Doctoral Thesis

Design of Computationally Efficient Private Information Retrieval Protocols (効率的な秘匿情報検索法の提案)

トマ　フランシス　ヴァネ
Thomas Francis Vannet

Abstract

A key element to the widespread adoption of electronic communication was its strong guarantees of privacy and security. Traditional Public Key Encryption (PKE) systems developed from the 1970s fulfill those requirements. Indeed, any two entities can rely on such schemes to communicate safely and privately under strong computational assumptions. So strong in fact that many countries attempted to legally restrict its usage over concerns of national security. These regulations however proved hard to enforce and truly private communication between clients and remote services became the norm as the Internet developed. However, new privacy concerns stemmed from these omnipresent relations in mainly two ways. Some services built their business model around the systematic gathering of private information to create user profiles and the selling of said profiles to advertisers. Other services unwillingly shared this private data either because they were legally bound to do it or because a third party illegally breached their security and unauthorized access occurred.

One way to address these issues is Private Information Retrieval (PIR). As the name implies, PIR protocols allow a client to recover data on a server he has access to without compromising his privacy. The concept was formally introduced in 1995 by Chor *et al.* and a number of solutions were quickly designed. Compared to other cryptographic primitives, the definition of PIR only offers limited power to the client. Namely, he can only recover bits of information already present on the database and already available to him if he requests them directly. This apparent simplicity makes PIR comparatively likely to be used on real-world systems in the near future. Meanwhile, this low degree of freedom is still sufficient to be applied in a variety of scenarios ranging from financial services to biometric authentication.

There is a very natural so called trivial PIR algorithm satisfying all the privacy requirements and consisting in merely sending all the data stored on the server to the client regardless of his query. While not practical, this algorithm has a reasonable computational cost, optimal in a lot of settings. Its bottleneck resides in the prohibitive communication cost since network

speeds tend to be far slower than local data processing. As such, most early contributions to this research area focused on reducing the amount of data that has to be transferred and proved to be very successful in that regard.

More recently, focus has shifted towards the design of lightweight schemes as the heavy computation cost became the last obstacle standing between these theoretical protocols and practicality. My research takes place in this setting. Two main issues were identified and improved on. First, most schemes deal with a low-level model where the clients know the physical address of the bits he wishes to recover. While sufficient, this model can be highly impractical. Second, almost every known PIR protocol operates by reading the entire database for every query received. This is a major limitation which can be improved upon.

A scheme that allows the execution of SQL-like queries on a server-hosted database efficiently and privately was designed. Since most data publicly available online is organized in such databases, this is a necessary consideration to provide realistic schemes. This situates itself in a generalized PIR framework called Extended Private Information Retrieval. The algorithms described distinguish themselves from others in the literature since they do not require independent data replication, also known as multi-server PIR. This latter setting only makes sense in some very specific instances while our approach is applicable to a wide range of situations.

Another contribution is a lightweight block PIR scheme performing very efficiently. It is built by combining different existing constructions and its security is based on a well trusted computational assumption known as the Approximate GCD assumption. Straightforward preprocessing methods speeding up the running time are detailed and shown to also apply to other published protocols.

A new and more complex approach to PIR with preprocessing is created under the name Partial Server Side Parameter Selection (PSSPS) PIR. It achieves lower overall computation and allows for a justified trade-off where the cost of running the algorithm can be split in many ways between client and server. This approach is applied to the Approximate GCD-based scheme introduced previously and to another one from the literature. Further developments of this approach are discussed along with guidelines to create more efficient schemes based on it.

Now, all of the methods described above are compatible with each other and can be used together to build an efficient scheme on real-world databases or used independently and combined with other PIR schemes to gain different combinations of desirable properties.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Online privacy is a growing concern for many Internet users all over the world. The issue of privacy over the Internet became a topic of mainstream media interest with the scandal that followed the Snowden leaks in 2012. Both governments and consumer associations pleaded for a more private experience online. Such privacy can take many shapes. Indeed, the revelation that large scale data collection [GP] was occurring was a reminder of the importance of privacy-oriented electronic security. Specifically, they outlined with great detail the secret close collaboration between major companies and some governmental organizations. Such relationships break the trust users have put in companies running online services. Similarly, targeted advertising and other inconspicuous recommendation systems systematically learn users' behavior in order to aggressively sell them more products. While these can potentially be helpful or beneficial, very often they are done without warning the user or acquiring his consent.

There really are only two avenues to solve these issues. One is to outlaw or regulate such behaviors, and many legislations all around the world have been passed to make sure citizens' privacy is not sacrificed for security or for profit. Another option is to make it impossible practically. This is what cryptography aims for. Schemes were designed, relying on computational or other assumptions to guarantee security. Meanwhile, time has proven that such mathematical assumptions are harder to break and less likely to be revoked than arbitrary written laws.

Now traditional encryption techniques such as Public Key Encryption and Symmetric Key Encryption allow the creation of private channels between a client and a server. Under appropriate assumptions, all communications on this channel are shielded from potential eavesdroppers. This of course does not prevent the server itself from sharing or exploiting the information he received without the user's consent. More complex primitives are therefore required.

We can mainly distinguish two cases for client-server interactions. Either the client is uploading and maintaining his own information on a cloud service, or he is downloading information that the server is willing to share with him. The former option falls under the broad category of cloud computing and cryptographic primitives such as homomorphic encryption [Gen09] or searchable encryption [CGKO06] offer solutions. The latter option's most common instances would be performing a search on a public website (search engine, shopping website, video delivery, some forms of social media, etc.) This encompasses a very significant portion of the Internet activity performed by average users [Jac] and is the setting we will be studying in this thesis.

Besides providing strong privacy for any "queryable" online service, we can identify some specific privacy-sensitive applications which can only be solved using this approach. For years it had been conjectured that stock exchange services could monitor investors' lookups and use that information for their personal gain. In 2013, the *Bloomberg Terminal Snooping* scandal [Cho] revealed that it had actually been happening for a long time, making it a justifiable fear. In the field of biometrics authentication, it is typical to have a server hosting the biometrical data for many users and another proxy server handling the (encrypted) credentials checking process [BCPT07]. This way, the proxy does not learn personal data and the data-hosting server does not learn when and who is logging in on a system. Such a setup however requires this type of privacy-enabling technology between the proxy and the other server. Finally, DNS servers are a necessity to allow users to convert memorable URLs into IP addresses. In doing so, they inherently learn the name of every website said users visit which is a major privacy shortcoming and could be solved. Naturally, every system relying on a user's location would benefit from these schemes as users can access the service without revealing a highly private data such as their location [KS09].

Now, in this standard setting we have one or more clients each downloading data independently from one or more servers. Informally, we need each client to know the result of their query while the server(s) must be unable to learn anything about said queries by the end of the interaction. This problem was formally defined under the name Private Information Retrieval or PIR in 1995 by Chor, Goldreich, Kushilevitz and Sudan [CGKS95]. Their original suggested solution required several independent servers hosting copies of the same database and had a communication complexity of $O(n^{1/k})$ where n is the number of bits in the database and k the number of servers.

Following the original construction, a groundbreaking result due to Kushilevitz and Ostrovsky in 1997 [KO97] showed that sufficient security could be achieved without the requirement on the existence of multiple independent servers. A variety of other schemes followed suit and were developed in the following decade, bringing the communication cost lower and lower until it reached a point where it was only a constant number of times larger than it would be for a non-private protocol in 2005 [GR05]. During this entire time, the prohibitive cost of sending large amounts of data over a potentially slow network had been the driving force behind the efforts to reduce communication as can be seen by the trove of protocols developed during that time [Amb97, AF02, BS03, CGKS95, GR05, GIKM98, Gol07, KO97, Lip05]. Note that all of those schemes either required a local server-side computation using the entire database as an input or a secure coprocessor attached to the server.

In 2007, Sion implemented the strongest contenders for practicality and compiled the results in a widely cited technical report [Sio07]. In particular, each implementation result was compared with the cost of sending the entire database over the network (which offers the same level of privacy) for different transfer speeds. The report's conclusion was that for all of the single server PIR schemes, processing one bit of data was actually slower than sending the same bit over the network, even on a slow channel. In other words, none of those schemes had any practical usefulness.

After this revelation, most subsequent proposals focused on improving the computational complexity of PIR algorithms, either theoretical or through more efficient implementation [MCG$^+$08, DC14]. In 2011, Sion's original report was revisited by Olumofin and Goldberg [OG12]. Their findings revealed that the newer schemes were able to perform one to two orders of magnitude faster than naive database sending. While encouraging, these results are still not practical for many potential applications of private information retrieval.

1.2 Related Works

Olumofin presented his PhD thesis [Olu01] in 2011 on a related topic. He had a very different approach however. First, his proposals were built for multi-server PIR, which gives a lot more freedom to the designer but is also compatible with only a small fraction of potential applications for PIR. Second, he did not design any preprocessing-based schemes. It is a well known fact that preprocessing is a necessity for sublinear complexity, although it has never truly been achieved in a generic setting so far. Third, his contributions, written in the wake of Sion's report [Sio07], aimed at showing that practical implementation was possible and actually outperformed the sending of the entire database. It is therefore very implementation-oriented while we focus mostly on theoretical findings (we only implement our results as a way to validate our claims).

1.3 Open Problems

The main issue with Private Information Retrieval protocols is their inadequacy for real-world systems, which is materialized in different ways.

First, a vast majority of schemes require reading the entire database for every query. This is not an option for most online services hosting large amounts of data. Other schemes have unrealistic assumptions such as many servers hosting copies of the data but somehow forbidden from communicating with each other.

Other protocols require the user to place some degree of trust in third-party issued hardware or certificates. While this is a fairly common assumption in other areas of cryptography, it does not fit the philosophy of PIR where clients not only do not trust third-parties, they do not even trust the other party they are interacting with.

The standard PIR model also expects the client to know the physical location of the data he wants to recover, which is very seldom the case. Most databases are hosted with a complex structure and data cannot be accessed meaningfully in such direct, low-level ways.

Finding a scheme that manages to avoid all of these shortcomings is the main problem we will be tackling here.

1.4 Contributions

PIR still faces many issues before it may be used in widespread large-scale systems. The formal setting under which it is defined offers great freedom but also hinders its practicality. Indeed, the standard definition indicates that a client selects the index of a bit and engages in a protocol with a server, at the end of which the client learns the data located at the index location while the server learns nothing about said index. A number of issues naturally arise when observing this definition. The privacy requirements imply that every single bit of the database must be read for every query. Meanwhile, there is an assumption that a client would know the physical location of the data he wishes to recover. Finally, only a single bit of data is recovered by the end of the protocol. While this last issue has received considerable attention and can arguably be considered solved, the first two problems do not yet possess a truly satisfactory solution.

Our contributions aim at addressing these concerns by extending the setting in which PIR is used in meaningful ways to provide privacy where it is actually needed. For instance, consider the problem of having to read the entire database to answer any query. In Chapter 3, we build around the idea that clients usually do not require perfect privacy but only need to hide specific information when querying. Specifically, we consider the very common case of SQL-like databases for which realistic usage scenarios are easy to envision and prepare for. With this limited approach to privacy, we are able to theorize a framework that addresses the first and second issues to some extent. In Chapters 4 and 5, we instead focus on a purely computational approach that partially solves the first and third problems. Both suggestions can further be used conjointly to build a system providing improvements in every direction.

We now detail the contributions.

A scheme that allows the execution of SQL-like queries on a server-hosted database efficiently and privately was designed. Since most data publicly available online is organized in such databases, this is a necessary consideration to provide realistic schemes. This situates itself in a generalized PIR framework called Extended Private Information Retrieval (EPIR). In this framework, clients are not only allowed to recover data directly, but also allowed to recover the result of a semi-private function on said data. Only a few function families have been shown to be compatible with EPIR schemes. We show that most common SQL queries can actually be used in such a way.

The algorithms described distinguish themselves from others in the literature since they do not require independent data replication, also known as multi-server PIR. This latter setting only makes sense in some very specific instances while our approach is applicable to a wide range of situations. Furthermore, our approach allows for much greater query expressivity and ease of use compared to existing generic solutions.

Another contribution is a lightweight block PIR scheme performing very efficiently. It is built by combining different existing constructions and its security is based on a well trusted computational assumption known as the Approximate GCD assumption. Straightforward preprocessing methods speeding up the running time are detailed and shown to also apply to other published protocols. The main interest of this scheme however is its compatibility with the new method introduced next.

Indeed, a new and more complex approach to PIR with preprocessing is created under the name Partial Server Side Parameter Selection (PSSPS) PIR. It achieves lower overall computation and allows for a justified trade-off where the cost of running the algorithm can be split in appropriate ways between client and server. This approach is applied to the Approximate GCD-based scheme introduced previously and to another one from the literature. This allows us to build a scheme able to go under the information theoretical lower bound on server-side computation cost in PIR protocols. This is the first time this bound is broken without any trust assumptions (secure coprocessor, non colluding servers) or running environment requirements (such as a high number of queries at any given time), although we show our scheme can run even faster under such conditions. Further developments of this approach are discussed along with guidelines to create more efficient schemes based on it.

Now, all of the methods described above are compatible with each other and can be used together to build an efficient scheme on real-world databases or used independently and combined with other PIR schemes to gain different combinations of desirable properties.

1.5 Organization

Chapter 2 provides the reader with all the necessary information to understand the technical contents of the thesis. It also provides formal definitions for the notions briefly mentioned above and used through the document. Finally, it gives summary examples of protocols from the literature that achieve various properties of interest or are representative of a specific setting.

Chapter 3 is our first contribution. It details a new protocol allowing clients to efficiently retrieve the result of a wide variety of SQL queries over a remote database. It is compatible with queries over any number of fields and with any condition that can be expressed as a logic formula over field-value comparisons, including `LIKE` queries and more generally regular languages testing. Its complexity will be dependent on the block PIR scheme used to recover the values once the query has been performed.

Chapter 4 is our second contribution. We designed a lightweight single-server PIR scheme that can outperform other known schemes — both single-server and multi-server — through the use of simple preprocessing optimizations. Its security relies on the Approximate GCD assumption. We also show how the precomputing techniques can be used on some other schemes for additional efficiency.

Chapter 5 is our third major contribution. It introduces the new concept of Partial Server-Side Parameter Selection which allows the server to preselect part of the information that is usually sent by the client. This lets the server perform some computations in an offline phase only happening once and before receiving any query. This reduces its online computational complexity but increases the client's complexity. Complexity trade-offs between server and client have to be considered and studied.

Chapter 6 shows how to combine all of the contributions presented before in our main proposal, an efficient PIR scheme designed for real-world databases, along with measurements and comparisons with other schemes.

Chapter 2

Preliminaries

In this chapter, we introduce all the techniques that will be relevant to the building of our main contributions. It will also help the reader better appreciate the broader setting in which this research is located.

2.1 Threat Models

Cryptography can be seen as the art of protecting information against attackers. The type of attacker will vary widely depending on the specific field of study. We describe the two main models frequently encountered in the privacy preserving area.

First let us clarify that in a private client-server interaction, not only third parties but also the server itself are considered adversaries. Security against third parties is the subject of very extensive research and we will not be covering it here. For all practical purposes, we can assume that every communication between client and server is encrypted using a trusted symmetric key cryptosystem such as the AES encryption [oST01] along with a secure key exchange protocol such as the Diffie-Hellman Key Exchange [DH06]. Whenever we talk about security in such privacy-oriented protocols, the only attackers will be the parties directly involved in the protocol.

We now describe the two adversary models for the server itself, namely the Honest but Curious model and the Malicious model. When performing a protocol in the Honest but Curious setting, the parties involved will follow every step of the protocol as described in its specification. On the side, the attacker may do additional computations using the data that he received in an attempt to break the client's privacy. The attacker (the server) will however always return what the client expects. Meanwhile, the malicious model will give greater freedom to the attacker, allowing him to stray from the protocol and send arbitrary data instead. A scheme provably secure (possibly under a number of assumptions) against the weaker assumption of a Malicious attacker will provide greater security overall. In practice, most of the PIR protocols we will study only have a single round of interaction. The client sends a query and the protocol ends when the server sends a response. In this situation, sending malicious data does not give the server any additional information.

A PIR scheme will be deemed secure if no Probabilistic Polynomial Time (PPT) adversary in the malicious model as described before is able to distinguish between two queries for different pieces of data (for instance, bit index) stored server-side. This security will be conditional on a security assumption, either decisional or computational.

Another consideration will be the number of parties involved. Oftentimes, the interaction takes place between a single client and a single server. However some schemes allow or even require more participants. Most commonly, a setting where one client interacts with several servers is the base for many protocols. It is then common to have a security conditional to the number of collaborating servers, in particular requiring that not every single server cooperates with each other. Most of our proposals however work in the single-server setting where such concerns do not arise.

A t-server PIR scheme will be deemed Information Theoretically secure (also known as ITPIR) up to a coalition of $k < t$ servers if no adversary controlling k servers or less can distinguish between two queries for different pieces of data. In particular this means that for such an adversary, any query could recover any piece of data with equal probability.

Other schemes allow for several clients to take part in one round of communication, adding potential concerns for malicious interaction between clients. Such a protocol will then be expected to guarantee that clients are unable to retrieve data related to each other's private information. In particular, a u-client PIR scheme is deemed secure if no PPT adversary controlling the server and up to $u-1$ clients is able to distinguish between two queries for different pieces of data sent by a client not under the adversary's control. If t servers are involved, the scheme is deemed information theoretically secure if no adversary controlling at most $t-1$ servers and at most $u-1$ clients is able to distinguish between two queries for different pieces of data sent by a client not under its control.

2.2 Homomorphic Encryption

Homomorphic Encryption is one of the most fundamental constructions available in cryptography with applications in a wide array of fields. As the base building block for many proposals, we describe its importance, history and the main construction that achieves it.

2.2.1 Background

A function F between two algebraic structures $(A, *_A)$ and $(B, *_B)$ is an homomorphism when $\forall a_1, a_2 \in A$, $F(a_1 *_A a_2) = F(a_1) *_B F(a_2)$. Common examples include linear polynomials over any field or exponentiation over integers.

Now let $\texttt{Enc}$ be an encryption function between a message space S_M and a ciphertext space S_C. We assume that both sets have internal operations $+$ and $*$. In most cases, those sets will be bit sequences which can be translated to integers on which such operations do indeed exist.

$\texttt{Enc}$ will be considered a homomorphic encryption scheme if there exist two operations $\bigstar_M$ and $\bigstar_C \in \{+, *\}$ such that $\texttt{Enc}(m_1 \bigstar_M m_2) = \texttt{Enc}(m_1) \bigstar_C \texttt{Enc}(m_2)$ for messages m_1 and m_2 in the message space S_M.

This property can be desirable as it allows a client to request some computation to be done on encrypted data remotely stored without having to go through the trouble of downloading it, performing the computations himself and then potentially uploading the result. In other settings, it is a nuisance as it allows malicious attackers to create ciphertexts for messages without the user's consent.

A number of schemes are naturally homomorphic (rather than by design) like the RSA encryption.

Definition 2.2.1

RSA Encryption.

For secret primes p and q, define $n = pq$ and find a public e and secret d such that $ed = 1 \mod (p-1)(q-1)$.

For $m \in \mathbb{Z}_n$, $\texttt{Enc}(m) = m^e \mod n$ and $\texttt{Dec}(c) = c^d \mod n$.

We clearly have $\texttt{Enc}(m_1 m_2) = \texttt{Enc}(m_1)\texttt{Enc}(m_2)$.

In these cases, a scheme can be made non homomorphic in many ways, for instance by padding the messages with a hash before encryption. We are however more interested in schemes which are homomorphic by design as this property allows the convenient building of private schemes.

2.2.2 Additively Homomorphic Encryption

Of particular interest to us are additively homomorphic encryption schemes. We will show how these can be used to build privacy-enabling protocols. In this section, we introduce a few of these.

Definition 2.2.2

Paillier Cryptosystem. *[Pai99] (simplified)*

For secret primes p and q, define $n = pq$.

The public key is (n, g) where $n = pq$ and $g \in \mathbb{Z}_{n^2}$. The secret key is $\lambda(n)$ where $\lambda(n) = lcm(p-1, q-1)$.

$\texttt{Enc}(m) = g^m r^n \bmod n^2$ *for a random $r \in \mathbb{Z}_n^*$*

$\texttt{Dec}(c) = \frac{L(c^{\lambda(n)} \bmod n^2)}{L(g^{\lambda(n)} \bmod n^2)} \bmod n$ *where $L(x) = (x-1)/n$*

Here we have $\texttt{Dec}(\texttt{Enc}(m_1)\texttt{Enc}(m_2)) = \texttt{Dec}(\texttt{Enc}(m_1 + m_2))$ and $\texttt{Enc}$ is additively homomorphic.

Definition 2.2.3

Goldwasser-Micali Encryption. *[GM82]*

For secret primes p and q, define $n = pq$.

The public key is (x, n) where $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1$. The secret key is (p, q).

$\texttt{Enc}(b) = x^b r^2 \bmod n$ *for a bit b.*

$\texttt{Dec}(c) = 0$ *if c is a quadratic residue,* 1 *otherwise.*

Here too we have $\texttt{Enc}(b_1)\texttt{Enc}(b_2) = \texttt{Enc}(b_1 + b_2)$

2.2.3 Somewhat Homomorphic Encryption

If a scheme supports both additive and multiplicative homomorphic encryption over single bits, then repeated application of these two basic notions allows the computation of any function that can be built using `OR` and `AND` gate. This notion is called Fully Homomorphic Encryption (FHE).

$\texttt{Enc} : \{0, 1\} \to S_C$ is such that $\texttt{Enc}(b_1) + \texttt{Enc}(b_2) = \texttt{Enc}(b_1 + b_2)$ and $\texttt{Enc}(b_1)\texttt{Enc}(b_2) = \texttt{Enc}(b_1 b_2)$ where $+$ is the modulo 2 addition.

The existence of such a secure primitive was conjectured since the 1980s but the first such secure construction was proposed by Gentry in 2009 [Gen09]. Following this publication, many other schemes were developed using more or less related techniques.

As it turns out, it is impossible to straightforwardly build a secure fully homomorphic scheme (see [GS16] for instance).

As such, we define Somewhat Homomorphic Encryption schemes as schemes supporting both additive and multiplicative homomorphism up to a certain degree. Typically, ciphertexts contain a certain amount of noise that grows with every additional homomorphic operation. When the noise crosses a set threshold, the ciphertext becomes undecipherable. This can in turn be translated in a threshold on the number of operations that can be performed on a "fresh" ciphertext.

Definition 2.2.4

Approximate GCD-Based Encryption[vDGHV10]

For a secret large odd number p, random q and noise ϵ small compared to p, we have:

$\texttt{Enc}(b) = pq + 2\epsilon + b$

$\texttt{Dec}(c) = (c \bmod p) \bmod 2$

In this case $\texttt{Enc}(b_1) + \texttt{Enc}(b_2) = \texttt{Enc}(b_1 + b_2)$ *and* $\texttt{Enc}(b_1)\texttt{Enc}(b_2) = \texttt{Enc}(b_1 b_2)$ *so long as the noise does not grow too high.*

Definition 2.2.5

Ring-LWE-Based Encryption.*[BV11] (simplified)*

The secret key is a small $s \in R_q = \mathbb{Z}_q[X]/\langle f(X)\rangle$ according to some specific Gaussian distribution. The message space is R_t (coefficients less than t).

$\texttt{Enc}(m) = (as + te + m, -a)$ *for some random $a \in \mathbb{Z}_q[X]/\langle f(X)\rangle$ and a small noise e according to the same distribution as s.*

$\texttt{Dec}(c_0, c_1) = c_0 + c_1 s \bmod t$

Once again $\texttt{Enc}(m_1)+\texttt{Enc}(m_2) = \texttt{Enc}(m_1+m_2)$ *and* $\texttt{Enc}(m_1)\texttt{Enc}(m_2) = \texttt{Enc}(m_1 m_2)$ *for a specific multiplication and so long as the noise does not grow too high.*

2.2.4 Fully Homomorphic Encryption

Starting with Gentry's original scheme in 2009 [Gen09], every secure FHE construction has relied on a technique called bootstrapping on top of a somewhat homomorphic scheme. Bootstrapping consists essentially in homomorphically executing a noise removal function (deciphering then re-encrypting with a low noise once again). The difficulty lies in finding an implementation of this noise resetting function that requires fewer operations than the scheme-specific threshold.

The design and implementation of such schemes is beyond the scope of this thesis and we only present the major scheme that is used in privacy-oriented constructions.

Definition 2.2.6

Fully Homomorphic Encryption over the Integers. *[vDGHV10]*
Using the Somewhat Homomorphic Encryption scheme from Definition 2.2.5, homomorphically reset the noise from the ciphertext using a bootstrapping function light enough that the noise does not outgrow p during the process.

2.3 Private Information Retrieval

2.3.1 Background

In this section, we define the specific field in which our research situates itself. Let us assume a server is hosting data that is available for viewing to a client. A Private Information Retrieval (PIR) protocol allows the client to recover part of that data without revealing which part is being retrieved.

Formally, the database $\mathcal{DB}$ consists of n bits $(b_1 \cdots b_n)$ where $b_i \in \{0, 1\}$. The client selects an index $x \in \{1, \cdots, n\}$. The protocol has to retrieve the value b_x while maintaining the index x secret from the server. A protocol may require one or more rounds of interaction between client and server. In practice, most schemes only have a single round.

We call a **query** the ordered set containing all the data exchanged between a client and a server excluding the final server reply. Similarly, we call the server's **response** the ordered set containing all the data returned by the server. In particular, in a single round scheme a query would be the data sent by the client initially while the response would be the data returned by the server.

The protocol can then be repeated as many times as necessary to recover any subset of the database $\mathcal{DB}$. This definition thus allows the private simulation of any non private client-server interaction over public data.

Trivial Algorithm

A very simple yet important way of achieving the aforementioned goals is to use the Trivial Algorithm.

This algorithm satisfies all the requirements from the definition of a PIR protocol. The server has to read and send n bits of data while the client has to download n bits of data (and discard all but one). While it is deeply inefficient and extremely simple, we will refer to this algorithm many times as a base to which other protocols can be compared.

Algorithm 1 Trivial PIR Algorithm

```
Require: Public $\mathcal{DB} = (b_1 \cdots b_n)$, secret index $x \in \{1, \cdots, n\}$
Ensure: Client learns $b_x$, server does not learn $x$
  Client sends 1 to server
  Server returns $res = \mathcal{DB}$
  Client recovers $b_x = res[x]$
```

Security

A PIR scheme is said to be secure if it is hard for an attacker to distinguish between queries for different bits. The specific definition will depend on the setting in which we work. A scheme can achieve information theoretical security or computational complexity. We detail these two cases in the following sections.

2.3.2 Information Theoretical Private Information Retrieval

A PIR scheme is information theoretically secure if there exists no way for the server to gain any information about the secret index. In other words, given a query q for a secret index x, we have $\forall i \in \{1, \cdots, n\}$, $Pr(x = i|q) = 1/n$.

For instance, the Trivial PIR Algorithm is information theoretically secure as the query received by the server (a constant 1) could have been a query for any index in $\mathcal{DB}$ with equal probability since its generation did not depend on x.

Single Server Setting

Interestingly enough, information theory dictates that in a single-server setting, such a scheme is only possible when the server's response has a size of at least n bits.

Theorem 2.3.1

Any information theoretically secure PIR protocol has a server response at least as large as the database size.

Proof: Let us call m the bit size or the server's response and assume $m < n$.

Now given a fixed query, we consider the function $F(\mathcal{DB})$ which returns the server's response. $F : \{0,1\}^n \to \{0,1\}^m$. Since $m < n$, F is not injective, we know that there exist $\mathcal{DB}_1, \mathcal{DB}_2 \in \{0,1\}^n$ two distinct databases such that $F(\mathcal{DB}_1) = F(\mathcal{DB}_2)$.

Since the client receives no information besides $F(\mathcal{DB})$, it is impossible for him to distinguish between $\mathcal{DB}_1$ and $\mathcal{DB}_2$. However, the protocol's correctness means that the client learns b_x from the server's response. As such, $Pr(x = i) = 0$ for every index i such that $\mathcal{DB}_1[i] \neq \mathcal{DB}_2[i]$.

Thus the scheme does not achieve information theoretic security. ■

Naturally, the same is true regarding the computational cost of the algorithm on the server's side. In other words, the server has to read at least n bits of information or the scheme cannot be information theoretically secure.

Theorem 2.3.2

If a server reads less than the entire database's size when answering a PIR query, then the scheme is not information theoretically secure.

Thus sending exactly n bits of data is the best we can hope to achieve in this setting. This is exactly the performance from the Trivial PIR Algorithm consisting in sending the entire database. It is optimal yet impractical which motivates us to look at different settings.

Multi-Server Setting

One way to go past this limitation is to work in a setting where one client will interact with several servers. The proof from Theorem 2.3.1 then does not apply since any indistinguishable databases on any one server's side may be distinguishable using the other servers' responses.

Here a number $t > 1$ of servers all host a copy of the exact same database $\mathcal{DB}$. The client enters an exchange with each of these independently and retrieve their individual answers, which he can then use to recover the value of b_x.

If the servers' independence is not maintained, the scheme potentially loses its information theoretic security. In particular, if all t servers collaborate and share the queries sent by one client, the scheme becomes equivalent to a single-server one and will only maintain its level of security if at least n bits of data are sent to the client overall. At best, a scheme will retain its security for coalitions of up to $t - 1$ servers. Several schemes available in the literature achieve this level of security. We give informal definitions of two such examples.

Definition 2.3.3

***Goldberg's Robust PIR Scheme**. [DGH12]*

There are k servers. The client wants to recover $b_{x,y}$. He selects k values α_t, $t \in \{1, \cdots, k\}$ and a random degree $k-1$ polynomial P_i for *every $i \in \{1, \cdots, \sqrt{n}\}$ such that $P_x(0) = 1$. He sends $\{P_i(\alpha_k), 1 \leq i \leq \sqrt{n}\}$ to server k.*

Server k returns $\sum_{i=1}^{\sqrt{n}} b_{i,j} P_i(\alpha_k)$ for every j.

The client interpolates $Q_j(X) = \sum_{i=1}^{\sqrt{n}} b_{i,j} P_i(X)$ and recovers $b_{x,y} = Q_y(0)$.

Definition 2.3.4

***CGKS**. [CGKS95]*

There are 2^d servers and $n = l^d$ for some l. b_x is recovered where index $x = \sum_{c=0}^{d} x_c l^c$ where $0 \leq x_c < l$.

For every t, server t is sent d subsets $S_c^{\sigma_c}$ where $t = \sigma_1 \cdots \sigma_d$ and $S_c^0 \oplus S_c^1 = x_c$ and he returns the exclusive-or of all the bits in the subcube defined by these sets.

The client then computes the exclusive-or of all the bits he receives and gets b_x.

2.3.3 Computational Private Information Retrieval

Unfortunately, very few real-life settings allow for multiple non-colluding servers to host the same database. Another more practical solution to the information theoretical limitations is Computational Private Information Retrieval (cPIR). In this setting, the security of a scheme relies on the hardness of distinguishing between queries for different indices in a probabilistic polynomial time in the database size n and some security parameter λ_s.

More specifically, suppose there exists an algorithm $\mathcal{A}$ that takes as input a query for a secret index x and that returns 0 or 1. Assume, for $i \in \{1, 2\}$, that $\mathcal{A}(q)$ returns 1 with probably σ_i when the input q is randomly generated according to the client's query generation algorithm for secret index x_i with $x_1 \neq x_2$. If $\mathbf{Adv}(\mathcal{A}) = |\sigma_1 - \sigma_2|/2$ is non negligible then $\mathcal{A}$ is a distinguisher for the protocol.

The protocol is computationally secure if no such algorithm $\mathcal{A}$ exists. Naturally, proving this non existence requires the assumption that some underlying problem is impossible to solve in a polynomial time.

Common Assumptions

We list some of the more common assumptions used to build computationally secure PIR schemes.

Definition 2.3.5

ϕ-hiding Assumption. *[CMS99]*

Let $M \in \mathbb{N}$ and p, q be two primes such that exactly one of them divides $\phi(M)$. It is assumed that it no PPT algorithm is able to distinguish which one of them does with non-negligible advantage.

There are a number of requirements on the bit size λ of p and q relative to the bit size of M to circumvent attacks and maintain security of the assumption. Most importantly, we require that both p and q be less than $M^{1/4}$ to prevent attacks due to Coppersmith on modular equations [Cop96a, Cop96b].

Another assumption we will rely on is the Approximate GCD assumption.

Definition 2.3.6

Approximate GCD Assumption.

Let $C = \{pq_i + \epsilon_i\}_i$ where p is an odd random integer over λ_p bits, q a random integer over λ_q bits and ϵ a random integer over λ_ϵ bits. It is assumed that there exists no PPT algorithm that returns p given a polynomially large collection C.

Once again, there are requirements on the parameters size to keep the assumption true. We will detail these in Section 4.4.5. Roughly, we require $\lambda_\epsilon > \lambda_s$ and $\lambda_q > \lambda_s \lambda_p(\lambda_p - \lambda_\epsilon)$ to achieve λ_s bits of security.

Multi-Server Setting

While this type of approach allows for efficient single server PIR, it can also be used with multi-server systems. For instance, Goldberg's Robust PIR Scheme (Definition 2.3.3) can be instantiated in a hybrid mode where the scheme is information theoretically secure as long as at least one server is honest and remains computationally secure if all the servers collude.

However, if the setting is such that there is any risk that all servers collude, it is much more efficient to have a single-server scheme relying exclusively on a computational assumption. We will therefore not study this situation.

Single Server Setting

This is the main setting of interest for this type of security model. Most PIR proposals in the literature fall in this class. So do our proposals.

We briefly present some schemes based on the assumptions mentioned earlier.

Definition 2.3.7

Gentry-Ramzan's Constant Communication Rate. *[GR05].*

Basically, after deciding on a set of n *powers of small prime numbers* $\{p_i^{c_i}, 1 \leq i \leq n\}$*, the server, using the Chinese Remainder Theorem, describes the set* $\{data_i, 1 \leq i \leq n\}$ *containing the whole database as a single integer* e *such that* $e \equiv data_i \mod p_i^{c_i}$.

To recover $data_i$ *privately, the client sends a generator* g *of a group of order* $qp_i^{c_i}$ *to the server which responds with* g^e*. Now the user computes* g^{eq} *which belongs to a subgroup of order* $p_i^{c_i}$ *in which he can perform a discrete logarithm in base* g^q*. The result of this computation is* $e \bmod p_i c^i = data_i$.

Definition 2.3.8

Cachin-Micali-Stadler's Polylogarithmic Communication. *[CMS99]*

Assume the client wants to recover bit b_x *from the database* $(b_1, \cdots, b_n)$.

Given a algorithm P *able to generate a fixed random sequence of prime number over* λ_p *bits for a seed* s*, write* $p = P(x, s, \lambda_p)$ *and pick a modulo* M *such that* M ϕ*-hides* p *(according to Definition 2.3.5) and a random value* $r \in \mathbb{Z}_M^*$*. Send* r, P, s, M, λ_p *to the server.*

The server computes $x^{\prod_{i=1}^n b_i P(i,s,\lambda_p)} \bmod M$ *and sends it to the client. Then the client checks whether this value is a* p*-th residue and learns* b_x.

Definition 2.3.9

Yi's Fully Homomorphic-Based Scheme *(basic scheme) [YKPB13]*

Assume the client wants to recover bit b_x *from the database* $(b_1, \cdots, b_n)$.

For any $i \in \{1, \cdots, n\}$*, we write* $i = \sum_{j=0}^{\lfloor \log_2 n \rfloor} i_j 2^j$.

Let $(\mathbf{Enc}, \mathbf{Dec})$ *be a Fully Homomorphic Encryption scheme. The client sends* $\{c_j = \mathbf{Enc}(x_j), 0 \leq j \leq \lfloor \log_2 n \rfloor\}$ *to the server. The server computes and returns:*

$$res = \sum_{i=1}^{n} b_i \prod_{j=1}^{\lfloor \log_2 n \rfloor} (i_j + c_j + 1)$$

The client recovers $b_x = \mathbf{Dec}(res)$.

In addition, any additively homomorphic scheme can be turned in a PIR protocol under the same assumption. Alternatively, any FHE scheme can also be turned into a PIR protocol with logarithmic communication. At this point, FHE schemes are far from practical and this construction remains purely theoretical.

2.4 Private Information Retrieval with Preprocessing

A variety of PIR protocols relying on preprocessing have been suggested in the literature.

Schnorr and Jakobsson [SJ00] mentioned PIR with preprocessing as an application of their encryption scheme back in 2000. In this scheme, the entire database is encrypted by the server and sent to the client in an offline phase. Then during the online phase the client further encrypts one of the records with its own key, sends it to the server who homomorphically decrypts the first layer of encryption and the client is then able to recover the unencrypted data using the client's private key. The applications for such a system are very limited.

Asonov *et al.* [AF03] developed in 2003 a system which allows constant time online queries but requires periodic preprocessing as each additional query between preprocessing rounds increases the query length. Its privacy however relies on a secure processor installed on the server but which the server is not able to read, which is against our principle of forbidding the need for trust in elements not under client's control.

Beimel *et al.* [BIM04] showed in 2004 that preprocessing could provide improvements without the need for secure hardware or high communication. Instead, he used a multi-server configuration which, as we mentioned, has limited applications. We are interested in more schemes usable in more generic settings.

Gasarch and Yerukhimovich [GY07] built in 2006 a scheme that uses preprocessing on the client's side. Namely, the client generates once a large number of parameters used to make queries. Then during the online phase, the client is able to efficiently take random subsets of those preprocessed values to generate queries inexpensively. Since the main bottleneck in PIR is the server side computation, this provides little improvement overall, especially in the common situation where a client sends a single query.

Finally, in 2014 Aguilar-Melchor *et al.* showed how to improve a Ring-LWE-based scheme [MBFK16] by preprocessing the database values so that their representation allows for linear computations during the online phase. This technique's objective was to linearize the scheme in some of the security parameters in which it would usually be quadratic. Since the largest parameter is always the database size, this only marginally improves the running time.

2.5 Related Privacy Enhancing Techniques

We here present other standard cryptographic primitives improving user privacy in various settings. While they are closely related to PIR and implementations may occasionally be based off the same assumptions, the constructions behind them and the goals they aim to achieve are widely different. The informal descriptions we provide justify the choice of PIR as a valuable research topic.

2.5.1 Oblivious Transfer

Oblivious Transfer was originally defined as the problem of sending a message with probability 1/2 without learning whether or not it was sent [Rab81]. It was eventually generalized to 1-out-of-n [BM89].

An 1-out-of-n Oblivious Transfer (OT) protocol is essentially a PIR protocol with the added restriction that it should not only be impossible for the server to find the secret index x being recovered but it should also be impossible for the client to recover the value of any b_i for $i \neq x$. In particular, this means that the Trivial PIR Algorithm is not an oblivious transfer protocol.

Because of this, OT is sometimes also called Symmetric Private Information Retrieval. While technically every OT scheme is a valid PIR scheme, the added server-side privacy requirement often translates to higher computational and communication costs. On the other hand, some schemes built with PIR in mind happen to satisfy the privacy requirements to be considered OT. While the added privacy never hurts, it is of very little interest in most of the PIR applications where the database queried is considered public.

2.5.2 Searchable Symmetric Encryption

Searchable Symmetric Encryption (SSE) [CGKO06] is another primitive which has been studied for about 15 years. It resembles PIR in many ways but has one key difference. In an SSE setting, a client will encrypt and upload its own private database onto a server (for instance, all of his personal emails). He then queries the remote database to retrieve some data.

Depending on security requirements, the query may be private or not. The server then executes a protocol to return the data the client was searching for. This step is reminiscing of PIR except for the fact that all the database setup was done by the client under a secret key of his choice.

Overall, these schemes can be seen as a somewhat easier version of PIR thanks to the extra setup step where the client is able to insert a trapdoor in the data for later efficient retrieval. As such, SSE protocols are not easily transformed into PIR ones.

2.5.3 Oblivious RAM

An Oblivious RAM scheme [GO96] allows a user to execute a secret program involving reading and writing data in memory (RAM) in such a way that an external observer cannot distinguish the memory access pattern for different inputs. Among various applications, this can be used to compute a secret function over private data that was encrypted and uploaded to a remote server. For instance, the function can simply be information retrieval. Conversely, by using a PIR protocol a user can retrieve the data needed to execute his secret program. Oblivious RAM is thus easier than PIR.

However, similarly to Searchable Symmetric Encryption, Oblivious Ram schemes require that the data read is encrypted by the user in a preliminary setup phase. This approach is completely incompatible with settings where the data is meant to be used by many unrelated users, as is common for PIR applications.

2.5.4 Onion Routing

Another approach to achieve privacy when retrieving data from servers is anonymous routing, also known as Onion Routing [SGR97]. Here, a client will build a “chain” or path of nodes (peers) in a network and then route his standard (non private) query through all of them before it reaches the server.

No single peer on the chain knows both the origin of the query and its destination and as such privacy is preserved. While PIR hides *what* is queried, anonymous routing hides *who* is querying. Compared with PIR, it has a slower overhead but communication speeds are greatly reduced. Security is also only maintained when at least one of the nodes on the chain is honest.

Unlike PIR, onion routing is susceptible to attacks based on traffic analysis or simply user profiling based on activity. For instance, Onion Routing cannot be used to access biometrical data which would reveal immediately *who* is querying. Similarly, in the field of financial markets study, there is no need for a malicious server to know *who* is querying as the analysis of query trends already provides valuable information.

The two primitives thus have mostly unrelated applications. Combining both techniques in one scheme is of course doable [MOT$^+$11], but practicality takes a severe hit. Besides, PIR can be used for premium or pay-per-view services while Onion Routing is usually not compatible with such business models.

2.5.5 Repudiative Information Retrieval

Asonov introduced the notion of Repudiative Information Retrieval (RIR) [AF02] as a more permissive alternative to Private Information Retrieval. An RIR protocol achieves the same goals as a PIR protocol but differ in its security definition.

In an information theoretical setting, an RIR protocol is secure if it is information theoretically impossible to prove that a specific entry in a public database was not requested. In particular, the user can always deny having queried an entry or claim to have queried another one. Compared to a PIR protocol, there is no requirement for the probability that an element was requested to be exactly the same for every element. In RIR, it simply needs to be non-zero.

While this definition made sense in the setting used by Asonov and which involved a secure coprocessor, it has not been used in other constructions or in computational PIR where schemes always achieve the stronger PIR requirements.

Chapter 3

Private Information Retrieval for SQL-like Queries

3.1 Introduction

The most basic requirement for a PIR scheme is the ability to privately recover one bit of data at a given secret index out of n bits available on a remote server. Only considering single-bit recoveries is convenient since it is clearly sufficient to simulate any operation yet allows the design of schemes relying on decisional assumptions thanks to their intrinsic binary nature. However we can easily identify a number of shortcomings this approach entails. First, most situations where PIR is used would require the reading of potentially long blocks of consecutive bits stored server-side. This issue is addressed by Block PIR protocols which have been extensively studied. We do not discuss them here. Second, there is an implicit requirement that the client needs to know the physical address of the data he wishes to retrieve. As the data is usually stored in a structured way, such an address is not directly available to the client. In some settings, this issue is solved by the use of Private Information Retrieval by Keywords schemes [BCN98]. Third, the privacy model requires the server to be unable to distinguish between two potential secret bit indices. In most scenarios, it makes little to no sense for most bits to be requested anyway and relaxing the privacy definition allows for more efficient schemes.

Looking at applications for PIR protocols including DNS servers, access to biometrical data or financial markets information, one realizes that most real life servers store data in a relational database such as SQL databases. This SQL-like structure allows clients to query information based on conditions using integer, floating point numbers, strings or arbitrary data. Emulating this type of common behavior on a low-level implementation of PIR protocols requires a great number of rounds between the client and server [BCN98] which, combined with the well-known requirement that the entire database has to be read for the query received every round to maintain privacy, makes those protocols impractical. We are interested in generic approaches which also give the server a greater freedom in setting up his database in any way he wants, allowing it to perform both private and non private queries on the same database. This is the main motivation behind our search for highly SQL-friendly PIR protocols.

Another unrelated motivation is purely theoretical. Many cryptographic primitives allow the computation of a function on some data in a private manner. Different privacy requirements will bring about diverse solutions including (Fully) Homomorphic Encryption [vDGHV10], Garbled Circuits [BHR12] or Extended Private Information Retrieval (EPIR) [BCPT07]. We are here interested in EPIR schemes which allow the computation of a partially public function on an input composed of secret client-selected values and public server-hosted data. Only equality testing, Hamming weight computing [BCPT07] and polynomial evaluation [BC09] have been shown to be EPIR-compatible. We are therefore interested in developing EPIR protocols for more complex functions such as the ones required for SQL queries.

In this chapter, we first propose a protocol to recover database entries that have some of their fields matching a secret pattern picked by the user. The query itself is not hidden away from the server and only the patterns are kept secret. This is usually of little importance as most real world databases have a specific purpose which is known (when performing a query on a DNS server-hosted database, revealing the fact that the user queries the IP address associated with a domain name is of no consequence, only the IP address itself should be kept private). Our protocol is efficient enough to be practically used on an actual SQL database with few entries (the standard restriction that every entry of the database has to be read for every query is still valid). This algorithm is based on the ϕ-hiding assumption which is extensively used in the field of PIR [CMS99].

We then extend this idea to more complex queries not only testing equality between a field and a value, but arbitrary comparisons (greater than, lower than, equal to), eventually leading us to consider the case of complex logic formulas using said field-value comparisons as predicates. As a final step, we suggest a scheme able to check if database entries belong in a secret regular language. In particular, this implies the possibility to perform any `LIKE` query in SQL.

3.2 Related Works

Olumofin described and implemented a framework called SQLPIR [OG10] that lets a client perform SQL queries privately using PIR techniques. His proposal has a few key differences with ours however.

First, his scheme is designed for a multi-server setting while ours is designed for single-server situations. This is a major distinction as multi-server PIR only has a handful of applications while single-server can potentially be applied to any existing system. Secondly, his approach mainly consists in filtering a database based on one parameter value, downloading the results and further filtering locally. In particular, *greater than* and *lower than* comparisons cannot be performed directly by the server. Comparatively, our approach executes such operations privately in an EPIR [BCPT07] fashion. For a much wider range of functions than ever shown before, we show that it is possible to compute their result on public data while keeping the functions private.

Private Information Retrieval with Keywords

In 1998, Chor, Gilboa and Naor introduced the concept of PrivatE information Retrieval with KeYwords ($\mathcal{PERKY}$) [BCN98]. For a server-hosted database consisting of n strings $s_1, \cdots, s_n \in \{0,1\}^l$, a $\mathcal{PERKY}$ scheme allows a client, given a secret keyword $w \in \{0,1\}^\ell$, to learn whether or not $\exists i \in \{1, \cdots, n\}$ such that $w = s_i$. The very generic scheme they presented is compatible with any PIR scheme and any data structure on the server-side. This can be seen as a primitive version of an SQL query testing for equality.

Informally, Chor *et al.*'s approach consists in performing an oblivious walk over the data structure. In particular, performance is achieved when the structure allows the search for a particular entry in a small (ie, logarithmic in n) number of lookups.

The approach we describe in Section 3.3 achieves the same result and its most simple version is indeed a $\mathcal{PERKY}$ algorithm as defined in [BCN98]. Two of the tree data structures described by Chor *et al.* require a number of rounds linear in the length ℓ of the strings. Using perfect hashing techniques, it is possible to reduce it to a constant number of rounds if the underlying PIR scheme has a constant number of rounds. However in every case the schemes cannot be used if the subset of data the client wishes to work on is not known in advance.

Indeed, for an actual SQL-like table over several fields, a query could be using any combination of fields for which no efficient data structure exists. In such cases, the algorithms proposed in [BCN98] would require n rounds of PIR.

3.3 Testing for equality

In this section we present an algorithm allowing a user to efficiently and privately perform SQL-like queries of the following form.

SELECT $\texttt{outField}_1$, ..., $\texttt{outField}_t$
FROM table
WHERE $\texttt{inField}_1$ = s_1
...
AND $\texttt{inField}_u$ = s_u

Here outField stands for output field while inField stands for input field. The s_i are the secret input values.

More formally, a table is a set of entries $entry_i$ for $i \in \{1, \cdots, n\}$ and each entry is a list of couples $entry_i = (field_1, value_{i,1}) \cdots (field_f, value_{i,f})$ where all the $value_{i,j}$ are bit strings. We call Fields the set $\{field_1, \cdots, field_f\}$. The client picks a set of input fields $\texttt{inField}_1, \cdots,$ $\texttt{inField}_u$ and output fields $\texttt{outField}_1, \cdots, \texttt{outField}_t$ in Fields along with a set of input parameters $s_1, \cdots, s_u$ where each s_i is a bit string. For every entry in the table containing all of the $(\texttt{inField}_j, s_j)$ couples, the user expects to learn the couples $(\texttt{outField}_k, value)$. The server should be unable to distinguish between two queries for secrets $(s_1, \cdots, s_u)$ and $(s'_1, \cdots, s'_u)$.

To achieve this, we first show how the problem is equivalent to a simpler one where $t = u = 1$. Then we show how to detect whether the table contains an entry satisfying the conditions or not. Finally, we show how to use this information to recover the value of the output fields associated to such an entry if it exists.

3.3.1 Assumptions

For the rest of the section, we will consider the simpler case where $t = u = 1$. Indeed, denoting by $\|$ the bit string concatenation, we can see that our problem is equivalent to performing the following SQL-like query:

SELECT $\texttt{outField}_1 \parallel \cdots \parallel \texttt{outField}_t$
FROM table
WHERE $\texttt{inField}_1 \parallel \cdots \parallel \texttt{inField}_u$ = $s_1 \parallel \cdots \parallel s_u$

Indeed, the user can separate the received bit string into the values for the t output fields and the server performs the query as if all the input fields behave like a single concatenated field.

All `outField` values are assumed to have fixed bit length. This is not an issue as the server's response must be as long as the longest value in the `outField` used to maintain privacy anyway. Under the widely accepted assumption that there exists collision-resistant hash functions, we can assume that the concatenated length of the `inField` values is rather short, regardless of the input length u. Essentially, for such a secure hash function h, the user performs this SQL-like query:

```
SELECT outField$_1$ $\|$ $\cdots$ $\|$  outField$_t$
FROM table
WHERE h(inField$_1$ $\|$ $\cdots$ $\|$  inField$_u$) = h($s_1$ $\|$ $\cdots$ $\|$ $s_u$)
```

Here $h(s_1\|\cdots\|s_u)$ is the secret value and has short, fixed length. Following this reasoning, we now assume that we work on a single input field of short length L and want to recover a single, possibly long, output bit string.

Furthermore, we can assume without loss of generality that there always exists at most one entry satisfying the input condition. Indeed, if the user is able to recover any output field value for a given condition when a single entry matches it, then he can do the same when several entries do using the following protocol. First, the server groups the entries of the database sharing the same values on the set of input fields and adds a temporary field `count` indicating the number of entries corresponding to this set of conditions. Note that this is actually what the standard SQL statement `GROUP BY inField`$_1$`, ..., inField`$_u$ does and requires no specific implementation on the server's part.

Then the user starts by only recovering the value of that temporary field. Afterwards, the server ungroups the entries and adds a temporary field `group_id` numbering the entries from 1 to the associated `count` within each group. Finally, the user adds `group_id` to the input fields and recovers the values one at a time on a set of conditions which now guarantees uniqueness.

Privacy Concerns

This may raise some privacy concerns as the number of queries sent by the client will reveal the value of the created temporary field `count` to the server. Knowing this value, the server can then deduce that the entries the client is recovering belong to a group of this size.

A simple solution would be to first have the server let the client know the maximum value of `count` after receiving the "private" query structure. We call this value *maxCount*. Then if the client actually wants to retrieve $k < maxCount$ values, he just sends extra queries for random elements so that *maxCount* queries are performed regardless.

This is obviously not very efficient and a waste of resources, but it unfortunately is optimal. Indeed, if we consider an efficient setting where the server's response is proportional to the data being received by the client, and if said client wants to retrieve k entries, then the server has to return at least k times the data that would be returned for a single entry.

In practice, we can assume that either the client does not require such a high level of privacy (otherwise generic PIR techniques can be used) or that the queries are specific enough to return a fixed number of entries (for instance a unique result or the first k result according to some order).

3.3.2 Detecting if an entry with a given secret value exists

We recall here the ϕ-hiding assumption as defined in [CMS99].

Definition 3.3.1

Let $M \in \mathbb{N}$ and p, q be two primes such that exactly one of them divides $\phi(M)$. It is assumed that it no PPT algorithm is able to distinguish which one of them does with non-negligible advantage.

In this section the user simply detects whether there exists an entry in the database satisfying the secret condition. Thus, no output field is involved. Here the parameters of our algorithm are $n \in \mathbb{N}$ the number of entries in the database, $L \in \mathbb{N}$ the bit length of our single input field and $s_1, \cdots, s_L \in \{0, 1\}$ the bit decomposition of our condition s.

For security parameters λ and f, the user chooses a large random prime p on λ bits and a random integer q (non prime) on $\lambda(f-1)$ bits such that $q_1 = pq + 1$ is a prime on λf bits. He also chooses a prime q_2 on λf bits and sets $M = q_1 q_2$, this way $p|\phi(M) = pq(q_2 - 1)$. In particular, we require $p < M^{1/4}$ to prevent an efficient attack using Coppersmith's results [Cop96a, Cop97b] on finding small root of polynomials.

Note that in this situation, the client does not know the actual factorization of $\phi(M)$. While this is not needed, it is convenient to consider the situation where all factors are known. To achieve this, the client sets $q = 2q'$, with q' prime and $q_2 = 2q'_2 + 1$ with q'_2 prime and then $\phi(M) = 2^2 pq'q'_2$. Besides 2, all the prime factors of $\phi(M)$ are very large and with overwhelming probability a random odd number in $\mathbb{Z}_M$ is coprime with M.

The user then chooses such a random value x coprime with M. The client builds two lists T_0 and T_1 of length L and fills all L entries $T_\delta[1], \cdots, T_\delta[L]$ of each list with random **even** values over μ bits ($2^\mu \gg p$) for $\delta \in \{0,1\}$. Finally he picks a random **odd** number r over $\mu - \lambda$ bits (such that rp has μ bits) and defines the value $S_0 = rp - \sum_{j=1}^{L} T_{s_j}[j]$.

We define the function $Sum(v_1 \cdots v_L) = S_0 + \sum_{j=1}^{L} T_{v_j}[j]$. Note that $Sum(s) = rp$. Also note that for every $v \in \{0,1\}^n$, $Sum(v)$ is odd as it is sum of an odd number S_0 and L even numbers. The target input field name `inField`, T_0, T_1, S_0, M and x are then sent to the server.

For every entry in the database, we call v_i the value such that the couple (`inField`, v_i) belongs to that entry. The server computes $Sum(v_i)$ and sends $res = x^{\prod_{i=1}^{n} Sum(v_i)} \bmod M$ to the client.

Upon receiving this value the client computes $\alpha = res^d$ where $d = \frac{\phi(M)}{gcd(p,\phi(M))}$ ($d = q(q_2 - 1)$ usually). This value is equal to 1 if and only if res is a p-th power residue. In this case, the client concludes that there exists an entry in the table with `inField` value s. Conversely, if $\alpha \neq 1$ then the client concludes that no entry has an `inField` value s.

Correctness

Theorem 3.3.2

With overwhelming probability, $\alpha = 1$ if and only if there exists some index $i \in \{1, \cdots, n\}$ such that $v_i = s$

Proof: We consider the two cases:

- As noted earlier $Sum(s) = rp$ so if the database contains an entry matching the condition, $\prod_{i=1}^{n} Sum(v_i)$ is a multiple of p and res will clearly be a p-th root residue since $p|\phi(M)$.
- Now let us assume the database contains no such entry. For a bit string $v \neq s$, there exists an index j_0 such that $v_{j_0} \neq s_{j_0}$. Therefore $T_{v_{j_0}}[j_0]$ was chosen randomly over μ bits and independently from S_0 and every $T_{v_j}[j]$ for $j \neq j_0$. For $0 \leq k < p$, let us write σ_k the probability that the sum of the terms besides $T_{v_{j_0}}[j_0]$ be congruent to k modulo p:

$$\sigma_k := \Pr(S_0 + \sum_{j \neq j_0} T_{v_j}[j] \equiv k \mod p) \tag{3.1}$$

Note that we have:

$$\sum_{k=0}^{p-1} \sigma_k = 1 \tag{3.2}$$

Thanks to the independence, the following holds:

$$\Pr(Sum(v) \equiv 0 \mod p) = \sum_{k=0}^{p-1} \sigma_k \Pr(T_{v_{j_0}}[j_0] \equiv p - k \mod p) \tag{3.3}$$

Since $2^\mu \gg p$, $\Pr(T_{v_{j_0}}[j_0] \equiv p - k \mod p) = 1/p$ for $0 \leq k < p$, which means that $\Pr(Sum(v) \equiv 0 \mod p) = 1/p$. Finally, the probability that all n values v_i satisfy $\Pr(Sum(v_i) \bmod p \neq 0)$ is $(1 - 1/p)^n \approx e^{-n/p} \approx 1 - n/p$ (since $n \ll p$) which is overwhelming when λ, the bit length of p, is large:

$$\Pr(p \nmid \prod_{i=1}^{n} Sum(v_i)) \approx 1 - n/p \tag{3.4}$$

Therefore with overwhelming probability $res = x^{\prod_{i=1}^{n} Sum(v_i)} \bmod M$ is not a p-th root residue since x was not. ■

Complexity

We recall here that M is the size of the group in which we work in and is large. In the first step, the client selects the parameters which only involves a polynomial (in λ) number of primality tests. When computing res, the server executes L sums over μ bits and one modular exponentiation over $\log M$ bits for every entry of the database. The complexity is then $n\mu(L + \log^2 M)$ elementary operations with naive multiplication. Then the client performs a single exponentiation in at most $\log^3 M$ elementary operations. The communication complexity is $O(\log M + \mu L)$, which depends on the database size n (since both μ and $\log M$ are larger than λ and we require that λ is larger than $\log n$).

Security

As mentioned earlier, security relies on the ϕ-hiding assumption. The server only knows M, x, S_0, T_0 and T_1. x is just a random number and gives no information. T_0 and T_1 are also filled with random numbers. S_0 is the only non-random value as it is part of a linear relation related to M via $\phi(M)$. Note that we assume that the server will uncover this relation (in a realistic scenario, the sought-after pattern is likely to be present in the database in which case the server would compute *rp* at some point). Intuitively, if the server is able to figure out which pattern the user requested, then he distinguishes between several integers the one that has a divisor also dividing $\phi(M)$.

Theorem 3.3.3

If there exists a PPT algorithm distinguishing between queries, then the ϕ-hiding assumption can also be broken in probabilistic polynomial time.

In order to prove this theorem, we will need a number of lemmas.

We write $\mathcal{Q}$ the set of all queries $q = (M, x, S_0, T_0, T_1)$ where $M \in \mathbb{Z}$ is the public modulo built as described in Section 3.3.2, $x \in \mathbb{Z}_M$, S_0, T_0 and T_1 are client-selected values also described in the same section.

Assume there exists an algorithm $\mathcal{A}(M, x, S_0, T_0, T_1) : \mathcal{Q} \to \{0, 1\}$ such that:

- $\mathcal{A}(q)$ returns 1 with probability σ_1 if q is uniformly drawn at random from the set of all queries for secret value s_1
- $\mathcal{A}(q)$ returns 1 with probability σ_2 if q is uniformly drawn at random from the set of all queries for secret value $s_2 \neq s_1$
- Advantage $\mathbf{Adv}(\mathcal{A}) = |\sigma_1 - \sigma_2|/2$ is non negligible

In other words, $\mathcal{A}$ is able to distinguish queries for a secret $s = s_1$ or $s = s_2$.

Lemma 3.3.4

Without loss of generality, we can assume that $\sigma_1 + \sigma_2 = 1$ ($\sigma'_1 = 1/2 + \boldsymbol{Adv}(\mathcal{A})$ and $\sigma'_2 = 1/2 - \boldsymbol{Adv}(\mathcal{A})$).

Proof: First, we can assume without loss of generality that $\sigma_1 > \sigma_2$ (if not switch the indices).

Second, if we have $\sigma_1 + \sigma_2 \neq 1$, we build an algorithm $\mathcal{A}'(q)$ that ignores input and returns 0 with probability α_0, ignores input and returns 1 with probability α_1 or returns $\mathcal{A}(q)$ with probability $1-(\alpha_0+\alpha_1)$ for the following parameters:

- If $\sigma_1 + \sigma_2 > 1$ then $\alpha_0 = 1 - \frac{1}{\sigma_1+\sigma_2}$ and $\alpha_1 = 0$ (note that $\alpha_0 \in [0,1]$)
- If $\sigma_1 + \sigma_2 < 1$ then $\alpha_0 = 0$ and $\alpha_1 = 1 - \frac{1}{2-(\sigma_1+\sigma_2)}$ (note that $\alpha_1 \in [0,1]$)

We write $\sigma_i' = \Pr(\mathcal{A}'(q) = 1 | s = s_i) = \alpha_1 + (1 - (\alpha_0 + \alpha_1))\sigma_i$ for $i \in \{1, 2\}$.

- If $\sigma_1 + \sigma_2 > 1$ then:

$$\sigma_1' + \sigma_2' = (1 - \alpha_0)(\sigma_1 + \sigma_2) = 1 \tag{3.5}$$

- Similarly, if $\sigma_1 + \sigma_2 < 1$ then:

$$\begin{aligned}\sigma_1' + \sigma_2' &= 2\alpha_1 + (1 - \alpha_1)(\sigma_1 + \sigma_2)\\ &= 2 - \frac{2}{2-(\sigma_1+\sigma_2)} + \frac{\sigma_1+\sigma_2}{2-(\sigma_1+\sigma_2)}\\ &= 1\end{aligned} \tag{3.6}$$

Proof: Definition of $\mathcal{A}'$ ensures that $\sigma_1' > \sigma_2'$ when $\sigma_1 > \sigma_2$, which guarantees that the new advantage $\mathbf{Adv}(\mathcal{A}') = \sigma_1' - 1/2$ is positive.

- If $\sigma_1 + \sigma_2 > 1$ then:

$$\begin{aligned}\mathbf{Adv}(\mathcal{A}') = \sigma_1' - 1/2 &= \frac{\sigma_1}{\sigma_1+\sigma_2} - 1/2\\ &= \frac{2\sigma_1 - (\sigma_1+\sigma_2)}{2(\sigma_1+\sigma_2)} = \frac{\mathbf{Adv}(\mathcal{A})}{\sigma_1+\sigma_2}\\ &> \frac{1}{2}\mathbf{Adv}(\mathcal{A}) \text{ since } \sigma_1+\sigma_2 < 2\end{aligned} \tag{3.7}$$

- If $\sigma_1 + \sigma_2 < 1$ then:

$$\begin{aligned}\mathbf{Adv}(\mathcal{A}') = \sigma_1' - 1/2 = 1 &- \frac{1}{2-(\sigma_1+\sigma_2)} + \frac{\sigma_1}{2-(\sigma_1+\sigma_2)} - 1/2\\ &= 1/2 + \frac{\sigma_1 - 1}{2-(\sigma_1+\sigma_2)} = \frac{\sigma_1-\sigma_2}{2(2-(\sigma_1+\sigma_2))}\\ &= \frac{\mathbf{Adv}(\mathcal{A})}{2-(\sigma_1+\sigma_2)} > \frac{1}{2}\mathbf{Adv}(\mathcal{A})\end{aligned} \tag{3.8}$$

■

Lemma 3.3.5

Let s_1, s_2 be two bit strings over L bits.

For $k \in \{1, 2\}$, we call χ_k the uniform distribution of all inputs (M, x, S_0, T_0, T_1) obtained from setting $S_0 = rp_k - \sum_{j=1}^{L} T_{s_{k,j}}[j]$ for uniformly random even values $T_\delta[j]$ over μ bits, uniformly random odd values r over $\mu - \lambda$ bits and uniformly random prime p_k over λ bits. Assume a PPT algorithm $\mathcal{A}(q)$ returns 1 with probability γ_k for $q \leftarrow \chi_k$. In other words, χ_k is the distribution of all queries for secret s_k.

This lemma states that $|\gamma_1 - \gamma_2|$ is negligible.

Proof: Let us assume that $|\gamma_1 - \gamma_2|$ is non negligible. Then $\mathcal{A}$ is by definition a distinguisher between the two distributions χ_1 and χ_2.

Note that the only difference between the input distributions is the value of S_0. If an attacker can distinguish between S_0 from χ_1 and χ_2, then he can distinguish between $S_0 + \sum_{j=1}^{L} T_{s_{1,j}}[j]$ from distributions χ_1 or χ_2.

This means distinguish between $z_1 = rp_1$ and $z_2 = rp_2 + \sum_{j=1}^{L} T_{s_{1,j}}[j] - T_{s_{2,j}}[j]$. z_1 is a product between a uniformly random prime over λ bits and a uniformly random number over $\mu - \lambda$ bits. z_2 is also a random number over μ bits.

However, it is a well known fact that the proportion of 2^λ-smooth integers (integers with no prime factor greater than 2^λ) less than 2^μ is $\rho(\mu/\lambda)$ where $\rho(u)$ is the Dickman-de Bruijn function [Dic, Gra08]. This function becomes negligible very quickly, guaranteeing the existence of prime factors of the order of p_1 for large enough parameters. z_1 and z_2 are both random numbers over μ bits with a prime factor of size at least λ.

The two distributions are thus indistinguishable. ■

We now prove Theorem 3.3.3 which we recall here:

Theorem

If there exists a PPT algorithm distinguishing between queries, then the ϕ-hiding assumption can also be broken in probabilistic polynomial time.

Proof: We are given a ϕ-hiding problem instance: M a semiprime with both factors over $f\lambda$ bits, p_1 and p_2 primes over λ bits such that with equal probability 1/2 either p_1 or p_2 divides $\phi(M)$ while the other is a prime uniformly selected at random. We build an algorithm $\mathcal{B}$ using $\mathcal{A}$ (working in the case $\sigma_1 + \sigma_2 = 1$) that guesses which p_i divides $\phi(M)$ with non negligible advantage.

Note that the distribution for the input tables of $\mathcal{A}$ is very simple. Every element of all L columns is a random even number over μ bits. S_0 is drawn from a specific distribution which we must mimic before invoking $\mathcal{A}$ to make sure that it acts as a distinguisher.

For $k \in \{1, 2\}$, we randomly pick every value in $T_{k,0}$ and $T_{k,1}$ as described in Section 3.3.2. We also pick x_k randomly in $\mathbb{Z}_M$ and define $S_k := r_k p_k - \sum_{j=1}^{L} T_{k,s_{k,j}}[j]$ for random r_k over $\mu - \lambda$ bits. We write $q_k := (M, x_k, S_k, T_{k,0}, T_{k,1})$ and $t := (\mathcal{A}(q_1), 1 - \mathcal{A}(q_2))$.

$\mathcal{B}$ is defined in the following way. If $t = (0, 0)$ or $(1, 1)$, $\mathcal{B}(M, p_1, p_2)$ returns 0 or 1 with equal probability 1/2. If $t = (1, 0)$ then $\mathcal{B}(M, p_1, p_2)$ returns 1 and if $t = (0, 1)$ then $\mathcal{B}(M, p_1, p_2)$ returns 0.

By Lemma 3.3.6, we know that B guesses correctly with non negligible advantage and the ϕ-hiding assumption is broken. ■

Lemma 3.3.6

$\mathcal{B}$ distinguishes between p_1 and p_2 with advantage $\boldsymbol{Adv}(\mathcal{A})/2$

Proof: By construction, if p_k divides $\phi(M)$ then $\mathcal{A}(q_k)$ is 1 with probability σ_k since q_k is a uniformly random query for s_k. Conversely, if p_k does not divide $\phi(M)$ then q_k is not a valid query. Let us assume that $\mathcal{A}$ returns 1 with some probability γ_k on the set of all such inputs.

By Lemma 3.3.5, we have $\gamma_1 \approx \gamma_2$ and we write γ their common approximate value. Now we have (recall that $\sigma_1 + \sigma_2 = 1$):

$$\begin{aligned}
&\Pr(t = (1,0)) = \frac{1}{2}(\gamma\sigma_2 + \sigma_1\gamma) = \gamma/2 \\
&\Pr(t = (0,1)) = (1-\gamma)/2 \\
&\Pr(t = (0,0)) = ((1-\sigma_1)\gamma + \sigma_2(1-\gamma))/2 \\
&\Pr(t = (1,1)) = (\sigma_1(1-\gamma) + (1-\sigma_2)\gamma)/2 \\
&\Pr(t = (0,0) \text{ or } t = (1,1)) \\
&\qquad\qquad = ((1-\sigma_1)\gamma + \sigma_2(1-\gamma) + \sigma_1(1-\gamma) + (1-\sigma_2)\gamma)/2 \\
&\qquad\qquad = (2\gamma + \sigma_1 + \sigma_2 - 2\gamma(\sigma_1 + \sigma_2))/2 = 1/2
\end{aligned} \tag{3.9}$$

Furthermore we have:

$$\begin{aligned}
\Pr(t = (1,0) | p_k \text{ divides } \phi(M)) &= \sigma_k\gamma \\
\Pr(t = (0,1) | p_k \text{ divides } \phi(M)) &= (1-\sigma_k)(1-\gamma)
\end{aligned} \tag{3.10}$$

From which we deduce:

$$\begin{aligned}\Pr(p_1 \text{ divides } \phi(M)|t=(1,0)) &= \frac{\sigma_1\gamma}{\sigma_1\gamma+\sigma_2\gamma} = \sigma_1\\ \Pr(p_2 \text{ divides } \phi(M)|t=(0,1)) &= \frac{(1-\sigma_2)(1-\gamma)}{(1-\sigma_1)(1-\gamma)+(1-\sigma_2)(1-\gamma)}\\ &= 1-\sigma_2 = \sigma_1\end{aligned} \tag{3.11}$$

So the probability that $\mathcal{B}$'s guess is correct is $\frac{1}{2}\cdot\frac{1}{2}+\frac{\gamma}{2}\sigma_1+\frac{1-\gamma}{2}\sigma_1 = 1/4+\sigma_1/2 = 1/2+\mathbf{Adv}(\mathcal{A})/2$. ■

3.3.3 Recovering the data associated with an entry containing a secret value

In this section we present two ways to recover the couple ($\texttt{outField}, data$) for the entry with the couple ($\texttt{inField}, s$) where s is the secret value selected by the client. The first way is a simple extension of the protocol described in the previous section but has a fairly high computational complexity while the second method uses any block retrieval PIR protocol to recover consecutive bits from the database efficiently.

First, the client executes the protocol of Section 3.3.2 to check if there exists an entry matching the search condition. Now, let's call L_o the length of the output field and $data_i = data_{i,1}\ldots data_{i,L_o}$ the bit decomposition of the output field value associated with the i-th entry of the database. For every $j \in \{1,\ldots,L_o\}$, the server computes $res_j = x^{\prod_{i=1}^{n} data_{i,j} Sum(v_i)}$ and sends it to the client. Now the client can test if res_j is a p-th root residue, in which case $data_{i,j} = 1$ ($data_{i,j} = 0$ otherwise). Complexities are similar to those detailed in section 3.3.2 repeated L_o times. Note that L_o may be large (for instance when recovering media files, $L_o > 2^{30}$ is possible) and a computation over the entire database has to be done for every bit sent to the user.

The other method is to use any traditional PIR block recovery algorithm. To apply such an algorithm in our situation, the user first needs to know the value of x such that entry number x in the database satisfies the conditions on the `inField` values. To do this, he can simply request it from the server using the first protocol described in this section, where only $\log_2 n$ bits have to be transferred. This is a reasonable overhead cost, especially when compared with the hassle of performing an entire SQL-like query entirely over PIR. Besides, PIR by Keywords protocols described by Chor *et al.* also require $\log_2 n$ rounds for standard data structures. In practice, this is the recommended solution.

See Section 2.3.3 for a short survey of possible candidates for the block recovery algorithm. Naturally, our own protocol described in Section 4 is a very fitting option. Such algorithms only have a reasonably small overhead compared to the non-private transfer of the same data when amortized over large enough blocks.

3.4 Testing for inequality

In this section we suppose the user wants to execute a query such as:

```
SELECT outField
FROM table
WHERE inField > s
```

on the database hosted by the server while keeping s secret. Other types of comparisons ($<$, $=$, $\geq$, $\leq$) are supported by applying some minor changes to the algorithm such as exchanging values.

3.4.1 Comparing a single entry to a secret value

In this section we show how the user can learn whether an entry of the database located on the server is equal, greater or less than a secret value he picks. By secret, we mean that the server does not learn the value. We only show how the scheme works for "greater than" comparisons, the other ones can be obtained with minor modifications.

Let us denote by L the bit length of `inField`. Recall that the secret value we compare to is $s = s_1 \cdots s_L$. Note that if s_L represents an integer with its Most Significant Bit first, then $s = \sum_{j=1}^{L} s_j 2^{L-j}$. Potentially, s could be any bit string however.

We assume that we are working in some set E (we will give examples of such sets later on).

The client chooses 3 distinct values $\alpha_j, \beta_j, \gamma_j \in E$ for every $j \in \{1, \cdots, L\}$ and another value $\alpha_0 \in E$.

He also selects 2 functions $F_{j,0}$ and $F_{j,1} : E \to E$ for every $j \in \{1, \cdots, L\}$ satisfying the following conditions ($\delta \in \{0, 1\}$):

$$F_{j,\delta}(\alpha_{j-1}) = \begin{cases} \alpha_j & \text{if } \delta = s_j \\ \beta_j & \text{if } \delta > s_j \\ \gamma_j & \text{if } \delta < s_j \end{cases} \tag{3.12}$$
$$F_{j,\delta}(\beta_{j-1}) = \beta_j \text{ if } j > 1$$
$$F_{j,\delta}(\gamma_{j-1}) = \gamma_j \text{ if } j > 1$$

To better understand this chapter, it is essential to have an intuition about what these values mean. If a function ultimately returns α_x then all bits up to x were equal between input field and secret value. If it returns β_x then the input field value was greater so far (and will forever stay greater). If it returns γ_x then it was smaller.

Now, the client sends all $2L$ functions to the server, along with the index i of the database entry to which he wishes to compare s. We call $v = v_1 \cdots v_L$ the value of `inField` for this i-th entry in the database.

The server computes and returns the function:

$$F_i = \bigcirc_{j=L}^{1} F_{j,v_j} = F_{L,v_L} \circ F_{L-1,v_{L-1}} \circ \cdots \circ F_{1,v_1} \tag{3.13}$$

The client then checks if $F_i(\alpha_0) = \beta_L$. If it is equal, then he deduces that $v > s$.

Note that this is not an efficient PIR protocol as simply having the server send the value in n bits would have been more efficient. However we use it on the whole database and gain significant per bit efficiency in Section 3.4.2.

Correctness

Clearly $F_i(\alpha_0) \in \{\alpha_L, \beta_L, \gamma_L\}$. If $F_i(\alpha_0) = \beta_L$ then $\exists j_0 \in \{1, \cdots, L\}$ such that $F_{j_0,v_{j_0}}(\alpha_{j_0-1}) = \beta_{j_0}$ and $\forall 1 \leq j < j_0$, $F_{j,v_j}(\alpha_{j-1}) = \alpha_j$.

This means that $\forall 1 \leq j < j_0$, $v_j = s_j$ and $v_{j_0} > s_{j_0}$, which is equivalent to saying that $v > s$.

We can show the same way that $F_i(\alpha_0) = \alpha_L$ if and only if $v = s$ and $F_i(\alpha_0) = \gamma_L$ if and only if $v < s$.

The scheme is deemed secure if an adversary is unable to efficiently distinguish between a query with secret parameter s_1 or s_2.

Matrix-Based Approach

The most natural way to instantiate this model is through the use of matrices. Each $F_{j,\delta}$ belongs to $\mathbb{Z}_p^{d \times d}$ for some values p and d.

Random vectors α_j, β_j and $\gamma_j \in \mathbb{Z}_p^d$ and random non-invertible matrices $F_{j,\delta} \in \mathbb{Z}_p^{d \times d}$ are picked such that the following conditions are satisfied (slight variant of the conditions 3.12):

$$\begin{aligned} F_{j,\delta}\alpha_{j-1} &= \begin{cases} \alpha_j & \text{if } \delta = s_j \\ \beta_j & \text{if } \delta > s_j \\ \gamma_j & \text{if } \delta < s_j \end{cases} \\ F_{j,\delta}\beta_{j-1} &= \beta_j \text{ if } j > 1 \\ F_{j,\delta}\gamma_{j-1} &= \gamma_j \text{ if } j > 1 \end{aligned} \tag{3.14}$$

In this case function composition is computed through matrix multiplication.

This approach is however not safe as an attacker can compute $K = Ker(F_{j+1,0}F_{j,0} - F_{j+1,1}F_{j,0})$ and learns information about $\beta_{j-1} \in K$ (identically, $\gamma_{j-1} \in K$). Naturally, the next step consists in studying nonlinear functions such as polynomial to prevent linear-algebra based attacks.

Polynomial-Based Approach

Here we present a function family that satisfy conditions 3.12. First let us consider a scheme based on univariate polynomials.

We can take $E = \mathbb{Z}_p$ for some large prime p and every $F_{j,\delta}$ is a truncated polynomial in $\mathbb{Z}_p[X]/\langle P(X)\rangle$ for some irreducible polynomial $P(X)$.

As is, this scheme is also vulnerable to simple attacks. We present a sketch for an attack on this weak scheme. An attacker can efficiently recover some information about the secret s since we have:

$$\begin{aligned} F_{3,0}(F_{2,0}(\beta_1)) &= F_{3,0}(F_{2,1}(\beta_1)) \\ &= F_{3,1}(F_{2,0}(\beta_1)) \\ &= F_{3,1}(F_{2,1}(\beta_1)) = \beta_3 \end{aligned} \tag{3.15}$$

This means that:

$$\begin{aligned} F_{3,0}(F_{2,0}(\beta_1)) - F_{3,0}(F_{2,1}(\beta_1)) = 0 \\ F_{3,1}(F_{2,0}(\beta_1)) - F_{3,1}(F_{2,1}(\beta_1)) = 0 \end{aligned} \tag{3.16}$$

So β_1 is a common root between polynomials $F_{3,0} \circ F_{2,0} - F_{3,0} \circ F_{2,1}$ and $F_{3,1} \circ F_{2,0} - F_{3,1} \circ F_{2,1}$. The same goes for γ_1. A polynomial GCD algorithm such as Euclid's algorithm will allow the attacker to recover those common roots. Then he can deduce β_i and γ_i for $j > 1$ by evaluating $F_{j,0}$ on β_{j-1}.

Either $F_{1,0}(\alpha_0) = \alpha_1$ or $F_{1,1}(\alpha_0) = \beta_1$ or $F_{1,0}(\alpha_0) = \gamma_1$ or $F_{1,1}(\alpha_0) = \alpha_1$. In the first case, $F_{2,0}(F_{1,1}(\alpha_0)) = \beta_2$ and $F_{2,1}(F_{1,1}(\alpha_0)) = \beta_2$.

We define $P(X) = F_{2,0}(F_{1,1}(X)) - \beta_2$ and $Q(X) = F_{2,1}(F_{1,1}(X)) - \beta_2$. Clearly, α_0 is a common root between P and Q which can be obtained through a polynomial GCD algorithm as before. At this point, the attacker can trivially recover the value s.

Now, observe that the attack succeeds because recovering the common root of two polynomials is an easy task. We can modify the scheme so that it uses multivariate polynomials instead and relate its security to a known hard problem.

Now, $E = \mathbb{Z}_p^m$ is the set of m-tuples of values in $\mathbb{Z}_p$. Each function $F_{j,\delta}$ can be seen as a set of m polynomials over m variables each modulo an ideal. We choose an ideal I generated by t polynomials $P_1, \cdots, P_t \in \mathbb{Z}_p[X_1, \cdots, X_m]$ such that:

- α_0 is a root of any P_i, $i \in \{1, \cdots, t\}$.
- The number of monomials in any $P \in \mathbb{Z}_p[X_1, \cdots, X_m]/I$ is a polynomial in m.

We call $S = (\mathbb{Z}_p[X_1, \cdots, X_m]/I)$. In this case, we write:

$$F_{j,\delta} = (F_{j,\delta,1}, \cdots, F_{j,\delta,m}) \in S^m \tag{3.17}$$

If $(a_1, \cdots, a_m) \in E$ then:

$$F_{j,\delta}(a_1, \cdots, a_m) = (F_{j,\delta,1}(a_1, \cdots, a_m), \cdots, F_{j,\delta,m}(a_1, \cdots, a_m)) \in E \tag{3.18}$$

Now to perform the attack sketched above, the attacker would have to solve a system of m equations over m variables, which is known to be a hard problem.

3.4.2 Detecting if there exists at least one entry in a database greater than a secret value

We now use the generic construction defined in the previous section and show how to apply it on the entire database. We assume that the database has n entries $v_1, \ldots, v_n$ for the input field and the user wants to know if at least one of those is greater than the secret s.

The server computes F_i for every $i \in \{1, \cdots, n\}$ and returns $F = \bigstar_{i=1}^{n} F_i$ for some adequate $\bigstar$ relation.

The client then checks if $F(\alpha_0)$ can be generated by a sum of values none of which being β_L. If it cannot, then he knows that there exists an $i \in \{1, \cdots, n\}$ such that $F_i(\alpha_0) = \beta_L$ and the client knows that at least one v_i is greater than s.

We now give an example based on the scheme detailed above.

Polynomial-Based Approach

As in the previous section, we are working with polynomials in $\mathbb{Z}_p[X_1, \cdots, X_m]$. Using the notations from the previous section, we set β_L such that $\gcd(\beta_L, p) \neq 1$ (we can naturally extended the definition with p a product of a small number of large primes). If p is prime, setting $\beta_L = 0$ also works. Similarly, we set α_L and γ_L such that $\gcd(\alpha_L, p) = \gcd(\gamma_L, p) = 1$.

We also define $F \bigstar G = FG$ (product of truncated polynomials). Now the client can easily test if β_L was a factor in $F(\alpha_0)$ by checking if $\gcd(F(\alpha_0), 1) = 1$ or not.

3.4.3 Finding the entry with the k-th highest value

We recall here that the value of the input field for the i-th entry in the database is an integer v_i on L bits. In this section, the client wants to find $Max(k) = v_{i_k}$ such that $|\{v_i | v_i > v_{i_k}\}| = k$ for every k such that $Max(k) > s$. In particular, $Max(0)$ is the maximum of all the values v_i, $Max(1)$ the second highest value, and so on.

One simple approach would be to ask the server to sort the database according to the values in the input field (note that the field name is public, only s is secret) and then use a standard block PIR protocol to recover the k-th entry in the database. However this approach will no longer work for complex queries involving conjunctions or disjunctions of different fields. Meanwhile the protocol we describe here will be naturally generalized in this scenario (see Section 3.5.3).

We now describe our new approach to find the value of $Max(k)$. First, the client finds the value of $Max(0)$. As in Section 3.4.2, the client checks if the database has at least one entry greater than s. If not, then the process is over. If such entries exist, the client knows that $s \leq Max(0) < 2^L$. He then finds the exact value of $Max(0)$ with a dichotomic binary search. Finding this number requires at most $\log_2 n$ executions of the protocol described in section 3.4.2.

Now let us assume that the client knows the value for $Max(k)$ for every $k < K$. We show that the client can find out the value of $Max(K)$. First, the client computes $x = \sum_{k<K} G(Max(k))$ where $G(v) = \bigcirc_{j=L}^{1} F_{j,v_j}$ is the function used by the server on a single entry. The client then executes the protocol of Section 3.4.2 and subtracts the locally computed value x from the result F sent by the server. If the value he ends up with shows the existence of an entry greater than s, then he knows that $s < Max(K) < Max(K-1)$ and he can find out the exact value of $Max(K)$ by binary search once again. Otherwise, he can stop the process as no other v_i is greater than s.

Denoting by K the number of elements in the database which verify the `inField` $> s$ condition of the query, $O(K \log_2 n)$ queries are performed and the communication complexity is $O(LK \log_2 n)$. Since KL bits of information are recovered, the communication per bit recovered is $O(\log_2 n)$. Note that in a non-PIR setting, this type of query based on a comparison would require $O(L)$ for the user and $O(LK)$ for the server anyway.

3.5 Testing combinations of several comparisons

In the previous section, we showed how to privately execute queries of the type:

```
SELECT outField
FROM table
WHERE CMP(inField,s) = c
```

where s and c are kept secret and

$$CMP(a,b) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a > b \\ -1 & \text{if } a < b \end{cases} \tag{3.19}$$

In this section we first extend this result to generic disjunction or conjunction queries of the form:

```
SELECT outField
FROM table
WHERE CMP(inField_1,s_1) = c_1
OR ...
OR CMP(inField_k,s_k) = c_k
```

```
SELECT outField
FROM table
WHERE CMP(inField_1,s_1) = c_1
AND ...
AND CMP(inField_k,s_k) = c_k
```

with all the s_i and a_i kept private. And then we show how to perform the query for any logical formula based on such field-value comparisons.

3.5.1 Generic conjunction of comparisons

Here, the client wants to find the list of rows r_i from the table where $\texttt{inField}_1 > s_1$ $\texttt{AND}\cdots$ $\texttt{AND inField}_f > s_f$. Note that the construction would be extremely similar for other comparisons so we only detail this one for simplicity's sake.

For every comparison, the client builds like in Section 3.4.1 a set of random functions $F^{\wedge}_{t,j,\delta}$ and random values $\alpha_{t,j}$, $\beta_{t,j}$ and $\gamma_{t,j}$ for $1 \leq t \leq f$, $1 \leq j \leq L$ and $\delta \in \{0,1\}$. He also picks a starting value α_0. The functions F have to satisfy the following conditions:

$$F^{\wedge}_{1,1,\delta}(\alpha_0) = \begin{cases} \alpha_{1,1} & \text{if } \delta = s_{1,1} \\ \beta_{1,1} & \text{if } \delta > s_{1,1} \\ \gamma_{1,1} & \text{if } \delta < s_{1,1} \end{cases} \tag{3.20}$$

For $1 < t \leq f$ and $j = 1$ (first bit for every comparison but the first one):

$$\begin{aligned} F^{\wedge}_{t,1,\delta}(\alpha_{t-1,L}) &= \gamma_{t,1} \\ F^{\wedge}_{t,1,\delta}(\beta_{t-1,L}) &= \begin{cases} \alpha_{t,1} & \text{if } \delta = s_{t,1} \\ \beta_{t,1} & \text{if } \delta > s_{t,1} \\ \gamma_{t,1} & \text{if } \delta < s_{t,1} \end{cases} \\ F^{\wedge}_{t,1,\delta}(\gamma_{t-1,L}) &= \gamma_{t,1} \end{aligned} \tag{3.21}$$

And for every $1 \leq t \leq f$ and $1 < j \leq L$ (subsequent bits):

$$\begin{aligned} F^{\wedge}_{t,j,\delta}(\alpha_{j-1}) &= \begin{cases} \alpha_{t,j} & \text{if } \delta = s_{t,j} \\ \beta_{t,j} & \text{if } \delta > s_{t,j} \\ \gamma_{t,j} & \text{if } \delta < s_{t,j} \end{cases} \\ F^{\wedge}_{t,j,\delta}(\beta_{j-1}) &= \beta_j \\ F^{\wedge}_{t,j,\delta}(\gamma_{j-1}) &= \gamma_j \end{aligned} \tag{3.22}$$

The client then sends all the functions to the server along with the ordered list of input field names that correspond to the lists.

We write $v_{i,t,j}$ the j-th bit of the value of input field number t for the i-th entry in the database. The server computes

$$\begin{aligned} F_i^{\wedge} &= \bigcirc_{t=f}^{1} \bigcirc_{j=L}^{1} F_{t,j,v_{i,t,j}}^{\wedge} \\ F^{\wedge} &= \bigstar_{i=1}^{n} F_i^{\wedge} \end{aligned} \tag{3.23}$$

Then he sends F to the client. Now the client checks if $F^{\wedge}(\alpha_0)$ can be expressed without $\beta_{f,L}$ as a term (see Section 3.4.2). If it cannot then not a single entry of the database satisfied all the conditions. Otherwise, the client knows that at least an entry satisfying all the conditions at once exists. He can recover it using the protocols detailed in section 3.4.3, generalized to f-tuples of values.

Concretely, he finds through binary search the f-tuple of input field values for which $F^{\wedge}(\alpha_0)$ has $\beta_{t,L}$ as factor and such that increasing any of those input values by 1 would provide an $F^{\wedge}(\alpha_0)$ that does not. This gives the client the values associated with the greatest entry and requires at most $f \log_2 n$ queries. He then iterates with the second greatest entry and so on until no entry satisfies the set of conditions. Once he knows the exact f-tuples associated to the values he is interested in, he recovers the values in the output field with the protocol of section 3.3.3 or any block PIR scheme.

3.5.2 Generic disjunction of comparisons

Here, the client wants to find the list of rows r_j from the table where $\texttt{inField}_1 > s_1$ $\texttt{OR} \cdots$ $\texttt{OR}$ $\texttt{inField}_f > s_f$. Like previously, the construction would be extremely similar for other comparisons so we only detail one for simplicity's sake. As in the previous section, the client builds the functions $F_{t,j,\delta}^{\vee}$ in the following manner.

$$F_{1,1,\delta}^{\vee}(\alpha_0) = \begin{cases} \alpha_{1,1} & \text{if } \delta = s_{1,1} \\ \beta_{1,1} & \text{if } \delta > s_{1,1} \\ \gamma_{1,1} & \text{if } \delta < s_{1,1} \end{cases} \tag{3.24}$$

For $1 < t \leq f$ and $j = 1$ (first bit for every comparison but the first one):

$$\begin{aligned}
F^{\vee}_{t,1,\delta}(\alpha_{t-1,L}) &= \begin{cases} \alpha_{t,1} & \text{if } \delta = s_{t,1} \\ \beta_{t,1} & \text{if } \delta > s_{t,1} \\ \gamma_{t,1} & \text{if } \delta < s_{t,1} \end{cases} \\
F^{\vee}_{t,1,\delta}(\beta_{t-1,L}) &= \beta_{t,1} \qquad\qquad (3.25) \\
F^{\vee}_{t,1,\delta}(\gamma_{t-1,L}) &= \begin{cases} \alpha_{t,1} & \text{if } \delta = s_{t,1} \\ \beta_{t,1} & \text{if } \delta > s_{t,1} \\ \gamma_{t,1} & \text{if } \delta < s_{t,1} \end{cases}
\end{aligned}$$

And for every $1 \leq t \leq f$ and $1 < j \leq L$ (subsequent bits):

$$\begin{aligned}
F^{\vee}_{t,j,\delta}(\alpha_{j-1}) &= \begin{cases} \alpha_{t,j} & \text{if } \delta = s_{t,j} \\ \beta_{t,j} & \text{if } \delta > s_{t,j} \\ \gamma_{t,j} & \text{if } \delta < s_{t,j} \end{cases} \qquad (3.26) \\
F^{\vee}_{t,j,\delta}(\beta_{j-1}) &= \beta_j \\
F^{\vee}_{t,j,\delta}(\gamma_{j-1}) &= \gamma_j
\end{aligned}$$

The rest of the protocol is identical to the conjunction case (Section 3.5.1). It is clear that if $F^{\vee}(\alpha_0) = \beta_{t,L}$ then at least one entry in the database satisfies at least one of the conditions.

Note that the protocols for conjunctions and disjunctions were indistinguishable from the server. As such, we can also assume that the nature of the query itself (in this case, conjunction or disjunction) is kept private from the server.

3.5.3 Arbitrary combination of conjunctions and disjunctions

This time the client sends the server a tree where the inner vertices are labelled by `AND` or `OR` and the leaves are conditions of the form `CMP(inField,` s`) =` c where the value for every s and c is kept secret from the server.

First, the leaves are given an index obtained by ordering them according to a depth-first search. The index goes from 1 to f where f is the total number of conditions. We call c_t the value of c in the label `CMP(inField,` s`) =` c of leaf t. Now, the client attaches to each leaf a set of functions $F_{t,j,\delta}$ where t is the leaf index, j goes from 1 to L and $\delta \in \{0, 1\}$.

We define NextOperator(t) as the label found on the "first" common ancestor in the tree between leaves indexed by t and $t+1$. The first common ancestor between two vertices v_1 and v_2 is the common ancestor v of v_1 and v_2 such that no child of v is an ancestor for both v_1 and v_2.

We write s_t the value s on the label for leaf with index t. $s_t = s_{t,1} \cdots s_{t,L}$ is the bit decomposition of s_t.

As before, α_0 is a random value and so are $\alpha_{t,j}$, $\beta_{t,j}$ and $\gamma_{t,j}$ for $t \in \{1, \cdots, f\}$ and $j \in \{1, \cdots, L\}$.

$$F_{1,1,\delta}(\alpha_0) = \begin{cases} \alpha_{1,1} & \text{if } \delta = s_{1,1} \\ \beta_{1,1} & \text{if } \delta > s_{1,1} \\ \gamma_{1,1} & \text{if } \delta < s_{1,1} \end{cases} \tag{3.27}$$

We assume for simplicity's sake that $c_t = 1$ (ie, we are doing "greater than" comparisons). If not, we can just switch the target value $\beta_{t,L}$ with either $\alpha_{t,L}$ or $\gamma_{t,L}$.

For $1 < t \leq f$, $j = 1$:

- If NextOperator($t-1$) = `AND` then the same conditions defined for $F^{\wedge}_{t,1,\delta}$ apply to $F_{t,1,\delta}$
- If NextOperator($t-1$) = `OR` then the same conditions defined for $F^{\vee}_{t,1,\delta}$ apply to $F_{t,1,\delta}$.

And for every $1 \leq t \leq f$ and $1 < j \leq L$ (subsequent bits):

$$\begin{aligned} F_{t,j,\delta}(\alpha_{j-1}) &= \begin{cases} \alpha_{t,j} & \text{if } \delta = s_{t,j} \\ \beta_{t,j} & \text{if } \delta > s_{t,j} \\ \gamma_{t,j} & \text{if } \delta < s_{t,j} \end{cases} \\ F_{t,j,\delta}(\beta_{j-1}) &= \beta_j \\ F_{t,j,\delta}(\gamma_{j-1}) &= \gamma_j \end{aligned} \tag{3.28}$$

Then as before the functions $F_{t,j,\delta}$ are sent to the server, along with the ordered list of input field names corresponding to the leaf ordering. There is no need to send the tree itself to the server who learns nothing but the number of comparisons. The server computes

$$\begin{aligned} F_i &= \bigcirc_{t=f}^{1} \bigcirc_{j=L}^{1} F_{t,j,v_{i,t,j}} \\ F &= \mathop{\bigstar}_{i=1}^{n} F_i \end{aligned} \tag{3.29}$$

As such, the user can detect if at least one entry verifies the formula in a single query to the server. He then recovers all the data needed as shown previously.

3.6 Other Expressions

3.6.1 Single Character Wildcard

Another very common operator in SQL-like queries is the `LIKE` operator which allows comparisons where some characters are wildcards. There are two kinds of wildcards: a single character or any number of characters. In this section, we show how the scheme from the previous section can be modified very easily to allow for single character wildcards _ (underscore).

Suppose we want to execute a query of the form:

```
SELECT outField
FROM table
WHERE inField LIKE s
```

Where $s = s_1 \cdots s_L$ with each $s_j \in \{0, 1, \_\}$. The query is expected to return the `outField` values of every entry in the database such that the corresponding `inField` has s_j has its j-th bit when $s_j \in \{0, 1\}$ (when $s_j = \_$ the j-th bit value of `inField` does not matter).

Exactly as before, the client picks random α_0, α_j, β_j and γ_j and defines a set of function $F_{j,\delta}$ for $1 \leq j \leq L$ and $\delta \in \{0, 1\}$ such that the following conditions are satisfied:

$$\begin{aligned} F_{j,\delta}(\alpha_{j-1}) &= \alpha_j \text{ if } s_j = \_ \\ F_{j,\delta}(\beta_{j-1}) &= \beta_j \text{ if } s_j = \_ \\ F_{j,\delta}(\gamma_{j-1}) &= \gamma_j \text{ if } s_j = \_ \end{aligned} \tag{3.30}$$

And if $s_j \in \{0, 1\}$:

$$\begin{aligned} F_{j,\delta}(\alpha_{j-1}) &= \begin{cases} \alpha_j & \text{if } \delta = s_j \\ \beta_j & \text{if } \delta > s_j \\ \gamma_j & \text{if } \delta < s_j \end{cases} \\ F_{j,\delta}(\beta_{j-1}) &= \beta_j \text{ if } j > 1 \\ F_{j,\delta}(\gamma_{j-1}) &= \gamma_j \text{ if } j > 1 \end{aligned} \tag{3.31}$$

Then the server computes F as described many times before and sends it to the client. The client then tests if $F(\alpha_0)$ contains α_L as a term (depending on implementation). If it does then at least one entry in the database satisfies the `LIKE` condition.

We can then naturally extend the scheme from Section 3.5.3 to also allow `LIKE` conditions using single character wildcards as predicates.

3.6.2 Regular Languages

The next natural step in extending the expressivity of the queries supported by this approach is to add support for the second wildcard operator `%` which is a substitute for zero or more characters and is used in `LIKE` comparisons. Combined with all the other query types we have described beforehand, this means that our construction will be able to recognize any regular language. We thus directly prove this stronger statement.

We define the set $\mathcal{R}$ of regular languages over a finite alphabet $\Sigma = \{\sigma_1, \cdots, \sigma_T\}$ as the smallest set containing $\emptyset$, $\{\sigma_t\}$ for all $\sigma_t \in \Sigma$ and stable under union, concatenation and Kleene Star $*$. Let us consider a regular language $\mathcal{L} \in \mathcal{R}$ over an alphabet Σ. We show that we can privately perform any query of the following form:

```
SELECT outField
FROM table
WHERE inField ∈ ℒ
```

Proof: Since $\mathcal{L}$ is regular, there exists a deterministic finite complete automaton $\mathcal{A}$ recognizing $\mathcal{L}$. We call S its set of states, with $S = \{S_k, 1 \leq k \leq K\}$. Without loss of generality, we assume there is a single initial state S_{k_i} and a single final state $S_{k_f} \in S$. The transition function is $Trans : S \times \Sigma \to S$. We also write $\Sigma = \{\sigma_1, \cdots, \sigma_T\}$.

To each state is associated an element $e(S_k) \in E$ such that $e(S_i) = e(S_j) \implies i = j$. We define the family of functions $F_{\sigma_t} : E \to E$ as follows.

For every σ_t, define $F_{\sigma_t} : E \to E$ random function such that $F_{\sigma_t}(e(S_k)) = e(Trans(e(S_k), \sigma_t))$ for every $1 \leq k \leq K$.

Now let v_i be the value of `inField` on the i-th entry. Since $v_i \in \Sigma^*$ we can write $v_i = v_{i,1} \cdots v_{i,L}$ where $v_{i,j} \in \Sigma$. Now the server computes:

$$\begin{aligned} F_i &= \bigcirc_{j=L}^{1} F_{v_{i,j}} \\ F &= \bigstar_{i=1}^{n} F_i \end{aligned} \tag{3.32}$$

Note that if $F_i(e(S_{k_i})) = e(S_{k_f})$ then $v_i \in \mathcal{L}$ since $\mathcal{A}$ is complete. ■

3.7 Discussion and Comparison

When discussing the results obtained in the previous sections, we have to consider protocols working if the same, or at least a similar, setting. While SQL-oriented PIR schemes have been designed in the past [OG10], they did not reach the same level of query expressivity, instead relying on higher data transfer and client-side filtering. Furthermore, Olumofin's approach worked in a multi-server setting which gives greater freedom to the designer at the cost of an inherent requirement on the client's part to trust the true independence of several third party entities. When it comes to PIR, most if not all protocols run in linear time. Comparing the efficiency of different schemes often cannot be achieved without actual implementation. In this scenario, the two schemes are near incomparable due to the fundamental differences between single and multi-server PIR protocols.

Now, the other main approach in this setting is the generic search with keywords [BCN98] or $\mathcal{PERKY}$. Since it can be used with any PIR protocol to perform the actual queries, it is a fair comparison with our proposal. In particular, they can both be instantiated in a single-server setting. $\mathcal{PERKY}$ can only perform equality queries and further filtering has to be executed client-side. For a query that would not include any equality as one of its conditions, this implies that every single entry would have to be sent over to the client. Comparatively, the schemes we described are able to realize a much wider range of `SELECT` queries server-side, minimizing communication costs. In both cases, the goal of the framework is to find the physical location of the data to be recovered. Once this location is known, the actual data recovery is done through a standard PIR protocol. We are therefore only interested in this first step.

There is a fundamental difference between Chor's and our approach. $\mathcal{PERKY}$ is a method to read and browse a remote data structure privately. It simulates reading the structure locally by sending a new query for each read operation. With clever organization, this usually means a logarithmic number of exchanges between client and server have to take place. Our approach is unique in that the server itself browses its data while performing a computation requested by the client. This minimizes the number of transactions and showcases the possibility of private function execution over public data.

3.8 Conclusion

We have introduced a high-level algorithm relying on the ϕ-hiding assumption, a well-studied security assumption to perform simple `SELECT` queries based on equality tests. It is able to detect the existence of results in a single query and can recover said results in another query through any PIR protocol. This translates into a efficiency improvement over previous options for complex databases in single-server settings.

We have also introduced a system allowing the execution of much more complex queries encompassing a great portion of queries actually used on real life databases. The framework could be instantiated on different families, offering different options and opening for additional research.

While the scheme performing equality tests, and others such as [BCPT07] computing Hamming weight are based on well understood security assumptions, we could not provide the same level of confidence for arbitrary comparisons. Conceptually, any scheme able to compute the evaluation of a private automaton on a public word (the input bits) has to rely on some noncommutative primitive. This is why polynomial composition and matrix multiplication were suggested.

Cryptography over non-abelian algebraic structures has received some attention for a long time but no truly trusted and efficient scheme currently exists based on such structures. If new schemes were to be devised in the future, they might be able to be used directly as instances for this construction.

On another level, the possibility of executing partially encrypted functions over unencrypted data is a very exciting idea which has received little attention compared to the notion of encrypted functions over encrypted data (see [Bre01] for a somewhat similar construction to ours using finite automata). Extended PIR is a useful application for such primitives which justifies further research.

Chapter 4

Approximate-GCD-Based Private Information Retrieval

4.1 Introduction

Private Information Retrieval (PIR) protocols take place between one or more clients and potentially multiple servers. The goal is for the client to retrieve part of the data publicly available on the server(s) without them knowing which bits were retrieved. With many applications in privacy-sensitive fields such as biometrics or financial markets study, the notion is intuitive and comes with an obvious catch: a simple way to achieve it is to send over the entire database to the client. Chor *et al.* [CGKS95] formalized the notion in 1995 and provided a first non-trivial solution with much lower transfer requirements.

Rapidly, more solutions requiring less and less bits transferred [CMS99, GR05] were devised and later on research focused on decreasing the computations required [HHG13, IKOS04, LG15, WDDB06, YDDB08] as it became clear they were the bottleneck preventing practical PIR [Sio07]. The main constraint preventing widespread usage is the inherent information theoretical lower bound on the complexity. Clearly, if the server reads less than the entire database upon receiving a query, he gains information about which bits were not requested by the client. While there have been numerous attempts at solving this issue, they can essentially be sorted in three categories.

First, it is possible to improve this bound if we add some trusted element to the scheme, such as trusted hardware on the server, a trusted third party or additional copies of the database on servers trusted to be independent. This is for instance the approach taken in the development of Popcorn [GCM$^+$16] where one server hosts nothing but decryption keys while other servers, trusted not to collude with each other, host the actual encrypted data. This currently seems to be the most efficient approach providing privacy at a reasonable cost (Gupta *et al.* claim a 3.87 price increase [GCM$^+$16] when moving from a non private to a private video delivery service). Since PIR's ultimate goal is to reduce the trust a client has to put in third parties, this approach can however be controversial.

A second approach consists in answering several queries batched together at once with a single reading of the database. This solution only works in settings where many queries are performed at any point and only provides a constant factor improvement. This is for instance the approach taken by Lueks and Goldberg in [LG15]. While we show that the scheme we present is compatible with such techniques in Section 4.4.8, this is not our main proposal.

Finally, a third approach consists in modifying the database through a one-time preprocessing step and then effectively reading less bits during the online phase without compromising privacy thanks to a computational assumption. Beimel used this technique [BIM04] in a multi-server setting, and similar ideas were used in [MBFK16] and [GY07].

We argue that single-server PIR with preprocessing based on computational assumptions is the least flawed approach providing clients with the highest degree of privacy. This type of scheme would have its limitation be the computational assumption, but so do most widely used cryptographic schemes. To the best of our knowledge, there is so far no scheme that manages to answer queries while reading less bits than the entire database size in a completely generic setting.

We point out strong structural similarities between a sublinear protocol by Beimel et al. [BIM04] and a well performing and high-security multi-server protocol by Goldberg [Gol07]. This allows us to combine the best of each protocol and turn Goldberg's protocol into a potentially sublinear one with a noticeably lower computational cost.

However this protocol, like any other information theoretically secure protocol, does not maintain client privacy when every server cooperates. Alternatively, the protocol cannot be used when there is only one server (or a single entity owning all the servers) hosting the database. While Goldberg also suggested a computationally secure protocol that solves this issue, it is particularly inefficient and our preprocessing technique cannot be used on it.

We thus also introduce a brand new single-server PIR protocol. Its computational security is derived from the Approximate GCD assumption [HG01], well-studied in the field of Fully Homomorphic Encryption [vDGHV10]. Its low communication rate is obtained using the common construction in Goldberg's and Beimel *et al.*'s protocols we pointed out earlier. Finally, its low computation complexity is possible using the precomputation techniques we introduce in the first sections of this chapter.

This chapter is organized as follows. First, we show how Goldberg's protocol is a perfect fit to allow for some preprocessing. Then we show how to do the actual preprocessing as efficiently as possible. In the second part, we present the Approximate GCD assumption and how it can be combined with Goldberg's approach to result in a new single-server PIR protocol. Then we show how precomputations can be achieved on this new protocol. We suggest and justify parameters to keep the scheme safe.

4.1.1 Notations

In Sections 4.2 and 4.3, we work in the usual multi-server PIR setting where several servers are hosting the same copy of a public database of size n. This database is $(b_1, \cdots, b_n)$, $b_i \in \{0,1\}$. We call $t \geq 2$ the minimum number of servers the client has to query for the protocol to work. In Section 4.4, we work in a single-server setting ($t = 1$).

We use the standard notation $\delta_{x,y} = 1$ if $x = y$, 0 otherwise.

For a variable x and a distribution $\mathcal{D}$, we use the notation $x \leftarrow \mathcal{D}$ to indicate that x is picked at random from the distribution.

4.2 The 2-dimensional database construction

Motivation

In this section, we present the construction which will serve as the central building block for the next sections. It was independently used by Goldberg [Gol07] and to some extent by Beimel et al. [BIM04]. It is also found in Gasarch and Yerukhimovich's proposal [GY07]. Essentially, it is a very simple structure which can be declined in many different ways. It usually provides a communication complexity of roughly $\tilde{O}(\sqrt{n})$, along with a computational complexity of $\tilde{O}(\sqrt{n})$ on the client side. While this seems very high compared to other schemes achieving $O(n^\epsilon)$ for any $\epsilon > 0$ or even constant rate on average, we argue that this is more than enough. This is due to the fact that a query recovers an entire block of $O(\sqrt{n})$ bits, giving us a near optimal per-bit complexity when at least $\sqrt{n}$ bits need to be recovered. As it turns out, this value seems to correspond to what real-world systems would need. Indeed, for a medium-sized database of 2^{30} bits which would likely contain text data, a query recovering a couple kilobytes of data makes perfect sense. Similarly, in a large database of 2^{50} bits likely containing media files, recovering a megabyte or more of data seems standard.

Besides, its structure is the perfect candidate for precomputations as we will detail in Section 4.3. Finally, this structure can be used in recursive schemes as in Gasarch and Yerukhimovich's protocol [GY07].

4.2.1 Goldberg's Robust Protocol

Here we present Goldberg's version, on which we will be able to add a layer of precomputions in Section 4.3 and replace the security assumption in Section 4.4.

Recall that the database is $(b_1, \cdots, b_n)$, $b_i \in \{0, 1\}$. We assume the database can be split into $\texttt{nb}$ blocks of $\texttt{wpb}$ words of $\texttt{bpw}$ bits each, such that $\texttt{nb}\ \texttt{wpb}\ \texttt{bpw} = n$. Note that $\texttt{nb}$ stands for *number of blocks*, $\texttt{wpb}$ for *words per block* and $\texttt{bpw}$ for *bits per word.* If n cannot be decomposed in such a way, we can easily pad the database with a few extra bits to make it so. We set up the database as a 2-dimensional array of words where every word is characterized by two coordinates. In other words, we rewrite $\{b_i\}_{1\leq i\leq n}$ as $\{w_{i,j} | 1 \leq i \leq \texttt{nb}, 1 \leq j \leq \texttt{wpb}\}$ where each $w_{i,j}$ contains $\texttt{bpw}$ bits b_i in a meaningful manner. We also call block x the set of words $\{w_{x,j} | 1 \leq j \leq \texttt{wpb}\}$. All $t \geq 2$ servers are using the same conventions for this 2-dimensional database.

Suppose the client wants to recover block x while keeping x secret. We set $\mathbb{S}$ a field with at least t elements. The client selects $\texttt{nb}$ polynomials $P_1, \cdots, P_{\texttt{nb}}$ of degree $t-1$ with random coefficients in $\mathbb{S}$, except the constant term always set such that $P_x(0) = 1$ and $P_i(0) = 0$ when $i \neq x$. He also selects distinct values $\alpha_1, \cdots, \alpha_t \in \mathbb{S}$. These values can be anything and do not have to be kept secret. For instance $\alpha_k = k$ will work. The protocol has only one round, the client sends a request, the server responds and finally the client deduces the value of every bit in block x from the response.

During the first step, the client sends a request to every server involved in the protocol. Specifically, for $1 \leq k \leq t$, the client sends $(P_1(\alpha_k), \cdots, P_{\texttt{nb}}(\alpha_k))$ to server k.

We consider $w_{i,j}$ as an element of S. Now we define, for $1 \leq j \leq \texttt{wpb}$, the degree $t-1$ polynomial Q_j with coefficients in S.

$$Q_j = \sum_{i=1}^{\texttt{nb}} P_i w_{i,j} \tag{4.1}$$

During the second step, every server answers. Specifically, server k computes the following values and sends them to the client.

$$\left\{ Q_j(\alpha_k) = \sum_{i=1}^{\mathtt{nb}} P_i(\alpha_k) w_{i,j} \right\}_{1 \leq j \leq \mathtt{wpb}} \tag{4.2}$$

After receiving the answer of all t servers, the client knows $Q_j(\alpha_1), \cdots, Q_j(\alpha_t)$ for every $1 \leq j \leq \mathtt{wpb}$. Since Q_j is of degree $t-1$, the client can interpolate all of its coefficients and compute

$$Q_j(0) = \sum_{i=1}^{\mathtt{nb}} P_i(0) w_{i,j} = w_{x,j} \tag{4.3}$$

Thus the client knows the value of every word in block x.

4.2.2 Security

Informally, as a variant on Shamir's secret sharing scheme [Sha79], this scheme is information theoretically as secure as possible. It is a well known fact that a protocol cannot be information theoretically secure if every single server cooperates, thus the best we can hope for is information theoretical protection when at least one server does not collaborate, which is what is achieved here. More formally, the scheme is information theoretically secure against a coalition of up to $t-1$ servers. Clearly if it is safe against a coalition of $t-1$ servers, it is safe against any coalition of less than $t-1$ servers.

4.2.3 Communication complexity

The client sends $\mathtt{nb}$ elements of $\mathbb{S}$ to each server. Similarly, each server returns $\mathtt{wpb}$ elements of $\mathbb{S}$. Total communication is $t(\mathtt{nb} + \mathtt{wpb}) \log |\mathbb{S}|$, which becomes $2t\sqrt{n} \log |\mathbb{S}| = O(t \log t \sqrt{n})$ when $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, $\mathtt{bpw} = 1$ and $\mathbb{S}$ has exactly t elements. Note that the computational complexity is still linear in the size of the database here since every b_i has to be read by the servers. The seemingly high communication compared to protocols with polylogarithmic [CMS99] or constant [GR05] communication rate is actually a non-issue, since asymptotically (and also in practice [Sio07]), the server's computational time will be much higher than the data transmission time.

4.3 Adding precomputations to Goldberg's Robust Protocol

In this section, we will show how we can improve the protocol from Section 4.2 by adding an offline step only executed once. First we use the exact same approach as Beimel *et al.*, which is not practical in our setting, and then we show how to modify it to make it practical and efficient.

4.3.1 Using precomputations

We now show we can improve Goldberg's protocol's running time. To reduce the computational complexity on the server side we add a precomputation step before the algorithm is run. A similar idea was introduced in 2004 in an article by Beimel [BIM04], which essentially used a variant of the 2-dimensional database structure described in Section 4.2. The concept of preprocessing to speed up computations can also be found in [GY07] where Gasarch and Yerukhimovich use it to reduce client-side computations. In our case, we focus on the server's side since a server is expected to answer many queries while most clients would only send a few. Regardless, this preprocessing step is only run once, has polynomial running time and can be done offline prior to any interaction with clients. Its complexity is thus irrelevant to the actual protocol running time.

Here is how we proceed. First, for every $1 \leq j \leq \texttt{wpb}$ (every word in a block), we partition the $m = \texttt{nb}\,\texttt{bpw}$ bits $b_{1,j}, \cdots, b_{m,j}$ into m/r disjoint sets of r bits each (without loss of generality, we assume r divides m). There are m/r such sets for every j, or n/r sets in total. We call these sets $C_{i,j}$, $1 \leq i \leq m/r$ and $1 \leq j \leq \texttt{wpb}$. For every $C_{i,j}$, the server computes all $|\mathbb{S}|^r$ possible linear combinations with coefficients in $\mathbb{S}$. That is, every $Pre(C, \Delta) = \sum_{d=1}^{r} \delta_d C[d]$ for any $\Delta = (\delta_1 \cdots \delta_r) \in \mathbb{S}^r$. This requires a total of $\frac{n}{r}|\mathbb{S}|^r \log_2 |\mathbb{S}|$ bits.

If we set $|\mathbb{S}| = t$ (which is the minimal number of distinct elements in $\mathbb{S}$, attained when $\texttt{bpw}$ is 1) and $r = \epsilon \log_2 n$ for some ϵ, this means a total of $\frac{n^{1+\epsilon \log_2 t} \log_2 t}{\epsilon \log_2 n}$ bits have to be precomputed and stored on the servers. This value is potentially very large if t is too large. We will solve this issue in Section 4.3.2. Computing one $Pre(C, \Delta)$ requires r operations and overall the preprocessing of the whole database requires $O(n^{1+\epsilon \log_2 t})$ operations.

During the online phase, each server only reads m/r elements of $\mathbb{S}$ to compute $Q_j(\alpha_k)$. Over all t servers, only $\frac{tn \log_2 |\mathbb{S}|}{r}$ bits are read. Note that the precomputed database should be ordered in such a way that, for a fixed i, all $Pre(C_{i,j}, \Delta)$ are stored consecutively. This way Δ, which is sent by the client, only has to be read by the server once. The nature of the algorithm (reusing the same Δ for every $1 \leq i \leq \texttt{nb}$) is what gives us a speed boost when different protocols would not benefit from the precomputations.

Once again, if we set $|\mathbb{S}| = t$, $\texttt{bpw} = 1$ and $r = \epsilon \log_2 n$ for some ϵ, a total of $\frac{nt \log_2 t}{\epsilon \log_2 n}$ bits are read during the online phase. If $t \log_2 t < \epsilon \log_2 n$, that means the server's computation is sublinear in the size of the database.

Communication complexity and security are exactly the same as detailed in Section 4.2. Indeed, a coalition of servers receives exactly the same information from a client as before. While this may seem obvious, it is interesting if one notes in particular that the potentially sublinear complexity would allow, from an information theoretical point of view, any individual server to distinguish some bits from the original database which are definitely not requested by the client (see the proof in Section 2.3.2). This could have potentially lead to an attack from a coalition of servers if it was not for the formal proof of Section 4.2. This shows yet again how fitting Goldberg's scheme is for this type of computational improvements through precomputation.

4.3.2 Decomposition over base 2

In this section, we present another approach to make precomputations even more efficient. This potentially improves Goldberg's complexity by several orders of magnitude.

Clearly the most limiting factor in the construction from the previous section (and in Beimel *et al.*'s approach [BIM04]) is the $|\mathbb{S}|^r$ factor. In some scenarios, we want t (and thus $|\mathbb{S}|$) to be large for redundancy purposes. Using words with more than a single bit can also be interesting. Besides, it is possible to create schemes based on a computational assumption which require very large groups $\mathbb{S}$, as we will show in Section 4.4. Alternatively, some implementations (including Goldberg's `percy++` [GDHH14]) set a fixed large $\mathbb{S}$ regardless of how many servers the client decides to use. In all of those situations, the space requirements on the server become prohibitive. We assume here that $\mathbb{S} = \mathbb{Z}_\ell$ for a possibly large ℓ over λ bits.

As before, in the first step the client sends to server k the following values.

$$(P_1(\alpha_k), \cdots, P_m(\alpha_k)) \tag{4.4}$$

We can however decompose those values as follows, since elements of $\mathbb{S}$ have a standard representation as integers.

$$P_i(\alpha_k) = \sum_{c=0}^{\lambda} P_{i,c} 2^c \text{ where } P_{i,c} \in \{0,1\} \tag{4.5}$$

Then the server's computation can be modified using the following equality.

$$\sum_{i=1}^{\texttt{nb}} P_i(\alpha_k) w_{i,j} = \sum_{c=0}^{\lambda} 2^c \sum_{i=1}^{\texttt{nb}} P_{i,c} w_{i,j} \tag{4.6}$$

Now once again we partition $\{1, \cdots, m\}$ as defined earlier into m/r disjoint sets $C_{i,j}$ of r elements each, where $1 \leq i \leq m/r$ and $1 \leq j \leq \texttt{wpb}$. For every such set C and every $\delta_1, \cdots, \delta_r \in \{0,1\}$, the server precomputes $\sum_{d=1}^{r} \delta_d C[d]$. Each precomputed value requires only $\log_2 r$ bits for storage.

If $r = \epsilon \log_2 n$, the following holds regarding the precomputed database size $|\mathcal{DB}'|$.

$$|\mathcal{DB}'| = \frac{m}{r} \texttt{ wpb } 2^r \log_2 r = n^{1+\epsilon} \frac{\log_2(\epsilon \log_2 n)}{\epsilon \log_2 n} < n^{1+\epsilon} \tag{4.7}$$

Thus less than $n^{1+\epsilon}$ bits are precomputed and stored in total. This value is independent of the size of $\mathbb{S}$ and thus indirectly independent of t. The one-time preprocessing requires $O(n^{1+\epsilon})$ elementary operations.

Now during the online phase, Q_j can be computed in $\frac{m}{r}\lambda$ operations. Overall, the t servers will return their results in $n\frac{t\lambda}{r} = n\frac{t\lambda}{\epsilon \log_2 n}$ operations, which is sublinear if $t\lambda < \epsilon \log_2 n$. Note that for real world values, $\log_2 r = \log_2(\epsilon \log_2 n)$ is small and summing m/r values should give a number that fits in a 64-bit register. This means that the online phase of the protocol can be efficiently implemented without using any large integer libraries for its most expensive step. Then only λ operations have to be performed using large integers. Because of this feature, the algorithm remains highly efficient even when $t\lambda > \epsilon \log_2 n$ as is the case with very large ℓ, t or small ϵ (to save space).

4.3.3 A note on Goldberg's computationally secure scheme

If all t servers collaborate, it is very easy for them to recover the secret x. Furthermore, we know it is impossible to make an information theoretically secure protocol with sublinear communication when every server cooperates. However it is still possible to make a computationally secure protocol in this situation as shown in [Gol07]. Note that because all of the servers can collaborate, we might as well consider that all t servers are the same, and that gives us a single server PIR protocol (although in some scenarios having several servers can also be convenient regardless of security concerns).

The basic idea behind the protocol is to encrypt the shares sent to the servers with an additively homomorphic scheme and to have the servers perform the computation on the ciphertexts. In practice, the Paillier cryptosystem is used. It has the interesting property that the product of ciphertexts is a ciphertext associated to the sum of the original plaintexts. It is not possible to add a precomputing step in this situation because the parameters (the modulus) must be chosen privately by the client. As such, all hypothetical precomputations would be done in $\mathbb{Z}$ instead of some $\mathbb{Z}/\ell\mathbb{Z}$ and reading a hypothetical precomputed product would require reading as many bits as reading the individual components of the product.

Instead, we can choose to use another additive homomorphic scheme like one based on the Approximate GCD problem. The scheme has however to be changed in several ways which we describe in the next section.

4.4 Single Server PIR Using the Approximate GCD Assumption

4.4.1 Computational assumptions

In this section we reuse the 2-dimensional construction of the database, but instead of securing it through Shamir's secret sharing scheme, we rely on the Approximate GCD assumption. The assumption, which has been the subject of widespread study thanks to its importance in the field of Fully Homomorphic Encryption, states that it is computationally hard to solve the Approximate GCD problem for sufficiently large parameters. This problem can be formulated as in Definition 4.4.1.

For any bit lengths λ_q and λ_ϵ and odd number p, we call $\mathcal{D}(\lambda_q, \lambda_\epsilon, p)$ the random distribution of values $z = pq + \epsilon$ where q has λ_q bits and ϵ has λ_ϵ bits.

In addition, for a bit $b \in \{0, 1\}$, we call $\mathcal{D}(\lambda_q, \lambda_\epsilon, p, b)$ the random distribution of values $pq + 2\epsilon + b$ where q has λ_q bits and ϵ has λ_ϵ bits.

Note that we will be using noise values $\epsilon \in \{0, \cdots, 2^{\lambda_\epsilon - 1}\}$. This makes notations easy as ϵ is also positive. However, this definition does not allow the subtraction of noise. In some other articles such as [vDGHV10] the noise is chosen uniformly around 0. In other words the noise ϵ' is picked from $\{-2^{\lambda_\epsilon - 1} - 1, \cdots, 2^{\lambda_\epsilon - 1}\}$. However the two notations are clearly equivalent as we can replace any $z = pq + \epsilon$ by $z' = z - 2^{\lambda_\epsilon - 1}$ before doing any sequence of k_a additions and k_s subtraction and then add $(k_s - k_a)2^{\lambda_\epsilon - 1}$ to the result.

Definition 4.4.1

(Approximate GCD) *Let $\{z_i\}_i \leftarrow \mathcal{D}(\lambda_q, \lambda_\epsilon, p)$ be a polynomially large collection of integers. Given this collection, output p.*

It was introduced in 2010 by Van Dijk *et al.* [vDGHV10] as a generalization of Howgrave-Graham's construction [HG01] from 2001. In the same paper, Van Dijk *et al.* also showed that the following problem can be reduced to the Approximate GCD problem.

Definition 4.4.2

(Somewhat Homomorphic Encryption) *Let $\{z_i\}_i \leftarrow \mathcal{D}(\lambda_q, \lambda_\epsilon, p)$ be a polynomially large collection of integers, $b \leftarrow \{0, 1\}$ a secret random bit and $z \leftarrow \mathcal{D}(\lambda_q, \lambda_\epsilon, p, b)$. Given $\{z_i\}_i$ and z, output b.*

Now here is how we construct the scheme.

As before we assume the database is $w_{i,j}$, $1 \leq i \leq \texttt{nb}$, $1 \leq j \leq \texttt{wpb}$, each $w_{i,j}$ containing `bpw` bits. Note that we have $\texttt{nb}\ \texttt{wpb}\ \texttt{bpw} = n$, the total bit size of the database. First we consider the convenient case where $\texttt{bpw} = 1$ (every word is a single bit).

Suppose the client wants to recover block x consisting of $\{w_{x,j}\}_{1 \leq j \leq \texttt{wpb}}$. The client picks a large random odd number p over $\lambda_p > \lambda_\epsilon$ which will be its secret key. He selects `nb` random $z_i = pq_i + 2\epsilon_i + \delta_{i,x} \leftarrow \mathcal{D}(\lambda_q, \lambda_\epsilon, \delta_{i,x})$.

Then for every j from 1 to `wpb`, the server computes the inner product $res_j = \sum_{i=1}^{\texttt{nb}} b_{i,j} z_i$. He then sends $\{res_j\}_{1 \leq j \leq \texttt{wpb}}$ to the client.

For every res_j received, if $\sum_{i=1}^{\texttt{nb}} 2\epsilon_i < p - 1$, the client can compute the following:

$$\begin{aligned}(res_j \bmod p) \bmod 2 &= (\sum_{i=1}^{\texttt{nb}} b_{i,j}(pq_i + 2\epsilon_i + \delta_{i,x}) \bmod p) \bmod 2 \\ &= \sum_{i=1}^{\texttt{nb}} b_{i,j}(2\epsilon_i + \delta_{i,x}) \bmod 2 \\ &= b_{x,j}\end{aligned} \tag{4.8}$$

As such, and so long as $\sum_{i=1}^{\mathtt{nb}} 2\epsilon_i < p-1$, he retrieves one bit of information $b_{x,j}$ for every $j \in \{1, \cdots, \mathtt{wpb}\}$ which once combined gives him the value of the entire block x.

It is important to note that this construction is somewhat homomorphic and has been used to create fully homomorphic encryption [vDGHV10], but here we only care about the additive property of the scheme. This is very important because without multiplications, the noise ϵ will progress very slowly and we realistically do not have to worry about it becoming too large. Also, our construction is unrelated to other PIR protocols based off generic somewhat or fully homomorphic encryption systems like Yi et al.'s [YKPB13] or Boneh et al.'s [BGH$^+$13] (see Section 2.3.3).

4.4.2 Complexity

First let us look at the communication complexity. The client sends nb values z_i and the server returns wpb values res_j. We call λ_p the bit size of the secret p and λ_q the bit size of the q_i values. We also write $\lambda = \lambda_p + \lambda_q$. The actual values of these security parameters will be discussed in section 4.4.4. Each z_i is thus λ bits long. Since res_j is essentially a sum of at most nb z_i values, its bit size is $\lambda + \log_2 \mathtt{nb}$.

The overall communication of the protocol is $\lambda\mathtt{nb} + \mathtt{wpb}(\lambda + \log_2 \mathtt{nb})$ to retrieve a block of wpb bits. When $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, and since $\log_2 \mathtt{nb} \ll \lambda$, the communication is $O(\lambda\sqrt{n})$ or $O(\lambda)$ per bit recovered.

Now let us detail the computational complexity. Multiplying a λ_p-bit integer with a λ_q-bit integer can be done with complexity $\lambda_p\lambda_q < \lambda^2$ or less depending on representations. The same applies for a modulo operation on λ-bit input values. Adding two λ-bit integers requires λ elementary operations. Now, the client performs $O(\mathtt{nb})$ multiplications to send the query, the server in turn performs $O(\mathtt{nb}\ \mathtt{wpb})$ additions to execute it. Finally the client performs $O(\mathtt{wpb})$ modulo computations to recover the block values from the server's reply.

The overall computational complexity of the protocol is $O(\mathtt{nb}\ \mathtt{wpb}\lambda)$. When $\mathtt{nb} = \mathtt{wpb} = \sqrt{n}$, which is the optimal value, this becomes $O(\lambda n)$.

4.4.3 Precomputations

As the scheme uses the 2-dimensional structure of the database, we can use precomputations in the exact same way we described in section 4.3.2. We write $z_i = \sum_{k=0}^{\lambda-1} z_{i,k} 2^k$, where $z_{i,k} \in \{0,1\}$. The server selects a precomputing parameter r and computes every possible sum of r consecutive words in the database. The client sends the $z_{i,k}$ values by groups of r bits for a fixed k. Now the server can compute the res_j values r times faster than previously by working with small integers and only performing λ large integer operations per res_j. See section 4.3.2 for details on this technique.

Communications are unchanged (the client sends the same amount of bits, simply changing their order) at $O(\lambda)$ bits transmitted for every bit recovered.

Computations are unchanged on the client side. On the server side, each r_j requires $\lambda\, \mathtt{nb}/r$ small integer operations and λ large integer operations. Overall computational complexity is $O(\lambda n/r + \lambda^2)$, which is asymptotically a r times improvement over the standard version.

4.4.4 Security

The security of a PIR scheme is defined by the following problem. Given two client queries for blocks x_1 and x_2 respectively, it should be computationally hard to distinguish them. In this scheme, this means that the server must distinguish two queries $\mathcal{q}_{x_1}$ and $\mathcal{q}_{x_2}$.

To simplify notations, we write $\mathcal{D}(\lambda, p, b)$ the distribution $\mathcal{D}(\lambda_q, \lambda_\epsilon, p, b)$ with $\lambda_q = \lambda - \log_2 p$ and $\lambda_\epsilon = \log_2 p - \log_2 \mathtt{nb}$. $\mathcal{q}_x$ is defined as:

$$\mathcal{q}_x = \{z_i | z_i \leftarrow \mathcal{D}(\lambda, p, 0) \text{ when } i \neq x, z_i \leftarrow \mathcal{D}(\lambda, p, 1) \text{ otherwise}\} \qquad (4.9)$$

Let us call $\mathcal{Q}_x$ the distribution of all possible queries $\mathcal{q}_x$. Now we can show that if an attacker can indeed distinguish these two queries when given access to polynomially many samples from $\mathcal{D}(\lambda, p, 0)$, it can break the Somewhat Homomorphic Encryption (Definition 4.4.2) and thus solve the Approximate GCD Problem (Definition 4.4.1).

Theorem 4.4.3

Given access to polynomially many values of $\mathcal{D}(\lambda, p, 0)$, if there exists a PPT algorithm distinguishing two queries for distinct blocks of a database with non-negligible advantage, then there exists a PPT algorithm able to recover the value p given these samples.

Proof: Let us assume there exists a distinguishing algorithm $\mathcal{A}(q)$ that returns 1 with probability σ if $q \in \mathcal{Q}_{x_1}$ and returns 1 with probability $\sigma + \alpha$ if $q \in \mathcal{Q}_{x_2}$. Now given an encryption $e = pq + 2\epsilon + b$ of a secret bit b, we build an algorithm $\mathcal{B}(e)$ that recovers b with probability at least $1/2 + \alpha/4$.

First, with probability $\alpha/2$, we return 0 and stop immediately. Otherwise, we draw $r \leftarrow \mathcal{D}(\lambda, p, 0)$ and $\mathtt{nb} - 2$ random elements from $\mathcal{D}(\lambda, p, 0)$. We generate a query q with the random elements in every position but x_1 and x_2. We randomly pick x_1 or x_2 and place e in the picked position, r in the other position. We then return $\mathcal{A}(q)$ if the picked position was x_1, $1 - \mathcal{A}(q)$ otherwise.

If b was equal to 0, then q contains only random encryptions of 0 and does not belong to some $\mathcal{Q}_x$. In this case, let us say $\mathcal{A}(q)$ returns 1 with probability σ'. Then $\mathcal{B}(e)$ returns 1 with probability

$$\begin{aligned} Pr(B(e) = 1|b = 0) &= (1 - \alpha/2)(1/2 \cdot \sigma' + 1/2 \cdot (1 - \sigma')) \\ &= 1/2 - \alpha/4 \end{aligned} \tag{4.10}$$

Now if b was equal to 1, then $\mathcal{B}(e)$ returns 1 with probability

$$\begin{aligned} Pr(B(e) = 1|b = 1) &= (1 - \alpha/2)(1/2 \cdot (\sigma + \alpha) + 1/2(1 - \sigma)) \\ &= 1/2 + \alpha/2(3/4 - \alpha/4) \\ &\geq 1/2 + \alpha/4 \text{ since } 0 < \alpha \leq 1 \end{aligned} \tag{4.11}$$

Then we use the result from Van Dijk et al. [vDGHV10] to show the reduction from recovering b to recovering the secret p and breaking the AGCD assumption. ■

Note that here we had to assume that the attacker has access to random encryptions of 0. This is actually a very fair assumption since the most common use of homomorphic encryption is cloud storage and computing. In such a scenario, it would be common for a server hosting user-submitted data to know some metadata about the plaintext which would include some always-0 bits.

4.4.5 Parameters Selection

Van Dijk *et al.* [vDGHV10] describe how to select parameters so that the Approximate GCD assumption is secure against known attacks. Let us set λ_s some security parameter (*i.e.*, the scheme should achieve λ_s bits of security).

Many efficient attacks rely on the assumption that an exact multiple of p is known, which also known as Partial AGCD problem. Brute force attacks on the noise value mean that λ_ϵ has to be at least λ_s when an exact multiple of p is known.

In our construction, no exact multiple is used. In this case, Chen and Nguyen [CN12] show that the scheme can be broken in $\tilde{O}(2^{3\lambda_\epsilon/2})$ and Coron *et al.* showed an improved heuristic attack in $\tilde{O}(2^{\lambda_\epsilon})$.

Most effective are lattice-based attacks. Writing $z_i = pq_i + \epsilon_i$, Van Dijk *et al.* describe an attack by applying the lattice reduction algorithm LLL [LLL] to the following matrix:

$$M = \begin{pmatrix} 2^{\lambda_\epsilon} & z_1 & z_2 & \cdots & z_t \\ & -z_0 & & & \\ & & -z_0 & & \\ & & & \ddots & \\ & & & & -z_0 \end{pmatrix} \tag{4.12}$$

We write:

$$\begin{aligned} v &= (q_0, q_1, \cdots, q_t) \cdot M \\ &= (q_0 2^{\lambda_\epsilon}, z_0 q_0 (\frac{z_1}{z_0} - \frac{q_1}{q_0}), \cdots, z_0 q_0 (\frac{z_t}{z_0} - \frac{q_t}{q_0})) \end{aligned} \tag{4.13}$$

The attack succeeds if v (of size about $2^{\lambda_q+\lambda_\epsilon}\sqrt{t+1}$) is the shortest vector in the lattice and if the lattice reduction finds it. It is likely to be the shortest when $t \geq \lambda/\lambda_p$. On the other hand, since there exist exponentially many vectors of size $2^{\lambda}\sqrt{t+1}$, the reduction has to find a better than $2^{\lambda_p-\lambda_\epsilon}$ approximation. Since in our case we have $\lambda_p \approx \lambda_\epsilon + \log_2 \mathtt{nb}$, that means it would have to return a better-than-nb approximation.

Using [Sch03] and its improvement [GHGKN06], we know that it takes time roughly $2^{t/k}$ to get a 2^k approximation. Since $t \geq \lambda/\lambda_p$, we need $t/k > \frac{\lambda}{\lambda_p \log_2 \mathtt{nb}} > \lambda_s$ which means $\lambda > \lambda_s \lambda_p \log_2 \mathtt{nb}$. Note that this is a very conservative approximation as we would likely need $t \gg \lambda/\lambda_p$ to guarantee v is the shortest vector and if $t = C\lambda/\lambda_p$ for a large C, then the condition becomes $\lambda > \lambda_s \lambda_p \log_2 \mathtt{nb}/C$

4.4.6 Multiple Bits Words

The process can easily be modified to recover words of more than 1 bit of data per word at little to no additional cost. Essentially, instead of picking the values z_i as $pq_i + 2\epsilon_i + \delta_{i,x}$, we can pick $z_i = pq_i + 2^{\mathtt{bpw}}\epsilon + \delta_{i,x}$ where bpw is the number of bits per word. The condition on the ϵ_i values becomes $2^{\mathtt{bpw}} \sum_{i=1}^{\mathtt{nb}} \epsilon_i < p/2^{\mathtt{bpw}} \iff \sum_{i=1}^{\mathtt{nb}} \epsilon_i < p/2^{2\mathtt{bpw}}$.

For instance, if λ_p is 160 bits and we process databases with less than 2^{30} blocks, we could pick $\lambda_\epsilon = 80$ bits which would allow $\texttt{bpw}$ to be as high as 25 bits per word.

The server's response in this case is unchanged, $res_j = \sum_{i=1}^{\texttt{nb}} z_i w_{i,j}$ where each $w_{i,j}$ is a word of $\texttt{bpw}$ bits. The client then recovers block x consisting of all the $w_{x,j}$ by computing $w_{x,j} = (res_j \bmod p) \bmod 2^{\texttt{bpw}}$.

The overall communication cost of the protocol is now $\texttt{nb}\ \lambda + \texttt{wpb}(\lambda + \texttt{bpw} + \log_2 \texttt{nb})$ but we now retrieve a block of $\texttt{wpb}\ \texttt{bpw}$ bits. Besides, we have $\texttt{bpw}\ \texttt{wpb}\ \texttt{nb} = n$ and $\texttt{nb} \ll 2^\lambda$, which means that when $\texttt{nb} = \texttt{wpb} = \sqrt{n/\texttt{bpw}}$ the communication cost becomes $O(\sqrt{\frac{n}{\texttt{bpw}}}(2\lambda + \log_2 \texttt{nb} + \texttt{bpw}))$. Since we recover $\texttt{wpb}\ \texttt{bpw} = \sqrt{n\ \texttt{bpw}}$ bits, per-bit-recovered complexity is $O(\frac{2\lambda+\log_2 \texttt{nb}}{\texttt{bpw}} + 1)$ which is roughly $O(\lambda/\texttt{bpw})$ (note that λ has to be larger than $\texttt{bpw}$).

Every sum $\sum_{i=1}^{\texttt{nb}} z_{i,k} w_{i,j}$ of $\texttt{nb}$ terms over $\texttt{bpw}$ bits each can still be performed in $\texttt{nb}$ elementary operations so long as $\log_2(\texttt{nb}) + \texttt{bpw}$ is smaller than the word size used by the processor (usually 64 or 128 bits). The overall computational complexity of the protocol is therefore unchanged at $O(\lambda \texttt{nb}\ \texttt{wpb})$. When $\texttt{nb} = \texttt{wpb} = \sqrt{n/\texttt{bpw}}$ and we precompute over r bits as described in Section 4.4.3, this becomes $O(\frac{\lambda n}{r\ \texttt{bpw}})$.

To sum up, both the communication per bit recovered and the overall computational cost are improved by a factor of $\texttt{bpw}$, which can be 10 or more. Note that it comes at the cost of recovering blocks of bits $\sqrt{\texttt{bpw}}$ times larger, which is a non-issue in a lot of settings.

The question this naturally brings up is whether or not this affects the security of the scheme. First, the scheme can be reduced to a version of the Somewhat Homomorphic Encryption scheme (Definition 4.4.5) on $\texttt{bpw}$ bits as we did in the security proof from Section 4.4.4. Furthermore, we can show that this version of the Somewhat Homomorphic Encryption scheme can be reduced to the AGCD problem for numbers $pq + 2^{\texttt{bpw}}\epsilon + b$, $b \in \{0, 1\}$. We detail these claims using the following lemmas.

Definition 4.4.4

Multiple Bit Distributions

- $\mathcal{D}_{\texttt{bpw}}(\lambda, p, b)$ *is the uniform distribution over the set* $pq + 2^{\texttt{bpw}}\epsilon + b$ *for* q *over* $\lambda - \log_2 p$ *bits,* $\epsilon < \frac{p}{2^{2\texttt{bpw}} nb}$ *and* $b \in \{0, 1\}$.
- $\mathcal{D}_{\texttt{bpw}}(\lambda, p)$ *is the uniform distribution over the set* $pq + 2^{\texttt{bpw}}\epsilon + b$ *for* q *over* $\lambda - \log_2 p$ *bits,* $\epsilon < \frac{p}{2^{2\texttt{bpw}} nb}$ *and* b *uniformly drawn from* $\{0, 1\}$.

Definition 4.4.5

Multiple Bit Somewhat Homomorphic Encryption Problem

Let $\{z_i\}_i \leftarrow \mathcal{D}_{\texttt{bpw}}(\lambda, p, 0)$ *be a polynomially large collection of integers,* $b \leftarrow \{0, 1\}$ *a secret random bit and* $z \leftarrow \mathcal{D}_{\texttt{bpw}}(\lambda, p, b)$. *Given* $\{z_i\}_i$ *and* z, *output* b.

Lemma 4.4.6

If there exists a PPT algorithm $\mathcal{A}$ *that can distinguish with non-negligible advantage between queries for secret index* x_1 *and* x_2, *then there exists a PPT algorithm* $\mathcal{B}$ *that can solve the* Multiple Bit Somewhat Homomorphic Encryption Problem *with non-negligible advantage.*

Proof: We proceed exactly as in the proof of Theorem 4.4.3. We briefly recall the reduction here. Suppose that $\mathcal{A}$ returns 1 with probability $1/2 + \mathbf{Adv}(\mathcal{A})$ for a randomly selected query for secret index x_1, and with probability $1/2 - \mathbf{Adv}(\mathcal{A})$ for secret index x_2.

Given a random z drawn from $\mathcal{D}_{\texttt{bpw}}(\lambda, p)$, we build a query $q = (z_1, \cdots, z_{\texttt{nb}})$. z_i is drawn randomly from $\mathcal{D}_{\texttt{bpw}}(\lambda, p, 0)$ when $i \notin \{x_1, x_2\}$. Then we do one of the following two options with equal probability:

- Option A: Set $z_{x_1} := z$ and z_{x_2} as a random element drawn from $\mathcal{D}_{\texttt{bpw}}(\lambda, p, 0)$.
- Option B: Set $z_{x_2} := z$ and z_{x_1} as a random element drawn from $\mathcal{D}_{\texttt{bpw}}(\lambda, p, 0)$.

We run $\mathcal{A}$ on the generated query and return $\mathcal{A}(q)$ if Option A was selected, $1 - \mathcal{A}(q)$ if Option B was selected. ■

Definition 4.4.7

Multiple Bit Approximate GCD Problem

Given a polynomial number of values z_i *drawn from* $\mathcal{D}_{\texttt{bpw}}(\lambda, p)$, *find the value of* p.

Lemma 4.4.8

If there exists a PPT algorithm $\mathcal{A}$ *that can solve the* Multiple Bit Somewhat Homomorphic Encryption Problem *with non-negligible advantage, then there exists a PPT algorithm* $\mathcal{B}$ *solving the* Multiple Bit Approximate GCD Problem *with non-negligible probability.*

Proof: We do not fully detail the proof of this step as it closely follows Van Dijk *et al.*'s [vDGHV10]. We recall the main steps in the reduction here.

Given $z = pq + 2^{\mathtt{bpw}}\epsilon + b$ for secret p, q, ϵ and b, we can easily guess the value of $q \bmod 2$. Indeed, the hypothetical algorithm $\mathcal{A}$ guesses the value of b with non-negligible advantage. We run $\mathcal{A}$ on z and call b' the output $\mathcal{A}(z)$. We call $a = (z \bmod 2) \oplus b'$. Clearly, $a = q \bmod 2$ if $\mathcal{A}$ correctly guessed the value of b (if $b = b'$).

We then build an algorithm that given two values z_1 and z_2 where $z_i = pq_i + 2^{\mathtt{bpw}\epsilon_i} + b_i$ computes the binary GCD [Knu97] of q_1 and q_2 (that is, $\gcd(q_1, q_2) \bmod 2$). With non negligible probability, this GCD is 1, which means the algorithm finds $\tilde{z} = 1 \cdot p + \epsilon$ (with small enough noise ϵ). Then the binary GCD algorithm is run a second time on inputs z_1 and $\tilde{z}$ which directly gives the binary decomposition of q_1, from which we easily recover $p = \lfloor z_1/q_1 \rfloor$.

Note that in [vDGHV10], the noise $\epsilon \in [-2^{\lambda_\epsilon - 1}, 2^{\lambda_\epsilon - 1}]$ which allows subtraction between values and is used in the binary GCD algorithm. This notation is however equivalent to ours by setting $y_i = z_i - 2^{\lambda_\epsilon - 1}$. ■

Theorem 4.4.9

If a PPT algorithm can distinguish between bpw-queries for secret indices s_1 and s_2 with non-negligible advantage, then the secret value p can be efficiently recovered with non-negligible advantage.

Proof: Immediate from lemmas 4.4.6 and 4.4.8. ■

This shows that this scheme is at least as secure as a version of the AGCD problem with the $\mathtt{bpw}-1$ least significant noise bits known. Alternatively, any generic AGCD problem instance can be turned into one of those instances by bruteforcing the value of said bits. As such, for small enough word sizes (say, up to 32 bits), the security of the scheme is mostly unaffected.

4.4.7 Updating the Database

Preprocessing techniques provide significant speed-up when answering queries at the cost of a one-time expensive preprocessing step. However, since the preprocessed values depend on the original database values, any modification of the original database will require the preprocessing step to be performed anew. Naturally, such schemes are better suited for somewhat static databases but even in those cases updates may still happen occasionally. In this section we study the complexity of updating the precomputed values when the original database is modified.

Suppose word $w_{i,j}$ from the original database is updated from v_0 to v_1. We call $\delta = v_1 - v_0$.

For every $(x_0, \cdots, x_{r-1}) \in \{0,1\}^r$, if $x_k = 1$ where $k = i - 1 \bmod r$, then there is one precomputed value v in the preprocessed database that has to be updated to $v' = v + \delta$. Overall, every modification of a word's value in the original database will require 2^{r-1} word updates to the preprocessed database, which is very reasonable if the database is only updated sparingly.

4.4.8 Query Batching

In works by Beimel, Ishai and Malkin [BIM04] or Lueks and Goldberg [LG15], query batching is used as a way to reduce complexity per query when many clients are involved in the protocol. The techniques used rely on the sub-cubic complexity of matrix multiplication. We here detail how the same approach can be used in our system to provide a significant improvement in the right setting.

Lemma 4.4.10

Let M, $N \in \mathbb{F}^{n \times n}$. The matrix product $MN \in \mathbb{F}^{n \times n}$ can be computed in $O(n^\omega)$ multiplications in $\mathbb{F}$ for $\omega < 2.376$ [CW87].

Now observe that for a query $q = (z_1, \cdots, z_{\mathtt{nb}})^\intercal \in \mathbb{Z}^{\mathtt{nb} \times 1}$ and a server-hosted database $\mathcal{DB} = (b_{i,j}, 1 \le i \le \mathtt{wpb}, 1 \le j \le \mathtt{nb}) \in \mathbb{Z}^{\mathtt{wpb} \times \mathtt{nb}}$, the server's response is $\mathcal{DB} \cdot q \in \mathbb{Z}^{\mathtt{wpb} \times 1}$.

More generally, if we have k (independent) queries $q_1, \cdots, q_k$ we can build the $\mathtt{nb} \times k$ matrix having query q_i as its i-th column. We call this matrix $q_{1\sim k}$. Now the server's response to the i-th query is the i-th column in $\mathcal{DB} \cdot q_{1\sim k}$.

Let's assume $\mathtt{nb} = \mathtt{wpb}$ which is the most efficient setting as described before. Three cases can be considered:

- If $k = \mathtt{nb}$ then both $q_{1\sim k}$ and $\mathcal{DB}$ are square matrices and $\mathcal{DB} \cdot q_{1\sim k}$ can be computed in $O(\mathtt{nb}^\omega)$ operations.

  This means that the server only performs $O(\mathtt{nb}^{\omega-1}) = o(\mathtt{nb}^2) = o(n/\mathtt{bpw})$ operations per query. Each operation here is elementary when splitting every $q_{1\sim k}$ as a sum of λ matrices with values in $\{0,1\}$ and so long as $\mathtt{bpw} + \log_2 \mathtt{nb}$ is smaller than the processor word size (see Section 4.4.6 for details).

  Overall, per query complexity is $O(\lambda(n/\mathtt{bpw})^{(\omega-1)/2}) < O(\lambda(n/\mathtt{bpw})^{0.7})$.

- If $k < \mathtt{nb}$ we can pad $q_{1\sim k}$ with 0 values to make it a square $\mathtt{nb} \times \mathtt{nb}$ matrix and process it using the first case.

  The computational complexity per query is $O(\lambda \mathtt{nb}^\omega / k)$ which is asymptotically less than the cost for the non-batching approach $\lambda \frac{n}{\mathtt{bpw}}$ (see Section 4.4.6) when $k > \mathtt{nb}^{\omega-2}$.

- If $k > \texttt{nb}$ we can process nb queries at a time using the first case and process then remaining $k_0 = k \bmod \texttt{nb}$ using the second case.

In the second case, we can also use a result by Coppersmith [Cop97a] stating that there exists an algorithm able to compute the product of an $\texttt{nb} \times \texttt{nb}$ matrix by an $\texttt{nb} \times \texttt{nb}^\alpha$ for a constant $\alpha > 0.294$ in $O(\texttt{nb}^{2+\epsilon})$ for any $\epsilon > 0$. In this case the computational cost per query is $O(\lambda\texttt{nb}^{2+\epsilon-\alpha}) = o(\lambda\texttt{nb}^{\omega-\alpha})$. This is however a mostly theoretical construction not practical for reasonable values of nb.

For smaller values of nb, the Strassen-Winograd algorithm [Win71] where $\omega = \log_2 7 \approx 2.8074$ can be used and provide significant improvement whenever $k > \texttt{nb}^{0.8074}$.

4.5 Discussion and Comparison

4.5.1 Implementation

Goldberg provided an implementation called percy++ [GDHH14] that included the robust protocol from [Gol07] along with other PIR protocols. Running tests showed that in multi-server settings, Chor *et al.*'s protocol [CGKS95] performed the fastest, followed by Goldberg's [Gol07]. Note that Goldberg's, and by extension our version described in Section 4.3.2, contains much stronger security features. In single-server settings, Aguilar-Melchor *et al.*'s scheme [MG07] was deemed the best performing. The security of this scheme is based off the so called *Differential Hidden Lattice Problem* which the authors introduced and studied. In comparison, our single-server scheme defined in section 4.4 relies on the computational Approximate GCD assumption on which one of the main candidates for fully homomorphic encryption is based and has thus received a lot of attention.

We made a straightforward implementation of our protocols without aiming for any kind of optimization. Our objective was the confirmation of theoretical complexities. For multi-server protocols, we always picked $t = 2$ servers, the minimum to keep the protocol safe while maximizing efficiency. In real world situations, a higher number would be desirable, multiplying the overall time complexity by a non negligible constant.

We only care about the computational complexity of the server since the communication and client computational complexities are asymptotically negligible. In situations where the preprocessing would generate files too large for our environment, we simulated reading from random files instead, which makes no difference from the server's point of view since the data received is indistinguishable from random data.

There are a lot of parameters that can be modified when running actual tests. For Goldberg's scheme, the preprocessing parameter r described in section 4.3.1, the number of bits per word and the size of the database. Our implementation shows that, as expected, the running time will decrease linearly as the preprocessing factor r is increased. This is true for databases small enough that the preprocessed version would still fit in RAM and for databases large enough that it does not fit in RAM even without preprocessing. For middle-sized databases, the added preprocessing forces the algorithm to read data directly from the harddrive in a semi-random manner, which can slow it down significantly depending on reading speeds.

For our single server scheme, a security parameter of 80 was chosen which, depending on database size, will translate into a value $\lambda > 64000$. In this setting, our implementation of the algorithm runs slower than alternatives. A tighter security bound than the one described in Section 4.4.5 would be an improvement. Most importantly, the techniques we will describe in the next chapter will solve this issue.

4.5.2 Discussion

PIR schemes have been proposed in a wide range of settings. In our approach, we tried to the setting most likely to be used in practice. That is, there exists a single server hosting a database, there are no non-computational assumptions required for scheme to be secure and any block of data may be recovered. As such, when comparing our results to others, we consider schemes able to provide the same level of freedom and security as our own.

Now, comparing PIR schemes is not an easy task. There exists a trivial algorithm performing in only n operations which is said to actually outperform more complex schemes [Sio07]. Conversely, there are PIR schemes in this setting that require more than n operations in total but outperforming the trivial algorithm [OG12]. Furthermore, different implementations for the same theoretical construction sometimes have very different practical running times.

The scheme we described in this Chapter has a number of appealing points but will only truly be worth studying when combined with the new techniques developed in Chapter 6. We will then further discuss in Section 6.6.

4.6 Conclusion

A completely practical PIR scheme has yet to be found. As long as the entire database has to be read for every query, such schemes will always perform too slowly. Based on the observation that preprocessing is a necessity to reach that goal, we first showed how such precomputations can be added to some already existing protocols and designed a new protocol compatible with this technique.

We presented the first PIR protocol allowing the recovery of a block of bits, strong protection against server collusion and Byzantine servers while performing more efficiently than previously possible. Its performance is theoretically better than other such algorithms and the design allows efficient implementations by mostly avoiding the need for large integers libraries. Our scheme can be seen as both a generalization of Goldberg's protocol and Beimel et al.'s protocol. Compared to Goldberg's, ours can perform several times faster but requires a polynomial expansion of the database. Even a quadratic expansion is sufficient to provide a faster protocol. Compared to Beimel et al.'s, the scheme provides much stronger security against several servers cooperating, a major weak point of multi-server PIR protocols.

We also presented an efficient single-server scheme with a simple structure. It is compatible for preprocessing techniques, a requirement for sublinear PIR, and relies on a computational assumption which is fairly new but has received and will likely continue to receive a lot of attention from the research community. Furthermore, the way we use it allows for relatively small parameters compared to FHE schemes based on it. New approaches to precomputing are however required to truly show its usefulness.

Chapter 5

Partial Server-Side Parameter Selection

5.1 Introduction

The scheme we detailed in Chapter 4 or the one Aguilar-Melchor *et al.* designed [MBFK16] are two lightweight constructions highly efficient but that do not reach the symbolic threshold of reading less bits that the trivial algorithm. In this chapter, we revisit these results and introduce new preprocessing techniques that can allow us to go under that threshold for large enough databases. The result is especially of interest theoretically as it shows that the information theoretical bound can asymptotically be broken under computational assumptions. To achieve this performance, we first have to increase the client-side computational cost. We further improve on this aspect after careful security considerations.

Increasing the computational cost of the client may seem counter-intuitive in a world focusing more and more on cloud computing which aims to achieve quite the opposite. It however makes sense in a PIR setting as the server does not benefit from the added computational cost. Indeed, privacy-centered systems are often not compatible with business models relying on monetizing clients' habits. It then makes sense that the recipient of the additional privacy should shoulder part of the cost through added computations on his side.

This chapter is organized as follows. In Section 5.2, we briefly go over the scheme from Chapter 4 that we will use as a building block for our main proposal. Section 5.3 is the new construction. We describe the new preprocessing technique and the adjusted steps for the client and the server. We detail parameter selection in Section 5.4 and discuss security concerns in Section 5.5. Section 5.6 shows two techniques to further improve the performance. Finally, Section 5.7 describes how the techniques detailed previously can also be applied to the Ring-LWE-based scheme from [MBFK16].

5.2 Working with Near Multiples

5.2.1 Notations

We work in the single-server PIR setting where a server is hosting a public database. The database is indexed over two dimensions, we write it $(b_{1,1}, \cdots, b_{\mathtt{nb},\mathtt{bpb}})$, $b_{i,j} \in \{0,1\}$. `nb` stands for **n**umber of **b**locks and `bpb` stands for **b**it **p**er **b**lock. Its total length is $n = \mathtt{nb}\ \mathtt{bpb}$. x is the index of the block that the client wants to retrieve. We use the notation ϵ to suggest a number is small compared to some other parameters. The exact requirements on the size will be detailed on a case by case basis.

For parameters p and q, λ_p is the bit length of p, λ_q the bit length of q and $\lambda = \lambda_p + \lambda_q$ the overall security parameter of the scheme. We call `I` the set of λ indices $\{0, \cdots, \lambda - 1\}$.

We try to use uniform notations throughout the chapter as much as possible. i will be an index on blocks, usually ranging from 1 to `nb`. j will be an index on bits within a block, usually ranging from 1 to `bpb`. Finally, k will be an index on bits from the input parameter, usually ranging from 0 to $\lambda - 1$ or a subset of `I`.

log designates the natural logarithm (base e) while $\log_b$ refers to logarithm base b.

For a variable z, the notations $z^{(c)}$ and $z^{(s)}$ refer to a splitting of z in a **c**lient part ($z^{(c)}$) and a **s**erver part ($z^{(s)}$). In other words, when there are parentheses, (c) and (s) are not exponents but labels.

5.2.2 Scheme Description

In this section, we briefly recall the scheme developed in Chapter 4.

Consider an n-bit public database made of `nb` blocks containing `bpb` bits each. Note that for now we do not consider the specific case where each block is made of words and each word has a fixed number of bits. We will discuss what happens when we do in Section 6.4. Recall that $b_{i,j}$ is the bit number j in block number i.

Assume a remote client wants to recover block number x from the database. He also wants to keep the index x secret for privacy reasons.

The protocol has only one round and aims at reducing computations as much as possible. In particular, it uses the additively homomorphic properties of Van Dijk et al.'s Somewhat Homomorphic Encryption scheme [vDGHV10].

First, the client selects at random a large odd secret number p and generates $\texttt{nb}$ close multiples of it: $\{z_i = pq_i + 2\epsilon_i + \delta_{i,x} | i \in \{1, \cdots, \texttt{nb}\}\}$ where $\delta_{i,x} = 1$ if $i = x$, 0 otherwise and $\epsilon_i < p/(2\texttt{nb})$ is small compared to p. The noise is picked from a uniform distribution as in [vDGHV10], but it could just as securely be picked from a Gaussian distribution (see [Reg04]).

The z_is are then sent to the server.

For every bit in a block $j \in \{1, \cdots, \texttt{bpb}\}$, the server computes $res_j = \sum_{i=1}^{\texttt{nb}} z_i b_{i,j}$ and sends these values to the client.

So long as $\sum_{i=1}^{\texttt{nb}} 2\epsilon_i + 1 < p$, the following holds and the client can effectively recover every bit from block x.

$$\begin{aligned}(res_j \bmod p) \bmod 2 &= \sum_{i=1}^{\texttt{nb}} 2\epsilon_i + \delta_{i,x} b_{i,j} \bmod 2 \\ &= \sum_{i=1}^{\texttt{nb}} \delta_{i,x} b_{i,j} = b_{x,j}\end{aligned} \tag{5.1}$$

Server side computational complexity is $\texttt{nb}$ sums over λ bits integers for every bit in a word. Overall, complexity is λn elementary operations. Client side complexity is $O(\texttt{nb})$ which is negligible compared to the server's computation.

Communication complexity is $\lambda\texttt{nb}$ bits to send the z_i values and $\lambda\texttt{bpb}$ bits to send back the res_j values. If $\texttt{nb} = \texttt{bpb} = \sqrt{n}$, the overall complexity is $O(\lambda\sqrt{n})$ bits to retrieve a $\sqrt{n}$-bit long block, or $O(\lambda)$ bits transferred per bit recovered.

The security proof from Section 4.4.4 shows that the scheme is secure under the Approximate GCD assumption with known 0 encryptions, which informally states that it is computationally hard to recover the value of p given a polynomial number of close multiples of it ($pq + \epsilon$ for a small ϵ) and a polynomial number of encryptions of 0 ($pq + 2\epsilon$ for a small ϵ). In the case of PIR protocols, security means that the server should be unable to distinguish between two queries for different blocks in reasonable time.

5.3 New Approach to Preprocessing

Let us consider a generic PIR protocol in a standard setting where there is no need for the client to trust some additional property (for instance trusting that several servers do not collude or using trusted hardware server-side). It is easy to see that in this situation, protecting client privacy while reading less than the entire database on the server requires some form of preprocessing (notion first introduced in 2004 by Beimel *et al.* [BIM04]). Indeed, let us assume the original database is left untouched. In this case it is obvious that the bits not read when answering a query were not the ones requested by the client.

Now, different kinds of preprocessing can be imagined. Some simply aim at speeding up computations in the non-preprocessed version of a scheme, for instance by processing several input bits at once as we did in Chapter 4 or using convenient representations of the data hosted by the server as used in the LWE-Ring based scheme [MBFK16]. Intuitively, this approach still requires a computation time linear in the size of the database. We detail these approaches in more details in Sections 5.6.2 and 5.7.2.

In this section, we introduce a brand new technique that allows the server to effectively take part in the choosing of the parameters. While this technique also provides a linear improvement, it is independent from the ones previously mentioned and thus can be combined with them. For some parameters, this allows the entire scheme to run securely while reading less than n bits of data. Furthermore, the technique itself is more involved than standard preprocessing approaches and is an interesting study case of what could lead to very efficient schemes.

5.3.1 Server-Side Parameter Selection

In the protocol detailed in the previous section, the client is free to choose the values of the z_i parameters before sending them to the server. We now show how to modify the protocol to allow the server to decide some of the values, use them to preprocess some computations and improve the overall complexity. In order to prevent the server from selecting potentially malicious parameters, we will assume that every random selection is done according to a public, unbiased generator (for instance, bits from π's base 2 decomposition or a more efficient widely used PRNG) and in a fixed order. Further security concerns will be discussed in Section 5.5.

We decompose each z_i as $z_i = \sum_{k \in \mathtt{I}} z_{i,k} 2^k$ where each $z_{i,k} \in \{0, 1\}$ (recall that $\mathtt{I} = \{0, \cdots, \lambda - 1\}$. Note that $res_j = \sum_{k \in \mathtt{I}} 2^k \sum_{i=1}^{\mathtt{nb}} z_{i,k} b_{i,j}$. We call $res_{j,k}$ the value $\sum_{i=1}^{\mathtt{nb}} z_{i,k} b_{i,j}$.

A subset of K indices larger than λ_p called `ClientIndices` (`CI`) $\subset$ `I` is selected. We also call `ServerIndices` (`SI`) the subset $\mathtt{I} \backslash \mathtt{CI}$ (containing $\lambda - K$ elements). A simple selection method is described in the security analysis of Section 5.5.

For every $k \in \mathtt{SI}$ and $i \in \{1, \cdots, \mathtt{nb}\}$, a random bit $z_{i,k}^{(s)} \in \{0, 1\}$ is selected. The server then precomputes for every $k \in \mathtt{CI}$ and $j \in \{1, \cdots, \mathtt{bpb}\}$ the value $res_{j,k} = \sum_{i=1}^{\mathtt{nb}} z_{i,k}^{(s)} b_{i,j}$. This computation is done prior to any client request and only once. Computing it merely requires `nb` elementary operations for every $res_{j,k}$, or $(\lambda - K)n$ operations in total. It also only requires $(\lambda - K)\mathtt{bpb} \log_2 \mathtt{nb}$ bits of storage.

5.3.2 Client-Side Parameter Selection

Assume once again that a client wants to recover the bits from block $x \in \{1, \cdots, \mathtt{nb}\}$ while keeping the actual index x private. In this section, we assume that the client has access to its own private, cryptographically secure random number generator. Every occurrence of randomness client-side is assumed to come from this secure generator unless explicitly specified. One should also note that because the server-side randomness generation method is public, all of the server-generated values (namely, the $z_{i,k}^{(s)}$) can also be assumed to be available to the client.

As in Section 4.2, the client selects a large random odd secret number p over λ_p bits. For every block index $i \in \{1, \cdots, \mathtt{nb}\}$, he uses Algorithm 2 to pick some values $\{z_{i,k}^{(c)}, k \in \mathtt{CI}\}$. He then sends all of these values to server: $\{z_{i,k}^{(c)}, 1 \leq i \leq \mathtt{nb}, k \in \mathtt{CI}\}$

5.3.3 Server Response

Upon receiving the values $z_{i,k}^{(c)}$ for every $i \in \{1, \cdots, \mathtt{nb}\}$ and $k \in \mathtt{CI}$, the server computes $res_{j,k} = \sum_{i=1}^{\mathtt{nb}} z_{i,k}^{(c)} b_{i,j}$ for every $j \in \{1, \cdots, \mathtt{bpb}\}$. Using the values precomputed for $k \in \mathtt{SI}$, he can now also compute $res_j = \sum_{k \in \mathtt{I}} 2^k res_{j,k}$.

Algorithm 2 Get$z_i^{(c)}(i)$

```
    Note: here $m_k$ is the $k$-th bit of $m$ and $\texttt{SET}[k]$ is the $k$-th element in $\texttt{SET}$
Input: Block index $i$
Output: Suitable parameters $z_{i,k}^{(c)}$, $k \in \texttt{CI}$ or Not Found exception
$L \leftarrow \emptyset$
$s \leftarrow \sum_{k \in \texttt{SI}} 2^k z_{i,k}^{(s)}$
for $m$ from $0$ to $2^K - 1$ do
    $z \leftarrow s + \sum_{k=0}^{K-1} 2^{\texttt{CI}[k]} m_k$
    if $z \bmod p < p/\texttt{nb}$ and $(z \bmod p) \bmod 2 = \delta_{i,x}$ then
        for $k$ from $0$ to $K$ do
            $z_{i,\texttt{CI}[k]}^{(c)} \leftarrow m_k$
        $L \leftarrow L \cup \{(z_{i,k}^{(c)}, k \in \texttt{CI})\}$
if $L = \emptyset$ then
    Throw exception Not Found
else
    Output random element from $L$
```

These bpb values are then all sent to the client. Finally, the client is able to recover $b_{x,j}$ from res_j exactly as described in Section 5.2.

The communication complexity for this step is unchanged at $\lambda\texttt{bpb}$. The computational complexity however went from λn down to Kn elementary operations ($K < \lambda$). See Section 5.4 for a discussion on K's value.

5.4 Parameter values and practical complexity

In this section we show practical values for the parameters used in Section 5.3. We then show that the overall scheme can run successfully while reading less than n bits, which is our main result. Further improvements are discussed in Section 5.6.

Theorem 5.4.1

If $K > 1 + \sigma + \log_2 \texttt{nb}$ then all nb *calls to* Get_$z_i^{(c)}$ *will be successful with probability at least $1 - 2^{-\sigma}$*

Please refer to Algorithm 2 for notations.

The key point here is the need for the z values created collaboratively between the client and the server to behave randomly enough when considered modulo p. This is the reason why the indices in `CI` are chosen to be larger than λ_p. In practice, just setting `CI` to be the set of most significant bits is sufficient. In this edge case, we have $z = 2^{\lambda-K}z^{(c)} + z^{(s)}$ where $z^{(c)}$ is a K-bit integer chosen by the client while $z^{(s)}$ is a $(\lambda - K)$-bit integer imposed by the protocol (server-side).

We want to find when equation 5.2 has solutions (z, ϵ) such that $0 < \epsilon < \frac{p}{2\mathtt{nb}}$ and $0 < z < 2^K$.

$$z2^{\lambda-K} + z_i^{(s)} = 2\epsilon + \delta_{i,x} \mod p \tag{5.2}$$

Note that we are only interested in knowing how large K should be to near-guarantee the existence of a solution. For clarity's sake, whenever an index goes under 0 or over the maximal value for which it is defined, we assume it is computed modulo this maximal value. The operator / indicates a (rounded) integer division and not a multiplication by modular inverse which is denoted by $^{-1}$. We assume the modulus p is prime therefore inversion is always possible. All the other computations described thereafter work on modular sets.

We write $E = \{0, \cdots, 2^K - 1\}$ and $\Delta = 2^{-1}2^{\lambda-K} \bmod p \in \mathbb{Z}_p$ and $c = 2^{-1}(z_i^{(s)} - \delta_{i,x}) \bmod p \in \mathbb{Z}_p$. We rewrite equation 5.2 as $z\Delta + c = \epsilon \bmod p$. Now suppose it has no solution, ie $\forall z \in E,\ z\Delta + c \bmod p \geq \frac{p}{2\mathtt{nb}}$.

Consider the set $S = \{(x, x\Delta \bmod p) \in E \times \mathbb{Z}_p \mid x \in E\}$. For any $i \in E$, we write $S[i] \in E$ the first coordinate of the i-th smallest element in S when ordered according to the second coordinate.

Definition 5.4.2 (*Distance*)

For z_1, $z_2 \in E$, let $x = (z_2 - z_1)\Delta \bmod p \in \mathbb{Z}$.

We write $d(z_1, z_2) = \begin{cases} x \text{ if } x \leq p/2 \\ -(p-x) \text{ if } x > p/2 \end{cases} \in \mathbb{Z}$ the signed shortest distance between elements z_1 and z_2. For $i \in E$, we write $d(S[i]) = (S[i+1] - S[i])\Delta \mod p \in \mathbb{Z}_p$ the (modular) distance between an element and the element following it in S, also called following-element-distance.

We say that z is to the right (respectively, left) of c if $d(z, x) \geq 0$ (respectively, $d(z, x) \leq 0$).

Definition 5.4.3 (*Gap Element*)

Since we assumed that for all $z \in E$, $z\Delta \bmod p \notin \{-c \bmod p, \cdots, -c + \frac{p}{2\mathtt{nb}} \bmod p\}$ (which contains $\frac{p}{2nb}$ consecutive elements), we know that there *exists $z \in E$ such that $d(z) \geq \frac{p}{2\texttt{nb}}$. We call any such z a gap element and write $M \geq 1$ the total number of gap elements in E. We define $G = \{(z, z\Delta \bmod p) \in E \times \mathbb{Z}_p \mid z \text{ gap element}\}$ the set of all M gap elements ordered according to the second coordinate with $G[m]$ being the first coordinate of the m-th smallest gap element indexed from 0 to $M-1$.*

Note that there can only be at most $2\texttt{nb}$ distinct gap elements (since they all belong to $\mathbb{Z}_p$ and are distant from each other by at least $\frac{p}{2\texttt{nb}}$). Therefore $M \leq 2\texttt{nb}$.

Definition 5.4.4 (*Cluster*)

For any gap element $z = G[m] = S[i]$, we define a set of elements of E containing z we call cluster and write C_m the set containing $G[m]$ and such that if $z \in C_m$ then every $x \in E$ within distance $p/(2\texttt{nb})$ of z is in C_m. Informally, it is the set containing every element between the $(m-1)$-th gap element (excluded) and the m-th gap element (included). Alternatively, cluster m contains the m-th gap element and its elements are not too far from each other. Note that every element of S belongs to a cluster.

Lemma 5.4.5

We now prove that if for some $k \in \mathbb{N}^*$, $z \in E$ and $m \in \mathbb{Z}_M$, both z and $(z+k)$ belong to the same cluster C_m then for any $z' \in E$ with $z+k < 2^K$, both z' and $(z'+k)$ belong to the same cluster $C_{m'}$ for some $m' \in \mathbb{Z}_M$.

Proof: If $d(z, z+k) < \frac{p}{2\texttt{nb}}$ then from the definition of distance we have $d(z', z'+k) = d(z, z+k) < \frac{p}{2\texttt{nb}}$ which implies that z' and $z'+k$ belong to the same cluster.

Now let us assume $d(z, z+k) \geq \frac{p}{2\texttt{nb}}$. Since z and $z+k$ belong to the same cluster, that means there exists a sequence $(s_1 = z, s_2, \cdots, s_{t-1}, s_t = z+k) \in E^t$ of elements between z and $z+k$ such that $d(s_i, s_{i+1}) < \frac{p}{2\texttt{nb}}$ for any $i \in \{1, \cdots, t-1\}$.

We define the sequence $(s'_1, \cdots, s'_t) \in E^t$ such that $s'_i = s_i + (z' - z)$. Once again, because of the properties of the distance function, we have $d(s'_i, s'_{i+1}) = d(s_i, s_{i+1}) < \frac{p}{2\texttt{nb}}$ for any $i \in \{1, \cdots, t-1\}$ which means that every s'_i belongs to the same cluster. In particular, $z' = s'_1$ and $z'+k = s'_t$ do. ■

We now show that this implies that there exists a cycle of clusters $(C_{m[0]}, \cdots, C_{m[M-1]})$ such that if $z \in C_{m[j]}$ then $z+1 \in C_{m[j+1]}$.

Proof: We consider the sequence of clusters $(C_{m[1]}, \cdots, C_{m[2^K]})$ such that for every $z \in E$, $z \in C_{m[z]}$. We assume K is large enough that $2^K > 2\texttt{nb}$, in which case we have $2^K \geq M$ and there exists $i < j \in E$ such that $m[i] = m[j]$ and every $m[k]$ for $i \leq k < j$ is distinct. Because of lemma 5.4.5, this means that the cluster sequence $(C_{m[i]}, \cdots, C_{m[j]})$ is a cycle (ie $m[j+k] = m[i+k]$ for any $k \in E$). Furthermore, by definition of a cluster, each cluster C_0 to C_{M-1} is non-empty so all M clusters must appear in the sequence. ■

The cyclic nature of the cluster sequence implies that every cluster contains (almost) the same number of elements, $2^K/M$ (or $2^K/M + 1$ for some clusters if 2^K is not an exact multiple of M). Furthermore, every non-gap element z in a cluster has the same distance $d(z)$ to its following element. This is because the element following z within a cluster is always either $z+M$ or $z - M$.

Proof: Indeed, consider the leftmost element z_0 in a cluster. We know that either $z_0 + M$ or $z_0 - M$ is in E. Suppose that $z_0 + M$ is (the other case is similar and we will not detail it). We write $z_1 = z_0 + M$. z_1 belongs to the same cluster and since we assumed z_0 is the first element, z_1 is to the right of z_0. Now suppose there exists an element z between z_0 and z_1. Then either $z - M$ is also in the cluster or z is the smallest element in the cluster. In the first case, $d(z - M, z) = d(z_0, z_1)$ but this however means that z is left of z_0 which is a contradiction. In the second case, that means that z_0 is not the smallest therefore $z_0 - M$ belongs to the cluster but $d(z_0 - M, z_0) = d(z_0, z_1) > 0$ which means that $z_0 - M$ is left of z_0: contradiction. ■

Now, we write Δ as $\frac{p}{M}\alpha+\gamma$ such that $\gamma < \frac{p}{M}$ and $\alpha < M$ (this decomposition is unique). Now for a non-gap element z, $d(z) = M\Delta \bmod p = M\gamma$. As before, we know that the following-element-distance for every gap element is the same. Since there are M clusters, this means that the distance between the first and last element of a cluster is less than p/M. Therefore, a cluster contains $2^K/M$ elements separated by a distance $M\gamma$ each and covering a distance less than p/M overall. This gives us $2^K\gamma < p/M$ or $\gamma < \frac{p}{M2^K}$.

To summarize, we have $\Delta = \gamma + \frac{\alpha}{M}p$ with $\gamma < \frac{p}{M2^K} \leq \frac{p}{2^K}$. In other words, the assumption that equation 5.2 has no solution implies that Δ is very close (within distance $\frac{p}{M2^K}$) to a fraction $\frac{\alpha}{M}$ of p with $M \leq 2\mathtt{nb}$.

The number of such distinct fractions is $\phi(M) \leq M$. Therefore an upper bound for the number of values Δ very close to a fraction $\frac{\alpha}{M}$ is $\frac{p}{2^K}$. Summing over M from 1 to $2\mathtt{nb}$ we get a conservative upper bound for the number of values Δ for which equation 5.2 has no solution: $\frac{2\mathtt{nb}\ p}{2^K}$. Since the value of $\Delta = 2^{-1}2^{\lambda-K} \bmod p$ is uniformly random in $\mathbb{Z}_p$ when p was picked uniformly randomly, this means that the probability that equation 5.2 admits solutions is at least $1 - \frac{2\texttt{nb}}{2^K}$.

Setting $K = 1 + \sigma + \log_2 \texttt{nb}$, this gives us a success probability greater than $1 - 2^{-\sigma}$. There are now essentially two ways to select K:

- For a very large value $\sigma = 80$, the scheme should fail with negligible probability. This case provides the strongest security since the client can essentially pick p truly randomly and have the protocol succeed, just as in the non-preprocessed scheme of Chapter 4.

- For a smaller σ such as $\sigma = 20$, one value p in 2^σ will actually be unusable. While rare, this would occasionally happen to some clients in practice. In such cases, the client has to select another random. With overwhelming probability, only a few attempts would ever be required. Security is not significantly affected since the secret key space is only negligibly reduced. See Section 5.5 for security concerns. In this setting however, computational complexity is improved by a factor of nearly 4 compared to case $\sigma = 80$ which is a very significant improvement.

Note that due to the construction used in the proof, if a given p is such that a solution exists for some $z_i^{(s)}$, then a solution will exist for *any* $z_{i'}^{(s)}$. In particular, that means that the probability that a solution exists for every single $i \in \{1, \cdots, \texttt{nb}\}$ is still $1 - 2^{-\sigma}$. Setting a somewhat low σ is therefore not a major concern.

Now, the server-side computational complexity is Kn while on the client side it is $\lambda_p 2^K \texttt{nb} = \lambda_p \texttt{nb}^2(2^\sigma + 1)$ (see Section 5.6.1 for ways to reduce it further). The overall complexity is $Kn + \lambda_p \texttt{nb}^2(2^\sigma + 1)$.

Recall that when solutions to equation 5.2 exist, one in every $2\texttt{nb}$ values tried in the `for` loop in Algorithm 2 will find a suitable $z_i^{(c)}$. Since $2^K = \texttt{nb}(2^\sigma + 1)$ is noticeably larger than $2\texttt{nb}$, we can replace Algorithm 2 with Algorithm 3 for generation of the $z_i^{(c)}$ values. Algorithm 3 only loops $2\texttt{nb}$ times on average (compared to 2^K times for Algorithm 2) at the cost of generating K bits of randomness in every iteration of the loop. Note that the table *perm* used in the algorithm only has to be generated once in time 2^K and can then be reused for all $\texttt{nb}$ calls to the algorithm. As such, the amortized cost of generating *perm* per algorithm call is only $O(\log \texttt{nb})$.

In either case, the client sends $\texttt{nb}K$ bits and the server returns $\texttt{bpb}\lambda$ bits giving an overall asymptotically sublinear communication at $\texttt{nb}(\sigma + 1 + \log_2 \texttt{nb}) + \texttt{bpb}\lambda$ bits.

```
Algorithm 3 Get_perm_$z_i^{(c)}(i, perm)$
Input:
    Block index $i$
    Table $perm$ of size $2^K$ containing every value from 0 to $2^K - 1$ in any order
Ensure: Table $perm$ always contains every value from 0 to $2^K - 1$ in any order
Output: Suitable parameters $z_{i,k}^{(c)}$, $k \in \mathtt{CI}$ or Not Found exception
    $s \leftarrow \sum_{k \in \mathtt{SI}} 2^k z_{i,k}^{(s)}$
    for $t$ from 0 to $2^K - 1$ do
        $r \leftarrow$ Random number in $\{t, \cdots, 2^K - 1\}$
        Switch $perm[t]$ with $perm[r]$
        $m \leftarrow perm[t]$
        $z \leftarrow s + \sum_{k=0}^{K-1} 2^{\mathtt{CI}[k]} m_k$
        if $z \bmod p < p/\mathtt{nb}$ and $(z \bmod p) \bmod 2 = \delta_{i,x}$ then
            for $k$ from 0 to $K$ do
                $z_{i,\mathtt{CI}[k]}^{(c)} \leftarrow m_k$
            Output $(z_{i,k}^{(c)}, k \in \mathtt{CI})$
    Throw exception Not Found
```

5.5 Security

The natural question that arises is whether or not reducing the range from which parameters are picked affects the security of the scheme. In this section, we assume the server tries to distinguish queries for blocks x_1 or x_2 using the data sent by the client.

It is important to understand the fundamental difference between our improved scheme and the original one which we already know is secure. Intuitively, our new scheme still selects the same parameters as the base secure scheme but instead of picking (q, ϵ) and setting $z = pq + 2\epsilon + \delta_{i,x}$, it instead selects z first and checks that suitable q and ϵ exist for it. The proof of Theorem 5.4.1 shows that the ϵ value is a random number and the same goes for q (since z, p are random and $q = \lfloor z/p \rfloor$). The underlying security is still based on the Approximate GCD assumption [vDGHV10] (breaking the assumption would immediately result in an efficient way to break client's privacy).

5.5.1 Security Proof

Important note: from here on, we restrict ourselves to the case where `CI` is exactly the set of the K most significant bits. This provides the scheme with a nice structure where each z_i can be written as $z_i^{(s)} + 2^{\lambda-K} z_i^{(c)}$ which we will use in the following formal reduction.

To simplify notations, we call CSPS (Client Side Parameter Selection) the base non-preprocessed scheme from Section 4.4.

The security model for the CSPS scheme was the following: given a query either for secret indice x_1 or x_2, it must be impossible to decide which index is queried with non negligible advantage in probabilistic polynomial time. While this model is standard and satisfies the intuitive security properties one would expect from a PIR protocol, we consider a more permissive model for our PSSPS scheme inspired by the IND-CPA model. We call it IND-CQA or *Indistinguishability Chosen Query Attack* and define it as follows.

We call $\mathcal{D}_x$ the uniform distribution over all queries $\{z_i^{(c)}, i \in \{1, \cdots, \texttt{nb}\}\}$ for secret index x, generated as described in Section 5.3.2. We summarize the IND-CQA model in Figure 5.1.

$$\begin{array}{ccc}
\text{Challenger} & & \text{Adversary} \\ \hline
i_1 \in \{1, \cdots, \texttt{nb}\} & \xrightarrow{i_1} & \\
 & \xleftarrow{q_1} & q_1 \leftarrow \mathcal{D}_{i_1} \\
 & \vdots & \\
i_k \in \{1, \cdots, \texttt{nb}\} & \xrightarrow{i_k} & \\
 & \xleftarrow{q_k} & q_k \leftarrow \mathcal{D}_{i_k} \\
\{x_1, x_2\} \in \{1, \cdots, \texttt{nb}\}^2 & \xrightarrow{\{x_1, x_2\}} & \\
 & \xleftarrow{q} & b \leftarrow \{1, 2\} \\
 & & q \leftarrow \mathcal{D}_{x_b} \\
b' & \xrightarrow{b'} & b = b'?
\end{array}$$

Figure 5.1: IND-CQA

In our model, an adversary and a challenger interact in two phases. The first phase consists of a polynomial number of rounds k. In round $j \in \{1, \cdots, k\}$, the adversary chooses an index $i_j \in \{1, \cdots, \texttt{nb}\}$, sends it to the challenger and the challenger returns a random query $\{z_i^{(c)}, i \in \{1, \cdots, \texttt{nb}\}\}$ for index i_j. In the second phase, the adversary sends $\{x_1, x_2\} \in \{1, \cdots, \texttt{nb}\}^2$ to the challenger. The challenger uniformly picks a random $b \leftarrow \{1, 2\}$ and returns a query for x_b. The adversary then guesses a value b'. The scheme achieves IND-CQA security if the adversary cannot guess b with non-negligible advantage, i.e. $\mathbf{Adv} = \Pr(b = b') - 1/2$ is negligible for any PPT challenger.

This model in particular reflects the fact that the different queries are not generated completely independently like in the non-preprocessed scheme. Indeed, each query is generated relative to the values $z_i^{(s)}$ which are shared between queries.

Let us introduce some notations. We call $\mathcal{Q}$ the set of all CSPS queries. Let $\mathcal{D}$ be the probability distribution over all queries in $\mathcal{Q}$ according to the generation algorithm from Section 4.4. For any query $q \in \mathcal{Q}$, we write $\sigma_q = \Pr(q' = q | q' \leftarrow \mathcal{D})$. For a set $S \subset \mathcal{Q}$, we write $\sigma_S = \sum\limits_{q \in S} \sigma_q$.

Definition 5.5.1 (*Near Completeness*)

A set of queries $S \subset \mathcal{Q}$ is called ***near complete*** *if, for every odd p over λ_p bits and every secret index x, S contains at least one query for those parameters with overwhelming probability.*

Definition 5.5.2 (*Weakness*)

Let $S \subset \mathcal{Q}$ be a near complete *set of queries. We turn S into a probability distribution $\mathcal{D}_S$ by assigning probability σ_q/σ_S to every $q \in S$. S is called* ***weak*** *if there exists a PPT algorithm $\mathcal{A}$ and two distinct indices x_1 and x_2 such that $\mathcal{A}$ can distinguish between queries for indices x_1 and x_2 with non negligible advantage over the distribution $\mathcal{D}_S$.*

Definition 5.5.3 (*Negligible Weakness*)

Let $P = \{P_1, \cdots, P_m\}$ be a partition of $\mathcal{Q}$ such that every P_j is near complete *and let $q \leftarrow \mathcal{D}$ be a randomly picked CSPS query. We call $P_q \in P$ the set to which q belongs. The CSPS scheme is deemed* ***negligibly weak*** *if the probability $\Pr(P_q \text{ weak} | q \leftarrow \mathcal{D})$ is negligible for any such partition.*

We assume the AGCD problem is negligibly weak (Definition 5.5.3).

Note that the assumption of **negligible weakness** from Definition 5.5.3 is a weaker assumption than the traditional security assumption stating that there exists no PPT algorithm able to distinguish queries for distinct indices x_1 and x_2 over $\mathcal{Q}$.

Proof: If there exists a PPT algorithm $\mathcal{A}$ able to distinguish queries for distinct indices x_1 and x_2, then the AGCD is clearly non negligibly weak (consider the partition $P = \{\mathcal{Q}\}$ where $\mathcal{Q}$ is near complete and weak because of $\mathcal{A}$).

Now let us assume the AGCD scheme is non negligibly weak. There exists a partition $P = \{P_1, \cdots, P_m\}$ such that each P_j is near complete and $Pr(P_q \text{ weak} | q \leftarrow \mathcal{D})$ is non negligible. If P_j is weak, we call $\mathcal{A}_j$ the PPT algorithm that can distinguish between indices over $\mathcal{D}_{P_j}$. If P_j is not weak, we write $\mathcal{A}_j$ the algorithm that returns 1 with probability 1/2. We could build an algorithm $\mathcal{A}$ distinguishing over $\mathcal{Q}$ by calling the right $\mathcal{A}_j$ for any input, but since there could be an exponential number of $\mathcal{A}_j$ there is no guarantee that $\mathcal{A}$ would be a PPT algorithm.

As such, the two notions are not necessarily equivalent. ■

Since the notion may not be equivalent to a standard well trusted assumption, we may wonder if it is a reasonable assumption. Clearly, if the CSPS scheme was non negligibly weak, then for a non negligible portion of queries we could build a PPT algorithm distinguish between indices for that query. Using the reduction from Section 4.4.4, we could actually show that given a polynomial number of near multiples of $p$we can build with non-negligible probability a PPT algorithm recovering p. This would be a major security concern for any application relying on the AGCD assumption. As such, we can only hope that the CSPS scheme is indeed non negligibly weak.

For any $Y = (y_1, \cdots, y_{\texttt{nb}}) \in \{0, \cdots, 2^{\lambda-K} - 1\}$, consider the set S_Y of all valid CSPS queries $(z_1, \cdots, z_{\texttt{nb}})$ for any p and any secret index x such that for all $i \in \{0, \cdots, \texttt{nb}\}$, $z_i \bmod 2^{\lambda-K} = y_i$.

Since the CSPS scheme is assumed to be non negligibly weak, with overwhelming probability there exists no PPT algorithm able to distinguish between two indices for all queries over S_Y when Y is picked randomly.

Note that such a PPT algorithm $\mathcal{A}_Y$ actually exists for some sets Y. For instance, if Y is such that many y_i are equal then with high probability at least two z_i will be equal (since there are only $2^K/(2\texttt{nb}) = 2^\sigma$ valid z_i on average for a given y_i). If $z_i = z_j$ then clearly they both are encryptions of 0 and neither i nor j is the secret index. This justifies our introduction of non-negligibility and its non-triviality.

Now, for two indices x_1 and x_2, let $q = \{z_i, i \in \{1, \cdots, \texttt{nb}\}\} \in \mathcal{Q}$ be a random query for the CSPS scheme retrieving either index x_1 or x_2 such that for all $i \in \{1, \cdots, \texttt{nb}\}$, $z_i \bmod 2^{\lambda-K} = z_i^{(s)}$ where $z_i^{(s)}$ are the parameters of the PSSPS scheme. Suppose the PSSPS scheme is not IND-CQA secure.

Then let us call $\mathcal{A}$ the algorithm that simulates the following interaction with a hypothetical adversary able to break the indistinguishability for x'_1 and x'_2 with non-negligible advantage. During the first phase, for every index i_j sent by the adversary, choose a random odd p_j over λ_p bits, build a random PSSPS query for index i_j and return it to the server. During the second phase, upon receiving $\{x'_1, x'_2\}$, return query $q' = \{z_i^{(c)}, i \in \{1, \cdots, \texttt{nb}\}\}$ where:

$$z_i^{(c)} = \begin{cases} \lfloor z_{x_1}/2^{\lambda-K} \rfloor \text{ if } i = x'_1 \\ \lfloor z_{x_2}/2^{\lambda-K} \rfloor \text{ if } i = x'_2 \\ \lfloor z_{x'_1}/2^{\lambda-K} \rfloor \text{ if } i = x_1 \\ \lfloor z_{x'_2}/2^{\lambda-K} \rfloor \text{ if } i = x_2 \\ \lfloor z_i/2^{\lambda-K} \rfloor \text{ otherwise} \end{cases} \tag{5.3}$$

We claim that q' is indistinguishable from a random query drawn from $\mathcal{D}_{x'_1}$ or $\mathcal{D}_{x'_2}$ where $\mathcal{D}_x$ is the distribution of all queries for secret index x. To show this, we prove that the most significant bits of a random CSPS query for index x with least significant bits equal to $z_i^{(s)}$ have the same distribution as a random query from $\mathcal{D}_x$.

Indeed, by construction Algorithm 2 returns for every $i \in \{1, \cdots, \texttt{nb}\}$ a value uniformly drawn among all near multiples of p with least significant bits matching $\{z_i^{(s)}\}$ since it computes *all* such near multiples and then returns one at random. Since two values generated for two distinct i are generated independently, this means that the query for the PSSPS parameter as a whole is generated uniformly among all CSPS queries where each z_i has its least significant bits matching $z_i^{(s)}$.

Besides, q is a query for x_b in the CSPS scheme if and only if q' is a query for x'_b in the PSSPS scheme. As such, we are able to distinguish with non-negligible advantage which index is retrieved by q using this adversary. This means that the set S_Y where $Y = (z_1^{(s)}, \cdots, z_{\texttt{nb}}^{(s)})$ which is near complete (because of K's selection in Section 5.4) is weak. By our security assumption (Definition 5.5.3), this means that such an adversary can only exist for a negligible portion of all queries q.

In other words, an instance of the PSSPS scheme set up with random parameters $\{z_i^{(s)}, i \in \{1, \cdots, \texttt{nb}\}\}$ is secure with overwhelming probability.

5.6 Additional Improvements

5.6.1 Alternative Client-Side Parameter Selection

In Section 5.3.2, we described an algorithm that allows the client to find bit values for which the values z_i will have the desired properties (namely, a close multiple of p with the right noise parity). In Section 5.4, we also detailed the actual complexity of this algorithm. While less than linear, it can still be quite expensive. We describe here a more clever approach that allows the client to find valid parameters much more efficiently.

Once again, we work in the convenient setting where CI is the set consisting of the K most significant bits of each z_i. For every $i \in \{1, \cdots, \mathtt{nb}\}$, the client needs to find $z_i^{(c)}$ such that $z_i^s + 2^{\lambda-K} z_i^{(c)} \bmod p = \epsilon < p/(2\mathtt{nb})$ and $(z_i^{(s)} + 2^{\lambda-K} z_i^{(c)} \bmod p) \bmod 2 = \delta_{i,x}$.

Since we want to use a lattice reduction algorithm to find such small solutions, we first "shift" the unknowns such that the solution range is symmetrical around 0. We write $\Delta = 2^{-1} 2^{\lambda-K} \bmod p \in \mathbb{Z}_p$, $\epsilon_{max} = \frac{p}{2\mathtt{nb}}$ and $c = 2^{-1}(z_i^{(s)} - \delta_{i,x}) \bmod p \in \mathbb{Z}_p$. We set $z' = z_i^{(c)} - 2^{K-1}$ and $\epsilon' = \epsilon - \epsilon_{max}/2$ and $c' = c - \epsilon_{max}/2 + \Delta 2^{K-1}$.

Solving equation $z 2^{\lambda-K} + z_i^{(s)} = 2\epsilon + \delta_{i,x} \mod p$ with $0 < \epsilon < \frac{p}{2\mathtt{nb}}$ and $0 < z < 2^K$ is equivalent to solving equation 5.4 for $-2^{K-1} \leq z < 2^{K-1}$ and $-\frac{1}{2} \cdot \frac{p}{2^{2\mathtt{bpw}}\mathtt{nb}} \leq \epsilon' < \frac{1}{2} \cdot \frac{p}{2^{2\mathtt{bpw}}\mathtt{nb}}$:

$$z'\Delta + c' = \epsilon' \mod p \tag{5.4}$$

Now we consider the dimension 3 lattice generated by the following base represented as the following 3×3 matrix:

$$\begin{pmatrix} \Delta' & 2^{\lambda_\epsilon - K - 1} & 0 \\ c' & 0 & \epsilon_{max}/2 \\ p & 0 & 0 \end{pmatrix} \tag{5.5}$$

Because of the way we selected K in Section 5.4, we know that if there exists a solution, then there should exist 2^σ solutions on average to equation 5.4. In such low dimensions, we are guaranteed to find one of those short solutions using reduction algorithms such as LLL [LLL].

Our target vector is:

$$(c_1\Delta' + c' \bmod p, c_1 2^{\lambda_\epsilon - K}, 2^{\lambda_\epsilon}) \text{ with } c_1 < 2^K \tag{5.6}$$

Since at least one vector in the reduced basis has a non-zero third coefficient c_2, we will have $|c_2| = 1$ with high probability for that vector. All the other coefficients of the vector will be smaller, ie $|\epsilon'| < 2_\epsilon^\lambda/2$ and $|c_1| 2^{\lambda_\epsilon - K} < 2^{\lambda_\epsilon - 1}$ which means $|c_1| < 2^{K-1}$. Now $\epsilon \bmod 2$ is random and equal to $\delta_{i,x}$ with probability 1/2. If it has the wrong parity, we repeat the process with slightly different coefficients until we find a satisfying combination.

There is a lot of freedom when it comes to the selection of the lattice reduction algorithm. In particular, since the input basis has dimension 3, we can use the results from [Sem01] which provides an algorithm running in time $O(\lambda_p^2)$ (whereas the regular LLL algorithm runs in time $O(\lambda_p^3)$)).

Alternatively, we can search for vectors close to $(-c', 0)$ in the two-dimensional lattice generated by $\begin{pmatrix} \Delta \\ 2^{\lambda_\epsilon - K - 1} \end{pmatrix}$ and $\begin{pmatrix} p \\ 0 \end{pmatrix}$. We know that if solutions to equation 5.4 exist then there are vectors $\begin{pmatrix} \epsilon' \\ z' 2^{\lambda_\epsilon - K - 1} \end{pmatrix}$ away from the target. Indeed, $z' \begin{pmatrix} \Delta \\ 2^{\lambda_\epsilon - K - 1} \end{pmatrix} + \begin{pmatrix} c \\ 0 \end{pmatrix} = \begin{pmatrix} \epsilon' \\ z' 2^{\lambda_\epsilon - K - 1} \end{pmatrix} + k \begin{pmatrix} p \\ 0 \end{pmatrix}$ for some $k \in \mathbb{Z}$.

In either case, once a solution has been found, other solutions can easily be discovered by finding small roots $(r_1, r_2) \in \mathbb{Z}_p$ of the linear equation $r_1 \Delta' = r_2$. Indeed, if (z', ϵ') is a solution then every solution has the form $(z' + r_1, \epsilon' + r_1 \Delta')$ for small r_1 and r_2.

For middle-sized to large databases (at least 2^{20} blocks), even using LLL on a small matrix with coefficients over λ_p bits will outperform an exhaustive search over K bits. This was confirmed by actual implementation of the client-side parameter search algorithm. Recall that exhaustive search as described in Section 5.3.2 required $\lambda_p \mathtt{nb}^2 2^{\sigma+1}$ operations to guarantee success with probably at least $1 - 2^{-\sigma}$.

Note that when using this algorithm, the distribution of $z_i^{(c)}$ (the K most significant bits of $pq_i + 2\epsilon_i$) and the distribution of ϵ_i is not uniform among all near multiples of p matching $z_i^{(s)}$ unless we find the full set of small roots and uniformly draw a root from this set. The uniformity of the distribution was used as an argument in the security proof of Section 5.5.

The distribution of the most significant bits seems widely irrelevant to the security of the scheme. The distribution of the noise could be an issue based on the fact that the lattice reduction algorithm will tend to minimize the noise value, therefore reducing security. However even the smallest noise value out of the $2^K = 2^{\sigma+1}\mathtt{nb}$ will still be on the order of $\frac{p}{\mathtt{nb}}$. If it is too low to satisfy the security conditions detailed in Section 4.4.5, the lattice reduction is run with slightly different coefficients.

5.6.2 Second layer of precomputations

Naturally, the preprocessing described in this section is compatible with the more straightforward techniques discussed in Chapter 4. Intuitively, the server can precompute short sums for every possible input and the online phase will then be performed several bits at a time. Overall, this allows the scheme to perform faster by a constant factor at an exponential spatial cost. This spatial limitation coupled with the necessity for a large security parameter meant the scheme was still not able to perform in sublinear time. We can now combine both techniques to achieve our improved result.

We use the same preprocessing parameter r described in Section 4.3.2. The server precomputes and stores all possible linear combination of r consecutive bits in adjacent blocks. This requires an expansion from an n-bit database to a $n2^r/r$-bit one and improves the computational complexity by a factor of r.

Combined with our preprocessing technique from Section 5.3, this puts the overall server-side complexity at $\frac{K}{r}n$ elementary operations and the total number of bits read server-side becomes $\frac{K\log_2 r}{r}n + o(n)$.

Now, the server-side computational complexity is $(K/r)n$ while on the client side it is $\lambda_p 2^K \texttt{nb} = \lambda_p \texttt{nb}^2 2^{\sigma+1}$. The overall complexity is $(K/r)n + \lambda_p \texttt{nb}^2 2^{\sigma+1}$.

If we set $r = K/\epsilon$ $(\epsilon < 1)$, the space requirement is $2^r n/r = \frac{\epsilon\left(\texttt{nb}2^{\sigma+1}\right)^{1/\epsilon}}{\log_2(\texttt{nb}2^{\sigma+1})}$ bits and offers a sublinear overall complexity at $\epsilon n + \lambda_p \texttt{nb}^2 2^{\sigma+1}$ when $\texttt{nb}$ is somewhat smaller than $\sqrt{n}$. The number of bits read server-side becomes less than n when $K < r\log_2 r$. While this is still prohibitive, it will be improved upon in Section 6.4.

When using the technique from Section 5.6.1, the client-side complexity becomes $O(\lambda_p^2)$ which is negligible in n even for the optimal value $\texttt{nb} = \texttt{bpb} = \sqrt{n}$, offering sublinear complexity.

The client sends $\texttt{nb}K$ bits and the server returns $\texttt{bpb}\lambda$ bits giving an overall sublinear communication at $\texttt{nb}(\log_2 \texttt{nb} + \sigma + 1) + \texttt{bpb}\lambda$ bits.

5.6.3 Updating the Database

In Section 4.4.7, we explained how preprocessing over r bits requires 2^{r-1} updates to preprocessed values whenever one bit of the original database is flipped, which is a reasonable amount. Now, we can also look at at the cost of modifying the original database when using PSSPS preprocessing.

Recall that in this setting, the server precomputes for $j \in \{1, \cdots, \texttt{bpb}\}$ and $k \in \texttt{SI}$ (with $|\texttt{SI}| = \lambda - K$) the following values:

$$res_{j,k} = \sum_{i=1}^{\texttt{nb}} z_{i,k}^{(s)} b_{i,j} \tag{5.7}$$

Clearly, when flipping bit $b_{i,j}$ from v_0 to v_1, the only preprocessed values requiring updating are $res_{j,k}$ for every $k \in \texttt{SI}$. One has to set $res'_{j,k} = res_{j,k} + z_{i,k}^{(s)}(v_1 - v_0)$.

In fact, we can go a step further. During the online phase, the values $res_{j,k}$ are never used by themselves. Only $res_j^{(s)} = \sum\limits_{k \in \mathtt{SI}} 2^k res_{j,k}$ is (see Section 5.3.3). Actual implementation would thus naturally only precompute and store $res_j^{(s)}$ for $j \in \{1, \cdots, \mathtt{bpb}\}$ rather than every $res_{j,k}$. The server should also further precompute and store $z_i^{(s)} = \sum\limits_{k \in \mathtt{SI}} 2^k z_{i,k}^{(s)}$. In this situation, flipping bit $b_{i,j}$ only requires an update: $res_j^{\prime(s)} = res_j^{(s)} + z_i^{(s)}(v_1 - v_0)$.

To summarize, flipping a bit in the original database requires 2^{r-1} updates with the method from Section 5.6.2, only 1 constant-time update with PSSPS and $2^{r-1} + 1$ when combining both (since the two methods are performed independently).

5.7 Ring LWE

In 2014, Aguilar Melchor *et al.* introduced a new PIR scheme aiming for computational efficiency. It is called XPIR and is based on the Ring-LWE assumption. While the existence of the scheme was already mentioned in [BV11], Aguilar Melchor *et al.* showed how efficiently it can be implemented and how some precomputations can speed it up. In this section, we show that our preprocessing technique is also compatible with this scheme.

5.7.1 Notations

We work in $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$. $\lambda = \log_2 q$ and N is a security parameter. χ is a Gaussian distribution of "small" polynomials from R_q with standard deviation r. The client has a secret key sk picked from χ. t_v designates the element of R_q with every coefficient equal to t. $a \otimes b$ is the product coefficient by coefficient while $a * b$ is the polynomial product modulo $X^N + 1$.

To encrypt a message $m \in R_t$ (every coefficient is less than t), return (q_1, q_2) where $q_1 \leftarrow R_q$ and $q_2 = q_1 * sk + e \otimes t_v + m$ with e some small noise drawn from χ. To decrypt (q_1, q_2), compute $q_2 - (q_1 * s) \bmod t$. Note that usually, we can think of t as equal to 2 (although larger values are possible).

We also define

$$\begin{gathered} Sum((a_1, a_2), (b_1, b_2)) = (a_1 + a_2, b_1 + b_2) \\ Absorb(k, (a_1, a_2)) = (k * a_1, k * a_2) \end{gathered} \tag{5.8}$$

Now to retrieve block x from the server, the client generates for every block a value $z_i \in R_q^2$ which is an encryption of $\delta_{i,x}$ (1 when $i = x$, 0 otherwise). The server returns $res_j = Sum_{i=1}^{\mathtt{nb}} Absorb(w_{i,j}, z_i)$. The client then recovers $w_{x,j}$ by using the decryption algorithm on res_j.

For more details about this protocol and its security, refer to [MBFK16].

5.7.2 Preprocessing Techniques

The main computational cost of this scheme is the product between polynomials in R_q. This can be made more efficient by using an NTT (Number-Theoretic Transform) representation for polynomials and CRT (Chinese Remainder Theorem) for integers. In this representation, both addition and multiplication can be done in near linear time in N. Usually the conversion between coefficient-based representation and NTT-CRT representation would cancel out the benefits from faster multiplication, but in the case of PIR such representations for the messages $w_{i,j}$ can be computed in advance and converting one z_i allows for `bpb` multiplications in linear time.

Yet another preprocessing trick involves the use of Newton Quotients to speed up scalar multiplication in $\mathbb{Z}_q$. At the cost of one integer division during preprocessing, further multiplications modulo q can be done in two integer multiplications, one bit shift and one conditional integer subtraction instead of one integer multiplication and one costly division without preprocessing.

Overall, the scheme performs much faster using these precomputations. Furthermore, the required storage for the precomputed values is minimal. However, this approach has no hope of bringing the computational cost under n, it merely linearizes the nonlinear parts of the computation.

5.7.3 Partial Server Side Parameter Selection for Ring-LWE

Recall that the client sends a value z_i for every $i \in \{1, \cdots, \texttt{nb}\}$. We write $z_i = (a_i, b_i)$.

Each $a_i \in R_q$ can be written as an N-dimensional vector over $\mathbb{Z}_q$: $a_i = (a_{i,1}, \cdots, a_{i,N})$ in the canonical embedding [BV11]. In this representation, both addition and multiplication are done coordinate by coordinate.

As before, each $a_{i,d}$ ($d \in \{1, \cdots, N\}$) can be split in a client part $a_{i,d}^{(c)}$ and a server part $a_{i,d}^{(s)}$. The same goes for b_i. We call K the number of bits for the client-side coefficients. $a_{i,d}^{(c)}$ is the K most significant bits of $a_{i,d}$ while $a_{i,d}^{(s)}$ is made of the remaining $\lambda - K$ least significant bits. Similarly, we write $sk = (sk_1, \cdots, sk_N)$. Now we have:

$$\begin{aligned} b_i - a_i sk &= (b_{i,1} - a_{i,1} sk_1, \cdots, b_{i,N} - a_{i,N} sk_N) \\ b_{i,d} - a_{i,d} sk_d &= 2^{\lambda-K}(b_{i,d}^{(c)} - a_{i,d}^{(c)} sk_d) + b_{i,d}^{(s)} - a_{i,d}^{(s)} sk_d \end{aligned} \tag{5.9}$$

We now place ourselves in a setting where every $a_{i,d}^{(s)}$ and $b_{i,d}^{(s)}$ is fixed. For instance, we can generate a random (a_i, b_i) according to the non-preprocessed scheme and set all the $a_{i,j}^{(s)}$ (respectively, $b_{i,j}^{(s)}$) to the set of the least $\lambda - K$ significant bits of each coefficient of a_i (respectively, b_i). The client is free to choose the values for the coefficients $a_{i,d}^{(c)}$ and $b_{i,d}^{(c)}$ over K bits.

The client selects a secret key sk from χ as usual.

Let $A_d(a,b) = 2^{\lambda-K}b + b_{i,d}^{(s)} - (2^{\lambda-K}a + a_{i,d}^{(s)})sk_d$. We build a distribution $\mathcal{D}$ over all 2^{2K} possible couples $(a,b) \in \{0, \cdots, 2^K - 1\}^2$.

We call $P_\chi(y,i) = \Pr(e \otimes t_v + \delta_{i,x} = y | e \leftarrow \chi)$. Note in particular that $P_\chi(y,i) = 0$ when $y \bmod t \neq \delta_{i,x}$.

We define $\sigma_i = \sum\limits_{(a,b)\in\{0,\cdots,2^K-1\}^2} P_\chi(A_d(a,b), i)$.

We define $\mathcal{D}$ such that $\Pr(X = (a,b) | X \leftarrow \mathcal{D}) = P_\chi(A_d(a,b), i)/\sigma$. $\mathcal{D}$ is well-defined when $\sigma \neq 0$, which is true when at least one couple $(a,b) \in \{0, \cdots, 2^K - 1\}^2$ is such that $A_d(a,b) \bmod t = \delta_{i,x}$. Note that this equation is true with probability $1/t$ (where t is usually 2) for any of the 2^{2K} couples.

The distribution $\mathcal{D}$ is an approximation of the distribution of all encryptions of $\delta_{i,x}$. For a random secret key picked according to distribution χ

Now K must be large enough that the noise from summing $\texttt{nb}$ values drawn from $\mathcal{D}$ does not go over $q/2$ with overwhelming probability. As a first approximation, we know that we need K large enough that for every $i \in \{1, \cdots, \texttt{nb}\}$, at least one value v out of the 2^{2K} possible is such that $v \bmod t = \delta_{i,x}$ and $|v| < q/(2\texttt{nb})$.

The first condition is satisfied with probability $1/t$. The second condition is satisfied with probability $1/\texttt{nb}$. These conditions match the ones detailed in Theorem 5.4.1. In other words, we need $2K > \log_2(t\texttt{nb}) + \sigma$ to guarantee the existence of such a v with probability at least $1 - 2^{-\sigma}$.

Now the server can precompute for every $j \in \{1, \cdots, \texttt{bpb}\}$ the value $Sum_{i=1}^{\texttt{nb}} Absorb(m_{i,j}, z_i^{(s)})$. During the online phase, the client computes $A(a,b)$ for every $(a,b) \in \{0, \cdots, 2^K - 1\}^2$, builds $\mathcal{D}$ as described earlier and samples a random $(a^{(c)_{i,d}}, b_{i,d}^{(c)})$ from that distribution. The client then sends all the $(a^{(c)_{i,d}}, b_{i,d}^{(c)})$ to the server.

The server only has to send back the precomputed values along with $res_j^{(c)} = Sum_{i=1}^{\texttt{nb}} Absorb(m_{i,j}, z_i^{(c)})$. All the precomputation techniques suggested by Aguilar-Melchor *et al.* still work in the exact same manner. Our technique simply reduces the input size from λ bits to K bits, thus speeding scalar multiplications involved in the polynomial multiplications. Depending on implementation, the expected speedup is about λ/K.

Security considerations are widely similar to those described in Section 5.5 and we do not detail them here.

5.8 Conclusion

We showed that in a Private Information Retrieval setting, a variety of protocols relying on additionally homomorphic schemes can be improved through the use of partial server-side parameter selection. This can lead to a speed up by an order of magnitude or more. The logic behind the scheme is generic enough that it could be used in completely unrelated areas of cryptography. Furthermore, this technique can be used on top of other preprocessing methods. This results in protocols performing in less than n elementary operations relying exclusively on computational assumptions.

The additional cost for the client is moderate and justifiable. As multi-server PIR schemes become very close to being practical, such improvements give hope that single-server schemes and their often stronger privacy properties will follow suit in the near future.

Chapter 6

Efficient Private Information Retrieval

6.1 Introduction

In this chapter, we discuss how to combine all the previous results in a very efficient scheme. Namely, we build a protocol able to privately execute complex SQL queries and retrieve the query result using the protocols developed and detailed previously.

6.2 Scheme Overview

Using the protocols detailed in Chapter 3, we can execute every query of the form:

```
SELECT outField$_1$, $\cdots$, outField$_{k_1}$
FROM table
WHERE Filter((inField$_1$, $s_1$), $\cdots$, (inField$_{k_2}$, $s_{k_2}$))
```

Here `Filter` is a logical formula using conjunctions and disjunctions with predicates being comparisons ($<$, $>$ or $=$) or `LIKE` operations between field `inField`$_i$ values and secret parameters s_i. This encompasses a very large portion of real-world queries used for essentially every online service.

In this case, executing a query simply means the client learns whether or not some entry in the database satisfies the `Filter` formula described in the query. Instead of directly recovering the bits from `outField`$_1, \cdots$ `outField`$_{k_1}$, the client will recover the entry index x in the database, which ranges from 1 to n and under which the entries are sorted. In particular, this means only $\lceil \log_2 n \rceil$ bits need to be recovered.

Now the client uses a block PIR protocol to retrieve the block associated with entry x for the output fields listed earlier. We will consider the use of our Approximate GCD-based protocol (Chapter 4) with all of its optimizations combined.

To achieve this, we use a precomputation over r bits when the input is decomposed over base 2 as detailed in Section 4.3.2 along with multiple bit words as described in Section 4.4.6, PSSPS for the parameter selection as in Section 5.4 and the efficient client-side parameter search from Section 5.6.1.

Security parameters and database decomposition has to be studied carefully to match the conditions required in each of these optimizations.

6.3 Recursive PIR

The 2-dimensional construction that was described in Section 4.2 and used extensively after that can be generalized to a d-dimensional construction in a natural way. This is a classic PIR trick which greatly reduces communication at the cost of some extra computation for the server. Under normal circumstances, we would not want to use such a trade-off since the communication time($O(\sqrt{n})$ in dimension 2) tends to be asymptotically negligible compared to the server's computation time (at least n). However this construction has another usually inconsequential side effect in that it significantly reduces the client's computational cost.

In our PSSPS construction, the client's computations can be very expensive, even approaching the server's. In this setting, a slight increase on the server side might be beneficial if the time saved client-side more than compensates it. We first present the construction and then discuss how it can be used together with the other optimizations we have developed.

We generalize over d dimensions the process from Section 4.2.

Set $n_1, \cdots, n_d \in \mathbb{N}$ such that $\prod_{i=1}^{d} n_i = n$ and index every b_i as $b_{i_1,\cdots,i_d}$ where $1 \leq i_j \leq n_j$. In practice, we tend to use $n_1 = \cdots = n_d = n^{1/d}$.

Suppose the client wants to recover $b_{x_1,\cdots,x_d}$. Then for every $c \in \{1, \cdots, d\}$, the client sends n_c values $z_{c,i}$ ($i \in \{1, \cdots, n_c\}$) such that $z_{c,i} = pq_{c,i} + 2\epsilon_{c,i} + \delta_{x_c,i}$ for appropriate q_{c_i} and $\epsilon_{c,i}$.

We write $res_{1,i_1,\cdots,i_d} = b_{i_1,\cdots,i_d}$.

The server computes $res_{c+1,i_{c+1},\cdots,i_d} = \sum_{i=1}^{n_c} z_{c,i} res_{c,i,i_{c+1},\cdots,i_d}$ whenever defined. He returns res_{d+1} to the client. Then the client knows that:

$$(res_{d+1} \bmod p) \bmod 2 = b_{x_1,\cdots,x_d} \tag{6.1}$$

The communication complexity is $\lambda \sum_{c=1}^{d} n_c$ which is $\lambda d n^{1/d}$ if all n_j are equal to $n^{1/d}$.

The client-side computational complexity is also $O(\lambda \sum_{c=1}^{d} n_c)$. The server-side computational complexity is $\lambda \sum_{i=1}^{d} \prod_{c=i}^{d} n_c$. In particular when $n_1 = \cdots = n_d = n^{1/d}$, it becomes $\lambda \sum_{i=1}^{d} n^{i/d} = \lambda n + o(n)$.

Note that the scheme can also be interrupted at step $c < d+1$. The server then returns $res_{c,i_c,\cdots,i_d}$ for all $n_c n_{c+1} \cdots n_d$ or $n^{(d-c)/d}$ values in total for the case $n_1 = \cdots = n_d$. In other words, the client is able to retrieve blocks of $n^{k/d}$ bits for any $1 \leq k \leq d$.

Compared to the simple 2-dimensional case, this reduces communication significantly and increases computation cost slightly. Note that the communication cost per bit recovered actually grows with the dimension. If a client recovers at least $\sqrt{n}$ bits — as is often the case in practice — then $d = 2$ provides the most efficient setting.

Looking at how the algorithm works in details, one notices that while $res_{2,i_2,\cdots,i_d}$ depends linearly on the client-sent values $z_{1,i}$ and the local database variables $b_{i_1,\cdots,i_d}$, all other values $res_{c+1,i_{c+1},\cdots,i_d}$ for $c > 1$ will depend linearly on the result of the previous computations (hence the recursive aspect of the scheme). This represents a major obstacle to the use of preprocessing techniques.

Indeed, both the PSSPS (Section 5.3) and base-2 preprocessing (Section 4.3.2) techniques require an offline phase where the server needs to know exactly which values will be multiplied with the input sent by the client. In the d-dimensional case, this can therefore only be used up to the computation of $res_{2,i_2,\cdots,i_d}$. Luckily, the cost of computing every $res_{2,i_2,\cdots,i_d}$ is $O(\lambda n)$ while the cost of computing $\{res_{j,i_j,\cdots,i_d}, j \in \{3,\cdots,d\}\}$ is only $O(\lambda n^{1-(j-2)/d}) = o(n)$.

So applying both techniques to the recursive scheme will still result on an asymptotic complexity of $O(Kn/r) + O(\lambda n^{(d-1)/d}) = O(Kn/r)$ as usual.

Let us now look at the client's computational complexity. When using PSSPS over the 2-dimensional scheme, it increases from $O(\lambda \texttt{nb})$ to $O(\lambda_p \; \texttt{nb} \; 2^K)$

for the exhaustive search approach or $O(\lambda_p^2\ \mathtt{nb})$ for the LLL-based approach. However in a d-dimensional construction, only $n_1 = n^{1/d}$ values $z_{1,i}^{(c)}$ need to be searched based on the fixed bits $z_{1,i}^{(s)}$. Every $z_{c,i}$ for $c > 1$ is generated randomly as before by picking $q_{c,i}$ and $\epsilon_{c,i}$ and setting $z_{c,i} = pq_{c,i} + 2\epsilon_{c,i} + \delta_{x_c,i}$. As such, client computational complexity is $O(\lambda_p n^{1/d} 2^K + \lambda(d-1)n^{1/d})$ for exhaustive search and $O(\lambda_p^2 n^{1/d} + \lambda(d-1)n^{1/d})$ is considerably faster than in dimension 2 in either case.

To sum up this whole section, as it turns out preprocessing for a d-dimensional construction offers almost the same improvement as on a 2-dimensional even though preprocessing can only be used on the first dimension. Meanwhile, communication is significantly reduced and so are the client's computations.

6.4 Using Words on Several Bits

The original scheme from Chapter 4, along with some others including [Gol07] could be used on words with more than a single bit. This leads to a significant improvement in computations at the cost of an increase in block size. In this situation, we will write the database as a set of words $w_{i,j}$ of `bpw`-bit (**b**its **p**er **w**ord) each with $1 \leq i \leq \mathtt{nb}$ and $1 \leq j \leq \mathtt{wpb}$ (**w**ords **p**er **b**lock).

In our case, using words on `bpw` bits would require a new definition for the parameters z_i selected by the client. We have to set $z_i = pq_i + 2^{\mathtt{bpw}}\epsilon_i + \delta_{i,x}$ for $i \in \{1, \cdots, \mathtt{nb}\}$ and the server returns $res_j = \sum_{i=1}^{\mathtt{n}} z_i w_{i,j}$ for $j \in \{1, \cdots, \mathtt{wpb}\}$. The condition on the ϵ_i values becomes:

$$\forall j \in \{1, \cdots, \mathtt{wpb}\},\ \sum_{i=1}^{\mathtt{nb}} \epsilon_i w_{i,j} < \frac{p}{2^{\mathtt{bpw}}} \tag{6.2}$$

Which, in particular, is satisfied if the following is true for all $1 \leq i \leq \mathtt{nb}$:

$$\epsilon_i < \frac{p}{2^{2\mathtt{bpw}}\ \mathtt{nb}} \tag{6.3}$$

When receiving the server's response res_j, the client now computes $(res_j \bmod p) \bmod 2^{\mathtt{bpw}}$ to recover the j-th `bpw`-bit word in block x.

However this minor difference changes the parameter selection described in Section 5.4. The probability that a single candidate z in Algorithm 2 is such that $z \bmod p < p/(2^{\mathtt{bpw}}\mathtt{nb})$ is $1/(2^{\mathtt{bpw}}\mathtt{nb})$. Meanwhile, the probability that $(z \bmod p) \bmod 2^{\mathtt{bpw}} = \delta_{i,x}$ is $1/2^{\mathtt{bpw}}$. Overall, the probability that z be a suitable candidate is reduced from $1/(2\mathtt{nb})$ in the single-bit version down to $1/(2^{2\mathtt{bpw}}\mathtt{nb})$.

One may wonder why setting $\mathtt{bpw} = 1$ does not give us the previous result. This is because we used a less conservative condition. The more accurate condition would be $z \bmod p < p/((2^{\mathtt{bpw}} - 1)\mathtt{nb})$ which is close to the one we use for large enough bpw. We now study how to select K under these conditions.

When working with words over bpw bits, the equation to solve is modified. It becomes:

$$z2^{\lambda-K} + z_i^{(s)} = 2^{\mathtt{bpw}}\epsilon + \delta_{i,x} \mod p \tag{6.4}$$

We are looking for a solution to it such that $0 < \epsilon < \frac{p}{2^{2\mathtt{bpw}}\mathtt{nb}}$ and $0 < z < 2^K$. As before, we write $\Delta = (2^{\mathtt{bpw}})^{-1}2^{\lambda-K} \bmod p \in \mathbb{Z}_p$ and $c = (2^{\mathtt{bpw}})^{-1}(z^{(s)} - \delta_{i,x}) \bmod p \in \mathbb{Z}_p$. We rewrite equation 6.4 as $z\Delta + c \bmod p = \epsilon$.

Following the steps used in the single-bit word case from Section 5.4, the condition on the number of clusters M becomes $M \leq 2^{2\mathtt{bpw}}\mathtt{nb}$ and we reach the following condition on K: setting $K = \sigma + \log_2(2^{2\mathtt{bpw}}\mathtt{nb}) = \sigma + 2\mathtt{bpw} + \log_2 \mathtt{nb}$, this gives us a success probability greater than $1-2^{-\sigma}$ for a uniformly random p.

This increase has several consequences. First, the client-side computations will increase by a factor of $2^{2\mathtt{bpw}-1}$ if using the exhaustive search algorithm (Algorithm 2). Since this becomes prohibitive, the LLL-based approach (Section 5.6.1) has to be used instead.

Finally, using words with more than one bit affects security slightly but not in any threatening way. See Section 4.4.6 for details. The overall performance of the scheme is significantly improved. Indeed, so long as $\mathtt{bpw}+\log_2 \mathtt{nb}$ is less than the word size used by the architecture (typically 64 although 128 bit operations are also common), working on bpw-bit words reduces the per-recovered-bit communication and the overall computation cost by a factor of bpw at the cost of recovering blocks $\sqrt{bpw}$ larger. See Section 4.4.6 for justification.

Overall, server-side computation cost becomes:

$$\begin{aligned}\frac{Kn}{\mathtt{bpw}} &= n\frac{2\mathtt{bpw} + \sigma \log_2 \mathtt{nb}}{\mathtt{bpw}} \\ &= n(2 + \frac{\sigma + \log_2 \mathtt{nb}}{\mathtt{bpw}})\end{aligned} \tag{6.5}$$

As detailed in Section 5.6.3 for the case $\texttt{bpw} = 1$, modifying the value of a word in the original database only requires a single constant-time update to the preprocessed database values. This approach is thus highly compatible with non-static databases.

6.5 Complete Construction

Now we can combine the construction over several words with the PSSPS optimization and observe that the server can still precompute over r bits sent by the server.

The expanded database size will be $n2^r/r$ which significantly limits the maximum value of r. However, since the other optimizations did not require additional space (thanks to the fact that every $z_i^{(s)}$ can be set to 0), this still seems realistic for $r \leq 8$.

The final server-side complexity is now r times faster than described in the previous section, or $\frac{Kn}{r\ \texttt{bpw}} = n(2/r + \frac{\log_2 \texttt{nb}+1+\sigma}{r\ \texttt{bpw}})$ which is well below n for real-world parameters.

By using the d-dimensional construction detailed in Section 6.3 on top of everything else, we further reduce the communication cost considerably, improve the client side computational cost and barely affect the server-side computational cost.

Note that the query-batching techniques described in Section 4.4.8 can also be added on top of everything else to add yet another layer of speedup. However, this require a less-generic setting than what we are mainly intested in. In particular, these techniques will only provide an improvement if the server receives more than $\texttt{nb}^\alpha$ queries before it can answer the first one, for an $\alpha > 0.8074$. A lot of PIR applications will not satisfy this specific condition.

SQL Compatibility

Note that this scheme is suited to be used as a recovery scheme that follows a private SQL query such as the system described in Chapter 3 or Chor's PIR by Keywords [BCN98]. When combined this way, we address all three major issues with Private Information Retrieval in general: the hardness to find the physical location of the desired information on the server, the sometimes superfluous privacy requirements and the need to perform at least as many operations as the database is large.

6.6 Discussion and Comparison

As described in Section 4.5, we are interested in schemes achieving the same level of privacy in the same realistic settings. We can discard pre-2007 schemes which were proven inefficient by Sion [Sio07].

While different schemes may have slight advantages for different parameters, when it comes to our setting where every known scheme performs linearly, the better scheme will almost always be faster regardless of parameters. Surveys comparing PIR schemes have been published in the past [Sio07, OG12] and one of the most recent notable constructions is XPIR [MBFK16] which is an implementation-focuses contribution for which the authors showed the superior processing speed of the described Ring LWE-based scheme. This was independently confirmed by Gupta *et al.* [GCM+16] who chose XPIR as the fastest PIR implementation for a real world application.

We therefore directly compare our work with XPIR which is free software and conveniently available. We are mainly interested in comparing the theoretical complexity and, to a lesser extent, running time from an actual implementation.

For each $j \in \{1, \cdots, \texttt{wpb}\}$, the XPIR scheme performs $2\texttt{nb}$ sums of elements of $R_q = \mathbb{Z}_q[X]\langle X^N + 1\rangle$ and $2\texttt{nb}$ products in R_q. The products are simplified by using precomputations detailed in Section 5.7.2, but still require at least N elementary operations each. Denoting by n the number of bits in the database, this means that at least $2n$ elementary operations are required to compute the sums and at least another $2n$ to compute the products. Even with an extremely conservative estimation, server-side computations would require at least $4n$ operations.

Meanwhile, theory indicates that our scheme eventually reaches a less-than-n number of operations for large enough preprocessing parameters.

Even for practical values used in a straightforward implementation (single-threaded, no pipelining or implementation-specific optimizations), we are able to match the server-side running times of XPIR. Consider the following set of parameters for a 1 Gigabit database:

$$\begin{aligned}
&n = 2^{30}, \texttt{nb} = \texttt{wpb} = 2^{13}, \texttt{bpw} = 16 \\
&\lambda_\epsilon = 80, \lambda_p = \lambda_\epsilon + 2\texttt{bpw} + \log_2 \texttt{nb} + 1 = 126, \lambda = \lambda_\epsilon \lambda_p \log_2 \texttt{nb} = 131,040 \\
&\sigma = 20, K = 2\texttt{bpw} + \sigma + \log_2 \texttt{nb} = 65
\end{aligned} \tag{6.6}$$

The first thing we notice is the tremendous improvement gained by replacing a $O(\lambda n)$ complexity by a $O(Kn)$ since K is about 2000 times smaller.

For this set of parameters, client-side parameter selection of all nb values $z_i^{(c)}$ took 1.36s while the server-side computation only required 0.40s. Asymptotically, client-side complexity grows linearly in nb while server-side complexity grows linearly in $n = \texttt{nb}^2$. The offline precomputation phase on the server required a modest 637s.

Here is the evolution for some other database sizes when every other parameter is picked the same way:

Database Size (bits)	2^{30}	2^{32}	2^{34}	2^{36}
Client Selection Time (s)	1.36	2.95	6.34	14.74
Server Computation Time (ms)	259	891	3170	12000

Note that for larger values, memory limitations make the testing environment unrealistic.

Now for the same database, letting XPIR auto-optimize its set of parameters and running the efficient Ring-LWE implementation, reported total processing time was 0.63s.

The main contribution remains the theoretical breaking of the n threshold even though it cannot be realized for practical parameters at this point.

We summarize this result by comparing the complexities of various important PIR schemes:

6.7 Conclusion

By combining a great number of independent optimizations all providing a linear improvement over the base case, we are able to reduce the constant factor in the complexity of the algorithm so much that it becomes less than 1 and the information theoretical restriction stating that n bits have to be read to provide security is finally broken. Now, considering the fact that some protocols were already able to perform on real world data for less than 4 times the cost of their non-private counterparts [GCM+16], this makes our final construction a prime contender for practical Private Information Retrieval over large databases.

Chapter 7

Conclusion

7.1 Concluding Remarks

We introduced a number of strategies allowing Private Information Retrieval protocols to become as close to practicality as we can reasonably expect them to be at this point. One main strategy is the ability to perform a broad array of SQL-like queries covering most practical situations. This required the development of a system that essentially built encrypted function which are executed over unencrypted data, compress it and still allow the client to recover information about the function's output from the compressed data. The system is an open framework which can be instantiated with different families of primitives. We suggest such a family based on systems of multivariate polynomials which has not been broken yet and may prove secure. Others can be designed in the future if the need arises. These findings are mainly theoretical and give insight into the great expressiveness of generalized definitions of private information retrieval.

The other main strategy is the development of generic optimization methods which can be used over a variety of schemes, along with a particularly efficient scheme thanks to its simplicity. The first optimization method consists in splitting a client's query into short chunks for which the output has already been computed at the cost of an exponential expansion of the database in the chunks' size. It had already been used by Beimel *et al.* [BIM04] in a multi-server setting. We showed how it can be beneficial to different multi-server schemes, but also to some single-server schemes.

The second optimization method consists in working on multiple bit words rather than individual bits. Taking advantage of the architecture on which the algorithm is executed allows for an important computational speed increase at no noticeable cost. This method had also been used in multi-server settings by Goldberg [Gol07] and once again we showed that it has a universality to it which should be exploited in single-server schemes also. While this very significantly reduces running time of the algorithm, it relies on the fact that modern computers naturally process multiple bits at once. This is thus less of a theoretical finding and more of a practical one.

The third optimization is brand new and reduces the size of the public parameters selected by the client. In the examples we provide, it allows for a very significant linear boost as it essentially bypasses the requirements for the security parameter. Suprisingly and perhaps counter-intuitively, this does not affect security in any significant way when careful parameter selection is done. In our case, the improvement remains linear but is by far the largest running time improvement. It has both theoretical and practical implications.

All of those optimizations being compatible with a scheme that naturally performs in a very efficient manner allows us to present a scheme that outperforms every other in the most generic setting, which is highly desirable. Indeed, it does not require any non mathematical assumption such as placing any level of trust on the server, its hardware or its independence from other services.

7.2 Future Directions

It is our hope that the Partial Server Side Parameter Selection can be used to build schemes, in particular lattice-based schemes, that would perform in sublinear complexity in the database's size and under a reasonable polynomial preprocessing expansion. Naturally, the ultimate long term goal for generic PIR would be the development of a $O(\log n)$ algorithm running only a constant times slower than non-private alternatives. While this is a very far-fetched goal, there is currently no research that would suggest that it is impossible. If such a scheme existed, PIR could be cheaply implemented in most online services. Considering the fact that preprocessing is a requirement for all sublinear schemes, our results are a significant step towards that long-term goal.

In another area, it seems entirely possible to build a system able to privately execute even more expressive SQL-queries. Over a single entry, any function could potentially be executed in a Fully Homomorphic Encryption fashion except running on unencrypted data. More specifically, dealing with nested queries and the like would prove useful. Finding secure instances for our SQL scheme is another project that could benefit from additional research.

A challenge posed by PIR with preprocessing is the heavy cost of updating the preprocessed values when the database is updated, which can be an extremely common occurrence on actual systems. The development of a scheme that requires a minimal number of preprocessing when an update is done is something to look forward to. Alternatively, a system that can work with both preprocessed values and newly updated one in parallel would limit the constant need for updating the preprocessed values.

Finally, one could start wondering whether some PIR schemes could be built specifically to target search engines. Research in this direction has proven unfruitful as the databases work in much more complex and dynamic ways than a simple list of $(field, value)$ which we used to modelize them. In particular, efficient set intersection computation would be required, which falls outside of the models we have developed.

Bibliography

[AF02] Dmitri Asonov and Johann-Christoph Freytag. Repudiative information retrieval. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES '02, pages 32–40, New York, NY, USA, 2002. ACM.

[AF03] Dmitri Asonov and Johann-Christoph Freytag. *Privacy Enhancing Technologies: Second International Workshop, PET 2002 San Francisco, CA, USA, April 14–15, 2002 Revised Papers*, chapter Almost Optimal Private Information Retrieval, pages 209–223. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Amb97] Andris Ambainis. Upper bound on communication complexity of private information retrieval. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, ICALP '97, pages 401–407, London, UK, UK, 1997. Springer-Verlag.

[BC09] Julien Bringer and Hervé Chabanne. Another look at extended private information retrieval protocols. In *Proceedings of the 2Nd International Conference on Cryptology in Africa: Progress in Cryptology*, AFRICACRYPT '09, pages 305–322, Berlin, Heidelberg, 2009. Springer-Verlag.

[BCN98] Niv Gilboa Benny Chor and Moni Naor. Private information retrieval by keywords. Cryptology ePrint Archive, Report 1998/003, 1998. `http://eprint.iacr.org/`.

[BCPT07] Julien Bringer, Hervé Chabanne, David Pointcheval, and Qiang Tang. Extended private information retrieval and its application in biometrics authentications. In *Proceedings of the 6th International Conference on Cryptology and Network Security*, CANS'07, pages 175–193, Berlin, Heidelberg, 2007. Springer-Verlag.

[BGH$^+$13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, pages 102–118, Berlin, Heidelberg, 2013. Springer-Verlag.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 784–796, New York, NY, USA, 2012. ACM.

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: Pir with preprocessing. *J. Cryptol.*, 17(2):125–151, March 2004.

[BM89] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 547–557, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[Bre01] Tønnes Brekne. *Encrypted Computation*. PhD thesis, Norwegian University of Science and Technology, 2001.

[BS03] Amos Beimel and Yoav Stahl. Robust information-theoretic private information retrieval. 2576:326–341, 2003.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 79–88, New York, NY, USA, 2006. ACM.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 41–, Washington, DC, USA, 1995. IEEE Computer Society.

[Cho] Amy Chozick. Bloomberg admits terminal snooping. *New York Times*. `http://nyti.ms/YDZbxd`.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[CN12] Yuanmi Chen and Phong Q. Nguyen. *Advances in Cryptology – EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, chapter Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers, pages 502–519. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Cop96a] Don Coppersmith. *Advances in Cryptology — EUROCRYPT '96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings*, chapter Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known, pages 178–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Cop96b] Don Coppersmith. *Advances in Cryptology — EUROCRYPT '96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings*, chapter Finding a Small Root of a Univariate Modular Equation, pages 155–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Cop97a] Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, March 1997.

[Cop97b] Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[CW87] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

[DC14] Changyu Dong and Liqun Chen. *Computer Security - ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, chapter A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost, pages 380–399. Springer International Publishing, Cham, 2014.

[DGH12] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 13–13, Berkeley, CA, USA, 2012. USENIX Association.

[DH06] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[Dic] K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude.

[GCM+16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107, Santa Clara, CA, March 2016. USENIX Association.

[GDHH14] Ian Goldberg, Casey Devet, Paul Hendry, and Ryan Henry. Percy++ project on sourceforge. `http://percy.sourceforge.net`, 2014. [version 1.0. Accessed Jan. 2015].

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[GHGKN06] Nicolas Gama, Nick Howgrave-Graham, Henrik Koy, and Phong Q. Nguyen. *Advances in Cryptology - CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings*, chapter Rankin's Constant and Blockwise Lattice Reduction, pages 112–130. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 151–160, New York, NY, USA, 1998. ACM.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[Gol07] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 131–148, Washington, DC, USA, 2007. IEEE Computer Society.

[GP] Barton Gellman and Laura Poitras. U.s., british intelligence mining data from nine u.s. internet companies in broad secret program. *Washington Post*.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32Nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 803–815, Berlin, Heidelberg, 2005. Springer-Verlag.

[Gra08] Andrew Granville. Smooth numbers: computational number theory and beyond, 2008. `http://www.dms.umontreal.ca/~andrew/PDF/msrire.pdf`.

[GS16] Kristian Gjøsteen and Martin Strand. Fully homomorphic encryption must be fat or ugly? Cryptology ePrint Archive, Report 2016/105, 2016. `http://eprint.iacr.org/2016/105`.

[GY07] William Gasarch and Arkady Yerukhimovich. Computational inexpensive cpir. 2007. `https://www.cs.umd.edu/~arkady/papers/pirlattice.pdf`.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *Revised Papers from the International Conference on Cryptography and Lattices*, CaLC '01, pages 51–66, London, UK, UK, 2001. Springer-Verlag.

[HHG13] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 262–271, New York, NY, USA, 2004. ACM.

[Jac] Jasper Jackson. Social media overtakes entertainment as favourite online activity. *The Guardian*. `http://gu.com/p/4c24p/sbl`.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 364–, Washington, DC, USA, 1997. IEEE Computer Society.

[KS09] Ali Khoshgozaran and Cyrus Shahabi. *Privacy in Location-Based Applications: Research Issues and Emerging Trends*, chapter Private Information Retrieval Techniques for Enabling Location Privacy in Location-Based Services, pages 59–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[LG15] Wouter Lueks and Ian Goldberg. *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, chapter Sublinear Scaling for Multi-Client Private Information Retrieval, pages 168–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proceedings of the 8th International Conference on Information Security*, ISC'05, pages 314–328, Berlin, Heidelberg, 2005. Springer-Verlag.

[LLL] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534.

[MBFK16] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR : Private information retrieval for everyone. *PoPETs*, 2016(2):155–174, 2016.

[MCG+08] Carlos Aguilar Melchor, Benoit Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre Rousseau. High-speed private information retrieval computation on gpu. In *Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, SECURWARE '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[MG07] Carlos AGUILAR MELCHOR and Philippe GABORIT. A lattice-based computationally-efficient private information retrieval protocol. Cryptology ePrint Archive, Report 2007/446, 2007. `http://eprint.iacr.org/2007/446`.

[MOT+11] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. Pir-tor: Scalable anonymous communication using private information retrieval. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 31–31, Berkeley, CA, USA, 2011. USENIX Association.

[NS09] Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Trans. Algorithms*, 5(4):46:1–46:48, November 2009.

[OG10] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 75–92, Berlin, Heidelberg, 2010. Springer-Verlag.

[OG12] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Proceedings of* *the 15th International Conference on Financial Cryptography and Data Security*, FC'11, pages 158–172, Berlin, Heidelberg, 2012. Springer-Verlag.

[Olu01] Femi Olumofin. *Practical Private Information Retrieval.* PhD thesis, University of Waterloo, 2001.

[oST01] Information Technology Laboratory (National Institute of Standards and Technology). *Announcing the Advanced Encryption Standard (AES).* Federal information processing standards publication. 2001.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[Reg04] Oded Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, November 2004.

[Sch03] Claus Peter Schnorr. *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science Berlin, Germany, February 27 – March 1, 2003 Proceedings*, chapter Lattice Reduction by Random Sampling and Birthday Methods, pages 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Sem01] Igor Semaev. *Cryptography and Lattices: International Conference, CaLC 2001 Providence, RI, USA, March 29–30, 2001 Revised Papers*, chapter A 3-Dimensional Lattice Reduction Algorithm, pages 181–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[SGR97] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 44–, Washington, DC, USA, 1997. IEEE Computer Society.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[Sio07] Radu Sion. On the computational practicality of private information retrieval. In *In Proceedings of the Network and Distributed Systems Security Symposium, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report*, 2007.

[SJ00] Claus-Peter Schnorr and Markus Jakobsson. Security of signed elgamal encryption. In *Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '00, pages 73–89, London, UK, UK, 2000. Springer-Verlag.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 24–43, 2010.

[WDDB06] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 49–64, Berlin, Heidelberg, 2006. Springer-Verlag.

[Win71] S. Winograd. On multiplication of 2 × 2 matrices. *Linear Algebra and its Applications*, 4(4):381 – 388, 1971.

[YDDB08] Yanjiang Yang, Xuhua Ding, Robert H. Deng, and Feng Bao. An efficient pir construction using trusted hardware. In *Proceedings of the 11th International Conference on Information Security*, ISC '08, pages 64–79, Berlin, Heidelberg, 2008. Springer-Verlag.

[YKPB13] Xun Yi, Md. Golam Kaosar, Russell Paulet, and Elisa Bertino. Single-database private information retrieval from fully homomorphic encryption. *IEEE Trans. on Knowl. and Data Eng.*, 25(5):1125–1134, May 2013.