Department of Informatics

Master Thesis

An Introduction to Information-Theoretic Private Information Retrieval (IT-PIR)

By: Tarald Riise
Supervisor: Øyvind Ytrehus

June 3, 2019

Preface

While there exists excellent, readily-available research on the field of Private Information Retrieval, the threshold for understanding topics of technical nature through the reading of academic papers tends to be quite high for the typical non-academic.

In order to accommodate interest from technical, but not necessarily academic, individuals, it seems an approachable, highly synthesized version of the most relevant research of the field may prove a valuable resource to the curious. This thesis will attempt to provide such a synthesized and approachable experience in paper form.

By gradually introducing relevant terminology, and thoroughly explaining constructs as we go along, it is my belief that this thesis may provide some useful insight to the current state of Private Information Retrieval, as well as its potential applications.

Naturally, as the field of Private Information Retrieval is vast and has currently ongoing research, I cannot claim to be able to provide a complete overview of the field. However, it is my aim that you, the reader, after having read this thesis, will feel that you have some degree of insight to and understanding of the field.

Abstract

Private Information Retrieval (PIR) is a way of querying a database server for a record, and getting that record back, without the server learning which record it returned, thus protecting the privacy of the user retrieving the record. This thesis will attempt to provide the reader with an introduction into the field of Information-Theoretic Private Information Retrieval (IT-PIR) so that the reader may gain an understanding of IT-PIR and its potential and hindrances as a privacy enhancing tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As the financial incentive for businesses to harvest and store data and meta-data pertaining to our online activity has grown significantly in the past twenty years, so has the privacy issue that this data poses. As commercial businesses arguably become the proprietors of the information obtained by processing our online requests, attempts can be made to extract intent and behavioural patterns relating to both our on- and offline presence from this information. This information may then be used for any range of motivations, may it be for improving user experience, advertising, or for malicious reasons. To better protect our online privacy, we may consider Privacy Enhancing Tools (PETs), such as Private Information Retrieval (PIR). In the case of PIR, we may for the sake of privacy want to utilize a method of obtaining information from a database server without revealing to the server what information we obtained.

Consider the following scenario. Alice has an idea for a website and wishes to buy a domain name to host her website on, for example `www.alicesgreatpage.com`. Alice visits the website of a not-so-honest Domain Name Registrar, henceforth *ShadyDNR*, offering a web-based form to let her check the availability of any domain name, as well as the price of buying a domain name if it is available. Alice enters her preferred domain name into the form, and clicks the submit button. The submit button triggers an HTTP-request to a ShadyDNR server, with the request containing the domain name Alice is interested in, namely `www.alicesgreatpage.com`. As this request is received at the ShadyDNR server, ShadyDNR preemptively purchases the domain name for themselves at the price of $5. The ShadyDNR server then responds to Alice's request that the domain name is not available, but that ShadyDNR happens to own this domain name, and that Alice may purchase the domain name directly from ShadyDNR at the price of $50. ShadyDNR will of course willfully omit that they themselves bought and artificially marked-up the price

of the domain name.

This example of the not-so-honest Domain Name Registrar should highlight one of the reasons why we may sometimes wish to obtain information from a server without revealing exactly what we are interested in. In this example we would like to request information about a domain name without revealing to the server which domain name we are interested in, but there exists a variety of problems where we would like to achieve something similar. These problems are instances of a more general problem known as Private Information Retrieval(PIR).

A perceptive reader may have already discovered a solution to the problem of PIR. In the case of the not-so-honest Domain Name Registrar, if Alice were to request the availability and price of all possible domain names, ShadyDNR could not in any way infer which domain name Alice is actually interested in, and ShadyDNR would be economically and practically deterred from applying their usual business practices. This is known as the *trivial solution* to the Private Information Retrieval problem, namely retrieving the entire database for the purpose of accessing only a subset of the records. While the trivial solution may be useful for small databases, it is extremely inefficient and impractical for larger databases, because of the overhead associated with downloading a multitude of records, only to use a subset. A somewhat fitting analogy would be to borrow every single book in a library to avoid the librarian knowing what book you truly wanted to read. While the Trivial Solution solves the Private Information Retrieval problem, it may be an extremely costly operation depending on the size of the server database, or conversely, in our library analogy, the size of the library.

As you may suspect by now, the field of Private Information Retrieval attempts to solve the problem of Private Information Retrieval in a manner more efficient than the trivial solution. Numerous PIR variations and schemes have been proposed in the literature, and some have been implemented[1][2][3].

1.2 Objectives

We will in this thesis attempt to provide the reader with an understanding of Information-Theoretic Private Information Retrieval from both a theoretical and practical standpoint. In regards to theory, this understanding will include the crypto-theoretic foundation needed to understand a IT-PIR scheme. From the practical perspective, we will show how PIR protocols may be used to solve a real world problem observed in the implementation of the onion routing protocol *Tor*. The thesis should be understandable to readers with limited knowledge of cryptography, as well as informative to readers with pre-existing knowledge of cryptography. We will in the conclusion reason around the potential future of PIR, and discuss whether or not we believe PIR will become widely

adopted and deployed.

1.3 Outline

An outline of this thesis may be given as an overview of the chapters.

- **Chapter 2** - My background for writing this thesis.
- **Chapter 3** - Introducing terminology and concepts relevant for subsequent chapters.
- **Chapter 4** - Properties of information retrieval protocols.
- **Chapter 5** - Exploring the first published PIR protocol.
- **Chapter 6** - Achieving robustness in PIR protocols.
- **Chapter 7** - Fetching multiple records at a time using PIR.
- **Chapter 8** - Querying Expressively in PIR.
- **Chapter 9** - Introducing anonymous information retrieval and Tor. Exploring how PIR may be used to scale Tor.
- **Chapter 10** - Conclusion

Chapter 2

Background

Private Information Retrieval is a field of research which is rather popular at the moment. This may be attributed to the fact that the technology is relatively new, with the first paper being published as recently as in 1995[4]. One might also argue that the public is becoming increasingly privacy-aware, and if true, this may attribute to increased interest in Private Information Retrieval research and other Privacy Enhancing Technologies(PETs).

With the popularity of the field, combined with my personal appreciation of PETs and my background in cryptography and computer security, it seems fitting that I write my master's thesis on Private Information Retrieval with the purpose of accommodating further interest.

This concludes my background for the thesis.

Chapter 3

Terminology

Before we will begin explaining properties, metrics, constructs and protocols, it may be useful to establish some terminology that allows us to more easily discuss these subjects. This chapter will therefore explain some terms often used when discussing cryptosystems.

3.1 *Information-Theoretic Security*

As we will primarily focus on information-theoretically secure PIR-protocols, it makes sense that we delve a bit deeper into the meaning of information-theoretic security. In Christof Paar's *Understanding Cryptography*, a definition of unconditional security, also known as information-theoretic security, is given as such:

Definition 3.1.1. Unconditional security: A cryptosystem is unconditionally or information-theoretically secure if it cannot be broken even with infinite computational resources[5].

3.1.1 Common misconceptions

Some readers may be under the impression that all cryptosystems attempt to achieve the information-theoretical security property as defined in definition 3.1.1, this is however incorrect.

One could argue that breaking the encryption of an 256-bit AES-encrypted email using fifty state-of-the-art supercomputers may take longer than $3 * 10^{51}$ years, exceeding the current lifetime of this universe an unfathomable number of times. While this task, for all practical reasons, is impossible today, this does not qualify AES as an information-

theoretically secure cryptosystem. If we truly had unlimited resources, as given in definition 3.1.1, we could simply build an infinite number of supercomputers, making the process instantaneous.

So if modern cryptosystems using adequately large keys does not yield information-theoretical security, what does?

3.1.2 Randomness

Randomness lies at the core of much of cryptography, and we will later see how randomness plays an integral part in creating a IT-PIR protocols. But what is randomness? If we look at this problem from a mathematical or physical perspective, we may say that a random event is an event that is non-deterministic. However, if we look at the problem from a computer-scientific perspective, we will see that there may be multiple interpretations.

PsuedoRandom Number Generators (PRNGs)

Most computer scientists have in their computer programs used a PsuedoRandom Number Generator (PRNG), and some programmers may believe that these generators are non-deterministic. However, upon closer inspection, we may notice that PRNGs are in fact deterministic, thus providing the "Pseudo" of the PseudoRandom Number Generators. To observe this, we may look at the internals of any PRNG. There, we will see that all PRNGs take a *seed*-argument, which sets the generator to a deterministic state. Upon instantiating a PRNG with a seed s_0 and calling the PRNG's *generate* function, the PRNG will generate a seemingly random value x. However, by creating a new instance of the same generator with the same seed s_0, and again calling the *generate* function, we will again be returned the very same value x. This is obviously deterministic behaviour.

If we put this into the context of the definition of information theoretic security (3.1.1), an adversary with unlimited computational resources may brute-force the seed parameter to obtain a copy of our PRNG, given that some output of our PRNG is somehow determinable to the adversary[1]. Any additional "randomness" generated by our PRNG may then be trivially obtained by the adversary, simply by calling the *generate* function.

[1]If no output of a PRNG is determinable to the adversary, the adversary has no way to determine whether the correct seed is found, even given unlimited computational resources.

Cryptographically Secure PRNGs (CSPRNGs)

Some readers may also have used a Cryptographically Secure PsuedoRandom Number Generator (CSPRNG), and may have been comforted by the "Cryptographically Secure" part of the nomenclature. However, the cryptographical security of the CSPRNG does not refer to the generator being non-deterministic. On the contrary, an adversary obtaining the initial seed-parameter of a CSPRNG will break the supposed randomness of the system, equivalent to any PRNG, and the adversary may look into the future of our generator, simply by calling the *generate* function. What makes these generators suitable for cryptography is a property known as *unpredictability*. The unpredictability property may be explained in the following manner:

> *Given n output bits of the CSPRNG, there exists no polynomial time algorithm that can predict the next bit s_{n+1} with better than 50% chance of success*[5].

Note that the quote gives a polynomial time algorithm as a bound, which is not a limitation for information-theoretic security. CSPRNGs can therefore be used in computationally secure cryptosystems, but as they are dependent upon the initial seed, they may never yield information theoretically secure cryptosystems if some output of the CSPRNG is determinable to the adversary.

True Random Events

True random events are non-deterministic, stochastic events, and may be observed in nature as thermal noise, radioactive decay, other miscellaneous quantum effects and events such as the flipping of a fair coin. Such events can not be described by any reliably measurable variables, and can therefore be used as a source of true randomness. In computer science, such events are usually reduced to a single bit-state b, such that $b \in \{0, 1\}$ with probability 50%. Hardware solutions using such true random events as a source of randomness exist[6] and are commercially available.

Since it is impossible to guess any true random event with a probability greater than 50%, and as all such events are independent, unlimited computational resources does not help an adversary in guessing the next outcome of a true random event.

3.1.3 The One Time Pad (OTP)

Now that we have a grasp of how we can achieve different types of randomness, we can look at applications of randomness in cryptography, and how we may use true randomness

to create an information-theoretically secure cryptosystem.

A Single-bit One Time Pad

Suppose I were to flip a fair coin and tell you the outcome in a secret meeting. If neither of us were to share the outcome with any third party, no-one but you and me would know the outcome, and the outcome may be referred to as our *shared secret key* $s_0 \in \{0, 1\}$. No matter how much computational power a potential adversary may have, the adversary can do nothing but guess, giving the chance of being correct or incorrect as simply 50%.

To make the example more exiting, we may hypothesize that I will at some point have access to inside information about a big company, and this information may prove profitable on the stock market. However, I can not act on this information myself, and I will need you to either *buy* or *sell* some stocks in the company at some time in the future. We obviously do not wish to leave any kind of evidence, as this practice is illegal, so I would like to be able to somehow securely communicate which of the two actions you should perform. To achieve this, we may use our shared secret key s_0.

To signal that you should buy or sell on any given day, I will raise a flag in my back yard, either the green flag, or the red flag. To make sure no one could learn which flag represents which action, I will use our shared secret key to encrypt the action. For example, if the result of the coin-toss was *heads*, the green flag signals the *buy* action and the red flag signals the *sell* action. Conversely, if the result of the coin-toss was *tails*, the green flag signals the *sell* action and red flag signals the *buy* action. These actions of buying and selling on any given day i may be expressed in binary as $x_i \in \{0, 1\}$, and which flag I raise on that particular day may be referred to as the encrypted output $y = E(buy|sell, key)$[2] and may be expressed in binary as $y_i \in \{0, 1\}$.

The encrypted output for day $i = 0$ may be formally expressed as

$$y_0 = x_0 \oplus s_0 \tag{3.1}$$

Equation (3.1) shows a secure single-bit one time pad where $\oplus$ is the bit-wise XOR operation.

As it is impossible for an adversary to know the outcome of the coin-flip s_0, it is also impossible for the adversary to determine x_0 with any other probability than 50%, even given the output y_0 and unlimited computational resources. This example demonstrates a single-bit One Time Pad that is information-theoretically secure.

[2]The E-function is standard notation in cryptography for a function encrypting some plain-text x by a key k s.t. $E(x, k) = y$, where y is the encrypted output. In this case, the E-function simply performs the bit-wise XOR operation of x and k, and returns the result as the output.

Multi-bit OTP

Given that our scheme proved successful, we may wish to repeat this hypothetical. However, an adversary could have looked at my flag at day 0, and correlated that to the stock market movement on day 1, effectively learning the secret for day 0, s_0. If we were to encrypt the action for day 1 using the same key s_0, it is trivial for the adversary to learn the new secret y_1.

$$\begin{aligned} i &= 1 \\ y_0 &= x_0 \oplus s_0 \\ y_1 &= x_1 \oplus s_0 \end{aligned} \tag{3.2}$$

Equation (3.2) shows a reuse of a one time pad key, which deems it insecure.

As using the same key s_0 for future communication makes the scheme insecure, we obviously need to change our key. So if we were to flip n coins in our secret meeting, we could use one key each day for n days without key reuse. Following the logic given by equation 3.1, we can rest assured that no adversary may learn what we are communicating.

$$\begin{aligned} y_0 &= x_0 \oplus s_0 \\ y_1 &= x_1 \oplus s_1 \\ &\vdots \end{aligned} \tag{3.3}$$

Equation (3.3) shows a provably secure, information-theoretic cryptosystem.

This means that if we want to be able to communicate for five different days, we would need to meet to share the results of five different coin tosses, or another true random event equivalent. Each true random event (key) may be used only once, which gives name to the One Time Pad, an information-theoretically secure cryptosystem.

One might think that by somehow cleverly combining used keys, we can arrive at new keys, but any such method is vulnerable to reverse engineering by an adversary with unlimited computational resources.

The Impracticality of the OTP

While the idea of having an unconditionally secure cryptosystem is a rather nice one, there are some serious practical limitations to the OTP.

To encrypt n bits, we need a key consisting of n bits. This means that to securely encrypt a 1GB file, we need 1GB of keys. Additionally, because key reuse is insecure,

we need to refill keys before we run out, or we risk being unable to communicate securely. Sending keys over a non-information-theoretically secure communications channel will deem the resulting OTP as information-theoretically **in**secure due to the transitive property. This means keys may have to be physically moved from one location to another. These impracticalities are the reasons we do not use One Time Pads for ordinary every-day secure communications.

3.2 *Computational Security*

Unlike information-theoretical security, in computational security, we model our adversary to have bounded computational resources. This relaxation allows us to create cryptosystems that are not provably information-theoretically secure. Instead, computationally secure cryptosystems must generally prove that an adversary must solve a *hard problem* in order to break the security of the system. The idea that some problems are hard to solve, is known as the *hardness assumption*.

3.2.1 The hardness assumption

The hardness assumption is the hypothesis that some problems cannot be solved by a computer in polynomial time. While this hypothesis remains exactly that, a hypothesis, researchers have put in decades of work without revealing polynomial time solutions to hard problems. Cryptanalysis has uncovered weaknesses in existing computationally secure protocols[3], though this is strictly related to implementations, and not to solving the underlying hard problems[4]. This allows us to take an optimistic view on the subject and conjecture that the hardness assumption holds. The hardness assumption can be looked at as closely tied to the $P \neq NP$ problem.

3.2.2 RSA and Integer Factorization

We will not explain the workings of RSA in this paper, but it is worth to briefly mention *integer factorization* as the hard problem at the core of RSA. It turns out that factoring

[3] As soon as a serious vulnerability is found in a cryptosystem, and subsequently disclosed, developers will replace the system with one without known, serious vulnerabilities. This effectively removes the system from the pool of acceptable systems. See the current phase-out of 3DES[7] as an example.

[4] As there is no proof that polynomial time solutions to hard problems do not exist, we can not exclude the possibility that such solutions are known, but being kept secret.

large numbers is a very hard problem for classical[5] computers, and RSA, simply put, takes advantage of this fact. If we were to find a way to efficiently factor large numbers, we could easily break the security of RSA. However, as no such procedure is currently known, we operate under the assumption that RSA is secure.

[5]Quantum computers can factor more efficiently than classical computers using Shor's Algorithm[8]. Currently, quantum computer are not big enough to tackle numbers at the size of those used in RSA.

Chapter 4

Information Retrieval

Before we start exploring the subject of Private Information Retrieval, it seems reasonable to give a definition of what an ordinary Information Retrieval protocol may look like. We will therefore in this chapter give a formal definition of an ordinary Information Retrieval protocol. Please note that someone defining information retrieval outside the context of PIR may end up with a definition quite different from what we will end up with. The reason for this, is that the definitions given in this chapter are meant to easily transform over to those of private information retrieval, as we will see in the next chapter.

The goal of an information retrieval protocol is, unsurprisingly, to retrieve information. While this is a very general goal, we will particularly look at retrieving information from a database stored at some remote server.

4.1 Sequences

An Information Retrieval protocol can be said to consist of four sequences, namely *Setup*, *Query*, *Respond*, and *Decode*. The Setup is performed once to initialize the system, and the other sequences are performed on every subsequent query.

4.1.1 Setup

The setup sequence is executed once in order to initialize the information retrieval system. As input, the setup sequence takes the number of servers $\ell \geq 1$ and the database X. The setup will output $(\Gamma; X^{(1)}, ..., X^{(\ell)})$, where Γ is the system parameters, and each $X^{(i)}$ is an internal state, often referred to as a *bucket*. These buckets will often contain the same data,

namely $X = X^{(1)} = X^{(2)} = ... = X^{(\ell)}$. Γ will be used to describe how the system is set up, and will appear as an implicit input to the other sequences. Γ may contain the number of servers ℓ, IP-addresses of said servers, and may contain cryptographic parameters such as *Common Reference Strings*. Once the setup is completed, the Information Retrieval system will be ready to receive requests in the form of queries.

4.1.2 Query

The query sequence takes as input a security parameter λ[1] and one or more record identifiers J[1]. The query sequence will output a sequence of query strings $q = (q_1, ..., q_\ell)$ where each query string q_i becomes the query for server i, and some private auxiliary information δ. Record identifiers will usually specify positions within X where the data we are interested in is located. The private auxiliary information λ may for example contain an ephemeral private decryption key for an established public key crypto-system.

4.1.3 Respond

The respond sequence will be executed for every i-th server, and takes as input its internal state $X^{(i)}$ and its part of the query component $q_i \in \{q_1, \ldots, q_\ell\}$. Every i-th server will output a response string r_i to be returned to the client.

4.1.4 Decode

The decode sequence takes as input the clients auxiliary information Δ and the ℓ response strings $r_1, \ldots, r_\ell$. The decode sequence outputs the record(s) requested by the client.

4.2 Properties

Below I will list some properties of information retrieval protocols. These properties are not necessarily found in all information retrieval protocols, but all are worth mentioning as they will give some context as we transition from regular information retrieval protocols to private information retrieval protocols.

[1]Note that this is one of the cases were our definition of an IR protocol is shaped to easily transform to that of a PIR protocol. A normal IR protocol may support searches, while this definition assumes the querier is already aware of the record identifier (index) of the record in which he is interested.

4.2.1 Correctness

Correctness, also refereed to as *completeness*, is a property that states that when a client requests records from a server which is not malfunctioning nor malicious, the server correctly returns these records to the client without error. In other words, when the client queries in the expected way, and the server responds in the expected way, the client will receive the correct response for his query.

Formally

$$\Pr\left[X \leftarrow DECODE(\Delta; r_1, \ldots, r_\ell) \;\middle|\; \begin{array}{c}(\Delta; q_1, \ldots, q_\ell) \leftarrow QUERY(1^\lambda; J) \\ \wedge(\Lambda_{j=1}^{\ell} r_j \leftarrow RESPOND(X^{(j)}; q_j))\end{array}\right] \geq 1 - \epsilon(\lambda) \quad (4.1)$$

Equation (4.1) showing the correctness property, where $X^{(j)}$ is the bucket(s) containing requested record(s), and J is the requested record identifier(s).

4.2.2 Robustness

We can image a scenario in where some of the ℓ servers may produce an incorrect response, or no response at all, either by failure, or by malicious intent, resulting in the failure of the decoding sequence. Robustness is a property that allows some servers to respond incorrectly, or not at all, while still preserving the correctness property. An Information Retrieval protocol can therefore be said to be robust if a client can successfully retrieve records even though some servers respond incorrectly or not at all.

Formally

$$\Pr\left[X \leftarrow DECODE(\Delta; r_1, \ldots, r_\ell) \;\middle|\; \begin{array}{c}(\Delta; q_1, \ldots, q_\ell) \leftarrow QUERY(1^\lambda; J) \\ \wedge(\Lambda_{j=1}^{\ell} r_j \leftarrow RESPOND(X^{(j)}; q_j)) \\ \wedge\{r_j | j \in C\} \leftarrow A(C; \{q_j\}_{j \in C})\end{array}\right] \geq 1 - \epsilon(\lambda) \quad (4.2)$$

Equation (4.2) showing the robustness property, where X is the requested record(s), J is the requested record identifier(s) and A is an algorithm which chooses a subset of the servers and their responses to pass to the decoder.

We will in this paper refer to protocols that allow k out of ℓ servers to not respond, while still being able to decode correctly as (k, ℓ)-correct. We will refer to protocols that allow v out of ℓ servers to respond incorrectly, while still being able to decode correctly as v-byzantine-robust. A protocol that allows some servers to not respond, and some servers to respond incorrectly, may be classified as a *v-Byzantine-robust k-out-of-ℓ information retrieval protocol.*

4.2.3 Non-triviality

As mentioned earlier, a trivial Information Retrieval protocol is not very interesting because of the overhead associated with downloading entire databases to only access a subset of the records. We will therefore be exclusively looking at non-trivial Private Information Retrieval protocols.

We may define an information retrieval protocol to be non-trivial if

$$\begin{gathered} \forall n \in \mathbb{N}, (\Gamma; X^{(1)}, ..., X^{(\ell)}) \leftarrow SETUP(X; 1, ..., \ell) \\ \text{s.t.} \\ |\Gamma| + Expected[|Q| + |R|] \in o(|X|) \end{gathered} \tag{4.3}$$

Equation (4.3) showing a definition of non-triviality, where Γ is the public parameters, Q are the query strings, R are the response strings, $X^{(i)}$ are the buckets and $|X|$ is the size of the database.

To explain further, equation (4.3) simply states that the total bandwidth used to fetch an element from the database should scale with little o of the size of the database, ensuring that we do not reach triviality.

Note that there are many different ways to define the non-triviality property. Some papers will measure communication costs using the average-case, while some will use the worst-case. Some definitions will account for bidirectional communication, while some will only consider download costs. Equation (4.3) encapsulates a rather reasonable way to define the non-triviality property. In addition, this definition should be applicable to most known private information retrieval protocols.

Chapter 5

The first PIR protocol

We will in this chapter explore the first ever PIR protocol, published by Chor et al. in 1995[4]. The protocol regards a database X as an n-bit binary string, and a record as a single bit contained in this string. We will show how the protocol achieves private information retrieval, and reason about how the protocol is information-theoretically secure, as well as discuss the communication costs. We will also explain step-wise improvements from a non-private, inefficient protocol to a private, efficient protocol.

5.1 One-dimensional Non-private IR

We will begin by look at an example of a **non-private** information retrieval protocol, where the querier wishes to retrieve a single bit from a database consisting of n bits. The example will differ in appearance from any ordinary information retrieval protocol. The reasons for this is so that we can add privacy at a later point, having the underlying protocol still recognizable. For a very approachable example, we will use a database consisting of $n = 4$ bits. If say, the user was interested in retrieving bit b_2, the user could send a query consisting of a standard basis vector of size n, containing a *1* in the position of b_2. The server can simply calculate the dot product of the standard basis vector and the database, yielding the singular bit that the user wanted.

$$\langle\ 0\ 0\ 1\ 0\ \rangle \cdot \langle\ b_0\ b_1\ b_2\ b_3\ \rangle = b_2 \tag{5.1}$$

Equation (5.1) showing how a singly bit may be retrieved from the database by computing the dot product of a standard basis vector and the database.

As previously stated, this query is non-private, and the reason for this should be obvious. By sending the query string to the server, the server can immediately interpret

that we are looking for b_2. In order to make this query private, we are going to replicate the database to $\ell > 1$ servers[1] and *secret share* the query using *Additive Secret Sharing in GF(2).*

5.2 Additive Secret Sharing in GF(2)

Additive secret sharing in GF(2) is rooted in the understanding of finite fields. However, we may for simplicity explain it as shortly as to say that GF(2) is a field with two elements $\{0, 1\}$, and the additive operation for elements within this field is simply the XOR operation.

Now, suppose we have some secret S that we wish to share with a group consisting of, for example, $\ell = 5$ members. We wish to have it so that each member has one *share* of the secret, and no entity may know the secret unless all shares are obtained. To achieve this, we view our secret as a member of the field $GF(2)$, i.e. $S \in \{0, 1\}$.[2] We generate shares for each of the members as follows:

We assign r_1 to r_4 as random values in $\mathbb{F}$, and compute r_5 to be the value so that the sum of all shares becomes S.

$$\begin{aligned} & & r_1 &\in_R \mathbb{F} \\ & & + r_2 &\in_R \mathbb{F} \\ & & + r_3 &\in_R \mathbb{F} \\ S \in \mathbb{F} & & + r_4 &\in_R \mathbb{F} \\ & & + r_5 &:= S - r_1 - r_2 - r_3 - r_4 \in_R \mathbb{F} \\ & & &= S \end{aligned} \tag{5.2}$$

Equation (5.2) showing how additive secret sharing in GF(2) may be achieved, with five resulting shares.

We now have five shares resulting from this operation. Suppose we distribute the five shares to five shareholders, such that each shareholder holds one of the shares. It should be quite obvious to see that even if the four shareholders holding r_1, r_2, r_3 and r_4 should choose to collude, they cannot determine S, as they all just hold random values. In fact, any four shareholders could choose to collude, but they will still only hold random values without the last share. It is only when someone holds all 5 shares that the secret S may be recovered.

[1] It is possible to achieve PIR using a single server setup. However, such a setup must be based on computational security and thus falls into the category of CPIR protocols. We will for the time being look exclusively at IT-PIR protocols, which require $\ell > 1$ servers.

[2] The secret can be any length n by repeating the construction n times, or by generating n-length bit-strings for each share.

We pointed out earlier that we may use CSPRNGs as a source of randomness to achieve information-theoretic security, given that the output of the CSPRNG is not determinable to the adversary, even given unbounded computational resources. If we use a CSPRNG as the source of randomness for the secret sharing, the output of the CSPRNG is only determinable to an adversary if he can obtain all shares, and by that point, any privacy resulting from secret sharing is already void. Thus, we are able to claim that this construct allows information-theoretic security within our privacy threshold, which in this case is $\ell - 1 = 4$.

Now that we have a way to share a secret, we will see how can we use secret sharing to make the query in section 5.1 private.

5.3 One-dimensional ℓ-server IT-PIR

To make our non-private query in section 5.1 private, we will use equation (5.2) to construct secret shares of our query vector $\vec{q}$. We will again be using a database consisting of four bits for our example, and we will for clarity build on the example previously established, where we are interested in bit b_2, and construct our standard basis vector accordingly.

$$\vec{q} = \langle\ 0\ 0\ 1\ 0\ \rangle, \quad X = \langle \square\square\square\square \rangle \tag{5.3}$$

Equation (5.3) showing a standard basis vector and a database with placeholder values.

Applying equation (5.2) we can generate $\ell - 1$ random vectors of length $|X|$, and a pseudorandom ℓ-th vector, also of length $|X|$, which together with the $\ell - 1$ random vectors sum to $\vec{q}$. These ℓ shares of the original query vector $\vec{q}$ may be referred to as *query-components* or *query shares*.

$$\begin{aligned} \vec{q_1} &\in_R GF(2)^n \\ \vec{q_2} &\in_R GF(2)^n \\ &\vdots \\ \vec{q_{\ell-1}} &\in_R GF(2)^n \\ \vec{q_\ell} &:= (\vec{q} - \vec{q_1} - \cdots - \vec{q_{\ell-1}}) \end{aligned} \tag{5.4}$$

Equation (5.4) showing additive secret sharing of the query vector $\vec{q}$ to obtain ℓ query-components.

If we follow the reasoning in section 5.2 on additive secret sharing and distribute the query-components to the ℓ servers as $\vec{q_1} \to S_1$, $\vec{q_2} \to S_2, \ldots, \vec{q_\ell} \to S_\ell$, no server may

learn our original query $\vec{q}$ without all servers colluding. We may refer to the assumption that servers are not colluding, as the *non-collusion assumption*. We will explore how to account for some degree of collusion in chapter 6 on robustness.

When a server S_i receives a query-component $\vec{q_i}$, the server will compute the dot product of the query-component $\vec{q_i}$ and its replica of the database X, and return the computed value, which is a single bit. Once every server has responded, the client will XOR the returned values to obtain the value of the bit specified by the original query vector $\vec{q}$.

$$(\vec{q_1} \cdot X) \oplus \cdots \oplus (\vec{q_\ell} \cdot X) = b_j \tag{5.5}$$

Equation (5.5) showing how XORing the results of each query-components dot product with the database yields the bit we were looking for.

Great! We have achieved private information retrieval, but there are some immediate issues with this protocol, first and foremost, the communication costs.

5.3.1 Communication Costs

If we consider the communication costs of the protocol so far, we note that each server only responds with a single bit. This gives us a total download cost of ℓ bits, which is optimally efficient. However, for each query-component we send to the server, we upload $|X|$ bits. This means the total upload cost is $\ell \cdot |X|$.

One could safely argue that downloading the entire database once, will almost always be better than uploading queries the same size as the database, many ℓ times. Because the total communication costs of this protocol exceeds the total communication cost of a trivial protocol, the non-triviality property is not satisfied. The protocol does however nicely explain the underlying concept of how a PIR-protocol could work, and it contains some, but not all of the properties we are looking for in a PIR protocol.

5.3.2 Properties

Satisfied properties:

- ℓ-correct
  The protocol will return the correct result if all ℓ servers respond as expected.
- $(\ell - 1)$ private
  Privacy is preserved via secret sharing for up to $\ell - 1$ colluding servers.

Unsatisfied properties:

- ℓ-private
  If all ℓ servers collude, our query is no longer private.
- $(\ell - 1)$-correct
  If a single server does not respond, we cannot decode to the requested record.
- 0-byzantine robust
  If more than zero servers responds incorrectly, we cannot decode to the requested record.
- Non-triviality
  Upload + download costs exceed the size of the database.

5.4 Vector-Matrix IR

In order to achieve a non-trivial PIR protocol, Chor et al. symmetrized the ℓ-server IT-PIR protocol in section 5.3. By symmetrizing, we mean lowering the total upload and download costs by utilizing some symmetrical structure[3]. This symmetrical structure may be obtained by introducing a new dimension to the database.

To explain how we may perform such a symmetrization of the protocol in 5.3, we will first look at a **non-private** example of such a construct. In order to obtain this symmetrical structure, we are going to go from thinking of the database as a one dimensional bit-string to start thinking about the database as a $m \times m$ matrix[4].

[3]Symmetrization may also be interpreted as equalizing some related metrics, such as upload and download costs.

[4]In the literature, the dimensions of the matrix is often written as $r \times s$, but we will for the time being stay with $m \times m$.

5.4.1 The Database as a matrix

If we begin with our database as a n-bit binary string, we may split the string into chunks of $m \approx \sqrt{n}$ bits, and consider these chunks as columns in a $m \times m$ matrix.

$$m = 5$$
$$|X| \approx m^2 \approx 25$$
$$X = \langle \square\square\square\square\square|\square\square\square\square\square|\square\square\square\square\square|\square\square\square\square\square|\square\square\square\square\square \rangle$$
$$\Downarrow$$
$$X = \begin{bmatrix} \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \end{bmatrix}$$

Figure 5.1: Showing how we may view an n-bit database as an $m \times m$ matrix.

5.4.2 Querying the matrix

To query the matrix X for a record j, we can construct an m-bit query vector $\vec{e_j}$ that queries each column of the matrix for the $(j \ mod \ m)$-th bit, effectively returning the entire j-th row of X as the response.

$$\vec{e_j} \cdot X = \vec{r}$$
$$\langle\ 0\ 0\ 0\ 1\ 0\ \rangle \cdot \begin{bmatrix} \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \end{bmatrix} = \langle\ \square\ \square\ \square\ \square\ \square\ \rangle \tag{5.6}$$

Equation (5.6) showing how we may extract a m-bit row from the database, using a vector of size m.

If we transpose the matrix, the operation becomes visually more intuitive.

$$\langle\, 0\,0\,0\,1\,0 \,\rangle \cdot X^T = \langle\, 0\,0\,0\,1\,0 \,\rangle \cdot \begin{bmatrix} \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square \end{bmatrix} = \langle\, \square\,\square\,\square\,\square\,\square \,\rangle \tag{5.7}$$

Equation (5.7) showing how we may view the database as transposed to more intuitively see how the response is computed.

We see that the product of our query vector and the database yields the entire j-th row of the database. In the retrieved row, only one bit is the bit we were looking for, while the $m - 1$ other bits are useless bits we can discard.

Another way to view this construct is that it allows us to fetch a complete database row, which may at first seem very useful, as traditional databases typically use rows to represent complete records. However, to achieve optimal communication costs, we must insist that each row/record and column has an equal length $r = s = \sqrt{|X|}$ [9]. Thus, this construct may deter us from supporting variable-length content such as media, as this would require padding to ensure all records are of equal length (the length of the biggest record), adding overhead. Additionally, this implication makes it non-optimal to add or remove rows to an established database of size $|X|$, as optimal costs are dependent upon the size of the records being the same size as the number of rows, given by $r \cdot s = |X|$.

5.4.3 Communication Costs

As we we are uploading a standard basis vector of size m, and downloading a row of size m, upload and download costs are now symmetrical. Note that this is a non-private single-server example.

$$\text{Upload} = \text{Download} = m = \sqrt{|X|}$$

$$Total\ Cost = 2m = 2\sqrt{|X|}$$

5.4.4 The Extra Bits

It is important to note that queries in this symmetrized construct returns data that the user did not ask for, namely the $m - 1$ extra bits. We are currently operating under the assumption that the entire database is publicly accessible. However, if this is not the case, any private protocol based on this non-private protocol may reveal information that the querier is not meant to learn.

Symmetric PIR

On the note of revealing information that the querier is not meant to learn, this seems like a fitting point to introduce the concept of Symmetric PIR (SPIR), also known as *oblivious transfer*.

An oblivious transfer protocol is a protocol that has privacy-guarantees for both the querier and the server. This means that the privacy for the querier must be preserved, as is the case in PIR, and additionally that the privacy of the server must be preserved. In practice, this means that the server must never reveal information that is not a subset of the information the querier should be able to learn.

This paper will not discuss SPIR protocols further than stating that they do exist[10].

5.5 Introducing Privacy to Vector-Matrix IR

Now that we have a more cost-efficient, although non-private protocol, we would like to introduce privacy. Like in section 5.3, we can take our query vector $\vec{e_j}$ and somehow securely share it between servers. We have a few different options on how to do this, and the options yield different branches of PIR, security-wise.

5.5.1 CPIR and Additively Homomorphic Encryption

If we were to encrypt $\vec{e_j}$ component-wise using some IND-CPA secure, additively homomorphic encryption scheme, the security of our query would rely on the computational hardness assumption. See the section on Computational Security in 3.2. This yields what we refer to as **CPIR**, namely Computationally Private Information Retrieval.

While CPIR is a reasonable approach, we will in this thesis look exclusively at IT-PIR protocols. This is partly because IT-PIR protocols are orders of magnitude faster than any known CPIR protocol[11][12], and partly to limit the scope of this thesis.

5.5.2 IT-PIR and Secret Sharing

As in section 5.3, we can choose to secret share our query and distribute the queries across the ℓ servers, thus protecting the privacy of our query. As we construct our query components as ℓ seemingly random unit vectors, there is no way to positively ascertain

our original query, even given unlimited computing resources, unless privacy is already broken by means of collusion. This yields Information-Theoretic security, as described in 3.1.

5.5.3 Other variants

There exists other PIR-variants in the literature, such as Hybrid-Security PIR[13] and Trusted-Hardware based PIR[14]. However, most PIR-schemes in the literature use either IT-PIR or CPIR[15].

5.6 2D 4-server IT-PIR

We will now look at an example of a functional vector-matrix IT-PIR protocol using secret sharing to distribute the query components. In addition to simply explaining the protocol, we will look at how we may reduce the costs even further than what we saw was possible in 5.4.3.

5.6.1 2D Queries

Let us again view the database as a square matrix, such that the length and width of the matrix is $m = \sqrt{|X|}$.

Recall that a standard basis vector of length m will return a single row of the database. If we were to consider that row as another database, we could query for a column element contained in the row using another unit vector, also of length m. Unsurprisingly, any 2D matrix element may be uniquely indexed by an index pair $(j_1, j_2) \in [1..m]^2$, where j_1 is the row index, and j_2 is the column index, and furthermore, any such element (j_1, j_2) may be queried using two standard basis vectors of length m, namely the row query $\vec{e_{j_1}}$ and the column query $\vec{e_{j_2}}$.

Note however, that any server learning $\vec{e_{j_1}}$ may know the row of the element we are retrieving, and any server learning $\vec{e_{j_2}}$ may know the column of the element we are retrieving. Any server learning both $\vec{e_{j_1}}$ and $\vec{e_{j_2}}$ will be able to learn the identity of the exact element (j_1, j_2) we were looking for, breaking privacy.

As stated in 5.5, we are going to focus exclusively on IT-PIR and thus securely distribute $\vec{e_{j_1}}$ and $\vec{e_{j_2}}$ using secret sharing.

5.6.2 Secret sharing 2D queries

Suppose we want to construct a private query for a single matrix element (j_1, j_2), represented by row query $\vec{e_{j_1}}$ and column query $\vec{e_{j_2}}$. To achieve this, we generate one random row query share[5] $\vec{q_0}^{(0)}$, and construct another row query share $\vec{q_0}^{(1)}$ such that the XOR of the two row query shares is the original row query $\vec{e_{j_1}}$. We repeat this for the column query, yielding two column query shares $\vec{q_1}^{(0)}$ and $\vec{q_1}^{(1)}$.

Formally

$$\begin{aligned} \text{Choose } \vec{q_0}^{(0)}, \vec{q_1}^{(0)} &\in GF(2)^m \\ \text{Set } \vec{q_0}^{(1)} &:= \vec{q_0}^{(0)} \oplus \vec{e_{j_1}} \\ \text{Set } \vec{q_1}^{(1)} &:= \vec{q_1}^{(0)} \oplus \vec{e_{j_2}} \end{aligned} \tag{5.8}$$

Equation (5.8) showing how we may secret share two a 2D query using additive secret sharing.

5.6.3 Distributing secret shared 2D subqueries

We will in this example show how we can securely distribute the four query shares $\vec{q_0}^{(0)}$, $\vec{q_0}^{(1)}$, $\vec{q_1}^{(0)}$ and $\vec{q_1}^{(1)}$. This example uses four servers, but we will later see how we may use more servers.

Let us first label the four servers involved in the protocol as 00, 01, 10 and 11. The query shares are distributed to the servers, so that servers 00 and 01 are both getting the same first share of the row request, namely $\vec{q_0}^{(0)}$, represented by the first digit in the server's labels. The other two servers, 10 and 11, are getting the other share of the row request, namely $\vec{q_0}^{(1)}$. The same practice also applies to the column query shares, where here the second digit of the server label refers to the share of the column query the servers receive. Generally, the first digit of the server labels corresponds to the row query, and which of the row query share that server should receive. The second digit corresponds to the column query, and which of the column query shares that server should receive.

To learn which element the querier was looking for, one would need to obtain both shares of the row query $\left(\vec{q_0}^{(0)}, \vec{q_0}^{(1)}\right)$, and both shares of the column query $\left(\vec{q_1}^{(0)}, \vec{q_1}^{(1)}\right)$. As long as we do not send both shares of either the column or the row query to a single server, no information may be learnt without multiple servers colluding.

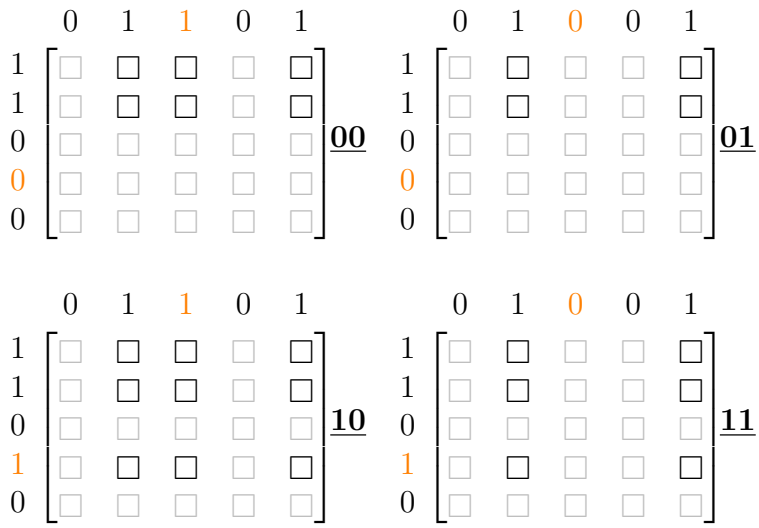
[5]These random query shares will be constructed such that each bit has a value of *0* or *1* with probability 50%.

We will look at a graphical example of distributing and decoding such a private query. Suppose we are looking for an element in **row four**, **column three**. Queries that capture this element may be generated as the following equation set.

$$\begin{aligned} \vec{q_0}^{(0)} &= \langle 1\ 1\ 0\ 0\ 0 \rangle \quad \vec{q_1}^{(0)} = \langle 0\ 1\ 1\ 0\ 1 \rangle \\ \vec{q_0}^{(1)} &= \langle 1\ 1\ 0\ 1\ 0 \rangle \quad \vec{q_1}^{(1)} = \langle 0\ 1\ 0\ 0\ 1 \rangle \end{aligned} \tag{5.9}$$

Equation (5.9) showing shares of row and column queries describing the position of an element in a matrix.

To each server, we may now send the two shares corresponding to the label of that server. Upon receiving the shares, the server can disregard all elements in either a zero-row or a zero-column, according to the shares it received. In figure 5.2, we see these elements as grayed out. To compute the result, each server may XOR the "surviving" bits. Each server will return that single-bit result to the querier.



(5.10)

Figure 5.2: Visualizing how secret shared row and column queries may be super-positioned over the data before XORing the surviving bits and returning the result.

Visually XORing the matrices in figure 5.2 will yield a single surviving bit in row four, column three. Thus, upon receiving all four response-bits (representing the result of each servers XOR operation of the two shares with the database), the user can simply XOR these four bits to obtain the value of the element in row four, column three, similarly as to the application of equation (5.5).

5.6.4 Communication Costs of the 2D 4-server IT-PIR

As we are uploading two shares of size $m = \sqrt{|X|}$ per server, and downloading a single bit per server, we may express the cost of this 2D 4-server protocol by the following table.

Total Upload:	$4 \cdot 2\sqrt{\lvert X\rvert}$	$= 8\sqrt{\lvert X\rvert}$	bits
Total Download:	$4 \cdot 1$	$= 4$	bits

Table 5.1: Communication costs of the 2D 4-server IT-PIR protocol.

5.6.5 Applying further symmetrization

We may notice that the upload and download costs in table (5.1) resembles that of the protocol in section 5.3 prior to symmetrization. In fact, we may apply symmetrization to this construct as well, following the logic given in 5.4 by introducing a new, third dimension.

Let us again consider the database as a n-bit binary string, where n is now a cube number with a base m. We split the database into chunks of m^2 bits, and split those chunks again into m bit subchunks. This allows us to consider our database as a three-dimensional, cubic matrix.

$$m = 3$$
$$|X| = m^3 = 27$$
$$X = \langle \square \rangle$$

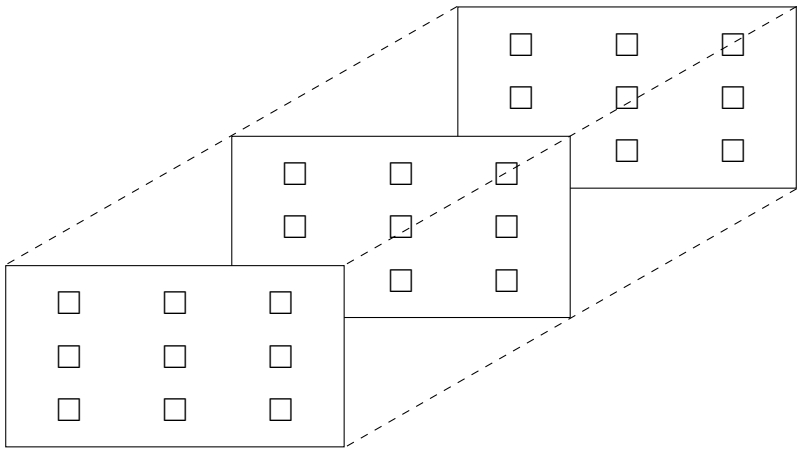


Figure 5.3: Visualizing the database in three dimensions.

Using the equivalent logic from section 5.4.2, we may now query the database using two unit vectors of size $\sqrt[3]{|X|}$, which would return a vector of size $\sqrt[3]{|X|}$

Formally, this yields the costs for an $\ell = 4$ setup as given by the following table.

Total Upload:	$4 \cdot 2\sqrt[3]{\lvert X\rvert}$	$= 8\sqrt[3]{\lvert X\rvert}$	bits
Total Download:		$= 4\sqrt[3]{\lvert X\rvert}$	bits

Table 5.2: Communication costs of the 3D 4-server IT-PIR protocol.

Now that we have applied the symmetrization trick twice, and both times reduced costs, we may wonder what is to stop us from applying the same trick again. We will now look at a generalization of this practice, where we can keep reducing costs by adding more dimensions and servers.

5.7 ND 2^N-server IT-PIR

By viewing our database as an increasingly higher-dimensional hypercube, we can continue to decrease our costs, as long as we have enough servers[15].

Because of the way the distribution of the subqueries works, (explained in section 5.6.3), we need the number of servers ℓ to be a power of two. Thus, in order to project the database into a N-dimensional hypercube, we would need $\ell = 2^N$ servers.

Projecting the database into a N-dimensional hypercube across 2^N servers yields the costs, pre-symmetrization, as given by the following table.

Total upload:	$2^N \cdot N \cdot \sqrt[N]{\lvert X\rvert}$	bits	Upload per server:	$N \cdot \sqrt[N]{\lvert X\rvert}$	bits
Total download:	2^N	bits	Download per server:	1	bits

Table 5.3: Upload and download costs when projecting the database as an N-dimensional hypercube, pre-symmetrization.

Applying the symmetrization trick yet again, we can obtain costs of fetching a record of size $\sqrt{|X|}$ as given by the following table.

Total upload:	$2^N \cdot (N-1) \cdot \sqrt[N]{\lvert X\rvert}$	bits	Upload per server:	$(N-1) \cdot \sqrt[N]{\lvert X\rvert}$	bits
Total download:	$2^N \cdot \sqrt[N]{\lvert X\rvert}$	bits	Download per server:	$\sqrt[N]{\lvert X\rvert}$	bits

Table 5.4: Upload and download costs when projecting the database as an N-dimensional hypercube, post-symmetrization.

Chapter 6

Robustness

As we saw in chapter 5, Chor's 1995 protocol satisfies neither the (k, ℓ)-correctness property nor the v-byzantine robustness property. This means that if one or more of the ℓ servers were to not respond, or respond incorrectly, we can not decode to a correct result. This is a problem that needs to be addressed before we can consider PIR for deployment in any practical environment. We will in this chapter look at two solutions to this problem. One of the solutions will provide the k-out-of-ℓ correctness property, and the other solution will provide both the k-out-of-ℓ correctness property and the v-byzantine robustness property.

6.1 BS'02 - (k, ℓ)-correctness

Beimel and Stahl released in 2002 their paper demonstrating a method of introducing the (k, ℓ)-correctness property, such that only k-out-of-ℓ servers are needed to respond in order to decode to the correct result[16]. We will now look at how to achieve the (k, ℓ)-correctness property, and we will begin by a single case of such a construct, namely the 2-out-of-4 case.

6.1.1 2-out-of-4-correct PIR

Suppose we have $\ell = 4$ servers, labeled 00, 01, 10 and 11. We would like to have a 2-out-of-4 PIR protocol, such that only two of these four servers need to respond in order to successfully return the record the user requested.

In order to accomplish this, we are going to run the query generation algorithm twice

for the same record j, creating two different, but equivalent, queries q_1, q_2.

Formally

$$\left((q_0^{(1)}, q_1^{(1)}), (q_0^{(2)}, q_1^{(2)})\right) \leftarrow QUERY(1^\lambda; j) \times QUERY(1^\lambda; j) \tag{6.1}$$

Equation (6.1) showing redundant query generation for a 2-out-of-4 correct PIR query.

Each of these queries will be secret-shared to two shares each, such that we now hold four shares $(q_0^{(1)}, q_1^{(1)})$ and $(q_0^{(2)}, q_1^{(2)})$. To server 00, we send the first share of both queries. To server 01, we send the first share of the first query and the second share of the second query, etc. As in the previous chapter, we see that each server's label refers to which two shares of the two queries the server is sent.

As we only need the responses to a complete set of shares, e.g. $(q_0^{(1)}, q_1^{(1)})$ or $(q_0^{(2)}, q_1^{(2)})$, in order to successfully decode to a correct answer, any two servers responding will always yield at least one such complete query q_1 or q_2. By extension, this also means that the privacy-threshold of this scheme may be given as $t = 2$, meaning that as soon at two servers collude, we no longer have privacy.

If the communication costs of performing a 4-out-of-4 PIR query is given by X, we see that the cost of performing an equivalent 2-out-of-4 query may be given as $2X$.

6.1.2 2-out-of-ℓ PIR

Of course, being limited to exactly four servers is not ideal. Fortunately, we can use the generalization of the 2-out-of-4 case to make any multi-server PIR protocol into a 2-out-of-ℓ protocol. To do this, we create $\log_2 \ell$ query share sets.

$$\left((q_0^{(1)}, q_1^{(1)}), \ldots, (q_0^{(\log_2 \ell)}, q_1^{(\log_2 \ell)})\right) \leftarrow QUERY(1^\lambda; j) \times \cdots \times QUERY(1^\lambda; j) \tag{6.2}$$

Equation (6.2) showing redundant query generation for a 2-out-of-ℓ correct PIR query.

For each server, we express the server as a binary string $i = b_i^{(1)} b_i^{(2)} \ldots b_i^{(\log_2 \ell)}$, and send the shares as $q_{b_i}^{(1)}, q_{b_i}^{(2)}, \ldots, q_{b_i}^{(\log_2 \ell)}$. The shares are distributed in the equivalent manner as in the 2-out-of-4 case, and given that any two servers respond, we can always successfully decode to the correct answer.

Additionally, given that the cost of perform a single query is given by X, it follows that the cost of performing any 2-out-of-ℓ query is $X \cdot \lceil \log_2 \ell \rceil$

6.2 Goldberg '07 - k-out-of-ℓ v-Byzantine Robustness

Ian Goldberg released In 2007 a paper titled "Improving the robustness of private information retrieval"[17]. The goal of this improved robustness was to allow for both servers not responding at all, and servers responding incorrectly, while still maintaining the ability to decode to the correct answer. Such robustness may also be referred to as k-out-of-ℓ v-byzantine robustness, where v refers to the number of incorrect responses we can tolerate while still being able to decode to the correct answer.

Let us consider the database as an $r \times s$ matrix over $\mathbb{F}$, where a record is a single row. A non-private query may be constructed as vector-matrix product of a standard basis vector and the database.

$$\langle 0\, 0 \ldots 1 \ldots 0 \rangle \cdot X = \vec{X_j} \tag{6.3}$$

Equation (6.3) showing how we may retrieve a record/row from a two-dimensional database by using a standard basis vector as our query.

In order to make queries such as the one in equation (6.3) private and robust, we are going to secret share the query using *Shamir's secret sharing*.

6.2.1 Shamir's Secret Sharing

In 1979, Shamir proposed a way of sharing a secret between multiple parties in such a way that a subset of the parties working collaboratively may recover the secret[18].

We will begin by looking at a single case of such a construct, namely the 2-out-of-3 case. We will then look at expanding this case to facilitate a greater number of shares required in order to recover the secret. We will then look at the general case, which may be applied to any setup of shareholders and privacy thresholds.

2-out-of-ℓ secret sharing

Suppose we want to generate a secret consisting of 3 shares, such that 2 shares is sufficient in order to recover the secret, i.e. a 2-out-of-3 secret sharing scheme. To achieve such a construct, we can generate a random slope on a graph and sample 3 random points on this slope, (x_1, y_1),(x_2, y_2) and (x_3, y_3). Recall that any 2 distinct points on a continuous slope may identify the slope's function as $f(x) = ax + b$ using interpolation. If we define the secret as the function's evaluation at $x = 0$, namely the y-intercept $y = f(0)$, any

two valid shares are now sufficient in order to recover this secret by interpolation to the function, and evaluation the function at $f(0)$.

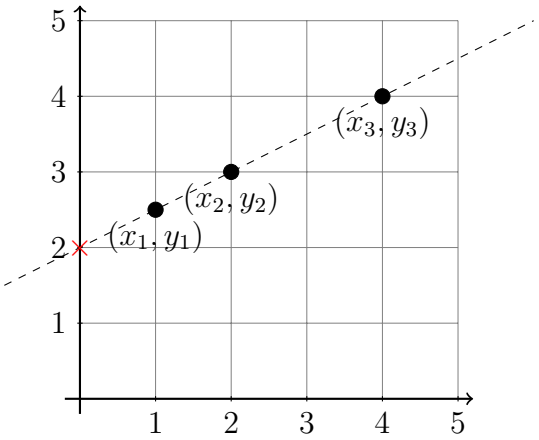


Figure 6.1: Visualizing a 2-out-of-3 Shamir's secret sharing scheme, with the secret equaling 2.

We may also secret share an existing secret S by placing the the secret on the graph as a point at $(0, S)$, adding a new random point x_0, and interpolating between these points, yielding a secret function we can now use to generate shares.

Note that a single shareholder may not learn the secret without another share, as attempting to interpolate from a single point to a linear function yields infinite possible functions and infinite possible results for evaluating those functions at $f(0)$. This retains the information-theoretical security-property using the same argument as in section 5.2 on additive secret sharing.

Note also that in order to share this secret with more than three shareholders, we can just measure new points on the slope, and send the new points as shares to the new shareholders. We can repeat this process to share this secret with privacy-threshold $t = 2$ to any number of shareholders ℓ.

If we wish to share a secret with a privacy-threshold greater than two, we can simply increase the degree of the polynomial.

4-out-of-ℓ secret sharing

In order to achieve a scheme where four out of some variable number of shareholders need to collaborate in order to recover the secret, we can simply adjust the degree of

the polynomial from what we saw in the previous example. In order to accommodate for 4 shares being required, we will need a cubic polynomial. A general rule is that a polynomial of degree t can be used to share a secret such that the privacy-threshold is $t + 1$.

We again sample ℓ points, in this example, $\ell = 5$, and distribute these to the shareholders.

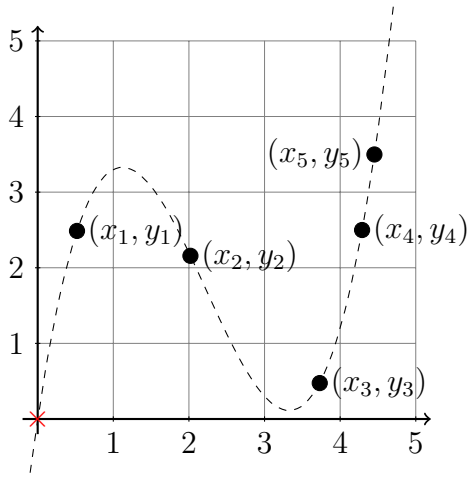


Figure 6.2: A 4-out-of-5 Shamir's secret sharing scheme, with the secret equaling 0.

Using interpolation, any four shareholders may now recover the function f by interpolation, and subsequently may recover the secret by evaluating the function at $f(0)$. To increase the number of shareholders, we can again simply measure and distribute new points.

Shamir's $(t + 1, \ell)$-secret sharing

In general, we can describe Shamir's $(t + 1, \ell)$-secret sharing as a way to share a secret S among ℓ shareholders, such that any subset of $k \geq t + 1$ shareholders can recover S, and no subset of $k \leq t$ shareholders can learn anything about S.

To achieve this, we choose a random function $f \in \mathbb{F}$, such that $f(0) = S$ and the degree of $f \leq t$. Each shareholder i receives the share $(x_i, f(x_i))$[1]. Any $k \leq t + 1$ shareholders

[1]Shareholder i may be given the share of any point on the function, not necessarily the point with x-coordinate i.

may now compute $f(0)$ from polynomial interpolation.

$$f(0) = \sum_{i=1}^{k} f(x_i) \cdot \prod_{j=1, j\neq i}^{k} x_j/(x_j - x_i)^{-1} \tag{6.4}$$

Equation (6.4) shows how $f(0)$ may be computed from the application of Lagrange interpolation, given at least t shares.

In addition to this, we make the observation that all operations involved in Shamir's secret sharing are linear.

$$\begin{aligned} f(0) &= a \\ h(0) &= b \\ (f+h)(0) &= a + b \end{aligned} \tag{6.5}$$

Equation (6.5) shows the linear, homomorphic property under the additive operation.

$$g(0) = a \Rightarrow c \cdot g(0) = c \cdot a \tag{6.6}$$

Equation (6.6) shows the linear, homomorphic property under scalar multiplication.

As the operations involved in Shamir's secret sharing are all linear, and the vector-matrix products involved in PIR are also linear, we may use secret sharing to share PIR queries. Additionally, because of this linearity, we can interpolate through each server's result of the vector-matrix product of its secret share and the database, and evaluate to $x = 0$ to obtain the record we were looking for.

6.2.2 Detecting Bad Shares in Shamir's Secret Sharing

Before applying Shamir's secret to the context of PIR, we first need to consider bad shares. A shareholder supplying a bad share may do so maliciously as a denial-of-service attack. If we thoughtlessly applied Shamir's secret sharing to PIR, we may still be susceptible to malicious servers, and we will miss out on our desired v-byzantine robustness property. In short, to obtain the v-byzantine robustness property, we have to be able to differentiate legitimate shares from illegitimate ones.

On a linear polynomial, given that we have more shares than we need, and most of the shares are legitimate, we can immediately tell by intuition which shares are bad. Below are examples of a 2-out-of-ℓ scheme with a bad share, and a 4-out-of-ℓ scheme with a bad share. We can imagine, as the number of bad shares increase, and the degree of the polynomial increases, that it becomes more difficult to differentiate the legitimate shares from the illegitimate shares.

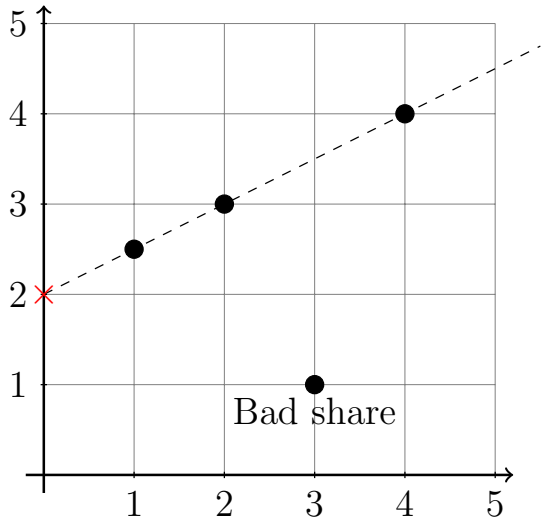


Figure 6.3: Showing the placement of a bad share on a linear polynomial.

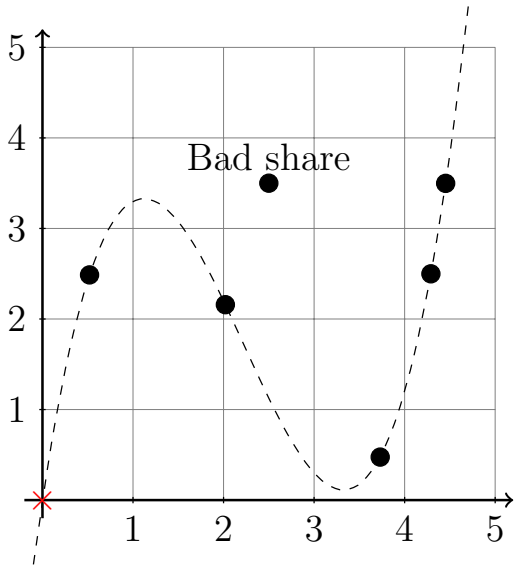


Figure 6.4: Showing the placement of a bad share on a quadratic polynomial.

Fortunately, efficient algorithms to tackle this problem exists[15]. I will not explain any such algorithms in depth, but I will say that they allow us to efficiently determine what shares, if any, are bad. Using such algorithms, we can detect illegitimate shares as long as we have enough legitimate shares. Given a sufficient amount of remaining legitimate shares, we will still be able to decode to the correct result. One such notable algorithm originates from Reed and Solomon's work on finite fields, which was originally designed for Reed-Solomon Error-correcting codes and published in 1960[19]. Other algorithms include Verifiable Secret Sharing algorithms, introduced by Chor et al. in 1985[20].

The fact that we may distinguish bad shares from good shares allows us to disregard the bad shares, and given that enough good shares remain, we may interpolate and arrive at the correct answer. This ability provides us with the v-byzantine robustness property.

6.2.3 Distributing Secret-Shared Queries

Now that we have a way of secret-sharing queries with robustness, we will look at an example of how we may apply this in practice. To reiterate, the idea is to take our query vector $\vec{e_i}$ and share it component-wise (per section 6.2.1) to obtain shares $(Shamir_1(\vec{e_i}), Shamir_2(\vec{e_i}), \ldots, Shamir_\ell(\vec{e_i}))$.

We pass one shares to each server, and each server computes the vector-matrix product of its share and the database, and returns the result. Upon receiving enough legitimate

responses, we may perform the interpolation and decode to the correct result.

$$X_i = \sum Shamir(\vec{e}_i) \cdot \prod x_j/(X_j - X_k)^{-1} \tag{6.7}$$

Equation (6.7) showing the interpolation from the response shares to the record.

6.2.4 Summarizing Goldberg '07

Now that we have a basic understanding of Goldberg's robust IT-PIR, we may attempt to formalize according to the template given in chapter 4 on general information retrieval.

Sequences

$$SETUP(X; 1, \ldots, \ell) \to ((r, s, \mathbb{F}); X, \ldots, X) \tag{6.8}$$

Equation (6.8) showing the setup sequence with corresponding output, where r is the number of rows, s is the number of columns and the buckets are simply copies of the database X.

$$QUERY(1^\lambda; j) \to (\Delta = \perp, (\vec{q_1}, \vec{q_2}, \ldots, \vec{q_\ell})) \tag{6.9}$$

Equation (6.8) shows the query sequence, where λ gives the security threshold, and there is no auxiliary information. The components are the result of $(\lambda + 1, \ell)$ component-wise secret sharing of $\vec{e}$ over $\mathbb{F}$.

$$RESPOND(X^{(i)}; \vec{q_j}) \to (\vec{r_i} := \vec{q_j} \cdot X^{(i)}) \tag{6.10}$$

Equation (6.10) shows the respond sequence, where each server returns the matrix-vector product of the database and their share of the query-vector.

$$DECODE(\Delta; \vec{r}_1, \ldots, \vec{r}_\ell) \to \left(\vec{X_j} = \sum_{i=1}^{\ell} \vec{r_i} \cdot \prod_{k=1, k\neq i}^{\ell} k/(k-1)^{-1} \right) \tag{6.11}$$

Equation (6.11) shows the decoding of responses by interpolation to obtain the correct result.

Properties

If we set $\ell > 1$ and choose a threshold $t \in [0 \dots \ell{-}1]$, the Goldberg '07 scheme is k-correct for any $k > t$ and perfectly $(t+1)$-private. Additionally, if $r + s \in o(rs)$, then it is non-trivial[17]. Moreover, the scheme is optimally v-robust for any $v < k - \sqrt{kt}$, using Guruswami-Sadams algorithm[21].

6.2.5 Further improvements to Goldberg '07 IT-PIR

Goldberg's robustness is provably optimal, in the regard that it is impossible to decode a single round of query-responses if there are more bad results than given by $v < k - \sqrt{kt}$. However, Devet, Goldberg and Heninger showed in 2012 that we may indeed improve this metric by repeating failed queries to gain information as to which servers are acting maliciously[22]. As repeated queries fail, we gain more and more information as to which servers are the culprits. Devet et al. showed that we may eventually decode any result, given that there are enough points on the correct polynomial, that is, $t + 1$ points. This holds even if every other participating server is malicious. This method uses unique decoding multi-polynomial codes[22].

This gives the improved byzantine robustness as:

$$v < k - t - 1 \tag{6.12}$$

Equation (6.12) showing optimal robustness using unique decoding multi-polynomial codes.

Chapter 7

Amortized PIR

From previous chapters, we have looked at retrieving either a single bit or a single s-length row from the database. In this chapter, we will look at how Amortized PIR allows us to construct queries that may return multiple rows more efficiently than repeated single-block queries. We will explore two general methods of achieving this, *multi-block queries*, and *batch codes*. We will in this chapter use the term *block* interchangeably with the term *row*.

7.1 Simple Multi-block Queries

Recall that by sending the server a standard basis vector, and having the server calculate the matrix-vector product of the standard basis vector and the database, this will return a row. If we wanted to fetch multiple rows, we could repeat this process y times to fetch y different rows. However, there is a simple, more efficient way to achieve the same result.

If instead of sending y standard basis vectors, we were to send a matrix Q of standard basis vectors, we may think of each row of the matrix as its own query. The server can then perform a matrix-matrix multiplication and retrieve multiple rows. It turns out that performing the $Q \cdot X$ matrix multiplication is faster than performing y vector-matrix multiplications[23], using efficient algorithms such as Strassen's algorithm or other equivalents.

While this simple construct provides a significant speedup over repeated single-block queries, there are even more efficient methods for performing multi-block queries available.

7.2 HHG '13 - Multi-block IT-PIR

Henry, Huang and Goldberg released in 2013 a paper on efficient Multi-block queries[9] using Ramp Sharing Schemes. To understand how this may be achieved, we first need to understand ramp sharing schemes. Note that the HHG'13 method is applicable only to PIR protocols using secret sharing.

7.2.1 Ramp Sharing Schemes

Ramp Sharing Schemes began as a field of study after the discovery of Shamir's Secret Sharing. The operating assumption of ramp sharing schemes is that while Shamir's secret sharing scheme may not reveal any information about the secret unless the privacy threshold t is met, a ramp sharing scheme may reveal *some partial information* if a coalition of $t \leq k < t + q$ servers collude. In return, we may encode more secret bits per share using ramp sharing schemes, namely q bits.

Ramp secret gives absolute privacy if $k \leq t$ shares are obtained, no privacy if $k > t+q$ shares are obtained, and something in between if $t \leq k < t + q$ shares are obtained[15]. We will in section 7.2.3 on v-Byzantine robustness look at an approach to dealing with the $t \leq k < t + 1$ case.

We will look at two different options of how we may use ramp sharing schemes for encoding additional bits. Both options will encode q secrets in a polynomial of degree $(t + q - 1)$.

Coefficients as the secrets

If we recall Shamir's secret sharing scheme, we remember that we may define the secret as the constant of a polynomial function.

$$f(x) = ax^2 + bx + c \tag{7.1}$$

Equation (7.1) shows an exponential function that may be used for Shamir's secret sharing, with the secret equaling $f(0) = c$.

If we want to attempt to use coefficients as the secrets, we can try to introduce another secret to the function in equation (7.1) as b. However, we should first recall that Shamir's secret sharing has the property of being homomorphic with respect to both addition and multiplication(fully homomorphic). With this in mind, we will observe that if we were to

perform some operation where we multiply two shares resulting from the function in (7.1), the constant term c behaves nicely, but the linear term b does not multiply appropriately.

$$(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd \tag{7.2}$$

Equation (7.2) showing the non-homomorphic property with regards to multiplication and the linear term b.

This non-homomorphism with regards to multiplication suggests we may run into difficulties when calculating products of shares. As this is something we need to be able to do in order to interpolate and recover secrets, we will rather consider another option.

Evaluations as the secrets

This other option is using function evaluations at the secrets. In short, we want the evaluation of $f(0)$ to be the first secret, and the evaluation of $f(1)$ to be the second secret, etc. To achieve this, we can randomly generate a point $s_0 = (x_0, y_0)$ to get our first point, and $s_1 = (x_1, y_1)$ to get our second point. Once we have our $q = 2$ points, we can simply interpolate to a polynomial of degree $d = t + q - 1$ capturing these points. We can then proceed to generate shares by sampling new points on this polynomial.

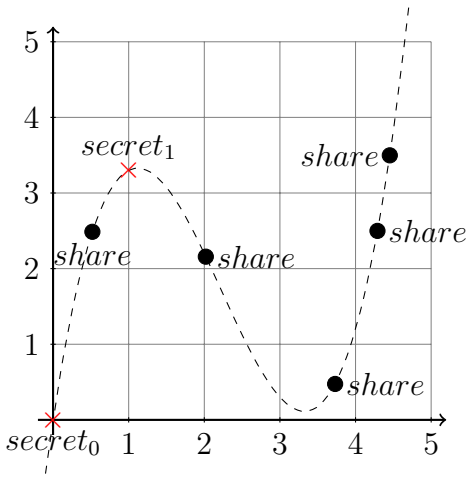


Figure 7.1: Showing how shares may be generated from $q = 2$ secret points as evaluations of their interpolated polynomial of degree 4, yielding a the privacy-threshold as $t = 3$

We observe that this method shows the homomorphic property with regards to multiplication. I.e., when we multiply two polynomials together, every point multiplies as well.

$$\forall i, (f \cdot g)(i) = f(i) \cdot g(i) \tag{7.3}$$

Equation (7.3) showing the homomorphic property with regards to multiplication.

This property allows us to use evaluations-as-the-secrets ramp secret sharing for PIR.

7.2.2 Column-wise Ramp-scheme Sharing

Now that we have a way to encode more secret bits in our shares by using ramp secret sharing, we will see how we may use this to achieve efficient multi-block IT-PIR requests.

To construct such an efficient query, let us first consider the q number of query vectors as a matrix.

$$Q = \begin{pmatrix} \vec{e_{j_1}} \\ \vec{e_{j_2}} \\ \vdots \\ \vec{e_{j_q}} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{7.4}$$

Equation (7.4) showing a vector of queries as a matrix.

Contrary to the theme of chapter 5, instead of component-wise secret sharing the queries, we are going to ramp-scheme share the matrix column-wise.

$$Q = \begin{pmatrix} \vec{e_{j_1}} \\ \vec{e_{j_2}} \\ \vdots \\ \vec{e_{j_q}} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} \tag{7.5}$$

$$\vec{F}(x) = \langle\ f_1(x),\quad f_2(x),\quad f_3(x),\quad f_4(x),\quad \dots,\quad f_r(x)\ \rangle$$

Equation set (7.5) shows the column-wise ramp-scheme sharing of the query matrix.

To ramp-scheme share the matrix column-wise, we take each column of the matrix, and compute a ramp scheme that encodes that column, yielding a polynomial. Doing this for every column yields a vector $\vec{F}(x)$ of polynomials. Note that $|\vec{F}(x)| = \sqrt{|X|}$.

Now, for every server i, we evaluate the vector component-wise at some x-coordinate x_i[1], yielding a vector of shares $\vec{F(x_i)}$ for each server. Each vector $\vec{F}(x_i)$ is sent to its intended server i, upon which, server i calculates the vector-matrix product of the database

[1]Server i does not necessarily compute the evaluation for x-coordinate i.

and the vector, and returns the result. The client, upon receiving the responses, may interpolate to 0 to retrieve the first block, at 1 to retrieve the second block, etc. In this fashion, we may fetch q blocks at a time. Note that the since the size of a multi-block query and a single-block query are the same, and both queries appear random to the servers, the servers can not differentiate between single-block queries and multi-block queries.

7.2.3 On v-Byzantine Robustness and Privacy

Given the $t \leq k < t + q$ case, we will not have the same level of privacy as we had in Goldberg's '07 IT-PIR protocol, given the same security parameters. In other words, to achieve these efficient multi-block queries while without changing security parameters, we must sacrifice some v-byzantine robustness[15]. However, Henry et al. writes in their 2013 paper[9] about the possibility of using a game-theoretic approach to assist in keeping malicious servers out of the server pool. The idea goes something like this:

If we issue multi-block queries, we are vulnerable to byzantine servers. However, if we issue single-block queries, the queries have optimal robustness. Servers cannot differentiate between single-block and multi-block queries. If we usually issue multi-block queries, and occasionally issue single-block queries, the single-block queries may act as checks to see that the servers are behaving honestly. If a single-block query with optimal robustness fails, we will be able to identify the servers responsible, and kick them out of the server pool if we deem them malicious. Given that the byzantine servers have some incentive to be part of the pool, they must act like all requests are single-block, optimally robust queries.

7.2.4 Summary

$$((r, s, \mathbb{F}); X, \ldots, X) \leftarrow SETUP(X; 1, \ldots, \ell)$$

Equation (7.2.4) shows the setup sequence with corresponding output, where r is the number of blocks, s is the number of columns and the buckets are simply copies of X.

$$(\Delta = \perp, (\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_\ell)) \leftarrow QUERY(1^\lambda; j_1, \ldots, j_q) \tag{7.6}$$

Equation (7.6) shows the query sequence, where λ gives the privacy threshold, and there is no auxiliary information. The query shares are the result of $(\lambda + 1, q, \ell)$-ramp secret sharing of $\vec{e}$ over $\mathbb{F}$.

$$(\vec{r}_j := \vec{q}_i \cdot X^{(i)}) \leftarrow RESPOND(X^{(i)}, \vec{q}_i) \tag{7.7}$$

Equation (7.7) shows the respond sequence, where each server responds with the matrix-vector product of the database and the query vector.

$$\vec{X}_{j_h} = \sum_{i=1}^{\ell} \vec{r}_i \cdot \prod_{k=1,k\neq i}^{\ell} (k-h+1)/(k-1)^{-1} \leftarrow DECODE(\Delta; \vec{r}_1, \ldots, \vec{r}_\ell) \tag{7.8}$$

Equation (7.8) shows the decoding of responses by interpolation to obtain multiple q records.

7.2.5 Properties

Given $\ell > 1$ and $t, q \in [0, \ldots, \ell - 1]$, the scheme is k-correct for any $k > t+q$ and perfectly $(t+1)$-private.

Additionally, given that $r + s \in o(rs)$, it is also non-trivial. Moreover, it is v-robust for any $v \leq k - t - q + 1$.

7.2.6 Cost Comparison with Goldberg'07

We may compare the costs of this protocol with the Goldberg'07 protocol. We will look at the costs of fetching q blocks. The costs are given per server as the following table.

	Goldberg '07	HHG '13	
Upload:	$q \cdot r$	r	$\mathbb{F}$ elements
Download:	$q \cdot s$	s	$\mathbb{F}$ elements
Computation:	$2 \cdot q \cdot r \cdot s$	$2 \cdot r \cdot s$	$\mathbb{F}$ elements
Storage:	$r \cdot s$	$r \cdot s$	$\mathbb{F}$ elements

Table 7.1: Costs of the Henry et al. multi-block IT-PIR protocol, compared to Goldberg'07.

7.3 Batch Codes

Ishai, Kushilevitz, Ostrvsky and Sahai introduced batch codes in their 2004 paper titled "Batch Codes and Their Application"[24]. Batch codes encode the database into buckets, and will allow us to fetch records from multiple buckets with some higher performing

metric than performing the repeated vector-matrix product. While the HHG'13 protocol is applicable only to secret sharing based PIR protocols, batch codes are generic constructs that may be applied to any PIR-protocol.

Definition 7.3.1. A (n, N, q, m, t)-batch code over $\mathbb{F}$ encodes a database $X \in \mathbb{F}^{r \times s}$ into an m-tuple of buckets in $\mathbb{F}^{\lceil r/m \rceil \times s}$ in such a way that a client can obtain any subset of q blocks from X by querying each bucket at most t times.[15]

One of the batch codes introduced by the 2004 paper is what we refer to as a *Subcube Code*. We will in this section look at applying a 2-way subcube code, a 3-way subcube code and a recursive subcube code.

7.3.1 2-way Subcube code

Consider that we have a database X of size $|X| = 60$ field elements, represented as a matrix. We split the database into two partitions, A and B.

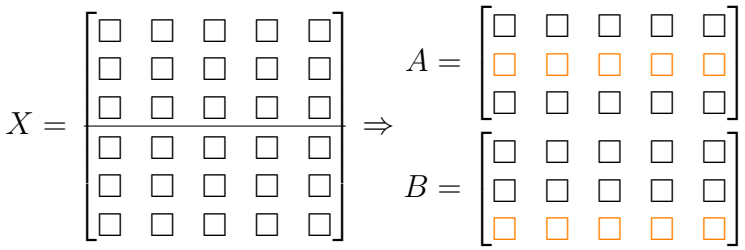


Figure 7.2: Partitioning the database into two buckets.

Suppose we wanted to request block $\vec{x}_2$ and $\vec{x}_6$ from X. If we were to send a query as $\vec{q} = \langle 010001 \rangle$, the query would be the same size as a single-record query, but the server can now return two blocks by splitting the query and super-positioning it over the partitions.

$$\vec{q} = \langle 010001 \rangle \cdot X \Rightarrow \begin{matrix} \vec{q_0} = \langle 010 \rangle \cdot A = \vec{x}_2 \\ \vec{q_1} = \langle 001 \rangle \cdot B = \vec{x}_6 \end{matrix} \tag{7.9}$$

Equation (7.9) showing how an m-bit query may be interpreted as two separate queries. Also showing how two blocks can be returned, given that they reside in different buckets.

We see that by splitting the database into two partitions, or buckets, we may retrieve two blocks by sending a single query, given that the blocks being requested reside in different buckets. This simple example nicely explains the concept behind subcube codes.

The costs of fetching two blocks, before and after introducing the 2-way subcube code, may be given by the following table.

Upload:	$2 \times 6 = 12$	$\mathbb{F}$ elements	$\Rightarrow$	Upload:	$2 \times 3 = 6$	$\mathbb{F}$ elements
Comp.:	$2 \times 60 = 120$	$\mathbb{F}$ elements	$\Rightarrow$	Comp.:	$2 \times 30 = 60$	$\mathbb{F}$ elements
Download:	$2 \times 5 = 10$	$\mathbb{F}$ elements	$\Rightarrow$	Download:	$2 \times 5 = 10$	$\mathbb{F}$ elements

Table 7.2: The cost of fetching two non-arbitrary blocks, before and after applying the 2-way subcube code with two buckets.

Notice from table 7.2 how the upload and computational costs are reduced by a factor of two. While we have achieved a nice performance increase, this approach does not work for all pairs of records, as there is no way to fetch two records from bucket A, or two records from bucket B. To support arbitrary two-block queries, we can introduce a third bucket, being the XOR of the two other buckets.

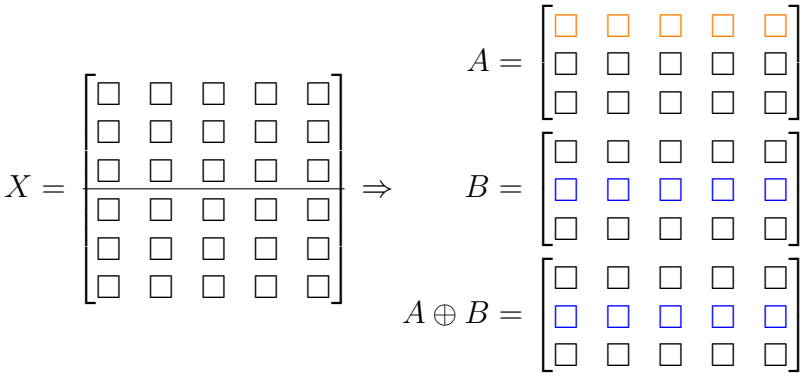


Figure 7.3: Partitioning the database into two buckets plus an additional bucket as the XOR of the other buckets.

Now, if we want to fetch two records from bucket A, for example $\vec{x}_1$ and $\vec{x}_2$, we may construct the query-vector as:

$$\begin{aligned} \vec{q}_0 &= \langle 100 \rangle \cdot A = \vec{x}_1 \\ \vec{q}_1 &= (\langle 010 \rangle \cdot B) \oplus (\langle 010 \rangle \cdot A \oplus B) = \vec{x}_2 \\ \vec{q} &= (\vec{q}_0, \vec{q}_1) \end{aligned} \tag{7.10}$$

Equation (7.10) showing how we may construct a query which can retrieve two arbitrary records.

We see that by splitting the database into two buckets A and B, and by creating a third bucket as the XOR of A and B, we may now retrieve any two arbitrary records by

sending a single query. The costs of fetching two blocks, before an after introducing the subcube-code with two plus one buckets, may be given by the following table.

Upload:	$2 \times 6 = 12$	$\mathbb{F}$ elements	$\Rightarrow$	Upload:	$3 \times 3 = 9$	$\mathbb{F}$ elements
Comp.:	$2 \times 60 = 120$	$\mathbb{F}$ elements	$\Rightarrow$	Comp.:	$3 \times 30 = 90$	$\mathbb{F}$ elements
Download:	$2 \times 5 = 10$	$\mathbb{F}$ elements	$\Rightarrow$	Download:	$6 \times 5 = 30$	$\mathbb{F}$ elements

Table 7.3: The cost of fetching two arbitrary blocks, before and after applying the 2-way subcube code with two plus one buckets.

As we see from table (7.3), we can reliably fetch two records with one query at a lower upload and computational cost than performing two single-block queries. However, the download cost has tripled. If the upload or computational cost is the bottleneck of a PIR scheme, this method might prove useful at the expense of higher download costs.

7.3.2 3-way Subcube code

If we build on the previous example, and now construct a subcube code with three buckets plus an additional bucket being the XOR of the three buckets, we may again fetch an arbitrary block pair. This construct lowers our storage and computational costs, at the expense of an increased download cost.

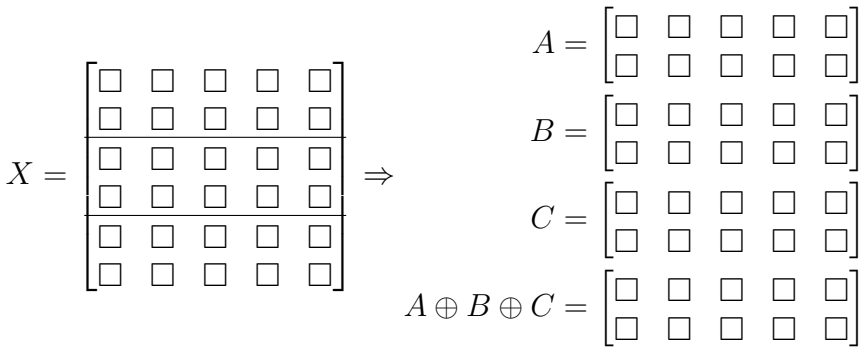


Figure 7.4: How we may partition the database to obtain lower storage and computational costs than we saw was the case for the 2-way subcube code.

We note that the download cost for three blocks has increased by a factor of 4 compared to two single-block queries, while the computational and upload-costs have only increased by a factor of $\frac{4}{3}$. If fact, we can continue to create more and more fragmented n-way subcube codes, and this relationship will continue linearly.

However, there exists a way to obtain better metrics by applying the idea of subcube codes recursively.

7.3.3 Recursive Subcube Codes

To achieve a recursive subcube code, we merely repeat the process we performed for the original database X, to each of the buckets A, B and $A \oplus B$ resulting from that subcube code.

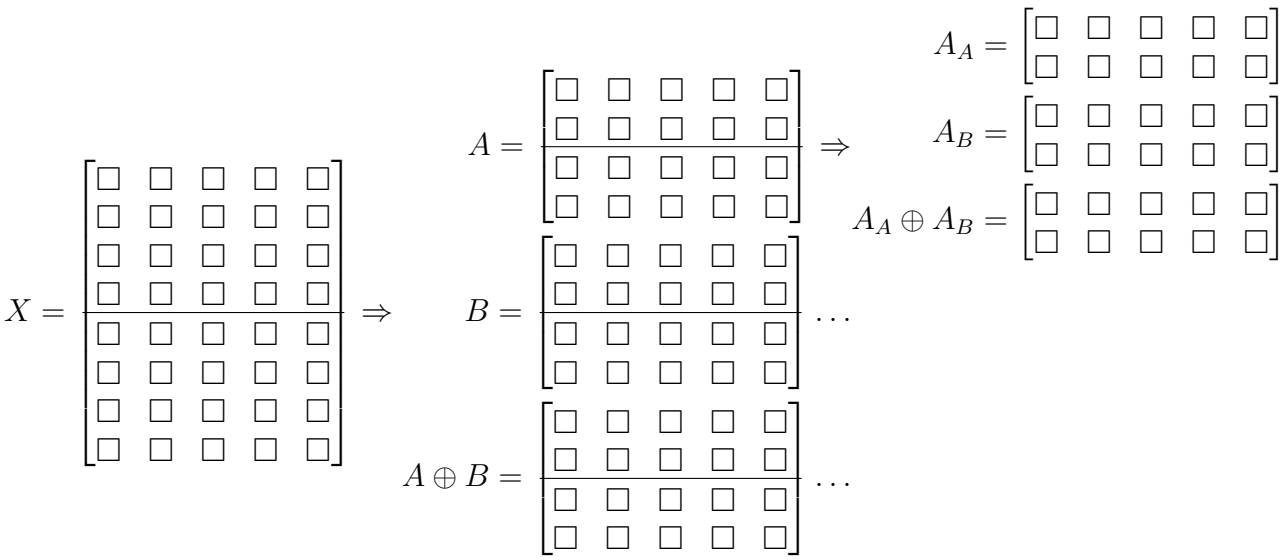


Figure 7.5: Showing how we may apply the idea of a subcube-code recursively.

By creating a recursive subcube code of recursion level k, we may fetch 2^{k+1} blocks per query, and the cost the query may be expressed by the following table.

Download:	$3^{k+1}s$	$\mathbb{F}$ elements
Upload:	$1.5r$	$\mathbb{F}$ elements
Comp.:	$1.5^{k+1}rs$	$\mathbb{F}$ operations
Storage:	$1.5^{k+1}rs$	$\mathbb{F}$ operations

Table 7.4: The costs associated with a k-level recursive subcube code.

Note that all servers must hold all buckets for subcube codes to be function.

7.3.4 The U-ary code

While schemes involving subcube codes require all servers to hold copies of all buckets, the U-ary code, proposed by Ryan Henry in 2016[25], introduces batch-codes where servers hold one unique bucket each. We will first look at how to setup in order to be able to retrieve a single block, then we will move on to see how we may fetch multiple blocks.

Single-block Retrieval - Setup

In order to set up for the U-ary code, we are going to begin by relabeling the database. We will go from considering the database as a matrix consisting of r rows and s columns to thinking of it as $\lceil r/u \rceil$ sub-matrices, each of which consisting of u rows and s columns.

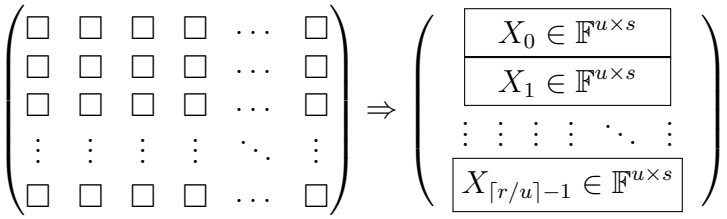


Figure 7.6: How we may view a single, large matrix as multiple, stacked matrices.

We will encode these stacked matrices using a ramp sharing scheme with a privacy threshold of zero. To do this, we encode each of the sub-matrices X_i column-wise, using interpolation to find the unique length-s vector of degree $(u-1)$ polynomials $\vec{d_i} = \langle d_{i0}(x), d_{i1}(x), \ldots, d_{i(s-1)}(x) \rangle$ such that $d_{ik}(h)$ is equal to the component in position (h,k) of X_i for every $k = 0, \ldots, s-1$ and every $h = 0, \ldots, u-1$.[25]

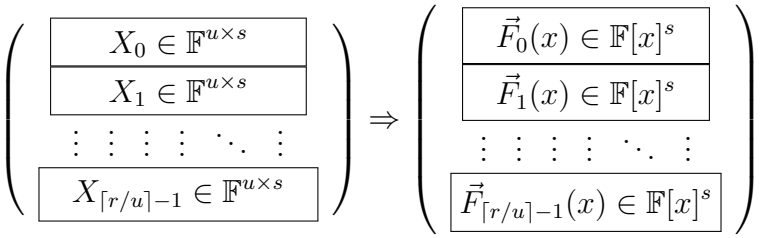


Figure 7.7: How we may go from a vector of sub-matrices to a matrix of polynomials

Evaluating the first polynomial of sub-matrix $\vec{F}_0$ at 0, we will obtain the record in the

first column, first row of the sub-matrix X_0. Evaluating the same polynomial at 1, yields the record in the first column, second row of the sub-matrix X_0, up to an evaluation of $u-1$, where the first column, last row of the sub-matrix may be retrieved. Evaluating the second polynomial of sub-matrix $\vec{F}_0$ at 0, yields the second column, first row of the sub-matrix, etc.

In order to distribute the buckets to the servers, we may now evaluate the matrix component-wise at different values $x \in [0, \ldots, \ell-1]$, and give the results to the ℓ servers as their bucket.

$$\begin{pmatrix} \vec{F}_0(x_0) \\ \vec{F}_1(x_0) \\ \vdots \\ \vec{F}_{r/u-1}(x_0) \end{pmatrix} \in \mathbb{F}^{\lceil r/u \rceil \times s}, \quad \begin{pmatrix} \vec{F}_0(x_1) \\ \vec{F}_1(x_1) \\ \vdots \\ \vec{F}_{r/u-1}(x_1) \end{pmatrix} \in \mathbb{F}^{\lceil r/u \rceil \times s}, \quad \ldots, \quad \begin{pmatrix} \vec{F}_0(x_{\ell-1}) \\ \vec{F}_1(x_{\ell-1}) \\ \vdots \\ \vec{F}_{r/u-1}(x_{\ell-1}) \end{pmatrix} \in \mathbb{F}^{\lceil r/u \rceil \times s}$$

Figure 7.8: How each server's bucket is created though column-wise ramp sharing of the data.

Single-block Retrieval

In order to query servers holding such a bucket for a record i, we first need to rewrite the index i. We use the division algorithm to find the quotient and the remainder from dividing i by u. The quotient is going to tell us which row of the polynomial matrix encodes the record we are looking for, and the remainder will tell us which polynomial of that row we must evaluate to retrieve the record we are looking for.

$$i = u \cdot i_q + i_r \quad (0 \leq i_r < 0) \tag{7.11}$$

Equation (7.11) showing how we may rewrite the index to be able to query the buckets.

Once we know in what row and what column the polynomial representing our record resides, we will secret share the standard basis vector having the 1 in the quotient position, and encode it, rather than at $x = 0$ as we typically do, at $x = i_r$.

$$SecretShare(\vec{e}_{i_q} \in \mathbb{F}^{1 \times \lceil r/u \rceil}) \quad \text{at} \quad x = i_r \tag{7.12}$$

Equation (7.12) showing how we construct the query to obtain a record from the distributed buckets.

We send one share of the query to each server. After we are returned the dot-product of each query share and bucket, we may interpolate the response to i_r, yielding the record we were looking for[25].

The cost of fetching a single record, per server, may be expressed as the following table:

	Goldberg'07	Henry'16	
Upload:	r	$\lceil r/u \rceil$	$\mathbb{F}$ elements
Download:	s	s	$\mathbb{F}$ elements
Comp:	$2 \cdot r \cdot s$	$2 \cdot \lceil r/u \rceil \cdot s$	$\mathbb{F}$ operations
Storage:	$r \cdot s$	$\lceil r/u \rceil \cdot s$	$\mathbb{F}$ operations

Table 7.5: The costs of fetching a single block, per server. Henry'16 u-ary codes compared to Goldberg'07.

We see that by using the u-ary code, we may reduce costs by a factor of u. However, this code may currently not be viewed as a batch code, as we are not fetching multiple records. To fetch multiple records, we will not look at the multi-block retrieval version of this construct.

Multi-block Retrieval

To arrive at how we may fetch multiple records, let us first remember how we fetched the j-th block from the k-th layer, where k is the x-coordinate we evaluated to. We recall that this was done by computing the standard basis vector at $x = k$. If we wish to also fetch the j'-th block from the k-th layer, we are unable to do so, as we cannot encode two different standard basis vectors, e.g. $sbv(j)$ and $sbv(j')$ at the same x-coordinate. On the other hand, if we want to query from a different layer, we can achieve this. If we evaluate the polynomial at k', we get another standard basis vector. We may use this observation to recover both blocks. However, we note that a single query may fetch at most one block from each layer, which is the same position we were in when we first started looking at subcube codes.

In order to fetch multiple, arbitrary blocks, regardless of layer, we can simply put every block in its own layer. Rather than encoding the x-coordinate, if we want the first row of the database, we may encode at $x = 0$, and the second row at $x = 1$, etc. Now, instead of dividing i by u to find the row in the polynomial matrix we want, we may encode at the x-coordinate corresponding to the actual row-index.

There is however a downside to this technique – The field we need to be able to construct polynomials of sufficient degrees must be larger than what we are used to. Usually, when we do Shamir's secret sharing-based PIR, we operate in rather small fields, for example $GF(2^8)$, as small fields provide fast computations. Now, depending upon the size of the database, we will need a bigger field.

$$|\mathbb{F}| \geq r + \ell \tag{7.13}$$

Equation (7.13) showing the required field size in order to encode every block in its own layer[15].

Now that we have a way of fetching q arbitrary blocks at a time, we may compare the costs per server to that of Goldberg'07.

	Goldberg'07	Henry'16	
Upload:	$q \cdot r$	$\lceil r/u \rceil$	$\mathbb{F}$ elements
Download:	$q \cdot s$	s	$\mathbb{F}$ elements
Comp:	$2 \cdot q \cdot r \cdot s$	$2 \cdot \lceil r/u \rceil \cdot s$	$\mathbb{F}$ operations
Storage:	$r \cdot s$	$\lceil r/u \rceil \cdot s$	$\mathbb{F}$ operations

Table 7.6: The costs of retrieval q block, per server. Comparing Henry'16 to Goldberg'07

We see from table (7.6) that by using Henry's u-ary multi-block retrieval, we may reduce our costs by a factor of u.

However, we will with this u-ary code meet the same implications to v-Byzantine robustness as we did with the HHG 13-paper[9]. Fortunately, we may also apply the game-theoretic perspective from section 7.2.3, which states that it may be feasible to achieve equivalent robustness to that achieved in section 6.2 from applying the game-theoretic strategy of sending single-block queries as honesty-checks.

Chapter 8

Expressive Queries

All the PIR-protocols we have gone through so far assumes that the querier somehow knows the index i of the record he wants to retrieve. However, if we think about how we normally access digital information, we may notice a sharp contrast to this assumption. For example, we rarely open our bookmark-folder in our web-browser and look for the bookmark at position $i = 42$. Rather, we are used to searching for a resource. For PIR to be useful in such scenarios, it seems like we need some way to perform more expressive and realistic queries. We will in this chapter look at some different methods for achieving expressive queries.

8.1 PIR by keywords

One way to achieve PIR by keywords is given by the Chor et al. 1997 paper titled "Private information retrieval by keywords"[26]. The paper shows how we may use a B+ tree, allowing the querier to recursively query the tree using PIR for the purpose of arriving at either the relevant data, or the index of the relevant data.

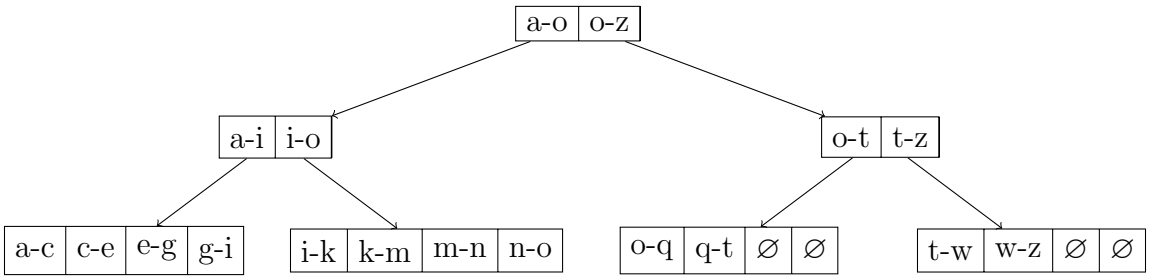


Figure 8.1: A B+ tree allowing us to query by keyword.

For example, let us say a querier is looking to query by the keyword "p" and the queryable structure is given by the B+ tree in figure 8.1. In order to perform a query by keyword over this tree, the querier will first download the root-node $\boxed{\text{a-o} \mid \text{o-z}}$. Trivial PIR may be used to download the root node, as we are querying by level, and this level only has one root, so there is no secret to protect yet. Upon having downloaded the root node, we can tell by the contents of the root node that in order to find the keyword "p", we must query the next level of the tree for the index contained in the right option of the root node, the $\boxed{\text{o-z}}$-index. We may now query the second level of the tree with PIR using this index to obtain the contents of the $\boxed{\text{o-t} \mid \text{t-z}}$ node. Upon having downloaded the contents of the $\boxed{\text{o-t} \mid \text{t-z}}$ node, we can now tell that to find the next relevant child, we may query the next level of the tree using the index contained in the left option of the $\boxed{\text{o-t} \mid \text{t-z}}$ node, the $\boxed{\text{o-t}}$-index. This yields the $\boxed{\text{o-q} \mid \text{q-t} \mid \varnothing \mid \varnothing}$ leaf node. The $\boxed{\text{o-q} \mid \text{q-t} \mid \varnothing \mid \varnothing}$ node, being a leaf node, may in the $\boxed{\text{o-q}}$ option either contain the data we were looking for, or indices to where we may find that data. Note that by constructing a larger tree, we may express more detailed keywords than simply "p".

If we look at the cost of performing a query-by-keyword of a B+ tree structure, we can start by observing that the cost of querying the bottom layer is the same as for a normal PIR query of the whole keyword database. We may give this cost of querying the complete keyword database as C. Now, if all but the bottom layer are structured as a binary tree, the size of the $j-1$-th layer is generally half the size of the j-th layer. Thus we may write the sum of costs of a complete query traversal as the following equation.

$$C + \frac{1}{2}C + \frac{1}{4}C + \frac{1}{8}C + \frac{1}{16}C + \cdots \approx 2C \tag{8.1}$$

Equation (8.1) giving the cost of a complete query traversal as $2C$.

8.2 SQL-type Queries

We will not explore any papers on SQL-type queries, but it is worth mentioning that such systems do exist[27][28], some of them using the PIR-by-keywords primitive given in 8.1[15].

8.3 Distributed Point Functions

Distributed Point Functions were introduced in 2014 by Gilboa and Ishai[29], and among other things, allow us to achieve a very compact additive secret sharing of a query vector

$\vec{e_j} \in \mathbb{F}^n$. We will in this section look at both how DPF can be used to achieve compact shared query vectors, and how DPF may be used to query by keyword.

> Informally speaking, a DPF is a representation of a point function $P_{x,y}$ by two keys k_0 and k_1. Each key individually hides x, y, but there is an efficient algorithm $Eval$ such that $Eval(k_0, x') \oplus Eval(k_1, x') = P_{x,y}(x')$ for every x'. Letting F_k denote the function $Eval(k, \cdot)$, the functions F_{k_0} and F_{k_1} can be viewed as an additive secret sharing of $P_{x,y}$[29].

8.3.1 Compact query in 2D using DPF

To understand how DPFs may be used for expressive queries, we will first look at how we may use a Distributed Point function to create a compact 2D query representation.

I will now show how we can represent the query vector as more compact, using a 2D distributed point function. To begin, we will first go from viewing our query vector $\vec{e_j}$ as a bit-string of length n, to a square matrix Q of size $x \times y$, such that $xy = n$.

$$\vec{e_j} = \langle \ 0\ 0\ 0\ 0\ 0 \ | \ 0\ 0\ 0\ 0\ 0 \ | \ 0\ 0\ 0\ 0\ 0 \ | \ 0\ 1\ 0\ 0\ 0 \ | \ 0\ 0\ 0\ 0\ 0 \ \rangle \tag{8.2}$$

Equation (8.2) showing how we may split the query vector into a square matrix.

To explain how we will form a more compact representation of Q using a DPF, I will first present the equation as equation (8.3). We will then step through the equation, and end up with an algorithm which outputs two compact shares of the original query-matrix.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(s_3) \\ G(s_4) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \cdot \vec{v} \right) \oplus \left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(\hat{s}_3) \\ G(s_4) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ 1 - b_3 \\ b_4 \end{bmatrix} \cdot \vec{v} \right) \tag{8.3}$$

Equation (8.3) showing the use of distributed point functions to form two shares of a query-matrix querying for the element in row four, column two.

We will start by generating $x + 1$ random, short seeds $s_0, s_1, s_2, s_3, \hat{s}_3, s_4$. For each seed s_i, we will use the seed as the input to a PRNG to generate a random bit-string of length y, presented in equation (8.3) as the $G(s_i)$-function. For every row of the query-matrix having only zeroes, we will provide the two shares with equal evaluations of the $G()$-function, namely $G(s_0), G(s_1), G(s_2), G(s_4)$. However, for the row of the query matrix containing the 1, i.e. row 4, we will provide the shares with PRNG-evaluations of two different seeds $G(s_3)$ and $G(\hat{s}_3)$.

$$\left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(s_3) \\ G(s_4) \end{bmatrix} \dots \right) \oplus \left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(\hat{s}_3) \\ G(s_4) \end{bmatrix} \dots \right)$$

So far we may note that every row of the two query-shares are identical, except for the 4th row, where $G(s_3)$ and $G(\hat{s}_3)$ appear to be two completely random, unrelated bit-strings. This means that by taking the XOR of these two query-shares will yield something along the lines of the following matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ ? & ? & ? & ? & ? \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

As we want to end up with our original query-matrix Q, it becomes apparent that we now need some way to ensure that the XOR of the two shares ends up with a 1 in the same position as in our original query-matrix, i.e. row four, column two.

To be able to do this, we will first generate and add one random bit per row as $b_0, b_1, \dots, b_4$. Both servers will be given the same bits, except for the bit corresponding to the row of the query vector containing the *1*-value, where we will give the servers opposite values, namely b_3 and $1 - b_3$.

$$\left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(s_3) \\ G(s_4) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \dots \right) \oplus \left(\begin{bmatrix} G(s_0) \\ G(s_1) \\ G(s_2) \\ G(\hat{s}_3) \\ G(s_4) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ 1 - b_3 \\ b_4 \end{bmatrix} \dots \right)$$

Now that the two rows both have randomness, plus an opposite bit-value in the row corresponding to the row we are interested in, we may compute a vector to vector-matrix multiply with each of the shares. The vector will cancel out on every row, except for the row with the differing bit-value, where one share will perform the vector-matrix multiplication, and the other will not. If we construct the vector correctly, we will be able to recover the content of the bit we were interested in.

To achieve this, we will construct the vector $\vec{v}$ as given by the following equation.

$$\vec{v} := G(s_3) \oplus G(\hat{s}_3) \oplus \langle \ 0 \quad 1 \quad 0 \quad 0 \quad 0 \ \rangle \tag{8.4}$$

Note that because of the one bit we introduced as opposites between the shares, b_3 and $1-b_3$, one share is going to perform the vector-matrix multiplication with the vector $\vec{v}$ at row four, and the other is not. If we consider the share that did perform the vector-matrix multiplication, we note that by having XORed this share with both $G(s_3)$ and $G(\hat{s}_3)$, the original randomness, being either $G(s_3)$ or $G(\hat{s}_3)$, is cancelled out, and additionally, we now introduced randomness that is exactly the same randomness as in the other share. If we then consider that we XOR these shares together, we will be left with the original content of the original query-matrix Q. All we need to do now, is to retrieve the bit by XORing the result with a standard basis vector containing a 1 in the position of the column. We may include this basis vector as an XOR in $\vec{v}$, as this the standard basis vector calculation will only be performed for one of the shares. This will yield the original query matrix Q.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Voila! We now have a way of querying that is more compact than the standard basis vectors of length n, given that the sum of seeds, random bits and the vector is smaller than the original query vector of size n. We may note take note of what data we needed to compute each share, which translates to what we must upload to each of the servers to perform a query.

$$\begin{aligned} k_1 = \{\ & (s_0, b_0) \\ & (s_1, b_1) \\ & (s_2, b_2) \\ & (s_3, b_3) \\ & (s_4, b_4), \vec{v}\} \end{aligned} \qquad \begin{aligned} k_2 = \{\ & (s_0, b_0) \\ & (s_1, b_1) \\ & (s_2, b_2) \\ & (\hat{s}_3, 1-b_3) \\ & (s_4, b_4), \quad \vec{v}\} \end{aligned} \tag{8.5}$$

Equation (8.5) showing how we may express two shares of a query vector.

This means that the cost of performing such a query is proportional to $\sqrt{n}$, as we have one PRNG-seed for every row x, plus a vector that is proportional to the number of columns in a row y, and both of these are approximately given as $\sqrt{n}$.

Formally, the upload cost can be given bt the following equation.

$$\approx \sqrt{n}(\log|\mathbb{F}| + |s| + 1) \text{ bits} \tag{8.6}$$

Equation (8.6) showing the upload cost of a compact 2D query using a PDF.

Additionally, we can apply this idea in higher dimensions.

8.3.2 Compact query in 3D using DPF

In order to achieve a three-dimensional query using DPF, we will begin by viewing our n-bit query as a cubic matrix with dimensions $x = y = z = \sqrt[3]{n}$.

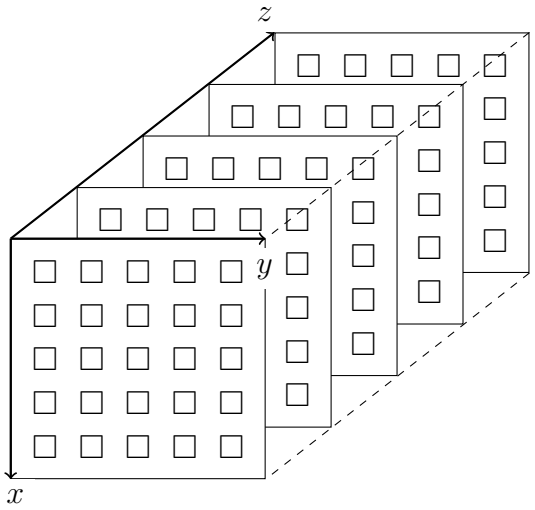


Figure 8.2: A 3D representation of a cubic query matrix, prior to DPF secret sharing.

The goal is to start with a query cube, and end up with a more compact, shared representation of the query cube. We will refer to the dimensions of the cube as rows, columns and *layers*. Similarly to the 2D DPF-query, we are going to begin by generating PRNG-seeds. However, the generated PRNG-seeds will themselves be used to generate a new generation of PRNG-seeds. The new generation of PRNG-seeds will be used to generate two identical psuedorandom shares of our query-cube, except for the layers corresponding to the 1-position of the original query-cube, where the seeds, and thus the evaluation of the seeds, and thus the values, will differ.

We will also generate a random bit for each layer of the original query cube, except for the layer corresponding to the 1-position of the query cube, where we generate two bits as b_i and $1 - b_i$, and provide these values to the corresponding layers of the two shares. As so, all layers of the two shares will be equal, except for one layer corresponding to the 1 in the original query cube, where the values appear random. Similarly to the 2D DPF construct, when we XOR the two query-shares together, we will end up with zeroes in every layer but the layer corresponding to the 1 in the original query cube, where all the values of that layer will appear random. Also similarly to the 2D DPF construct, we will now need to construct vectors to allow us to retrieve the exact part of the layer we are interested in. We must construct two vectors, as we need to recover a single value from a two-dimensional matrix.

For the following example, we are recovering the value in the third layer, second column and fourth row, i.e. $z = 3$, $y = 2$, $x = 4$. As we have already recover layer $z = 2$ from the stated generation of seeds and random bits, we may proceed to construct the row and column vectors $\vec{v}_0$ and $\vec{v}_1$.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} k_1 = \{\ & (s_0, b_0) \\ & (s_1, b_1) \\ & (s_2, b_2) \\ & (s_3, b_3) \\ & (s_4, b_4), \\ & \vec{v}_0, \vec{v}_1 \ \} \end{aligned} \qquad \begin{aligned} k_2 = \{\ & (s_0, b_0) \\ & (s_1, b_1) \\ & (\hat{s}_2, 1 - b_2) \\ & (s_3, b_3) \\ & (s_4, b_4), \\ & \vec{v}_0, \vec{v}_1 \} \end{aligned}$$

Figure 8.3: The query cube layer $z = 2$. Also the complete keys needed to represent the position of the *1*-value in the original query cube.

The first vector $\vec{v}_0$ must be constructed to recover some representation of the fourth row, given by the following equation.

$$\vec{v}_0 := G(s_2) \oplus G(\hat{s}_2) \oplus \langle 0\ 0\ 0\ G(\hat{s}_2)[3]\ 0 \rangle \tag{8.7}$$

Equation (8.7) showing how we construct a vector to recover a representation of the fourth row.

The second vector $\vec{v}_1$ must be constructed to recover some representation of the second column of the fourth row, given by the following equation.

$$\vec{v}_1 := G(G(s_2)[3]) \oplus G(G(\hat{s}_2)[3]) \oplus \langle 0\ 1\ 0\ 0\ 0 \rangle \tag{8.8}$$

Equation (8.8) showing how we construct a vector to recover a representation of the appropriate column.

Having given these values, we may indeed recover the original query cub from the two shares constructed, and we may therefore also recover the value in the database corresponding to the *1*-position in the cubic query matrix. Also note that we are able to more compactly represent our original query cube by two keys, k_1 and k_2, given that the size of a key is smaller than n.

Formally, the upload cost may be given by the following equation.

$$\approx \sqrt[3]{n}(2 \log|\mathbb{F}| + |s| + 1) \text{ bits} \tag{8.9}$$

Equation 8.9 showing the upload cost of a performing a query in a three dimensional database of size $xyz = n$, using a 3D DPF shared Query.

8.3.3 DPF queries in ND

Now that we have applied this construct in both two and three dimensions, we may apply equivalent logic to obtain DPF-constructs of higher dimensions.

By applying this structure in N dimensions, the upload costs may be given by the following table[15].

N dimensions	Upload Cost		
3	$\approx$	$(2\log\|\mathbb{F}\| + \|s\| + 1)\sqrt[3]{n}$	bits
4	$\approx$	$(3\log\|\mathbb{F}\| + \|s\| + 1)\sqrt[4]{n}$	bits
$\vdots$		$\vdots$	
k	$\approx$	$((k-1)\log\|\mathbb{F}\| + \|s\| + 1)\sqrt[k]{n}$	bits
$\vdots$		$\vdots$	
$\log n$	$\approx$	$((\log n - 1)\log\|\mathbb{F}\| + \|s\| + 1)2$	bits

Table 8.1: Upload costs of performing DPF-based queries in k dimensions for a database of size n

What relevance has DPF-based queries for PIR-by-keywords?

While we now have a way to represent our standard basis vector very compactly, recall that our goal is to perform efficient PIR-by-keyword queries. To achieve this, we will look at specifically the $N = \log n$ case of the DPS-based query.

8.3.4 PIR for key-value stores via $(\log n)$D DPF queries

If we apply the method for constructing compact, private queries for the maximum possible dimension $N = \log n$, we will observe that this construct may be viewed as a binary

tree.

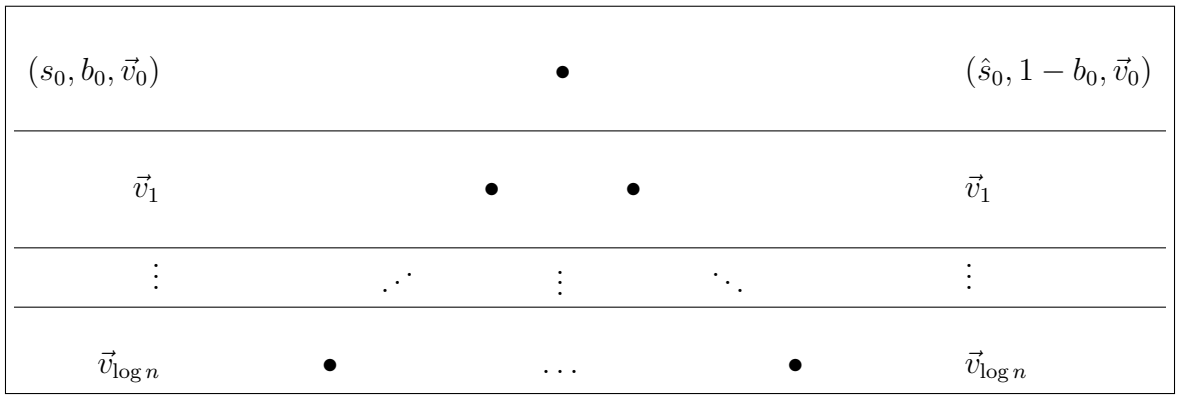


Figure 8.4: Showing how we may view a $N = \log n$-dimensional DPF query as a binary tree, with each of the $\log n$ vectors having size 2.

Now that we can represent queries for records using logarithmic space, we may consider the tree to have an exponential set of leafs, such that each leaf corresponds to a possible keyword. Querying such an exponential space using a logarithmic traversal technique will yield polynomial time[15].

Using this idea, the querier may construct the query with vectors such that the query will end up with a *1*-value in the position of a particular keyword, and zero for everything else.

To do this, we will relabel the query vector $\vec{e}$ as:

$$\vec{e_j} = \langle \square\square \dots 1 \dots \square \rangle \Rightarrow \begin{bmatrix} DPF(k_1; keyword_1) \oplus DPF(k_2, keyword_1) \\ DPF(k_1; keyword_2) \oplus DPF(k_2, keyword_2) \\ \dots \\ DPF(k_1; keyword_j) \oplus DPF(k_2, keyword_j) \\ \dots \\ DPF(k_1; keyword_n) \oplus DPF(k_2, keyword_n) \end{bmatrix}^T \tag{8.10}$$

Equation (8.10) showing the relabelling of the query vector using DPF to search for keywords.

The servers will evaluate all representations of keywords in the query-vector, and so we may search for multiple keywords. Each server will respond with a share of a vector containing a *1*-value in every row-index a keyword was found. After the querier decodes this vector, the querier can then use the indices of the *1*-values to perform standard PIR queries, and obtain the records.

$$\begin{bmatrix} DPF(k_1; key_1) \oplus DPF(k_2, key_1) \\ DPF(k_1; key_2) \oplus DPF(k_2, key_2) \\ \dots \\ DPF(k_1; key_j) \oplus DPF(k_2, key_j) \\ \dots \\ DPF(k_1; key_n) \oplus DPF(k_2, key_n) \end{bmatrix}^T \cdot \left[\begin{array}{c|cccc} key_1 & X_{11} & X_{12} & \dots & X_{1s} \\ key_2 & X_{21} & X_{22} & \dots & X_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ key_n & X_{n1} & X_{n2} & \dots & X_{n2} \end{array}\right] \tag{8.11}$$

Equation (8.11) showing querying by keywords. The query will return standard basis vectors describing in which positions the keywords were found.

We can use this construct to search quite efficiently. The upload cost of a search query may also be given by table 8.1.

Chapter 9

Anonymous Information Retrieval

As we now have some degree of understanding as to how private information retrieval may function, it would be interesting to give a practical example of a case where PIR may solve an existing problem in a practical, deployed environment. To give such a case, we will first give a short introduction into the field of anonymous information retrieval and Tor. Then we will look at how PIR may be used to solve a problem in the current Tor protocol.

9.1 Properties

9.1.1 Anonymity

Anonymity is the notion of hiding one's identity. However, in the online world, this is easier said than done. Commercial actors stand to profit from knowing who you are and for example what ads they should serve you. Such actors may attempt to track your activity as you browse the internet, effectively removing your online anonymity.

In addition to this, internet routers are dependent upon knowing the sender and receiver-addresses of IP-packets for successfully routing traffic, and this information is not encrypted. Even if we encrypt the contents of our messages, an eavesdropper may deduce valuable information from simply looking the recipient of the messages and tying that to our identity. What we would like, is a way to obtain *perfect anonymity*. Perfect anonymity, in the context of Anonymous Information Retrieval, is the notion that a request sent from a client C, to a server S, over a network N, should never compromise the identity of C, even if an eavesdropper E is able to capture the request within N. Of course, perfect anonymity can only account for interactions within N. Device security,

such as client and server security, falls well outside the scope of Anonymous Information Retrieval.

9.1.2 Confidentiality

Anonymity is usually coupled with **confidentiality**, meaning that if an eavesdropper were to capture a message in transit through a network, the eavesdropper could not make sense of the contents, i.e., messages are encrypted.

On the internet, we have come to expect TLS to provide confidentiality where needed. If we were to authenticate securely to a server using TLS, our username and password would be encrypted. But as previously stated, our anonymity is not guaranteed using only TLS. In one solution to the anonymous information retrieval problem, we will see how we may obtain both anonymity and confidentiality by using nested layers of encrypted messages over a network of relays, known to most as *Tor*.

9.2 Tor - The Onion Router

One way to obtain both anonymity and confidentiality on a network can be through a technique called *Onion Routing*. Tor, an acronym of The Onion Router, is a popular implementation of the onion routing protocol.

9.2.1 Tor game

Below is an example of a simplified Tor request and response game.

A client C wants to send a request R to a server S, and wishes to do so anonymously. If C were to send the request directly to S, an eavesdropper E listening on the connection could see that C is sending a request to S. This is an obvious problem if C wishes to protect his anonymity when communicating with S.

To solve this issue, Tor establishes a network of nodes willing to relay and encrypt/decrypt messages, these nodes are often referred to as *relays*. Messages are encrypted and decrypted layer-wise, with each relay in the chain only having the key to encrypt/decrypt a single layer. Messages are also constructed so that a single relay may only see the IP-address of the previous and next relay in the chain, but no other relays.

Let's look at a Tor example with a chain consisting of three relays, as visualized in figure (9.1).

Client	$C_R =$ "GET https://www.simula-uib.com"
Client	$E_R = E(E(E(C_R, K_3), K_2), K_1)$
N_1	$M_1 = D(E_R, K_1) = E(E(C_R, K_3), K_2)$
N_2	$M_2 = D(M_1, K_2) = E(C_R, K_3)$
N_3	$C_R = D(M_2, K_3) =$ "GET https://www.simula-uib.com"
Server	$S_R =$ "HTTP/1.1 200 OK..."
N_3	$R_3 = E(S_R, K_3)$
N_2	$R_2 = E(R_3, K_2) = E(E(S_R, K_3), K_2)$
N_1	$R_1 = E(R_2, K_1) = E(E(E(S_R, K_3), K_2), K_1)$
Client	$S_R = D(D(D(R_1, K_3), K_2), K_1) =$ "HTTP/1.1 200 OK..."

Figure 9.1: Showing an example of a Tor-request round trip with encryption and decryption steps.

Client C chooses three relays participating in the network, N_1, N_2 and N_3. C computes shared cryptographic keys with each of N_1, N_2 and N_3 using a key exchange protocol such as Diffie Hellman, yielding keys K_1, K_2 and K_3.

A message containing the client request C_R is constructed as layers of encryption over the original message. In Tor, the symmetric encryption scheme AES is used with a secret key for each layer.

As we can see from M_1, by C having encrypted the outer layer of C_R with K_1, only N_1 and C with their shared key K_1 may remove the outer layer of encryption from the message. Once N_1 has removed the outer layer, N_1 cannot remove another layer as N_1 lacks the key K_2, which only N_2 and C have, and so forth.

The message is sent along the chain and decrypted with each node's key until the message is no longer encrypted, and the node currently holding the message, referred to as an exit-node, executes the request C_R and obtains the server response S_R from S.

As we can see from the Response return route, a reverse procedure of applying layers of encryption as the response traverses back trough the network is performed until the message with all 3 layers of encryption arrives at C. C, knowing K_1, K_2 and K_3, may now decrypt all layers of the message and read the response S_R.

9.2.2 Chain Ignorance and Minimal Chain Length

Not explicitly shown in the Tor game is that all messages passed through the network contain only the IP-address of the next and previous relay, but no *other* relays. Decrypting a layer reveals the next set of IP-addresses. This mitigates the problem of revealing the sender and recipient of a request or response to a potential eavesdropper. Because each layer of the message contains the address of the previous and next relay, messages can be step-wise passed along using the Internet Protocol with TLS, to be decrypted/encrypted with the shared keys, and passed along again, and so forth.

By using more than one intermediary node, no node may obtain the IP-address of both C and R without breaking the encryption or colluding with the other relays in the chain. This ignorance is what preserves the privacy of C. Note that due to the ease of correlating traffic between only 2 intermediary nodes using *traffic correlation*, Tor uses 3 nodes by default.

9.3 Why the current Tor protocol does not scale

If we look back to our game-based Tor example in figure (9.1), we started by selecting three relays N_1, N_2 and N_3 to allow us to anonymously execute our request. However, we did not discuss how we were able to select these relays.

The information describing what relays are available is currently[1] being served by *directory authorities*, i.e. servers holding information on all the relays participating in the Tor network[30], often referred to as the *global map* or the *global view*.

An initial thought is that we may ask an authority directory to supply us with three relays from this map. However, this leaves us vulnerable if the authority directory turns out not to be trustworthy, as the directory may provide us with a subset designed to compromise our identity[2].

To obtain information on the relays without risking a malicious subset, the client downloads the complete map from a directory authority. This may correctly remind us of trivial private information retrieval. We may recall that trivial private information becomes impractical if the database becomes to big.

We may remark that as relays enter and exit the network, the information in the directory authorities must be updated. For clients to have a somewhat recent version of the

[1]The current stable version of the Tor directory protocol was at the time of writing version 2.

[2]Supplying a malicious subset makes it easy to compromise anonymity through *route fingerprinting attacks[31].*

map, the map must be updated and subsequently re-downloaded quite often. Additionally, as the number of relays grows larger, so does the size of the map. Moreover, as the number of clients increase, so does the number of map downloads. From these facts, we may deduce that as the size of the network grows larger, so does the bandwidth required to distribute the global map. Interestingly, McLachan et al. showed that in the case of Tor, if the number of relays and number of active users continues to grow, the bandwidth required to maintain and distribute the map will in the near future end up becoming greater than the bandwidth used to communicate privately within the network[32]. With this in mind, we may look at alternative strategies for distributing the map, other than trivial private information retrieval.

9.3.1 Scaling Tor with peer-to-peer architecture

Some approaches to scaling Tor suggest a peer-to-peer architecture. Whilst peer-to-peer distribution of the global view would be able to scale to millions of relays[12], this approach may introduce new attack vectors into the system; The security community have in fact been widely successful in breaking privacy in proposed, modern peer-to-peer anonymity systems[33][34][35]. Thus, we may be sceptical of taking a peer-to-peer approach for scaling Tor.

Luckily, we have just learnt of another technology that will allow us to privately retrieve information from the map, with lower communication costs than the trivial approach.

9.3.2 Scaling Tor with PIR

Mittal et al. suggested in 2012[12] an alternative architecture of Tor that allows clients to query for subsets of the global map, using PIR as the protocol.

The paper proposes two PIR protocols as candidates for this purpose, namely the Goldberg'07 IT-PIR scheme introduced in section 6.2 and a lattice-based CPIR scheme proposed by Agular-Melchor and Gaborit[36].

Mittal et al. found that by using lattice-based CPIR, one could reduce the cost of fetching a single circuit, i.e. three relays, by an order of magnitude compared to trivial PIR. For IT-PIR, this improvement was determined to be two orders of magnitude.

In cases where the client wishes to establish multiple circuits, Mittal et al. determined that the CPIR variety quickly approached the costs of a trivial download. For IT-PIR however, the communication overhead was determined to be an order of magnitude smaller than that of a trivial download.

Mittal et al. concluded that a PIR-based architecture could easily sustain a 10-fold increase in both relays and clients[12][3].

[3]Tor has since the release of the Mittal et al. paper been optimized to serve the map more compactly. This means that the Mittal et al. estimate may now serve as a rough approximation.

Chapter 10

Conclusion

I have in this thesis attempted to provide insight into the field of private information retrieval. This has included:

- Defining ordinary information retrieval protocols and private information retrieval protocols.
- Exploring the 1995 Chor et al. paper introducing PIR and IT-PIR.
- Achieving robust IT-PIR using Beimel and Stahl's 2002 paper.
- Achieving improved robust IT-PIR using Goldberg's 2007 paper.
- Exploring varieties of amortized PIR.
- Exploring methods of achieving expressive queries in PIR.

While we have seen how we may technically achieve and utilize PIR, we have not discussed whether we believe it probable to see widespread adoption of PIR protocols "in the wild", so to speak. As our introductory objective states, we will in this conclusion reason about to what degree we can expect to see widespread adoption of PIR protocols. To assist in this goal, I will present some observations and thoughts I had on this subject while writing this thesis, in addition to naming some domains I deem more likely to adopt PIR than others.

Optimal PIR database dimensions are impractical for most cases. One consideration we noticed as we explored PIR protocols, was the assumption that our database was sized according to some relationship $r \approx s \approx \sqrt{|X|}$. While such database structures

are achievable, they do however limit flexibility compared to most conventional databases. This limited flexibility may act as a deterrent for commercial actors and developers considering PIR for a project. However, if bandwidth and computational resources are plentiful, this consideration may be disregarded.

The non-collusion assumption is void for proprietary data. When operating with IT-PIR protocols, we must assume that no coalition exceeding the privacy-threshold will collude. However, if we consider that some commercial actor, for example Google, were to implement IT-PIR for some proprietary database, we will in many cases assume that they are not willing to publicly release such a database for other actors to co-host. In such cases, given that a single actor controls all servers, we may argue that IT-PIR gives no added privacy.

A size-wise lower bound may be given for a database to be reasonably served through PIR. For a PIR-implementation to make sense, we assume that the database is large enough for a trivial download to become less practical than adopting PIR. This gives a lower bound on database size for any practical applications of PIR. Additionally, one may argue that for commercial projects, the investment required to implement a PIR scheme may drive one to surpass this lower bound without ever implementing PIR. If we also consider that bandwidth is generally increasing, we can argue that this lower threshold on the database size will continue to move higher as time progresses and as more data may practically be retrieved privately by trivial download.

There exists a PIR-disincentive for information-dependent commercial actors. Actors such as Google, Facebook and Amazon all have large monetary incentives for knowing who you are, and what you are doing on their platforms. I would go as far as to deem it naïve to expect these actors to willingly use technologies designed to decrease their information flow, and thus their profits, for no benefit except the privacy of their users. As an experiment, you may download the Tor Browser, and attempt to search a term on *google.com*. My result of this experiment was at time of writing an error message from Google stating: *"[...]To protect our users, we can't process your request right now.[...]"*. This may be interpreted as indicative of Google's current views on profits versus privacy. It is on the other hand worth mentioning that Google has, at the time of writing, researchers employed researching PIR. However, whether this is for the sake of providing better privacy for their users, or simply for appearances and goodwill, remains to be seen.

Government and police as adopters of PIR. In the talk from Ryan Henry at ACM 2017, which I have cited extensively, Henry talks about being approached by, at the

time, the head of IARPA[1]. This person supposedly told Henry a hypothetical, where a government agency was trying to locate a terrorist. The agency suspected the terrorist of staying at a hotel in New York, and wanted to find out if this was true. The agency may have proceeded in two ways, either by visiting the hotel and asking the receptionist if the name of the terrorist was in their records (information retrieval), or by compelling the hotel to hand over their records (trivial private information retrieval). Naturally, as the agency would not be inclined to reveal the name of the terrorist, they would perform the trivial private information retrieval by compelling the records. However, this would pose a privacy issue for the innocent guests of the hotel, and by extension, possibly every other hotel in New York. It could be possible for such hypothetical and non-hypothetical use-cases that PIR may improve the privacy of the innocent guests, simply by allowing the government agency to query privately. Henry also talked about the possibility of creating a PIR-scheme with public key infrastructure, such that only a digital warrant, signed with the private key of a judge, may perform a PIR-query from an otherwise restricted database.

I suspect there may be other such practical use-cases for PIR situated in the domain of government and policing.

Already Privacy-oriented projects as adopters of PIR. As we saw in the chapter on anonymous information retrieval, Tor is a potential candidate for the adoption of PIR. I think it is reasonable to assume that projects that are inherently privacy-related may be more likely to adopt PIR than projects which are not inherently privacy-related.

Summarizing my views on PIR. While PIR is exciting, and progress in the field continues to be made, I cannot help but to feel that the conditions for truly widespread adoption of the technology is not yet met – at least not commercially. And is often the case, commercial adoption can be the deciding catalyst for truly accelerating a technology.

However, in an another time, when a privacy-centric mindset is the default for all, PIR might play a larger role in securing our privacy.

10.0.1 Meta

I would like to thank my supervisor, Øyvind Ytrehus, for his guidance and patience during my time as his student. I would also like to thank Ryan Henry of Indiana University, whose talk at ACM CCS in 2017[15] laid the foundation for many of this thesis' examples.

[1]The Intelligence Advanced Research Projects Activity (IARPA) is an organization within the Office of the Director of National Intelligence responsible for leading research to overcome difficult challenges relevant to the United States Intelligence Community.

Given more time to improve and expand this thesis, I would write out a section on SQL-type Queries in the chapter on Expressive Queries. I would also include a chapter on a single-server CPIR protocol, and a chapter on an SPIR protocol.

Bibliography

[1] *PIR in C++*. URL: `http://percy.sourceforge.net/` (visited on 05/26/2019).

[2] Carlos Aguilar-Melchor, Joris Barrier, and Marc-Olivier Killijian. *XPIR*. URL: `https://github.com/XPIR-team/XPIR` (visited on 05/26/2019).

[3] Daniel Demmler, Amir Herzberg, and Thomas Schneider. *RAID-PIR*. URL: `https://github.com/encryptogroup/RAID-PIR` (visited on 05/26/2019).

[4] Benny Chor et al. "Private Information Retrieval." In: *Proc. of 36th IEEE Conference on the Foundations of Computer SCIENCE (FOCS)*. Vol. 36. IEEE, 1995, pp. 41–50.

[5] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 9783642041006.

[6] Berk Sunar, William J Martin, and Douglas R Stinson. "A provably secure true random number generator with built-in tolerance to active attacks." In: *IEEE Transactions on computers* 56.1 (2007), pp. 109–119.

[7] NIST. *Update to Current Use and Deprecation of TDEA*. July 2017. URL: `https://csrc.nist.gov/news/2017/update-to-current-use-and-deprecation-of-tdea` (visited on 05/24/2018).

[8] Peter W Shor. "Algorithms for quantum computation: Discrete logarithms and factoring." In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

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

[10] Felipe Saint-Jean. *Java implementation of a single-database computationally symmetric private information retrieval (cSPIR) protocol*. Tech. rep. YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE, 2005.

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

[12] Prateek Mittal et al. "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval." In: *Proc. of the 20th USENIX conference on Security*. USENIX Association Berkeley, 2011, pp. 31–31.

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

[14] Yanjiang Yang et al. "An efficient PIR construction using trusted hardware." In: *International Conference on Information Security*. Springer. 2008, pp. 64–79.

[15] Ryan Henry. *ACM CCS 2017 - Private Information Retrieval - Ryan Henry*. Youtube. Nov. 2017. URL: `https://www.youtube.com/watch?v=XEYwMPwPxNI` (visited on 05/24/2018).

[16] Amos Beimel and Yoav Stahl. "Robust Information-Theoretic Private Information Retrieval." In: *Journal of Cryptology* 20 (2002), pp. 295–321.

[17] Ian Goldberg. "Improving the robustness of private information retrieval." In: *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE. 2007, pp. 131–148.

[18] Adi Shamir. "How to share a secret." In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[19] Irving S Reed and Gustave Solomon. "Polynomial codes over certain finite fields." In: *Journal of the society for industrial and applied mathematics* 8.2 (1960), pp. 300–304.

[20] Benny Chor et al. "Verifiable secret sharing and achieving simultaneity in the presence of faults." In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE. 1985, pp. 383–395.

[21] Venkatesan Guruswami and Madhu Sudan. "Improved decoding of Reed-Solomon and algebraic-geometric codes." In: *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*. IEEE. 1998, pp. 28–37.

[22] Casey Devet, Ian Goldberg, and Nadia Heninger. "Optimally robust private information retrieval." In: *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012, pp. 269–283.

[23] Wouter Lueks and Ian Goldberg. "Sublinear scaling for multi-client private information retrieval." In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 168–186.

[24] Yuval Ishai et al. "Batch codes and their applications." In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM. 2004, pp. 262–271.

[25] Ryan Henry. "Polynomial batch codes for efficient IT-PIR." In: *Proceedings on Privacy Enhancing Technologies* 2016.4 (2016), pp. 202–218.

[26] Benny Chor, Niv Gilboa, and Moni Naor. *Private information retrieval by keywords*. Citeseer, 1997.

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

[28] Frank Wang et al. "Splinter: Practical private queries on public data." In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 299–313.

[29] Niv Gilboa and Yuval Ishai. "Distributed point functions and their applications." In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2014, pp. 640–658.

[30] The Tor Project. *Tor directory protocol, version 3*. (Visited on 05/26/2018).

[31] G. Danezis and R. Clayton. "Route fingerprinting in anonymous communications." In: *Proc. of International Conference on Peer-to-Peer Computing (P2P 2006)*. IEEE Computer Society, 2006, pp. 69–72.

[32] J. McLachlan et al. "Scalable onion routing with Torsk." In: *Proc. of 16th ACM Conference on Cumputer and Communications Security*. Ed. by S. Jha and A. D. Keromytis. ACM, 2009, pp. 590–599.

[33] Nikita Borisov et al. "Denial of service or denial of security?" In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, pp. 92–102.

[34] George Danezis and Paul Syverson. "Bridging and fingerprinting: Epistemic attacks on route selection." In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2008, pp. 151–166.

[35] George Danezis and Richard Clayton. "Route fingerprinting in anonymous communications." In: *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*. IEEE. 2006, pp. 69–72.

[36] Carlos Aguilar-Melchor and Philippe Gaborit. "A lattice-based computationally-efficient private information retrieval protocol." In: *Cryptol. ePrint Arch., Report* 446 (2007).