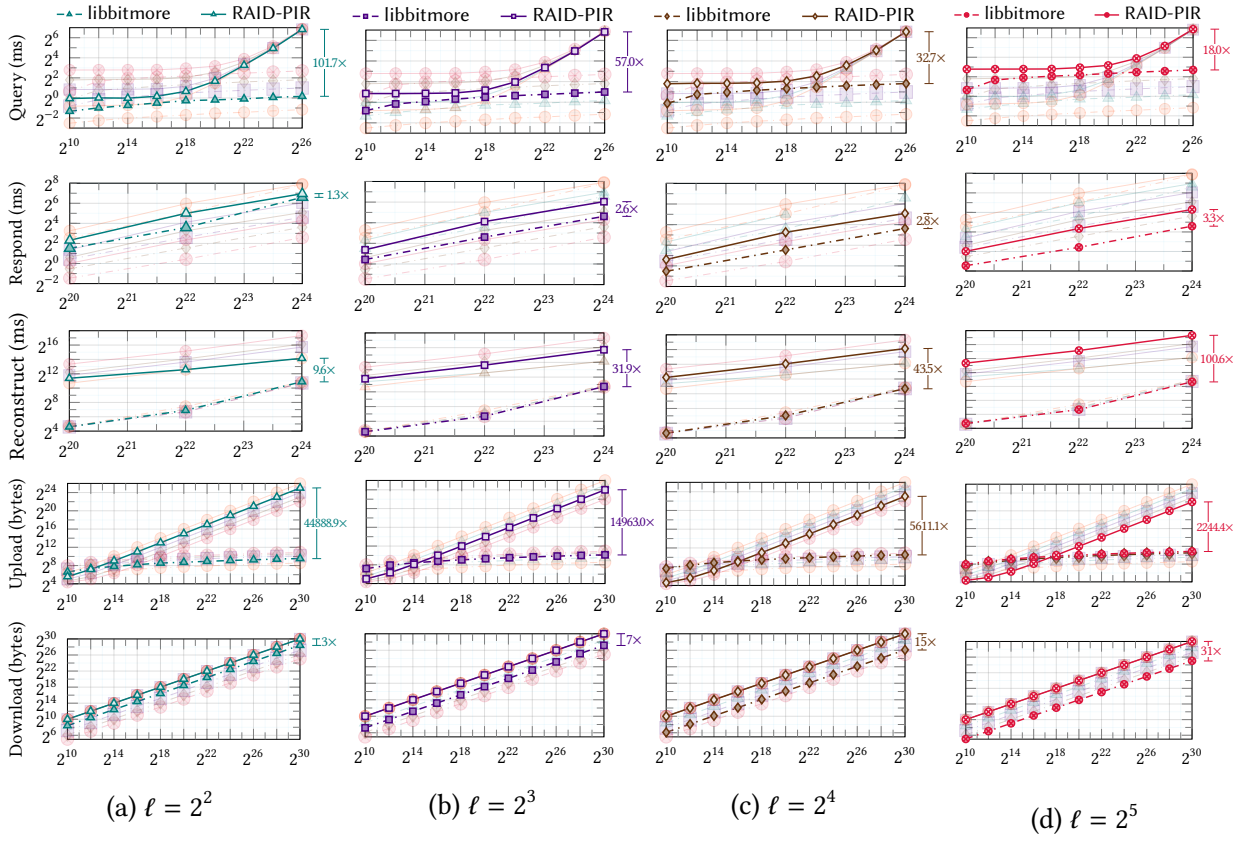


(a) Query construction

(b) Response generation ($r = 256$)

(c) Reconstruction

Figure 5.7: Head-to-head comparison with various 1-private, 2-server instances from Percy++ v1.0. The faint plots near the bottoms Figures 5.7a, 5.7b, and 5.7c show corresponding costs from Figures 5.2a, 5.5, and 5.6, respectively.



(a) $\ell = 2^2$ (b) $\ell = 2^3$ (c) $\ell = 2^4$ (d) $\ell = 2^5$

Figure 5.8: Head-to-head comparison with computationally 1-private RAID-PIR v0.9.5 instances for ℓ ranging from 2^1 to 2^{32}. The scale of some experiments was limited because RAID-PIR v0.9.5 cannot handle databases that exceed physical memory.

5.9 CHAPTER SUMMARY

In this chapter, we revisited the one-extra-bit PIR construction of Shah et al. with an eye toward rigor and efficiency, transforming that construction from a (highly impractical) theoretical result into what we believe to be the most efficient PIR protocol currently in existence, albeit under an extreme trust assumption. We have implemented our protocols as an open-source library and demonstrated their efficiency relative to existing techniques, including the "folklore optimal" protocols of Chor et al. and Boyle et al.For future work, we plan to explore how our approach extends to the setting of computationally t-private protocols for thresholds $t > 1$.

CHAPTER 6

TUNICA: PIR IN THE WILD

6.1 INTRODUCTION

Tor is a low-latency anonymous communications system [47] that supports both *initiator anonymity*—Tor users can access network resources without disclosing their network addresses to the providers of those resources—and *responder anonymity*—service providers can make network resources available to Tor users without disclosing the network addresses of those resources. Network resources that rely on Tor's responder anonymity feature to protect their network addresses are called *onion services* (formerly *hidden services* [63]).[1] Onion services have garnered a good deal of attention in recent years [15, 48, 81, 88], which has led to the identification and exploitation of several weaknesses in their design.

This chapter proposes Tunica, a family of protocols that leverage a cryptographic primitive called *private information retrieval (PIR)* to mitigate the risks posed by most known attacks on Tor's onion services architecture. Specifically, Tunica protocols prevent so-called *Hidden Service Directories (HSDirs)* from distinguishing among requests for different *onion service descriptors*, thereby frustrating attempts to construct popularity histograms over those descriptors and outright thwarting specific intersection-style attacks against the initiator anonymity of users that access onion services.

[1] Outside of the privacy research community, the collection of onion services are sometimes referred to by the pejorative 'the *dark web*'.

What's in a name? In botany, an outer layer of loose membranous skin on an organism is called a *tunic* or *tunica* [35]. A tunica typically serves as a protective membrane to insulate inner layers from damage due to desiccation. An organism protected by a tunica is said to be *tunicate*, with onions being the quintessential example of *tunicate bulbs*.

We chose the name Tunica—and we refer to onion service descriptors protected by Tunica systems as *tunicate onion descriptors*—because of the way our new protocols wrap Tor's onion service descriptors in an additional protective "skin" of cryptographic privacy protections designed to help insulate onion service descriptors against the leakage of access patterns and associated aggregate statistical information.

6.2 PRELIMINARIES

6.2.1 NOTATION AND BASIC DEFINITIONS

$\mathbb{F}$ always refers to a finite field. Attackers are modeled as PPT algorithms. Given a finite set A and nonnegative integer k, we write (i) $a \in_{\mathrm{R}} A$ to denote the uniform random selection of an element a from A, (ii) $\binom{A}{k}$ to denote the set of all size-k subsets of A, and (iii) $B \subseteq_k A$ as shorthand for $B \in \binom{A}{k}$.

A function $\varepsilon\colon \mathbb{N} \to \mathcal{R}+$ is *negligible* if it vanishes faster than the reciprocal of any positive polynomial; that is, if $\forall c > 0$, $\exists N_c \in \mathbb{N}$ such that $\varepsilon(n) < n^{-c}$ for all $n > N_c$.

6.2.2 PRIVATE INFORMATION RETRIEVAL

Informally, a private information retrieval (PIR) scheme is an interactive protocol using which a user can fetch one or more records of its choosing from a remote database without disclosing to the database operator which records it is fetching. The Tunica constructions we present herein each leverage single-round PIR protocols that guarantee privacy against computationally bounded database operators under suitable computational hardness assumptions.

Figure 6.1 illustrates the flow of information in a single-round PIR protocol involving $\ell \geq 1$ database servers (each of which holds an identical copy of the database D): given a security parameter λ and some "identifier" for a desired record in D (described by a random variable $\boldsymbol{I}$) as input, the user sends a *query string* to each server (described by the random variables $q[\lambda]^{(1)}, \ldots, q[\lambda]^{(\ell)}$), and each server replies with a *response string* (described by the random variables $\boldsymbol{R}_\lambda^{(1)}, \ldots, \boldsymbol{R}_\lambda^{(\ell)}$), from which the user produces its final output (described by the random variable $\boldsymbol{E}$).

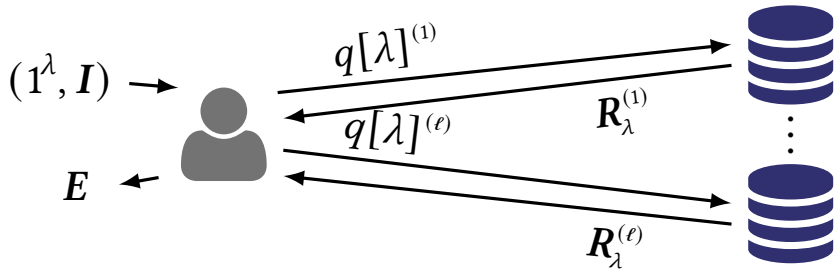


Figure 6.1: Information flow in single-round, ℓ-server PIR.

In light of Definitions 3.1.2–3.1.6, we update the definition of ℓ-server PIR schemes in Definition 6.2.1 providing the three properties we require in our Tunica constructions; namely, *(one-time) computational privacy against a single malicious server*,[2] *correctness*, and *non-triviality*. Fixing $\ell = 2$ specializes Definition 6.2.1 to the 2-server case we consider in Section 6.5, while fixing $\ell = 1$ specializes it to the single-server case we consider in Section 6.7.

[2]Notably, Definition 6.2.1 does *not* address privacy against non-singleton coalitions among the ℓ servers; indeed, in our constructions, we consider only single-server protocols (in which case non-singleton coalitions cannot possibly exist) and 2-server protocols that are trivially non-private against a coalition comprising both servers. Furthermore, Definition 6.2.1 considers only a *one-time* notion of privacy—it makes no promises about privacy if the user queries twice. Of course, one may obtain *many-time* privacy from one-time privacy by requiring the querier to choose fresh keying material for each and every query (a precaution upon which our constructions insist—indeed, as Tunica systems seek to preserve anonymity for the queriers in addition to hiding the contents of their queries, it is already imperative for users to *never* reuse keying material, lest doing so provides a way to link their queries back to a common source). We opt for these security notions over the stronger, standard ones because (i) they are easier both to state and to satisfy, and (ii) they suffice for our purposes.

Definition 6.2.1. *An ℓ-server PIR protocol over a database D is a triple of PPT algorithms $\Pi := (\textsc{Query}, \textsc{Respond}, \textsc{Decode})$ with the following syntax:*

1. $(q[\lambda]^{(1)}, \ldots, q[\lambda]^{(\ell)}; \gamma) \leftarrow \textsc{Query}(1^\lambda; i)$ *is a* query algorithm*, which is run by the user on input a security parameter λ and an identifier i for a record $\vec{\mathbf{D}}_i$ in D, and which outputs an $(\ell+1)$-tuple $(q[\lambda]^{(1)}, \ldots, q[\lambda]^{(\ell)}; \gamma)$ consisting of a query string $q[\lambda]^{(j)}$ for each server j and private auxiliary information γ to be retained by the user;*

2. $\boldsymbol{R}_\lambda^{(j)} \leftarrow \textsc{Respond}(D; q[\lambda]^{(j)})$ *is a* response algorithm*, which is run by each server j on input the database D and the jth query string $q[\lambda]^{(j)}$, and which outputs a response string $\boldsymbol{R}_\lambda^{(j)}$; and*

3. $\boldsymbol{E} \leftarrow \textsc{Decode}(\gamma; \boldsymbol{R}_\lambda^{(1)}, \ldots, \boldsymbol{R}_\lambda^{(\ell)})$ *is a* response decoding algorithm*, which is run by the user on input private auxiliary information γ and ℓ response strings $\boldsymbol{R}_\lambda^{(1)}, \ldots, \boldsymbol{R}_\lambda^{(\ell)}$, and which outputs the final response $\boldsymbol{E}$.*

The scheme is (one-time) computationally private against a single malicious server if, for every $j \in [1..\ell]$ and every PPT algorithm $\mathscr{A}$, there exists a negligible function $\varepsilon\colon \mathbb{N} \to \mathcal{R}+$ such that

$$\left| \Pr\left[\mathscr{A}(q[\lambda]^{(j)}) = 1 \;\middle|\; \boldsymbol{I} = i\right] - \Pr\left[\mathscr{A}(q[\lambda]^{(j)}) = 1 \;\middle|\; \boldsymbol{I} = i'\right] \right| \leq \varepsilon(\lambda)$$

for all i, i' in the support of $\boldsymbol{I}$; it has correctness error κ if

$$\Pr\left[\boldsymbol{E} = \vec{\mathbf{D}}_i \;\middle|\; \boldsymbol{I} = i\right] \geq 1 - \kappa(\lambda),$$

for all i in the support of $\boldsymbol{I}$, where $\vec{\mathbf{D}}_i$ denotes the record with identifier i in D; and it is non-trivial if

$$\sum_{j=1}^{\ell} \left(|q[\lambda]^{(j)}| + |\boldsymbol{R}_\lambda^{(j)}|\right) \in \mathrm{o}\big(|D|\big).$$

All probabilities are over the random coin tosses of the user.

The two Tunica constructions in this chapter are built atop two recent PIR protocols from the literature. Both constructions require only *black-box access* to the underlying PIR algorithms; thus, we detail here only the syntax and important distinguishing features of the two PIR schemes.

6.2.2.1 PIR WITH DISTRIBUTED POINT FUNCTIONS

Our first construction, $\mathsf{Tunica}_{\mathsf{DPF}}$, is built atop a 2-server PIR scheme due to Gilboa and Ishai [56, §4]. The Gilboa–Ishai PIR scheme treats the database as a *key-value store*,

$$D = \left\{(i_1, \vec{\mathbf{D}}_{i_1}), \ldots, (i_r, \vec{\mathbf{D}}_{i_r})\right\} \subseteq \{0,1\}^{\mu} \times \mathbb{F}^s,$$

in which each record $\vec{\mathbf{D}}_i$ is represented by a vector from $\mathbb{F}^s$ and is associated with a μ-bit identifier $i \in \{0,1\}^{\mu}$.

The user requests the record $\vec{\mathbf{D}}_{i^*}$ associated with identifier i^* by (i) sampling a pair of keys, say (K_1, K_2), for a 2-*party distributed point function (2-DPF)* [23, Definition 2.3], $F\colon \{0,1\}^{\lambda} \times \{0,1\}^{\mu} \to \mathbb{F}$, to satisfy

$$F(K_1;\, x) + F(K_2;\, x) = \begin{cases} 1 & \text{if } x = i^* \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

and then (ii) sending $q[\lambda]^{(1)} \coloneqq K_1$ and $q[\lambda]^{(2)} \coloneqq K_2$ to the first and second server, respectively. Upon receiving $q[\lambda]^{(j)}$ from the user, the jth server responds with

$$\boldsymbol{R}_{\lambda}^{(j)} \coloneqq \sum_{k=1}^{r} F(q[\lambda]^{(j)};\, i_k) \cdot \vec{\mathbf{D}}_{i_k},$$

a vector from $\mathbb{F}^s$. The user's output is $\boldsymbol{E} \coloneqq \boldsymbol{R}_{\lambda}^{(1)} + \boldsymbol{R}_{\lambda}^{(2)}$.

Boyle, Gilboa, and Ishai [23, Figure 1] present an efficient construction for 2-DPFs from any pseudorandom generator (PRG). The size of each 2-DPF key in their construction—and, hence,

per-server upload cost—is just $(\lambda + 2)(\lceil \lg \mu \rceil + 2) - 2 \in \Theta(\lambda \lg \mu)$ bits (where, as above, λ is the security parameter and μ is the bit length of record identifiers).

Theorem 6.2.2. *Instantiating the Gilboa–Ishai PIR scheme with Boyle et al.'s 2-DPF construction yields a 2-server PIR scheme that (i) is (one-time) computationally private against a single malicious server (provided the given PRG is, in fact, a PRG), (ii) has correctness error $\kappa(\lambda) = 0$, and (iii) is non-trivial, with $\sum_{j=1}^{2} \left(|q[\lambda]^{(j)}| + |\boldsymbol{R}_\lambda^{(j)}| \right) \in \Theta\left(|\mathbb{F}|s + \lambda \lg \mu \right)$.*

We note that the Gilboa–Ishai scheme offers no privacy if the two servers collude, as it is trivial to deduce the target identifier i^* given access to both 2-DPF keys.

6.2.2.2 PIR WITH LATTICE ASSUMPTIONS

Our second construction, $\mathsf{Tunica}_{\mathsf{LWE}}$, is built atop a single-server PIR scheme due to Aguilar-Melchor, Barrier, Fousse, and Killijian [1, §2.2]. The Aguilar-Melchor et al. PIR scheme is a special instance of a generic construction described by Stern [125], which treats the database as a *fixed-width flat file*, $D \in \mathbb{F}^{r \times s}$, in which each record $\vec{\mathbf{D}}_i$ is represented by a vector from $\mathbb{F}^m$ and is identified by the integer $i \in [1..r]$ indicating position (i.e., its 1-based row number) in the file.

The user requests the record $\vec{\mathbf{D}}_{i^*}$ occupying row i^* by (i) generating an ephemeral public-private key pair (sk, pk) for some IND-CPA–secure, additively homomorphic encryption scheme, and then (ii) sending a component-wise encryption of the i^*th length-r unit vector $\vec{\boldsymbol{e}}_{i^*} \in \mathbb{F}^r$ to the server. Upon receiving $q[\lambda]^{(1)} \coloneqq \mathrm{Enc}(pk; \vec{\boldsymbol{e}}_{i^*})$ from the user, the server uses the additive homomorphism to respond with a length-m vector of ciphertexts $\boldsymbol{R}_\lambda^{(1)} = \langle z_1, \ldots, z_s \rangle$ encrypting the vector-matrix product $\vec{\boldsymbol{e}}_{i^*} \cdot D$.[3] The user's output is $\boldsymbol{E} \coloneqq \langle \mathrm{Dec}(sk; z_1), \ldots, \mathrm{Dec}(sk; z_s) \rangle$, a component-wise decryption of the server's response string.

[3] Recursive variants of the scheme reinterpret the response string $\boldsymbol{R}_\lambda^{(1)}$ as a new, smaller fixed-width flat database that the user can privately query, thereby enabling the user to retrieve (or submit further recursive queries against) a fixed-length, contiguous substring of $\vec{\mathbf{D}}_i$.

Aguilar-Melchor et al. [1] present an instance of the above protocol using a lattice-based encryption scheme of Brakerski and Vaikuntanathan [24] with parameters optimized for efficient PIR implementations. The encryption scheme is proven IND-CPA secure under Lyubashevsky, Peikert, and Regev's *Learning with Errors Over Rings (Ring-LWE)* assumption [94]; it results in a ciphertext expansion of 5–6x for reasonable security parameters [1, Figure 1].

Theorem 6.2.3. *Aguilar-Melchor et al.'s instantiation of Stern's (non-recursive) PIR scheme is a single-server PIR scheme that (i) is (one-time) computationally private against a single malicious server (under the Ring-LWE assumption), (ii) has correctness error $\kappa(\lambda)$ negligible in λ, and (iii) is non-trivial, with $\left(|q[\lambda]^{(1)}| + |\boldsymbol{R}_{\lambda}^{(1)}|\right) \in \Theta\left(|\mathbb{F}|(r+s)\right)$.*

The Aguilar-Melchor et al. scheme is computationally intensive compared with the Gilboa–Ishai scheme, but it has the distinct advantage of not requiring any assumptions about collusion (or lack thereof) among multiple untrusted servers.

6.3 TOR AND THE RENDEZVOUS SPECIFICATION

Tor is an overlay network (Figure 6.2) and associated software stack that employs a technique called *onion routing* to support low-latency anonymous communications and traffic-analysis-based attack resistance over the Internet. The present-day Tor network (as of December 2020) comprises several thousand volunteer-operated servers [110] that fulfill several distinct roles.

A special SOCKS proxy called an *onion proxy (OP)* sits between each Tor user and the Internet, intercepting, encrypting, and forwarding network-bound traffic through a series of (typically three) hops called *onion routers (ORs)* in a "telescoping" manner: The first hop—the so-called *entry guard*—learns the network address of the user and the second hop, but neither that of the third hop nor the endpoint; the second hop—the so-called *middle node*—learns the network address of the entry guard and the third hop, but neither that of the user nor the endpoint; the third hop—the so-called *exit node*—learns the network address of the middle node and the endpoint, but neither

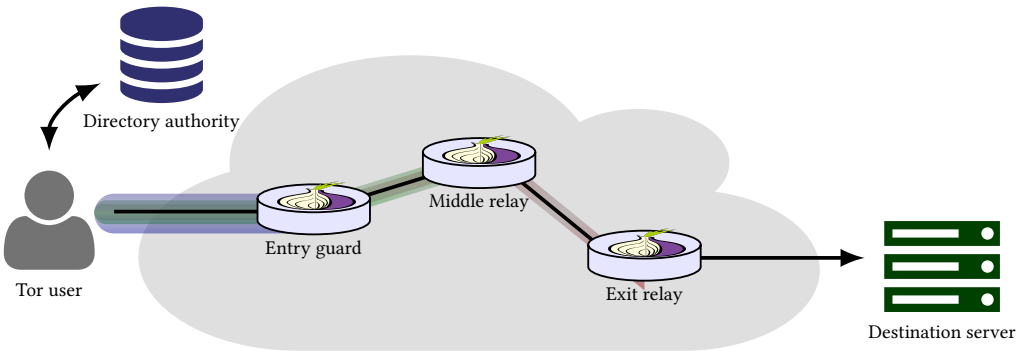


Figure 6.2: Tor network.

that of the user nor the entry guard. Such a telescopically encrypted, three-hop path through the Tor network is called a *circuit*.

Users learn about the availability and attributes of ORs through which to construct their circuits by periodically downloading a *network consensus* document from any member of a predefined set of *directory authorities*, semi-trusted servers whose network addresses are hard-coded into the OP software. The network consensus lists all known ORs, along with relevant attributes including their network addresses, public keys, and any number of *flags* that describe important characteristics, capabilities, and suitabilities for various tasks (for instance, whether a given OR should be used as an entry guard or whether it can be used as an exit node and, if so, for which kinds of traffic).

6.3.1 TOR RENDEZVOUS SPECIFICATION–VERSION 2

In its "default" use case, Tor provides for *initiator anonymity*: Tor users can access network resources without disclosing their network addresses to the service providers that offer those resources. The *rendezvous specification* [45, 46] describes a mechanism through which Tor additionally provides for *responder anonymity*: service providers (i.e., onion services) can make network resources accessible through Tor without disclosing their network addresses to the users

that access those resources. The combination of initiator and responder anonymity yields *mutual anonymity*, wherein neither party to the communication learns the other's network address.

As a first step toward realizing responder anonymity, an onion service must somehow advertise its existence, its public key, and some way for users to initiate contact through the Tor network. To facilitate this, a subset of ORs (roughly 4100 [111] out of about 6700 total [110] as of December 2020) are designated in the network consensus to be *hidden service directories (HSDirs).* The HSDirs collectively form a distributed hash table (DHT) in which onion services publish *onion service descriptors*, signed documents that list the onion services' public keys, and some number (typically 3) of ORs designated to act as *introduction points (InPs)* through which clients can initiate contact with the onion service.

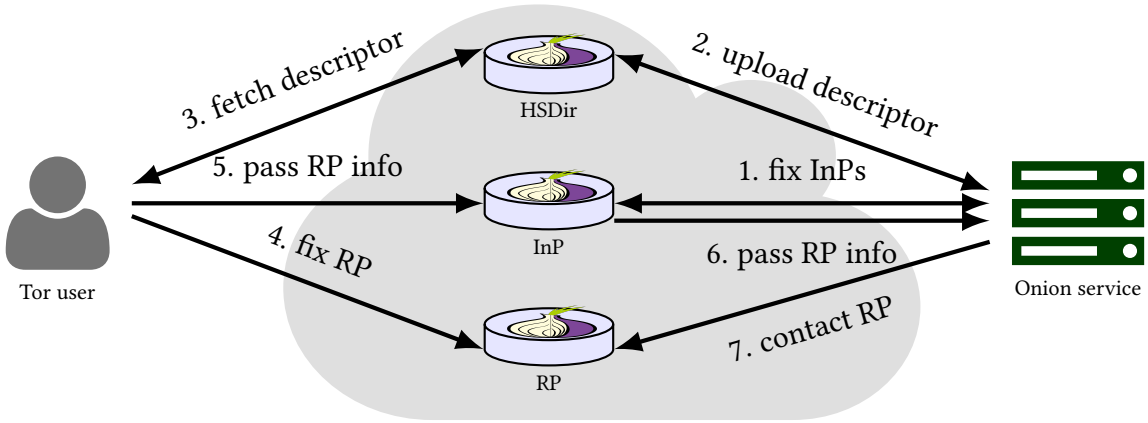


Figure 6.3: Pictorial overview of the Tor rendezvous specification.

The onion service builds a long-lived anonymous circuit to each of its InPs (Step 1 of Figure 6.3), thus enabling the InPs to relay messages to the onion service without disclosing to the InPs its network address. The onion service also uploads its descriptors to HSDirs (Step 2). To establish a connection with the onion service, a Tor user first fetches its onion service descriptor from the DHT (Step 3). It then builds an anonymous circuit to a randomly selected OR and asks it to

serve as a *rendezvous point (RP)* by telling it a one-time secret (Step 4). The one-time secret acts much like a port number, allowing an incoming circuit from the onion service to "connect" to the client's existing circuit. Meanwhile, the user builds another anonymous circuit to one of the onion service's InPs (selected randomly from those listed in the onion service descriptor). It then forwards both the network address of its chosen RP and the one-time secret through the InP and the onion service (Step 5 and 6). From here, the onion service builds an anonymous circuit to the RP (Step 7) who, upon receiving the one-time secret through that circuit, fuses the onion service's circuit with the user's circuit. The latter "fused" circuit serves as the medium for subsequent, mutually anonymous communications between the user and the onion service.

Any client who wants to access an onion service computes its descriptor ID using their knowledge of the generation of the onion service's onion address. The client identifies the HSDir onion router address to use it to fetch the descriptor that contains the InP credentials. Using the InP, the client sends the RP onion router's location to the onion service via the *introduction* circuit. Afterward, the client and the onion service perform communication through this RP onion router. This communication assures bi-directional anonymity.

The descriptor ID on which the client looks up in the DHT of HSDirs consists of a hashed value of 80 bits of the onion service's public key, period, descriptor cookie, and replica number. The descriptor itself contains the identifier, version number, public key, InP credentials, etc. The InP credentials include an identifier, IP-address, onion port, onion key, and public key.

The onion service OP learns the current list of active HSDirs by filtering the directory authorities' consensus document. An OR in the consensus document which opens its directory port has the HSDir flag enabled and is operational for at least 25 hours is eligible to act as an HSDir. At any given time, there are two or more sets of 3 or more HSDirs responsible for holding replicas of the descriptor of an onion service. Here three or more HSDir onion routers of each set have consecutive onion identity digests following the descriptor ID in the DHT table's circular list. Note that all requests and replies are performed by HTTP messages for the communication with HSDir

routers. Each pair of adjacent HSDirs has a subset of common onion service descriptors. Each pair of HSDirs one or more hop away has a smaller subset of common onion service descriptors. We investigate these critical characteristics in the coming section.

6.3.2 TOR RENDEZVOUS SPECIFICATION–VERSION 3.

One of the goals of the version 3 proposal [46,63] is to make it difficult for attackers to observe the popularity of any onion service by an unintended entity and to censor onion service descriptors. Each descriptor is located at different positions on the (DHT) hash ring periodically in the proposal. A single directory or a set of directories will not be an easy Denial of Service (DoS) attack target for removing the onion service. During a given period, a random shared value agreed upon by the directory authorities is used along with the onion key or identity key of the HSDir onion router to determine which HSDir will store the onion service descriptor. Furthermore, the descriptor's body is encrypted with the symmetric key derived from the onion service credential.

An onion service has a long-term *master identity key pair* that is exclusively used to generate the *blinded signing key pair*. The onion address of an onion service consists of a checksum, the public part of this master identity key pair, a version number, and the string ".onion" appended to the end of the onion address. A client can derive the public part of the blinded key pair using their knowledge of the master identity key pair and optional secret data. The public blinded signing key is employed by a client to find the onion service index in the DHT-like directory structure for a descriptor. The blinded signing keys are used to sign the transient *descriptor signing keys*. A *descriptor signing key* is used to sign an onion service descriptor. Unlike other asymmetric key pairs, the secret part of Descriptor Signing Keys is kept online by onion service hosts. Finally, the public part is located in the unencrypted portion of the descriptor.

For each period, an HSDir host signs the descriptor signing keys using the private segment of a different blinded signing key pair. The default length of each period is 1440 minutes (one day). Each period starts at 12:00 UTC. To decrypt the onion service descriptors for each period, a client

derives the unchangeable credential (hash of constant string and public part of the master identity key pair) and the changeable sub-credential (hash of a constant string, credential, and public part of the blinded key pair). It is important to note that the onion service host (HSDir) cannot derive the credential even with the knowledge of the sub-credential and private part of the blinded key pair.

The blinded key pair can be used to get a signature of the tunicate (private) descriptor. Assume the basepoint of an ECC group is B, the private component of the master identity key pair is a, and the public component is $A = aB$. In order to derive a blinded key pair (a', A'), a nonce N, an optional secret s, and a `BLIND_STRING` are used to compute the blinding factor $h = H(\texttt{BLIND\_STRING} \mid A \mid s \mid B \mid N)$. The public key for the period is $a' = h \cdot a$. The private key for the period is $A' = h \cdot A = h \cdot a \cdot B$. A deterministic random-looking r is used to generate the signature $(R, S) = (r \cdot B, r + \text{hash}(R, A', M) \cdot a \cdot h)$. The client can verify the signature by checking whether $S \cdot B = R + \text{hash}(R, A', M) \cdot A'$. Thus, this proposal provides an extra protection for onion service descriptors.

The location of an onion service descriptor changes over time. The onion router that hosts such descriptor depends upon several factors; i.e., the current period, a day-based sub credential, the identity digests (public keys) of the router, the consensus parameters, and the periodically changing shared random value. A Tor client may want to access an onion service; the consensus documents provide the random shared value of the day and the random shared value of the previous day every hour. To prevent the HSDirs from being predictable far ahead of time, the descriptor ID; i.e., *hs_index(replica_num)*, and HSDir indexes, i.e., *hsdir_index(node)*, both change over time. The position in the hash ring for an onion service index (descriptor ID) is calculated from the public part of the blinded key pair, the replica number, and the period. A Tor onion router that obtains the HSDir3 flag has *hsdir_index* computed using node identity key, shared random value, and period.

The number of HSDir replica sets and the number of consecutive HSDirs (in a DHT structure)

in a set can be changed by modifying three parameters. As before, the default number of HSDir replica sets and the number of consecutive HSDirs in each replica set are 2 and 3, respectively. The parameter *hsdir_n_replica* can be set in an integer range $[1, 16]$ to change the number of HSDir replica sets a descriptor can appear in. The parameter *hsdir_spread_fetch* can be configured in an integer range $[1, 128]$ to change the number of consecutive HSDirs in a set that the user can initially fetch from. The parameter *hsdir_spread_store* determines the number of consecutive HSDirs a descriptor is stored for each replica set. *hsdir_spread_store* can be adjusted in an integer range $[1, 128]$. They have *hsdir_spread_store* set to a larger number than *hsdir_spread_fetch* allows the system to be better at tolerating the disappearance of the HSDirs.

The descriptor has the following fields in its body: the version number, the lifetime of the descriptor, the descriptor signing key certificate along with an extension of the public part of the blinded key pair, the revision counter, an encrypted string of the introduction circuit-related confidential data, and the signature of the all previous fields using the descriptor signing key. One of the significant changes described in the proposal is the mandatory encryption of the onion service descriptor. Without knowing the public part of an onion service's master identity key pair, an entity cannot decrypt the descriptor's encrypted part under computational hardness assumption.

For the mandatory first layer encryption, confidential data consists of the public part of the blinded key pair and a constant string. The symmetric encryption key (SEC_IV and SEC_KEY) is derived from that confidential data, constant line, sub-credential, and revision counter information along with salt. The encrypted data format is as follows:

$$SALT \mid (STREAM(SEC\_IV, SEC\_KEY) \texttt{ XOR } D) \mid MAC.$$

6.3.3 ATTACKS ON ONION SERVICES ANALYSIS

After the deployment of Tor's onion service protocol, several attacks have been investigated that finally deanonymized the onion services [15, 81, 88]. Biryukov et al. [15] ranked all onion services to build a popularity histogram after observing the client's access frequency to onion services, which made it easy to target a particular onion service of interest and carried a denial of service (DoS) attack on the responsible HSDir(s) by making the associated descriptors unavailable for the clients to fetch. They harvest all onion service descriptors of the compromised HSDirs and come up with a process to disclose the guard nodes (of the circuit) of an onion service and finally deanonymize the onion service's network address a probabilistic nature. One crucial step of these attacks is to observe the querying pattern of tor client, which makes it imperative to protect the observation of users' descriptor access patterns from one or more compromised HSDir(s).

Biryukov et al. [15] investigates attacks that hinder access to particular onion services and eventually deanonymize them. Additionally, they identify a process to build a popularity histogram for arbitrary onion services by impersonating HSDirs. They cheat the bandwidth scanning by directory authorities to inflate their injected relays' bandwidth to be selected by the path selection algorithm. To insert a relay working as an HSDir, they compute the descriptor ID of the targeted onion service and find the responsible HSDirs from the network consensus. Afterward, they run a brute force search for the key pairs where the SHA-1 hash of the public key falls between the descriptor ID and the fingerprint of responsible HSDirs. With the key pair, they run the relay and wait for the HSDir flag. Subsequently, new descriptors of the targeted onion service are uploaded to it, and clients fetch them from it. Thus, the attacker can inject multiple HSDirs to collect usage statistics and even censor arbitrary onion service. To gather all onion service descriptors of the then 1200 HSDirs, they lease 50 network addresses and run 24 relays on each of them to cover all gaps in the DHT table. To deanonymize an onion service location, the first step is to reveal a guard node of the onion service. The attacker sets several circuits from its own Rendezvous Point to the onion service with the target to take control of the middle node and the guard node by an

opportunistic traffic confirmation attack.

Mitigating the vulnerabilities demonstrated by the attacks against onion services has become a significant concern over the Tor research community. For example, a Tor project blog post [84] listed various shortcomings of the onion service infrastructure and proposed vital research questions regarding them. As a response to the Tor project blog post, there was a *onion service hackfest* [85] where the next generation onion service protocol has been proposed. Some of the key topics covered in the proposal 224 include (i) making the system faster, (ii) using provable cryptographic primitive to protect the content of the descriptors, (iii) featuring more secure onion addresses, (iv) offering advanced security properties like improved DoS resistance for the descriptors, and (v) keeping identity keys of the onion services offline. The features in the new proposal are summarized in Section 6.3.2. Nevertheless, privacy leakage of the access pattern of onion service descriptors has yet to be considered.

To establish a connection with an onion service, the client first has to fetch the descriptor from one of the HSDirs. The descriptor contains information about the introduction points. When an adversary takes control of a directory, they can monitor the downloading and uploading of onion service descriptors and subsequently perform a DoS attack against the most attractive onion service after preparing the popularity histogram. Design-fit symmetric encryption can secure illegitimate access to the content of the descriptors. According to the proposal 224 [46], symmetric encryption is proposed to encode the crucial information of the descriptor. The author of the Tor project blogpost [84] envisioned that tracking for onion service popularity can be made harder by operating a Private Information Retrieval (PIR) protocol. Because PIR is such a cryptographic technique that allows a user to fetch data from a remote database server without letting it know which data is getting fetched. Despite this suggestion, there were no proposals that studied this idea any further. This paper investigates how the descriptor retrieval pattern can be made private by applying a feasible PIR scheme.

6.4 THREAT MODEL

The adversaries are probabilistic polynomial time (PPT) bounded. An adversary can take control of any HSDir(s) and make them malicious. The straightforward way to take control is to inject relays in the Tor network and try to achieve the HSDir3 flag. For fruitful attack against $\mathsf{Tunica}_{\mathsf{DPF}}$ construction, an adversary tries to install as many as the required number of HSDirs so that 2 of the injected HSDirs are chosen by the same client to fetch descriptor of an onion service. If the attempt is successful, then the user's query's privacy immediately gets broken according to to Definition 6.2.1.

In the upcoming two sections, we present two Tunica constructions. Both variants leverage PIR to mitigate information leakage: the first variant, $\mathsf{Tunica}_{\mathsf{DPF}}$, is based on 2-server PIR from distributed point functions, while the second variant, $\mathsf{Tunica}_{\mathsf{LWE}}$, is based on single-server PIR. To employ multi-server PIR, we assume the database copy of the servers is identical.

6.5 $\mathsf{Tunica}_{\mathsf{DPF}}$ CONSTRUCTION

Most of the multi-server PIR schemes need the database copy of the servers to be identical. However, the HSDirs that contain descriptors of onion services do not have an identical set of descriptors, which makes it very challenging to incorporate those PIR techniques. Fortunately, the Distributed Hash Table (DHT) is built to create implicit partitions of descriptors. Each HSDir holds multiple partitions and shares some of them with neighbor HSDir(s); we investigate it next. The position-dependent PIR query will not work in the multi-server case for the shared partition's dynamic nature. However, keyword-based PIR query supported schemes are the ideal candidate. Hence, we employ the Distributed Point Function-based PIR technique, which offers keyword-based private searching with a single round of interaction.

Let us assume the Tor network parameters (i) *hsdir_n_replica* as r, (ii) *hsdir_spread_store* as ℓ, (iii) *hsdir_spread_fetch* as k, and (iv) m as the number of HSDirs the client contacted, while

they are delineating its previously discussed meaning. In Section 6.3.2, we note that the hash ring comprises the *hsdir_index*es of the Tor relays with *HSDir3* flag. Since descriptors of the same replica number of a particular onion service are uploaded in ℓ number of consecutive HSDirs, there is an ℓ number of implicit yet identifiable partitions among all the descriptors of any HSDir of them. For example, the descriptors in the first partition of an HSDir are also present in the previous $\ell - 1$ number of HSDirs; the second partition is shared with $\ell - 2$ prior and 1 next HSDir. Similarly, the last partition is common to $\ell - 1$ next HSDir nodes. Consequently, any two adjacent nodes share the $\ell - 1$ number of distributed partitions.

To explain it further, we find that any subset of HSDir relays of a replica set has at least one shared partition which contains the same descriptors responsible for several onion services. This feature leverages the finding of the intersection set of the datasets of descriptors from $m \leq k$ number of HSDir relays. Note that this common partition resides in different positions among all partitions in the dataset of the descriptors of an HSDir. At this point, we have Figure 6.4 to explicate the notion. For instance, we set $\ell = 3$, so onion service uploads descriptor to three HSDirs, each HSDir dataset has three partitions that are colored differently in the figure. For example, there are 5 labeled partitions; e.g., $P_1, P_2, \ldots, P_5$ and 3 labeled HSDirs; e.g. R_1, R_2, R_3. Any two consecutive nodes share two common partitions; any three nodes share only one common section, as we discussed. Two consecutive HSDirs R_1 and R_2 share two partitions; i.e., $R_1 \cap R_2 = P_2 \cup P_3$; similarly $R_2 \cap R_3 = P_3 \cup P_4$. Finally, we observe that three consecutive HSDirs R_1, R_2, R_3 share one common partition P_3 and we have $R_1 \cap R_2 \cap R_3 = P_3$.

We consider each HSDir S_b holds a descriptor x as a key-value pair $(x := (w, d))$ in the dataset, where w is the unique ID of the descriptor and d is the body of it. We denote by $Replica(i; x)$ the ith replica set of x. Moreover, if A is a set, then $\binom{A}{n}$ is the set of size-n subsets of A (see §6.2). The method $\text{Parts}(S_b)$ outputs each partition of an HSDir with a label; i.e., P_j, where $j \in [1..\ell]$. We illustrate the main protocol of $\text{Tunica}_{\text{DPF}} := (\text{Query}, \text{Respond}, \text{Decode})$ in Algorithm 6.8.

Firstly, client selects a replica set $Replica(i; x)$ uniformly at random (step 2 of Algorithm 6.5).

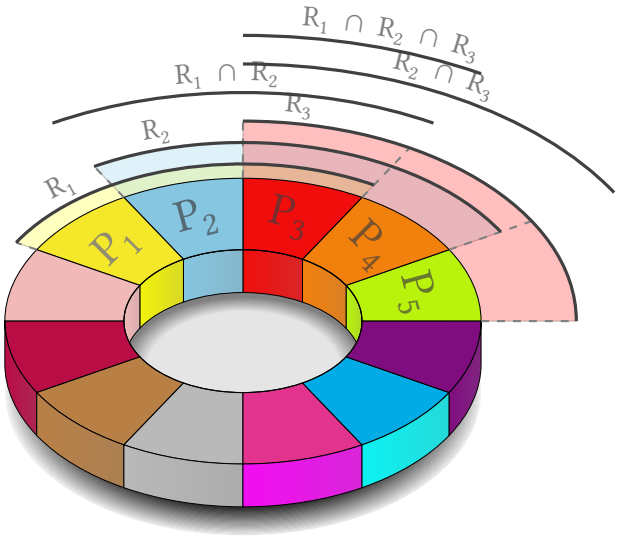


Figure 6.4: Distributed Hash Table of the HSDirs demonstrating explicit partitions when $\ell = 3$.

From the replica set it chooses two HSDirs (S_1, S_2) uniformly at random out of k $(\leq \ell)$ options. The client knows what partition of the HSDirs holds the descriptor in question, as we discussed recently. Following the key generation method *KeyGen* of 2-party DPF protocol, the client generates two keys and sends them as a query to the chosen HSDirs. After receiving the key, each HSDir runs the *Eval* function on each key of the key-value pair of each partition's descriptors regardless of which partition has the required descriptor. The result for each partition is grouped and sent back to the client. After receiving 2ℓ responses, the client filters the necessary responses and add exactly 2 responses to reconstruct the required descriptor. The deterministic nature of the DHT enables the user to choose the correct partition from each set. Note that the remaining $2(\ell - 1)$ number of responses are discarded by the user.

From the above discussion of the technique, it is clear that in the constructions of $\text{Tunica}_{\text{DPF}}$, we take a couple of non-obvious decisions. For instance, The client chooses the replica set uniformly at random and further selects 2 out of the k $(\leq \ell)$ number of HSDirs from the replica set uniformly at random. This is very important because if the client selects a fixed replica set and/or a particular pair of HSDirs, it can stimulate the adversary to compromise the HSDir pair to collude them to

function QUERY($1^\lambda, w$)
  $i \in_R [1..r]$
  $\{S_1, S_2\} \in_R \binom{Replica(i;\, x)}{2}$
  $(k_1, k_2) \leftarrow KeyGen(1^\lambda;\, w)$
  return $\big((S_1, k_1, p_1), (S_2, k_2, p_2)\big)$
end function

Algorithm 6.5: QUERY is invoked by the client to initiate fetching an onion service descriptor.

function RESPOND(k_b)
  for each $\big(P_j \in \text{Parts}(S_b)\big)$ **do**
    $r_{bj} \leftarrow 0$
    for each $\big((w_i, d_i) \in P_j\big)$ **do**
      $r_{bj} \leftarrow r_{bj} + Eval(w_i) \cdot d_i$
    end for
  end for
  $R_b := (r_{b1}, \ldots, r_{b\ell})$
  return R_b
end function

Algorithm 6.6: RESPOND is invoked by server S_b upon receiving a DPF key k from the client.

function DECODE($(R_1, p_1), (R_2, p_2)$)
  Parse R_1 as $(r_{11}, \ldots, r_{1\ell})$
  Parse R_2 as $(r_{21}, \ldots, r_{2\ell})$
  $d \leftarrow r_{1p_1} + r_{2p_2}$
  return d
end function

Algorithm 6.7: DECODE is invoked by the client upon receiving responses from the servers.

Figure 6.8: $\text{Tunica}_{\text{DPF}}$ algorithms

disclose the query client.

To generate the response by the contacted HSDir, one straightforward way is to run the evaluation function upon the shared partition descriptors only. However, this approach helps the adversary to identify the other involved HSDir immediately. Quantitatively, when an HSDir S

sees a client is sending a query for a particular partition, then the technique loses computational privacy according to Definition 6.2.1. To explain it, if S denotes the random variable that a server is associated with a query and P represents the fact that the query is for a particular partition, then we have

$$\Pr\left[S = S_i \;\middle|\; P = P_i\right] \neq \Pr\left[S = S_j \;\middle|\; P = P_i\right].$$

From the Figure 6.4, if R_3 HSDir is compromised by an adversary, and it watches a client sends a query for P_3 partition, then it immediately understands that the client is also querying R_1 or R_2 because only they contain the partition. Hence, each HSDir computes ℓ number of responses against ℓ number of partitions. This choice increases the communication cost linearly in ℓ but provides better privacy.

6.5.1 COST ANALYSIS

The upload cost for the client is $O(\lambda|w|)$. Client downloads $O(\ell|d|)$ responses from each HSDir. Each HSDir has to execute $O(|\text{Parts}(S_b)||w|)$ number of PRG operations. Besides, we can apply 3 or more multi-party Function Secret Sharing (FSS) schemes [21] where the client has to query $m \geq 3$ number of HSDirs what will make it harder for an adversary to compromise all of these relays together. However, the communication and computation overhead for multi-party FSS is noticeably higher than the 2-party case, and hence it is slower, which does not make it an acceptable solution. So, our constructions employ 2-pary DPF based FSS or single server $\text{Tunica}_{\text{LWE}}$.

6.5.2 SYNCHRONIZATION BETWEEN HSDIRS

Duo to the way onion services upload descriptors via HTTP Post announcements to HSDirs, there is a risk that the common partition of the HSDirs of a replica set might not be synced at a given time as expected. However, the 2-server DPF based PIR solution requires the partition to be in synchronization. We have two answers to encounter this issue. Firstly, the client can use *Bloom*

Filters [18] to identify the intersection set of the common partitions of the related HSDirs. As a result, the client has to perform an additional level of interaction before following our basic protocol Section 6.5, which incurs communication and computation overhead. Secondly, an HSDir in the hash ring can keep its ℓ number of partitions synced by itself by communicating with the ℓ number of corresponding neighbor HSDirs. If the HSDirs periodically performs the auto-synchronization, the client will have no extra work beyond the underlying protocol. However, Tor has to enforce this policy for Tor relays, which have the HSDir3 flag. Moreover, to avoid this synchronization issue, we provide another potential solution $\text{Tunica}_{\text{LWE}}$, a single-server-based cPIR built on Ring-LWE lattice-based cryptosystem in Section 6.7.

6.6 SECURITY ANALYSIS OF $\text{Tunica}_{\text{DPF}}$

We consider an attack model in which the attacker may choose to act as a Tor relay operator that controls several HSDirs. The attacker's incentive for controlling a group of HSDirs is to collect statistical information about the onion services that store their descriptors in the attacker operated HSDirs. Regardless of whether the attacker wants to target a particular onion service or wants to collect information on arbitrary onion services, the attacker needs to inject an m number of relays so that some of them reside in any replica set of l consecutive relays that are capable of collecting the statistics.

With collusion among the m malicious HSDirs (servers), the non-collusion assumption of the Distributed Point Function-based $\text{Tunica}_{\text{DPF}}$ collapses. However, note that, even if the collusion among m relays happens, the privacy level of the onion service protocol that contains "tunicate onion descriptors" will not deteriorate more than the privacy of the 2nd or 3rd version of the onion service Protocol described in Section 6.3. Both versions of the onion service Protocol allow attackers to collect statistics on any of the one or more HSDirs they control. When the non-collusion assumption breaks, our protocol turns into the current protocol concerning privacy.

6.6.1 ATTACKS AGAINST ARBITRARY ONION SERVICES

Let n be the number of HSDirs in the consensus document during a given time. We consider the ring of the directories (in the DHT) as a circular array where the last index of the array is adjacent to the first index of the array. The location of an HSDir is determined by a hash function that should distribute the directories uniformly at random. An attacker that wants to collect information on arbitrary onion services can disregard the particular position of where their malicious Tor relays are injected. The attacker only cares about inserting relays that are in the sets of ℓ consecutive relays. Note that the attacker would prefer to have adjacent relays to experience a larger intersection of the descriptors to get the maximum amount of statistics on one or many onion services.

Assuming that the attacker only injects m relays within the ring of HSDirs, the probability an attacker has m relays out of ℓ consecutive relays of an arbitrary replica set is $\frac{r\binom{l}{m}}{\binom{n}{l}}$. For the simplest case, when $r = 2, \ell = k = 3, m = 2$, the probability becomes $\frac{2\binom{3}{2}}{\binom{n}{3}} = \frac{6}{\binom{n}{3}}$. There are currently around 4000 relays with the HSDir flag [111]. Therefore, the probability after injecting only 2 relays to get useful statistics of an arbitrary onion service is near 5×10^{-10}, i.e., significantly low. Hence, it is evident that the attacker needs to inject plenty of relays to collect onion service statistics feasibly. We discuss the required number of injections with an even chance in the next part.

6.6.2 OPPORTUNISTIC ATTACKS AGAINST ARBITRARY ONION SERVICES

Suppose there are n relays total, out of which the attacker controls m relays. (Hence, there are $n - m$ non-attacker relays.) We are interested in the probability that no two attacker-controlled relays exist within the k hops of one another in the DHT. From the heuristic assumption that relays are distributed uniformly at random around the DHT, we can quickly determine the desired probability using an elementary counting argument based on the ubiquitous "stars-and-bars"

method.

Lemma 6.6.1 (Stars and bars). *The number of ways that n identical objects can be distributed among m labelled bins is $\binom{n+m-1}{m-1}$.*

Specifically, we regard the m attacker-controlled relays (the "bars") as partitioning the $n - m$ non-attacker relays (the "stars") into m disjoint bins within the DHT: those between the first and second attacker relays, those between the second and third attacker relays, and so on, up to those between the last and first attacker relays. Now, there are $\binom{n-1}{m-1}$ possible assignments of the m attacker relays into the n positions in the DHT (up to cyclic permutation);[4] this will serve as the denominator in our probability expression.

For the numerator, we are interested in the number of assignments of the $n - m$ non-attacker relays in which each bin contains at least k such relays. This number is equal to the number of ways to assign the remaining $(n - m) - km$ non-attacker relays after first placing k non-attacker relays into each of the m bins. From Lemma 6.6.1, the number of ways to assign these remaining relays is $\binom{(n-m)-km+m-1}{m-1} = \binom{n-km-1}{m-1}$; this will serve as the numerator in our probability expression.

Combining the above two calculations, we obtain the following theorem.

Theorem 6.6.2. *The probability that no two attacker-controlled relays reside within k hops of one another in the DHT is*

$$\binom{n-km-1}{m-1} \Big/ \binom{n-1}{m-1}. \tag{6.1}$$

Figure 6.9 uses Equation (6.1) to plot the theoretically predicted probability that an attacker controlling m out of $n = 3\,200$ HSDirs does not control any pair of relays that reside within $k = 2$ hops in the DHT. The number $n = 3\,200$ was selected to (approximately) coincide with the number of active HSDirs reflected in the Tor consensus at the time of writing [111].

[4] Think of the first attacker relay as defining the "start" of the circle, and then assign the remaining $m - 1$ attacker nodes to the remaining $n - 1$ positions.

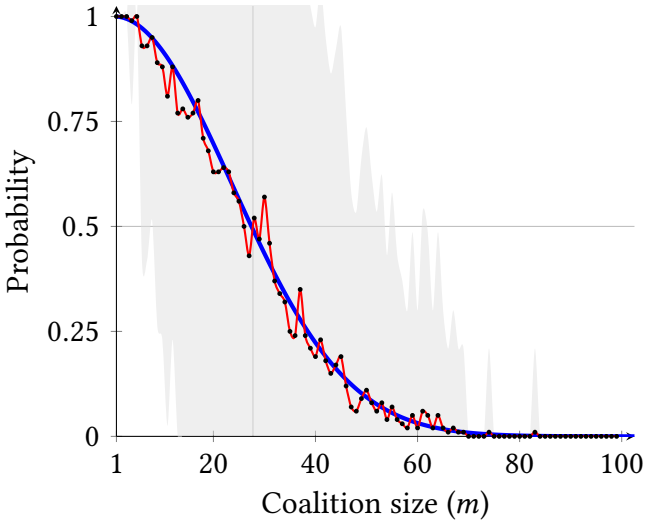


Figure 6.9: Probability that an attacker who controls $\boldsymbol{m}$ (out of $\boldsymbol{n = 3\,200}$) does not control *any* pair of nodes residing within $\boldsymbol{k = 2}$ hops of one another in the DHT.

6.6.3 ATTACKER'S EXPECTED COVERAGE

The preceding analysis characterizes the probability that an attacker fails to deanonymize *any* queries. Hence, to compute the success probability to deanonymize a user's query, we substruct the probability estimate that an attacker fails to deanonymize any query from 1. The graph of Figure 6.9 illustrates that the attacker has to insert at least a 28 number of HSDirs in the DHT so that it has an even chance to be selected in a replica set by a client. Moreover, if the attacker injects almost 100 HSDirs, it is guaranteed that it will deanonymize a query. Next, we analyze the *expected coverage* that an attacker can hope to achieve; that is, we analyze for what fraction of relays the attacker holds two or more copies that may be involved in a given query and, therefore, at least occasionally subject to deanonymization by the attacker.

Initially, we wish to count the number of assignments of HSDirs into the DHT such that exactly $k - i$ hops separate exactly one pair of attacker relays, and k or more hops separate all other pairs of attacker relays. This is equivalent to counting the number of ways we can place $k - i$ non-attacker relays into one bucket, k non-attacker relays into each of $m - 1$ other buckets, and then arbitrarily distribute the remaining non-attacker relays into those $m - 1$ same buckets.

Of course, this experiment is only possible when $km - i \leq n - m$, since otherwise, we cannot even complete the initial step where we fill the buckets. This latter inequality is equivalent to $i \geq (k+1)m - n$. Define $k^* = (k+1)m - n$.

Using the bars-and-stars LEMMA 6.6.1 and the above observation, we get that this number is:

$$= \begin{cases} 0 & \text{if } i \geq (k+1)m - n \\ \binom{m}{1}\binom{n-km+i-2}{m-2} & \text{otherwise.} \end{cases}$$

We can sum this formula over all i from 1 to k to get the number of assignments in which exactly one pair of nodes is less than k hops apart:

$$\binom{m}{1} \sum_{i=1}^{\min(k,k^*)} \binom{n-km+i-2}{m-2}$$

Notice that the min function in the upper bound of the summation takes care of the two cases in the piecewise probability formula. This will prove very useful going forward. As a next step, we wish to count the number of assignments in which exactly two pairs of *consecutive* non-attacker relays are separated by $k - i$ and $k - j$ hops, respectively, with all other pairs of consecutive attacker relays being separated by k or more hops. Using the same approach, this is:

$$= \begin{cases} 0 & \text{if } i + j \geq (k+1)m - n \\ \binom{m}{1,1,m-2}\binom{n-km+i+j-3}{m-3} & \text{if } i \neq j, \text{ and} \\ \binom{m}{2,m-2}\binom{n-km+i+j-3}{m-3} & \text{if } i = j. \end{cases}$$

As before, we can sum over all choices for i and j from 1 to k, subject to $i \leq j$. Naively, this would require a double summation; however, i and j always occur together in the formula as $i + j$; indeed, the double summation is equivalent to:

$$\binom{m}{2}\sum_{i=2}^{\min(2k,k^*)}\binom{n-km+i-3}{m-3}$$

More generally, we can compute the probability for ℓ pairs as:

$$\binom{m}{\ell}\sum_{i=\ell}^{\min(\ell k,k^*)}\binom{n-km+i-\ell-1}{m-\ell-1}$$

And we can now sum over all ℓ from 1 to m:

Theorem 6.6.3. *The expected number of HSDir-HSDir-partition tuples that an attacker can observe is*

$$\sum_{\ell=1}^{m}\left[\sum_{\substack{1\le i_1\le\cdots\le i_\ell\le k\\(i_1+\cdots+i_\ell\le k^*)}}\binom{m}{C_0,C_1,\ldots,C_k}\binom{n-km+\sum_j i_j-\ell-1}{m-\ell-1}\right], \tag{6.2}$$

where, for each $j=1,\ldots,k$, C_j denotes the number of elements in $i_1,\ldots,i_\ell$ that are equal to j and where $C_0=m-\sum_{j=1}^{k}C_j$.

6.6.4 ATTACKS AGAINST TARGETED ONION SERVICE

A particular onion service has r replica sets; each has ℓ consecutive relays. For the attacker to target a particular onion service, such as Free Haven, it would first need to focus on one of the directories that contained that onion service. We compute here what is the probability if the attacker injects m number of relays. However, no pair of relays fall in any of the replica sets of the targeted onion service. The number of ways m attacker relays can be inserted so that not a single one fall in any replica set is $\binom{n-r\ell}{m}$. The number of ways m attacker relays can be inserted so that only one of them resides in any of the r replica sets is $\binom{n-r\ell}{m-1}\binom{r}{1}\binom{\ell}{1}$. Similarly, the number of ways m attacker relays can be injected so that each of r attacker relays fall in each replica set is $\binom{n-r\ell}{m-r}\binom{r}{r}\binom{\ell}{1}^r$. Therefore, the probability of inserting m attacker relays but no two of them fall in any of r replica sets of a particular onion service is:

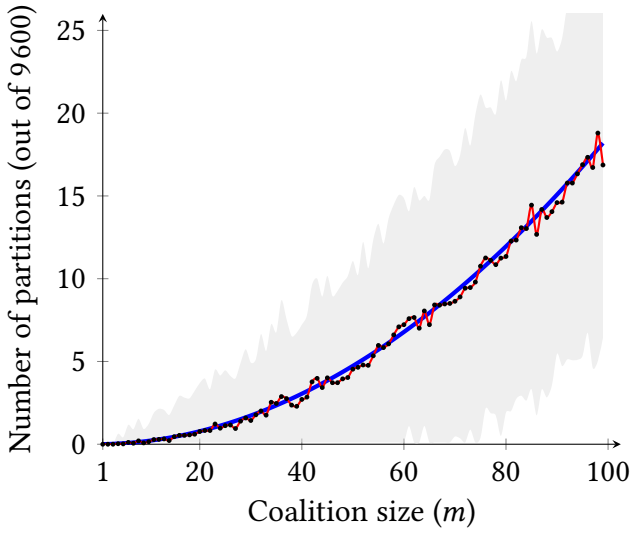


Figure 6.10: Expected coverage of partitions by an attacker controlling $\boldsymbol{m}$ out of $\boldsymbol{n = 3\,200}$ HSDirs. The smooth blue line is the theoretically predicted value; the jagged red line is the empirically observed average from 100 simulation trials; the shaded grey region is a (currently made up) 90% confidence interval obtained from the simulation results.

Theorem 6.6.4. *The probability that an attacker can observe a portion of the queries to a particular onion service is*

$$\sum_{i=0}^{r} \binom{n - r\ell}{m - i} \binom{r}{i} \binom{\ell}{1}^{i} / \binom{n}{m} \tag{6.3}$$

6.7 Tunica$_{\mathsf{LWE}}$ CONSTRUCTION

Here we outline our second construction using Ring-LWE based single server cPIR protocol from Section 6.2.2.2. The employment of the protocol is straightforward, except the only challenge in the query phase; i.e., the client has to know the original position (row number) of the intended descriptor in the database of the selected HSDir. Nevertheless, the layout of the database of the descriptors changes very frequently, and the position of a particular descriptor responsible for a specific onion service is dynamic duo to the onion service protocol (§6.3) and the nature of the

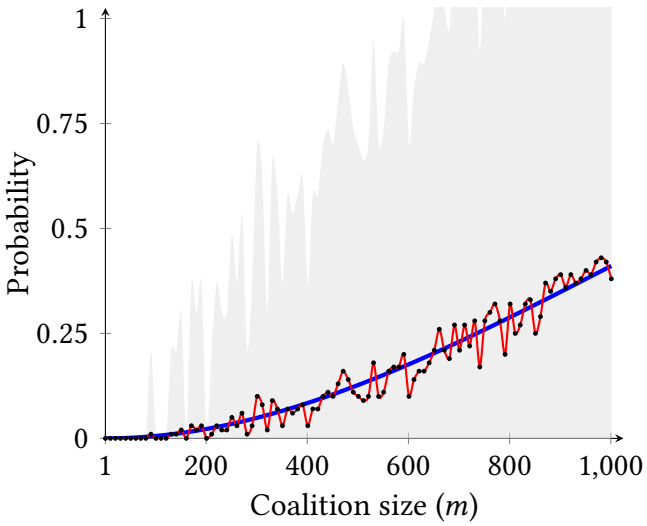


Figure 6.11: Probability that an attacker who controls $\boldsymbol{m}$ (out of $\boldsymbol{n = 3200}$) relays and *one* pair of them reside in any of the $\boldsymbol{r}$ replica sets of a particular onion service.

uploading of the descriptors by the onion service host. We encounter this problem by adding an extra level of interaction between the client and the HSDir, which we explain next.

First we discuss how the *FindIndex* function in the step 3 of Algorithm 6.12 works. The client fixes a keyed hash function and sends the key to the selected HSDir server S. The server creates a hash table indexed by the row number of each descriptor. It computes the hash of each descriptor ID w using the client selected keyed hash function and populated the hash table. Afterward, the complete hash table is sent back to the client. The client computes the hash of its targeted descriptor ID w and looks for it on the table. Following a hash function-based solution, the client will not know any information about other descriptors ID the HSDir is holding. After knowing the current row number, index i, of the targeted descriptor d, the client follows the underlying protocol of XPIR [1, §2.2] according to the Algorithms 6.12–6.14.

6.7.1 COST ANALYSIS

Say the expansion factor of the underlying encryption scheme is f. The client uploads $f \cdot r \cdot |\mathbb{F}|$ to the server and downloads $f \cdot s \cdot |\mathbb{F}|$ from the server. For the additional round of interaction, the

function QUERY($1^\lambda, w$)
    $S \in_R Replica(i;\ x)$
    $i \leftarrow FindIndex(S, w) \in_R [1\,..\,r]$
    $(sk, pk) \leftarrow KeyGen(1^\lambda)$
    $\langle q_1, \ldots, q_r \rangle \coloneqq \mathrm{Enc}(pk; \vec{e}_i)$
    return $(S, \langle q_1, \ldots, q_r \rangle)$
end function

Algorithm 6.12: QUERY is invoked by the client to initiate fetching an onion service descriptor.

function RESPOND($\langle q_1, \ldots, q_r \rangle$)
    for j from 1 to s **do**
        for i from 1 to r **do**
            $z_j \coloneqq q_j \cdot d_{ij}$
        end for
    end for
    $R \coloneqq \langle z_1, \ldots, z_s \rangle$
    return R
end function

Algorithm 6.13: RESPOND is invoked by server S upon receiving the query from the client.

function DECODE(R)
    Parse R as $(z_1, \ldots, z_s)$
    for j from 1 to s **do**
        $d_{ij} \coloneqq \mathrm{Dec}(sk; z_j)$
    end for
    $d \coloneqq \langle d_{i1}, \ldots, d_{is} \rangle$
    return d
end function

Algorithm 6.14: DECODE is invoked by the client upon receiving the response from the server.

Figure 6.15: $\mathsf{Tunica}_{\mathsf{LWE}}$ algorithms

client sends hash parameters and receives a hash table from the server. Note that using recursive techniques, the communication costs can be further reduced. Computation cost is high for each bit of the database as it involves operations with ciphertexts. XPIR [1] and SealPIR [4] proposed

smart techniques to reduce the computation cost as well as communication cost (especially the upload cost).

6.8 RELATED WORK

PIR has been proposed to employ in Tor previously by Mittal et al. [97] but in a different purpose—especially to find relays to construct Tor circuits. Maintaining a global presence of all relays, as network consensus, to all users is expensive and not scalable in the current Tor. A solution to this problem is the Peer-to-Peer approach, which scales to millions of relays but does not provide well-understood security and hence, prone to many attacks. As an alternative, Mittal et al. [97] applied PIR to allow a user to fetch information of a few numbers of relays instead of the entire database of all relays. Sasy and Goldberg [117] employed oblivious RAM and Trusted Executive Environment together to solve the same problem. Since directory servers will not know which relay(s) the user is interested in, a router fingerprinting attack is not feasible. They name it PIR-Tor and propose two constructions based on two different types of PIR. To establish single circuit cPIR-based construction offers one order of magnitude improvement over the trivial solution of downloading the entire database. IT-PIR-based one shows two orders of magnitude betterment. PIR-Tor achieves scalability, provable, and equivalent-to-current-Tor security, efficient circuit creation, and minimal changes to the current Tor by updating directory functionality and relay selection mechanism.

For CPIR-based construction, any relay can work as a PIR server which downloads the signed global view of network consensus from directory authorities. The client makes 2 PIR queries to retrieve descriptors for the middle node and exit node. For IT-PIR-based architecture, precisely three guard nodes are used as PIR servers that download the PIR database from directory authority. The client makes a single query to fetch the exit node descriptor. In both constructions, they maintain three separate databases for entry, middle, and exit nodes. The relays are sorted in the

databases by their bandwidth, which is the current relay selection algorithm for Tor. Every 10 minutes, a new circuit has been created, so the IT-PIR case is faster than the cPIR case having an extra PIR query. As a result, to speed up, they offer to reuse descriptors during multiple circuit rebuilds, which weakens privacy protection (unlinkability of circuits) but does not harm the user's anonymity. They improve the situation by keeping various descriptors in a single block of the PIR database.

Compared to the current literature of utilizing PIR to solve privacy and security problems in Tor network, to the best of our knowledge, ours is the first work to deploy PIR to protect onion service descriptor lookup for safeguarding onion services as well as the clients.

6.9 MICROBENCHMARK EXPERIMENTS

After designing some microbenchmarks, we discuss our experiment to measure the amount of complexity our protocol incurs concerning time and space. We assume $\ell = 3$ HSDirs in the replica set available for a client to fetch a descriptor from. The client submits a request to 2 of them. In particular, each HSDir has a dataset of descriptors with three partitions. The partition common in both of the queried HSDirs consists of a different number of descriptors where the requested descriptor is one of them. The client uses $\mathsf{Tunica}_{\mathsf{DPF}}$ scheme to retrieve the descriptor from both of the HSDirs.

Table 6.1 represents the experimental results after executing the primary $\mathsf{Tunica}_{\mathsf{DPF}}$ protocol for the microbenchmarks. Query generation and Data recovery columns are measured on the client-side, and Response generation is computed by the HSDir server. We run the client and HSDir servers on the same machine, which has Intel(R) Core(TM) i5-2500 @ 3.30GHz CPU and 8GB primary memory resources. We used the GO implementation of the Function Secret Sharing (FSS) library open-sourced by the group of [133] and adjusted the library to testify our protocol. The mock descriptors resided in the shared partition of the HSDir servers were generated by the

Table 6.1: Experimental results for different number of descriptors in a common partition. All timing experiments are executed for 100 trials to get a standard deviation upto one significant digit.

Number of descriptors	Size of descriptors	Query generation	Query size	Response generation	Response size	Data recovery
100	246 KB	360.1 ± 52.3 μs	597 B	13.1 ± 2.2 ms	2400 B	104.7 ± 18.3 μs
513	1213 KB	332.8 ± 33.0 μs	597 B	59.1 ± 2.9 ms	2364 B	104.7 ± 33.8 μs
1018	2408 KB	324.2 ± 24.5 μs	597 B	114.8 ± 4.9 ms	2364 B	100.1 ± 24.1 μs

Python package "leekspin 2.1.1" created by Isis Lovecruft [92]. Each descriptor body is padded with white characters up to the longest descriptor's length to keep it fixed in size.

We have examined multiple cases for three different number of descriptors in the shared partition in the table, e.g., 100, 513, and 1018 descriptors. The size of the current partitions is depicted in the second column of the table. The query generation and the data recovery time at the client side are nearly identical irrespective of the partition's size in all cases, as should be. Response generation by the server depends linearly on the number of descriptors the partition holds. The size of the key sent to a server is half a KB, and the response size is precisely equal to the most extended descriptor body. We inspected from [111] that there are 52000 unique onion services active and 4000 relays having HSDir flag. If we assume that the network parameters have default values, then each onion service has 6 descriptors. We observe that an HSDir relay holds around 80 descriptors on average in its dataset with this information. Hence, from our first case of the experiments, it is evident that our protocol incurs less than 15 milliseconds overhead to accommodate all the steps of our scheme. This finding shows that our method of acquiring an extra level of protection for the onion services is remarkably feasible since Tor is a low-latency network.

6.10 CHAPTER SUMMARY

We propose two Tunica constructions leveraging PIR to enable the onion service client to make a private query to fetch an onion service descriptor. This additional protection atop status-quo

Tor hinders hidden service directories (HSDirs) build popularity histogram of onion services and stops intersection-style attack against client's anonymity. We demonstrate rigorous security and privacy analysis of $\mathsf{Tunica}_{\mathsf{DPF}}$ construction, which involves two HSDirs and has a non-collusion assumption between them. $\mathsf{Tunica}_{\mathsf{LWE}}$ works with a single server but incurs an additional round of interaction. We plan to work with the Tor community to integrate our approaches to turn onion service infrastructure more secure and private for itself and the clients alike.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This dissertation aims to bring private information retrieval (PIR), a robust privacy-preserving cryptographic technique, more usable in practice. "Indexing" to view and access the database is one of the most effective and standard means in modern database management. Allowing indexing with PIR databases supporting an information-theoretic privacy guarantee (§4) is an outstanding contribution of this thesis. Supporting indexing with PIR databases put PIR forward to practice. Even with a strong non-collusion assumption—no pair of participating servers can communicate each other—we achieve optimal-rate performance in all metrics; e.g., upload cost, download cost, computation costs, and round requirement, in many-server PIR settings for both information-theoretic and computational privacy guarantee (§5).

We materialize our theoretical contributions through implementation and experimentation and release open-source libraries. This journey of studying practical PIR forces us to formally define PIR properties (§3) which were absent as a whole in the literature. Finally, we investigate PIR constructions' deployment to enable Tor users to access onion services without sharing their choice of onion services (descriptors lookup), which essentially improve the security and privacy of onion services and anonymous communication systems. With these powerful improvements, we envision that PIR gets closer to be deployed in various applications in the wild to enrich privacy-enhancing technologies (PETs). We next portrait some plans of our future directions in each thrust we discussed in this dissertation, slated to be conducted immediately.

Indexes of compound queries. In real database systems, the service provider uses multiple layers of indexing to facilitate complex user queries. In addition to advanced constructions like indexes of batch queries and indexes of aggregate queries, we aim to propose indexes of compound queries. This exciting possibility allows multiple indirections as a way of composition of multiple indexes of queries matrices.

Optimal-rate t-private PIR. One limitation of our bit-more-than-a-bit protocols is it assumes no-pair of server colludes with each other. We plan to inspect how we can permit a threshold number of servers may collide with each other with the guarantee of privacy. We like to propose a t-private PIR protocol that minimizes servers work as low as possible. Satisfying the requirement of holding t-privacy, we like to build a system of equations to find the lowest cost by optimization under a set of assumptions. The minimum work by the servers in a 1-private PIR protocol, such as bit-more-than-a-bit, is equal to the database's size due to Beimel et al. [13]. However, we argue that a t-private protocol, where $t > 1$, cannot achieve this optimal limit.

Redesign Tor onion service structure. Tor onion service system has been developed on an ad-hoc basis. We feel that a complete redesign of the onion service infrastructure can be more robust against current attacks. For instance, an onion service provider may upload its descriptors to more than the current limit, targeting a particular tunicate onion service more difficult. We plan to play with changing various parameters of our proposed Tunica constructions and inspect the merits of the outcomes in better security, privacy, availability, and anonymity.

BIBLIOGRAPHY

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2016(2):155–174 (April, 2016).

[2] Carlos Aguilar-Melchor, Benoît Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre Rousseau. High-speed private information retrieval computation on GPU. In *Proceedings of SECURWARE 2008*, pages 263–272, Cap Esterel, France (August, 2008).

[3] Carlos Aguilar-Melchor and Philippe Gaborit. A fast private information retrieval protocol. In *Proceedings of ISIT 2008*, pages 1848–1852, Toronto, ON, Canada (July, 2008).

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *Proceedings of IEEE S&P 2018*, pages 962–979, San Francisco, CA, USA (May, 2018).

[5] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, Savannah, GA (November, 2016). USENIX Association.

[6] Gilad Asharov, T-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Locality-preserving oblivious ram. Cryptology ePrint Archive, Report 2017/772, 2017. `https://eprint.iacr.org/2017/772`.

[7] Dmitri Asonov. Private information retrieval - an overview and current trends, 2001.

[8] Dmitri Asonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*, volume 3128 of *LNCS*. Springer, Berlin, Germany (June, 2004).

[9] UN General Assembly. Universal declaration of human rights (217 [iii] a), 1948.

[10] Karim A. Banawan and Sennur Ulukus. Private information retrieval from Byzantine and colluding databases. In *Proceedings of Allerton 2017*, pages 1091–1098, Monticello, IL, USA (October, 2017).

[11] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology: Proceedings of CRYPTO 1986*, volume 263 of *LNCS*, pages 311–323, Santa Barbara, CA, USA (August, 1987).

[12] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Ilan Orlov. Share conversion and private information retrieval. In *Proceedings of CCC 2012*, pages 258–268, Porto, Portugal (June, 2012).

[13] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151 (March, 2004).

[14] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. In *Proceedings of SCN 2002*, volume 2576 of *LNCS*, pages 326–341, Amalfi, Italy (September, 2002).

[15] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor hidden services: Detection, measurement, deanonymization. In *Proceedings of IEEE S&P 2013*, pages 80–94, Berkeley, CA, USA (May, 2013).

[16] Simon Blackburn and Tuvi Etzion. PIR array codes with optimal PIR rate. *arXiv:CoRR*, abs/1607.00235 (July, 2016).

[17] Simon R. Blackburn, Tuvi Etzion, and Maura B. Paterson. PIR schemes with small download complexity and low storage requirements. In *Proceedings of ISIT 2017*, pages 146–150, Aachen, Germany (June, 2017).

[18] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426 (July, 1970).

[19] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Advances in Cryptology: Proceedings of CRYPTO 2007*, volume 4622 of *LNCS*, pages 50–67, Santa Barbara, CA, USA (August, 2007).

[20] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. In *Proceedings of PETS 2015*, pages 4–24, Philadelphia, PA, USA (June–July, 2015).

[21] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology: Proceedings of EUROCRYPT 2015 (Part II)*, volume 9057 of *LNCS*, pages 337–367, Sofia, Bulgaria (April, 2015).

[22] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *Advances in Cryptology: Proceedings of CRYPTO 2016 (Part I)*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA (August, 2016).

[23] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of CCS 2016*, pages 1292–1303, Vienna, Austria (October, 2016).

[24] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *Advances in Cryptology: Proceedings of CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524, Santa Barbara, CA, USA (August, 2011).

[25] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology: Proceedings*

of EUROCRYPT 1999, volume 1592 of *LNCS*, pages 402–414, Prague, Czech Republic (May, 1999).

[26] Justin Cappos. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In *Proceedings of FC 2013*, volume 7859 of *LNCS*, pages 386–394, Okinawa, Japan (April, 2013).

[27] Terence H. Chan, Siu-Wai Ho, and Hirosuke Yamamoto. Private information retrieval for coded storage. In *Proceedings of ISIT 2015*, pages 2842–2846, Hong Kong (June, 2015).

[28] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Forensics and Security (TIFS)*, 4(4):611–627 (December, 2009).

[29] Z. Chen, Z. Wang, and S. A. Jafar. The capacity of t-private information retrieval with private side information. *IEEE Transactions on Information Theory*, 66(8):4761–4773, 2020.

[30] Benny Chor and Niv Gilboa. Computationally private information retrieval (Extended abstract). In *Proceedings of STOC 1997*, pages 304–313, El Paso, TX, USA (May, 1997).

[31] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report CS 0917, Technion-Israel Institute of Technology, Haifa, Israel (February, 1997).

[32] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of FOCS 1995*, pages 41–50, Milwaukee, WI, USA (October, 1995).

[33] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981 (November, 1998).

[34] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. In *Proceedings of ANTS X (2012)*, volume 1, number 1 of *The Open Book Series*, pages 271–293, San Diego, CA, USA (July, 2012).

[35] Wikipedia contributors. Tunica (biology) — Wikipedia, the free encyclopedia. *[Online; accessed 2017-11-18; rev 788720943]*

[36] Nvidia Corporation. Tesla® Kepler™ GPU Accelerators. `http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf`. *(Accessed: February 16, 2017).*

[37] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Proceedings of IEEE S&P 2015*, pages 321–338, San Jose, CA, USA (May, 2015).

[38] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of IEEE S&P 2003*, pages 2–15, Oakland, CA, USA (May, 2003).

[39] Daniel Demmler, Amirr Herzberg, and Thomas Schneider. RAID-PIR; version 0.9.5 [computer software]. Available from: `https://github.com/encryptogroup/RAID-PIR` (October, 2016).

[40] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: Scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018(4):159 – 178, 2018.

[41] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[42] Danniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: Practical multi-server PIR. In *Proceedings of CCSW 2014*, pages 45–56, Scottsdale, AZ, USA (November, 2014).

[43] Casey Devet and Ian Goldberg. The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency. In *Proceedings of PETS 2014*, pages 63–82, Amsterdam, The Netherlands (July, 2014).

[44] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *Proceedings of USENIX Security 2012*, pages 269–283, Bellevue, WA, USA (August, 2012).

[45] Roger Dingledine and Nick Mathewson. Tor rendezvous specification. In *torspec—Tor's Protocol Specification*. The Tor Project. *[Online; accessed 2017-11-19; commit d3a93e1]*

[46] Roger Dingledine and Nick Mathewson. Tor rendezvous specification–Version 3. In *torspec—Tor's Protocol Specification*. The Tor Project, 2014. *[Online; accessed 2017-11-19; commit ef5f940]*

[47] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *Proceedings of USENIX Security 2004*, San Diego, CA, USA (August, 2004).

[48] Roger Dingledine (as *arma*). Tor security advisory: "relay early" traffic confirmation attack. Blog post on *Tor Blog* (July 30,, 2014). *[Online; accessed 2017-11-17]*

[49] Klim Efremenko. 3-query locally decodable codes of subexponential length. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 39–44, New York, NY, USA, 2009. Association for Computing Machinery.

[50] Torbjörn Granlund et al. GNU multiple precision arithmetic library; version 6.1.2 [computer software]. Available from: `https://gmplib.org/` (December, 2016).

[51] Arman Fazeli, Alexander Vardy, and Eitan Yaakobi. Codes for distributed PIR with low storage overhead. In *Proceedings of ISIT 2015*, pages 2852–2856 (June, 2015).

[52] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, Emil Stefanov, Dimitrios Serpanos, and Srinivas Devadas. A low-latency, low-area hardware oblivious ram controller. Cryptology ePrint Archive, Report 2014/431, 2014. https://eprint.iacr.org/2014/431.

[53] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A random server model for private information retrieval or how to achieve information theoretic PIR avoiding database replication. In *Proceedings of RANDOM 1998*, volume 1518 of *LNCS*, pages 200–217, Barcelona, Spain (October, 1998).

[54] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Science (JCSS)*, 60(3):592–629 (June, 2000).

[55] Niv Gilboa. Two party rsa key generation. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 116–129, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[56] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *Advances in Cryptology: Proceedings of EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658, Copenhagen, Denmark (May, 2014).

[57] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of IEEE S&P 2007*, pages 131–148, Oakland, CA, USA (May, 2007).

[58] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ / PIR in C++; version 1.0 [computer software]. Available from: git://git-crysp.uwaterloo.ca/percy (October, 2014).

[59] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of STOC 1987*, pages 218–229, New York, NY, USA (May, 1987).

[60] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473 (May, 1996).

[61] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[62] Google Scholar. `https://scholar.google.com/`. *(Accessed: February 16, 2017).*

[63] David Goulet (as *dgoulet*). What's new in Tor 0.2.9.8. Blog post on *Tor Blog* (December 19,, 2016).

[64] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1591–1601, New York, NY, USA, 2016. ACM.

[65] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *Proceedings of NSDI 2016* (March, 2016).

[66] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath T. V. Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with Popcorn. In *Proceedings of NSDI 2016* (March, 2016).

[67] Syed Mahbub Hafiz and Ryan Henry. Querying for queries: Indexes of queries for efficient and expressive IT-PIR. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1361–1373, New York, NY, USA, 2017. ACM.

[68] Syed Mahbub Hafiz and Ryan Henry. A bit more than a bit is more than a bit better: Faster (essentially) optimal-rate many-server PIR. *Proceedings on Privacy Enhancing Technologies*, 2019(4):112 – 131, 2019.

[69] Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 126–143, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[70] Ryan Henry. Polynomial batch codes for efficient IT-PIR. In *Proceedings on Privacy Enhancing Technologies (PoPETS)*, volume 2016(4), pages 202–218, Darmstadt, Germany (July, 2016).

[71] Ryan Henry. dpf++; version 0.0.1 [computer software]. Available from: `https://www.github.com/rh3nry/dpfplusplus` (July, 2019).

[72] Ryan Henry and Syed Mahbub Hafiz. libbitmore; version v0.0.1 [computer software]. Available from: `https://www.github.com/rh3nry/libbitmore` (July, 2019).

[73] Ryan Henry, Kevin Henry, and Ian Goldberg. Making a Nymbler Nymble using VERBS. In *Proceedings of PETS 2010*, volume 6205 of *LNCS*, pages 111–129, Berlin, Germany (July, 2010).

[74] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In *Proceedings of NDSS 2013*, San Diego, CA, USA (February, 2013).

[75] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical PIR for electronic commerce. In *Proceedings of CCS 2011*, pages 677–690, Chicago, IL, USA (October, 2011).

[76] IACR Cryptology ePrint Archive. `https://eprint.iacr.org/`. *(Accessed: August 31, 2015).*

[77] Alexander Iliev and Sean Smith. Private information storage with logarithmic-space secure hardware. In Yves Deswarte, Frédéric Cuppens, Sushil Jajodia, and Lingyu Wang, editors,

Information Security Management, Education and Privacy, pages 201–216, Boston, MA, 2004. Springer US.

[78] Yuval Ishai and Eyal Kushilevitz. On the hardness of information-theoretic multiparty computation. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 439–455, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[79] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of STOC 2004*, pages 262–271, Chicago, IL, USA (June, 2004).

[80] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *Proceedings of FOCS 2006*, pages 239–248, Berkeley, CA, USA (October, 2006).

[81] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The Sniper Attack: Anonymously deanonymizing and disabling the Tor network. In *Proceedings of NDSS 2014*, San Diego, CA, USA (February, 2014).

[82] Z. Jia and S. A. Jafar. On the asymptotic capacity of x-secure t-private information retrieval with graph-based replicated storage. *IEEE Transactions on Information Theory*, 66(10):6280–6296, 2020.

[83] Z. Jia, H. Sun, and S. A. Jafar. Cross subspace alignment and the asymptotic capacity of x -secure t -private information retrieval. *IEEE Transactions on Information Theory*, 65(9):5783–5798, 2019.

[84] George Kadianakis (as *asn*). Hidden services need some love. Blog post on *Tor Blog* (April 22„ 2013). *[Online; accessed 2017-11-20]*

[85] George Kadianakis (as *asn*). A hidden service hackfest: The Arlington accords. Blog post on *Tor Blog* (August 4„ 2015). *[Online; accessed 2017-11-20]*

[86] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. In *Proceedings of PETS 2015*, volume 2, pages 222–243, Philadelphia, PA, USA (June–July, 2015).

[87] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of FOCS 1997*, pages 364–373, Miami Beach, FL, USA (October, 1997).

[88] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *Proceedings of USENIX Security 2015*, pages 287–302, Washington, DC, USA (August, 2015).

[89] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: an efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115 – 134, 2016.

[90] Peeter Laud. A private lookup protocol with low online complexity for secure multiparty computation. In Lucas C. K. Hui, S. H. Qing, Elaine Shi, and S. M. Yiu, editors, *Information and Communications Security*, pages 143–157, Cham, 2015. Springer International Publishing.

[91] Tancrède Lepoint and Mehdi Tibouchi. Cryptanalysis of a (somewhat) additively homomorphic encryption scheme used in PIR. In *Proceedings of WAHC 2015*, volume 8976 of *LNCS*, pages 184–193, San Juan, Puerto Rico (January, 2015).

[92] Isis Lovecruft. Leekspin—An Onion Router descriptor generator (July, 2016). *[Computer software; version 2.1.1]*

[93] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *Proceedings of FC 2015*, volume 8975 of *LNCS*, pages 168–186, San Juan, Puerto Rico (January, 2015).

[94] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):43:1–43:35 (November, 2013).

[95] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Efficient private file retrieval by combining ORAM and PIR. In *Proceedings of NDSS 2014*, San Diego, CA, USA (February, 2014).

[96] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296, 2018.

[97] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *Proceedings of USENIX Security 2011*, pages 475–490, San Francisco, CA, USA (August, 2011).

[98] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. X-search: Revisiting private web search using intel sgx. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, Middleware '17, page 198–208, New York, NY, USA, 2017. Association for Computing Machinery.

[99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of STOC 1999*, pages 245–254, Atlanta, GA, USA (May, 1999).

[100] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology: Proceedings of CRYPTO 1999*, volume 1666 of *LNCS*, pages 573–590, Santa Barbara, CA, USA (August, 1999).

[101] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 448–457, USA, 2001. Society for Industrial and Applied Mathematics.

[102] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 111–125, Washington, DC, USA, 2008. IEEE Computer Society.

[103] Arvind Narayanan and Vitaly Shmatikov. Myths and fallacies of "personally identifiable information". *Commun. ACM*, 53(6):24–26 (June, 2010).

[104] Arvind Narayanan and Vitaly Shmatikov. Myths and fallacies of "personally identifiable information". *Communications of the ACM (CACM)*, 53(6):24–26 (June, 2010).

[105] Femi G. Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of PETS 2010*, volume 6205 of *LNCS*, pages 75–92, Berlin, Germany (July, 2010).

[106] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Proceedings of FC 2011*, volume 7035 of *LNCS*, pages 158–172, Gros Islet, St. Lucia (February, 2011).

[107] Femi G. Olumofin, Piotr K. Tysowski, Ian Goldberg, and Urs Hengartner. Achieving efficient query privacy for location based services. In *Proceedings of PETS 2010*, volume 6205 of *LNCS*, pages 93–110, Berlin, Germany (July, 2010).

[108] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In *Proceedings of PKC 2007*, volume 4450 of *LNCS*, pages 393–411, Beijing, China (April, 2007).

[109] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1002–1019, New York, NY, USA, 2018. ACM.

[110] The Tor Project. Relays and bridges. In *Tor Metrics Portal: Servers. [Online; accessed 2017-12-05]*

[111] The Tor Project. Relays by relay flag (HSDir). In *Tor Metrics Portal: Servers. [Online; accessed 2017-11-20]*

[112] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005.

[113] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-complete private information retrieval. Technical Report CACR 2007-34, University of Waterloo, Waterloo, ON, Canada (December, 2007).

[114] Irving S. Reed and Gustav Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 8(2):300–304 (June, 1960).

[115] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *Proceedings of ESORICS 2006*, volume 4189 of *LNCS*, pages 313–326, Hamburg, Germany (September, 2006).

[116] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon Gate: A secure method of pseudonymous mail retrieval. In *Proceedings of WPES 2005*, pages 1–9, Alexandria, VA, USA (November, 2005).

[117] Sajin Sasy and Ian Goldberg. Consensgx: Scaling anonymous communications networks with trusted execution environments. *Proc. Priv. Enhancing Technol.*, 2019(3):331–349, 2019.

[118] Nihar B. Shah, K. V. Rashmi, and Kannan Ramchandran. One extra bit of download ensures perfectly private information retrieval. In *Proceedings of ISIT 2014*, pages 856–860, Honolulu, HI, USA (June–July, 2014).

[119] Adi Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613 (November, 1979).

[120] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious ram with o((logn)³) worst-case cost. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, page 197–214, Berlin, Heidelberg, 2011. Springer-Verlag.

[121] Victor Shoup. NTL: A library for doing number theory; version 10.5.0 [computer software]. Available from: `http://www.shoup.net/ntl` (July, 2017).

[122] Radu Sion and Bogdan Carbunar. On the practicality of private information retrieval. In *Proceedings of NDSS 2007*, San Diego, CA, USA (March, 2007).

[123] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683–695, 2001.

[124] Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. *J. ACM*, 65(4) (April, 2018).

[125] Julien P. Stern. A new efficient all-or-nothing disclosure of secrets protocol. In *Advances in Cryptology: Proceedings of ASIACRYPT 1998*, volume 1514 of *LNCS*, pages 357–371, Beijing, China (October, 1998).

[126] H. Sun and S. A. Jafar. Multiround private information retrieval: Capacity and storage overhead. *IEEE Transactions on Information Theory*, 64(8):5743–5754, 2018.

[127] Hua Sun and Syed A. Jafar. The capacity of symmetric private information retrieval, 2017.

[128] Hua Sun and Syed Ali Jafar. The capacity of private information retrieval with colluding databases. In *Proceedings of GLOBECOM 2016*, pages 941–946, Washington, DC, USA (December, 2016).

[129] Hua Sun and Syed Ali Jafar. The capacity of private information retrieval. *IEEE Transactions on Information Theory*, 63(7):4075–4088, 2017.

[130] Hua Sun and Syed Ali Jafar. The capacity of robust private information retrieval with colluding databases. *IEEE Transactions on Information Theory*, 64(4):2361–2370, 2018.

[131] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ-private information retrieval. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2016(2):184–201 (October, 2016).

[132] Jonathan T. Trostle and Andy Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In *Proceedings of ISC 2010*, volume 6531 of *LNCS*, pages 114–128, Boca Raton, FL, USA (October, 2010).

[133] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 299–313, Berkeley, CA, USA, 2017. USENIX Association.

[134] Luqin Wang, Trishank Karthik Kuppusamy, Yong Liu, and Justin Cappos. A fast multi-server, multi-block private information retrieval protocol. In *Proceedings of GLOBECOM 2015*, pages 1–6, San Diego, CA, USA (December, 2015).

[135] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In *Proceedings of ESORICS 2006*, volume 4189 of *LNCS*, pages 49–64, Hamburg, Germany (September, 2006).

[136] Peter Williams and Radu Sion. Usable PIR. In *Proceedings of NDSS 2008*, San Diego, CA, USA (February, 2008).

[137] Erica Y. Yang, Jie Xu, and Keith H. Bennett. Private information retrieval in the presence of malicious failures. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, COMPSAC '02, pages 805–812, Washington, DC, USA, 2002. IEEE Computer Society.

[138] Andrew Chi-Chih Yao. How to generate and exchange secrets. SFCS '86, page 162–167, USA, 1986. IEEE Computer Society.

[139] Sergey Yekhanin. New locally decodable codes and private information retrieval schemes. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(127) (October, 2006).

[140] Sergey Yekhanin. Private information retrieval. *Communications of the ACM (CACM)*, 53(4):68–73 (April, 2010).