The Best of Both Worlds: Combining Information-Theoretic and Computational Private Information Retrieval for Communication Efficiency

by

Casey Devet

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

Author's Declaration

Abstract

The goal of *Private Information Retrieval* (PIR) is the ability to query a database successfully without the operator of the database server discovering which record(s) of the database the querier is interested in. There are two main classes of PIR protocols: those that provide privacy guarantees based on the computational limitations of servers, called computational PIR or CPIR, and those that rely on multiple servers not colluding for privacy, called information-theoretic PIR or IT-PIR. These two classes have different advantages and disadvantages that make them more or less attractive to designers of PIR-enabled privacy enhancing technologies.

We present a hybrid PIR protocol that combines two PIR protocols: one CPIR protocol and one IT-PIR protocol. Our protocol inherits many positive aspects of both classes and mitigates some of the negative aspects. For example, our hybrid protocol maintains partial privacy when the security assumptions of one of the component protocols is broken, mitigating the privacy loss in such an event. We have implemented our protocol as an extension of the Percy++ library so that it combines a PIR protocol by Aguilar Melchor and Gaborit with one by Goldberg. We show that our hybrid protocol uses less communication than either of these component protocols and that our scheme is particularly beneficial when the number of records in a database is large compared to the size of the records. This situation arises in applications such as TLS certificate verification, anonymous communications systems, private LDAP lookups, and others.

The server-side computation involved in the PIR protocols that we discuss in this thesis all lend themselves to parallelization. As an extension to the Percy++ library we have implemented parallelized server computation for each of these protocols using both multithreading and distributed computation. We show that using parallelization allows the servers to reduce the latency involved in serving PIR queries.

Acknowledgements

Thank you to all of the people who were a constant support while writing this thesis. I would especially like to thank Ian Goldberg for sticking with me and for his constant support and guidance. My wife Laura also deserves thanks for pushing me and always being there to help out.

A huge thank you to Ryan Henry for the idea that inspired me to explore this new type of Hybrid PIR.

This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

Table of Contents

List of Figures **vii**

List of Tables **viii**

List of Algorithms **ix**

Nomenclature **x**

1 Introduction **1**

1.1 Private Information Retrieval 1

1.2 Notation 3

2 Background **5**

2.1 Computational PIR 5

2.1.1 Aguilar Melchor and Gaborit's Protocol 6

2.2 Information-Theoretic PIR 10

2.2.1 Chor et al.'s Protocol 10

2.2.2 Goldberg's Protocol 12

3 Hybrid PIR **17**

3.1 Our Hybrid PIR Protocol 17

3.1.1 Notation 17

3.1.2 Protocol 18
3.2 Analytical Evaluation 20
3.2.1 Inner Protocols 20
3.2.2 Communication 21
3.2.3 Computation 24
3.2.4 Privacy 26
3.2.5 Robustness 26
3.3 Implementation and Empirical Evaluation 27
3.3.1 Communication 27
3.3.2 Computation 28
3.3.3 Total Query Time 30
3.4 Summary 32

4 Parallelized Server Computation **34**
4.1 Parallelizing PIR 34
4.1.1 More Than One Way to Split Computation 34
4.1.2 More Than One Way to Split a Request 35
4.2 Parallelizing Recursive PIR 39
4.3 Empirical Evaluation 41
4.3.1 Multithreading 41
4.3.2 Distributed Computation 43
4.3.3 A Combination 46
4.4 Summary 46

5 Conclusion **47**
5.1 Future Work 48

References **49**

List of Figures

3.1 Comparison of Communication Costs 28
3.2 Comparison of Request Encoding Times 29
3.3 Comparison of Server Computation Times 29
3.4 Comparison of Response Decoding Times 30
3.5 Comparison of Total Query Times over 9/2 Mbsp Connection 31
3.6 Comparison of Total Query Times over 20/5 Mbsp Connection 31
3.7 Comparison of Total Query Times over 100/100 Mbsp Connection 32

4.1 Server computation for G07, C95 36
4.2 G07 and C95 when partitioning queries 36
4.3 G07 and C95 when partitioning database records 37
4.4 G07 and C95 when partitioning the words of records 38
4.5 Server-side computation for AG07. 39
4.6 AG07 when partitioning queries 39
4.7 AG07 when partitioning database records 40
4.8 AG07 when partitioning words of records 40
4.9 Multithreading on G07 42
4.10 Multithreading on G07 and Hybrid 42
4.11 Distributed Computation on G07 44
4.12 Distributed Computation on G07 and Hybrid 44
4.13 Combination Parallelization on G07 (per thread) 45
4.14 Combination Parallelization on G07 (per thread) 45

List of Tables

3.1 Comparison of Communication Costs 23
3.2 Useful range of hybrid protocol 24

List of Algorithms

2.1 AG07 Query Generation . . . 7
2.2 AG07 Server Computation . . . 8
2.3 AG07 Response Decoding . . . 9
2.4 C95 Query Generation . . . 11
2.5 C95 Server Computation . . . 11
2.6 C95 Response Decoding . . . 12
2.7 G07 Query Generation . . . 14
2.8 G07 Server Computation . . . 14
2.9 G07 Response Decoding . . . 15

3.1 Hybrid Query Generation . . . 18
3.2 Hybrid Server Computation . . . 19
3.3 Hybrid Response Decoding . . . 20

Nomenclature

d	Depth used for a recursive PIR protocol (p. 6)
D	The database (p. 3)
D_i	The i^{th} record of the database D (p. 3)
δ_u	In the hybrid protocol, the number of actual records contained in each virtual record at iteration u (p. 18)
$\mathbb{F}$	Field used for arithmetic; exact definition depends on protocol begin used (p. 13)
γ_u	In the hybrid protocol, the database is split into γ_u virtual records at iteration u (p. 18)
i_0	Index of the database record that a client wants to retrieve (p. 4)
i_u	For the hybrid protocol, the index of record i_0 in the result of iteration u (p. 18)
k	Number of PIR servers that respond in a multiserver protocol (p. 13)
ℓ	Number of PIR servers in a multiserver protocol (p. 11)
L	In AG07, each record is a matrix of size $L \times N$; L depends on N (p. 6)
m	In G07, Number of w_G-bit words per record (p. 13)
N	AG07 security parameter: Each record is a matrix of size $L \times N$; typically $N = 50$ (p. 6)
n	Number of records in the database (p. 4)
p	Modulus of the AG07 field: $p \approx 2^{3w_{AG}}$ (p. 6)

Φ	The single-server CPIR protocol used in the hybrid protocol (p. 17)
π_u	In the hybrid protocol, the index of the virtual record containing the record the client wants (record i_0) at iteration u (p. 18)
Ψ	The multiserver IT-PIR protocol used in the hybrid protocol (p. 17)
q	AG07 hard noise constant: $q \approx 2^{2w_{AG}}$ (p. 6)
s	Size of each database record in bits (p. 4)
t	G07 privacy level: queries are private if at most t servers collude (p. 13)
v	Number of Byzantine PIR servers that may respond incorrectly in a multiserver protocol (p. 13)
w_{AG}	AG07 word size in bits; typically $w_{AG} = 20$ (p. 6)
w_G	G07 word size in bits; typically $w_G = 8$ (p. 13)

Chapter 1

Introduction

One major goal of *privacy enhancing technologies* (PETs) is to give control over the dissemination of personal information to the users that the information pertains to. PETs rely on underlying primitives to provide a guarantee of privacy to users; these are often primitives from fields such as cryptography and information theory, but may also use secure hardware or a trusted third party. Many PETs protocols use cryptographic primitives, relying on assumptions about the infeasibility of solving a specific problem with a limited amount of computing resources. Other PETs are built on information-theoretic primitives which, under non-computational assumptions such as limited collustion, have the advantage of providing a guarantee that no amount of computing resources will allow an adversary to discover the user's private information. However, using information-theoretic primitives instead of cryptographic ones requires some alternative assumption to support the protocol's privacy guarantees. An assumption used in many PETs, including mix networks [DDM03], secret sharing [Sha79], onion routing [DMS04] and some voting protocols [CCC+09, RS06], is that no more than some threshold of agents are colluding against the user to discover the private information.

1.1 Private Information Retrieval

Private Information Retrieval (PIR) [CGKS95, CKGS98] is a PET that allows a user to query a database for some records without letting the operator of the database server learn anything about the query or the retrieved records. The most trivial form of PIR is for the client to download the entire database from the server and do the query herself. This is private because the user has not revealed any information about which record she is interested in, yet she still retrieves the record by finding it in the content of the entire database. In a 2007 study, Sion and Carbunar concluded that no single-server PIR protocol would likely outperform this trivial download PIR protocol [SC07]. However, more recent work has shown that there are indeed non-trivial PIR protocols that perform better than downloading the entire database [OG11]. PIR has applications in many privacy-sensitive applications, including patent databases [Aso01], domain name registration [OG10], anonymous email [SC05], anonymous communication networks [MOT$^+$11], and electronic commerce [HOG11].

As a PET, a PIR protocol gets its privacy guarantees from its underlying primitives. One class of PIR protocols, called *computational PIR* (CPIR), encodes the query in such a way that the database server can serve records, while learning nothing about the queries or retrieved records. The privacy guarantees of CPIR protocols are based on the assumption that some problem is hard or impossible to solve given a limit on computational power. Olumofin and Goldberg [OG11] showed in 2011 that it is possible for a CPIR protocol to outperform the trivial download protocol. In particular, they showed using empirical results that the CPIR protocol by Aguilar Melchor and Gaborit [AG07] is faster than trivial download when using typical network connections. One advantage of many CPIR protocols is the ability to use *recursion* to reduce the communication costs, a technique that is illustrated by Aguilar Melchor and Gaborit with their CPIR protocol [AG07].

The other class of PIR protocols, called *information-theoretic PIR* (IT-PIR), does not rely on the assumption that a cryptographic primitive is hard to solve with limited computing resources. In 1995, Chor et al. showed that non-trivial IT-PIR is impossible when there is only a single database server [CGKS95]. To combat this result, they designed a multi-server IT-PIR protocol that guarantees privacy as long as not all of the servers are colluding together against the user. Several IT-PIR protocols have since been proposed [BIM04, GGM98, Gol07, HOG11] that use similar non-collusion assumptions. Olumofin and Goldberg [OG11] also showed that a number of these multi-server IT-PIR protocols perform better than the trivial download PIR.

There are five contributing factors to the speed of a PIR query for a particular protocol:

1. the time for the client to generate a private query;

2. the communication time required to send the query to the server(s);

3. the time for the server(s) to apply the query to the database;

4. the communication time required for the response from the server to the client; and

5. the time for the client to decode the response(s).

Over time, proposed PIR protocols have incrementally improved some or all of these time factors. This thesis begins by comparing three PIR protocols, one CPIR and two IT-PIR. We observe that the CPIR protocol we discuss uses recursion to improve its communication efficiency and that the IT-PIR protocols we discuss are very communication and computation efficient.

Our main contribution is a novel *hybrid* PIR protocol that incorporates aspects of both classes, including the recursive property of single-server CPIR and the low communication and computation costs of IT-PIR. Our hybrid protocol has lower costs, while incorporating the positive properties of the CPIR and IT-PIR protocols of which it is composed.

Our protocol is particularly well suited for databases that consist of a large number of relatively small records. As a practical example of where PIR over databases of this shape would be beneficial, consider the problem of determining the validity of web server certificates for Transport Layer Security (TLS), the ubiquitous mechanism used to protect secure websites. A web client, on receiving a TLS certificate from a server, must check to see whether the certificate is revoked, typically with the Online Certificate Status Protocol (OCSP) [SMA+13], or with the recently proposed Certificate Transparency (CT) Protocol [LLK13]. However, doing these lookups will reveal the site the client is visiting to the OCSP or CT servers. PIR has been proposed [Kik04] as a way for clients to privately determine the validity of these certificates. Other applications of PIR over databases of this shape could include sensor network data retrieval [Xiv14], private LDAP lookups [Ser06], and efficient retrieval of network information in anonymous communications systems [MOT+11].

In an effort to improve the server-side computation time of PIR, we implement parallel computation for all of the protocols discussed in this thesis. The growing use of cloud computing and multiprocessing computers provides the opportunity to perform computationally intensive procedures quicker and easier. We show that by making use of these types of resources, large-scale PIR is not just possible, but practical.

1.2 Notation

For clarity, we will use the following notation throughout the thesis:

- D denotes the database.

- D_i denotes the i^{th} record of the database D.
- n is the number of records in the database.
- s is the size of each record in bits.
- i_0 is the index of the database record that a client wants to retrieve.

Additional notation will be introduced in Sections 2.1.1, 2.2.2 and 3.1.1 to support the protocols presented in those sections.

Chapter 2

Background

2.1 Computational PIR

One class of PIR contains all protocols that assume that the server(s) are computationally bounded to make their privacy guarantees. These protocols are built so that breaking the security of their system would require an adversary to solve a problem that is believed to be hard. These types of assumptions are often used in cryptography, security and privacy; for example, the RSA public-key cryptosystem assumes that factoring large numbers is hard when an adversary has limited resources.

Computational PIR was first introduced by Chor et al. in 1997 [CGN97]. They showed that weakening the adversary to a computationally bounded entity improves the communication costs of PIR. Their work was soon followed by a protocol by Kushilevitz and Ostrovsky [KO97] that used the same computationally bounded adversary in their model, but did not require multiple servers as previous CPIR protocols did. This protocol relied on the assumption that the Quadratic Residuosity problem [McC90] is difficult to solve.

One advantage of single-server CPIR protocols is that they can be used recursively to improve the communication cost of PIR. This idea was introduced by Kushilevitz and Ostrovsky in addition to their new single-server CPIR protocol [KO97]. To apply recursion, we evenly split our database into a set of virtual records, each one containing an equal number of the actual records. The client then queries the server for a particular virtual record, but instead of returning the result to the client, the server holds on to it. The result of the first query is treated as a virtual database containing smaller virtual records. The client then queries for one of the virtual records of this virtual database. The scheme continues in this fashion until we are left with the response for a single (actual) record, which is sent to the client. This idea will be further explored in the next section.

2.1.1 Aguilar Melchor and Gaborit's Protocol

Without being faster than the trivial download protocol for modest-sized databases, a PIR protocol is not very useful. The main problem with the CPIR protocols already discussed is that they do not generally perform queries faster than the trivial protocol. In 2007, Aguilar Melchor and Gaborit introduced a lattice-based single-server CPIR scheme with promising results [AG07]; we denote this protocol as AG07. In 2011, Olumofin and Goldberg [OG11] empirically showed that this protocol outperforms the trivial protocol, thus suggesting that CPIR may indeed be practical.

The idea behind their protocol is to add noise to the query in a way that the server cannot discover which record the client is interested in, but with the secret information that the client has, she can remove the noise from the server's response.

Notation

For this protocol we add the following notation:

- Each record in the database is encoded as an $L \times N$ matrix of w_{AG}-bit words, where N is a security parameter and $L = \left\lceil \frac{s}{w_{AG} \cdot N} \right\rceil$.
- $q \approx 2^{2 \cdot w_{AG}}$ is the *hard noise constant.*
- $p \approx 2^{3 \cdot w_{AG}}$ is the prime modulus of the field used for arithmetic. All matrices in the protocol are over $\mathbb{Z}_p$; the entries in the above database record matrices just happen to have relatively small values ($< 2^{w_{AG}}$) in $\mathbb{Z}_p$.
- d is the number times that the protocol is used recursively.

Aguilar Melchor and Gaborit [AG07] suggest the values $w_{AG} = 20$, $N = 50$, $q = 2^{40}$, and $p = 2^{60} + 325$ for the above parameters.

Algorithm 2.1 AG07 Query Generation

```
Input: Desired record index: $i_0$
1: Generate two uniform random $N \times N$ matrices $A$ and $B$ such that $A$ is invertible.
2: For each $i \in \{1, \ldots, n\}$ generate uniform random invertible $N \times N$ matrix $P_i$.
3: Generate the uniform random scrambling matrix $\Delta$, a diagonal invertible $N \times N$ matrix.
4: For each $i \in \{1, \ldots, n\}$ generate soft noise matrix $C_i$, an $N \times N$ matrix over $\{-1, 1\}$.
5: Convert $C_{i_0}$ to a hard noise matrix by setting all diagonal terms to $q$.
6: For each $i \in \{1, \ldots, n\}$ compute the disturbed $N \times 2N$ matrices
       $$M'_i = [P_i A | P_i B + C_i \Delta].$$
7: Choose uniform random column permutation $\mathcal{P}$ and for each $i \in \{1, \ldots, n\}$ compute
   $M_i = \mathcal{P}(M'_i)$.
8: Send $M_1, \ldots, M_n$ to the server.
```

Protocol

A client wants to retrieve record i_0 from the database. For each database record, she generates two matrices, one that has been made noisy and one that has not. For the query matrices corresponding to record i_0 she adds hard noise (relatively large disturbances) and for the others she adds soft noise (small disturbances). The privacy of this protocol relies on the assumption that the server cannot distinguish between query matrices with hard noise and soft noise. For details, see Algorithm 2.1.

When the client sends the query to the server, the amount of communication (in bits) is $6N^2 w_{AG} \cdot n$.

To process the query, each record in the database is represented as an $L \times N$ matrix whose terms are words of size w_{AG} bits. When the server receives the query, it multiplies each database record D_i by the corresponding query matrix M_i and adds the results to get R. For details, see Algorithm 2.2.

The server sends the response R back to the client. The amount of communication (in bits) for this step is six times the size of each record, or $6s$.

Finally, when the client receives the response, she removes the soft noise to reveal the database record D_{i_0} that she requested (see Algorithm 2.3).

The privacy of this protocol relies on the assumption that the *Hidden Lattice Problem*

Algorithm 2.2 AG07 Server Computation

```
Input: Query from the client: matrices $M_1, \ldots, M_n$
    Database records represented as $L \times N$ matrices with $w_{AG}$-bit elements: $D_1, \ldots, D_n$
1: Compute response matrix
       $$R = \sum_{i=1}^{n} D_i M_i \pmod{p}.$$
2: Send $R$ to the client.
```

and the *Differential Hidden Lattice Problem* are hard to solve by computationally bounded adversaries [AG07]. Aguilar Melchor and Gaborit use related problems in coding theory to justify these assumptions.

Recursive AG07

As stated above, this CPIR protocol can be performed recursively to improve the communication cost of the scheme. We get optimal communication for a given recursive depth d if we split our database into $\sqrt[d]{n}$ virtual records at each iteration.

For example, if we have a database with 125 records and we are performing this recursive protocol with depth 3, then in the first iteration we separate the database into $\sqrt[3]{125} = 5$ virtual records, each one a matrix of size $25L \times N$ with w_{AG}-bit entries, thus containing 25 actual records. This client will query the server for the virtual record that her wanted record belongs to, but instead of sending the result R_1, a $25L \times 2N$ matrix with $3w_{AG}$-bit words, back to the client, the server will hold onto it. In the second iteration, the server transforms the result R_1 from the first iteration into 5 virtual records, each one a $6 \cdot 5L \times N$ matrix with w_{AG}-bit entries encoding 5 actual records. The client will query the server for the virtual record that contains her desired record and again the server holds onto the result R_2. Finally, for the last iteration, the server transforms the result R_2 from the second iteration into 5 virtual records, each one a $6^2 \cdot L \times N$ matrix with w_{AG}-bit entries encoding one actual record. The client queries the server for the record that she is interested in and the server sends the result R_3 of this last iteration to the client. The client must then perform the decoding algorithm (Algorithm 2.3) 3 times, once for each iteration, to recover the database record.

By using recursion we improve the client-to-server communication cost of AG07 to $d \cdot 6N^2 w_{AG} \cdot \sqrt[d]{n}$ bits. However, each iteration of the protocol increases the size of the

Algorithm 2.3 AG07 Response Decoding

```
Input: Response from the server: $R$
    Random values generated in Algorithm 2.1: $A, B, \Delta, \mathcal{P}$
1: Undo the column permutation
       $$R' = \mathcal{P}^{-1}(R) = [V_U|V_D]$$
   where $V_U$ and $V_D$ are $L \times N$ matrices
2: Compute scrambled noise matrix
       $$E' = V_D - V_U A^{-1} B.$$
3: Compute the unscrambled noise matrix
       $$E = E'\Delta^{-1}.$$
4: For each term $e_{x,y}$ of $E$ remove noise by computing
       $$f_{x,y} = e_{x,y} - \epsilon_{x,y}$$
   with
       $$\epsilon_{x,y} = \begin{cases} e_{x,y} \mod q & : e_{x,y} \mod q < \frac{q}{2} \\ (e_{x,y} \mod q) - q & : \text{otherwise} \end{cases}$$
5: Recover database record $D_{i_0}$ by computing words
       $$d_{x,y} = f_{x,y} \cdot q^{-1}.$$
```

result by a factor of 6. This makes the server-to-client communication cost $6^d s$ bits. Thus, it is important to find the appropriate recursive depth to balance out this decrease in client-to-server communication and the increase in server-to-client communication.

Advantages and Disadvantages

One advantage of this protocol is that it only requires a single server. As shown later in Section 2.2, multi-server protocols generally assume that some threshold of the servers are not colluding. CPIR protocols, however, remain secure even if all servers (or the one server in the single-server case) are trying to discover the client's private query.

The AG07 protocol also has the advantage that it can be used recursively, with a relatively low compounding overhead factor (6). As shown above, we can use this property to significantly improve the communication cost incurred by the protocol.

The main disadvantage of this scheme is that the security is based on lattice problems that are not well understood. Because of this, some clients may not completely trust the privacy of their queries. As stated by Aguilar Melchor et al. in a subsequent paper [ACG$^+$08] and by Olumofin and Goldberg [OG11], the protocol resists known lattice-based attacks, but the protocol and its privacy assumptions are new and may not be secure.

Another disadvantage of the AG07 protocol is that it is considerably slower than many IT-PIR schemes [OG11]. This is due to the amount of computation involved in encoding the queries and because the server is performing a matrix-by-matrix multiplication (as compared to a vector-by-matrix multiplication used by some IT-PIR schemes).

2.2 Information-Theoretic PIR

The other class of PIR protocols, information-theoretic PIR (IT-PIR), includes all PIR schemes whose privacy guarantees hold no matter how computationally powerful and adversarial the server(s) may be. In 1995, Chor et al. [CGKS95] showed that any single-server IT-PIR scheme must have communication cost at least that of the trivial protocol. To avoid this problem, they developed IT-PIR protocols that used multiple servers. Since then, a variety of multiple-server IT-PIR schemes have been formulated [BIM04, GGM98, Gol07, HOG11], making improvements on Chor et al.'s protocols. One of these improvements is *robustness*—the ability to retrieve the correct database records even when some of the servers are down or return incorrect or malicious responses.

An advantage to multiple-server IT-PIR is that it generally incurs smaller communication and computation costs. Like CPIR protocols, multiple-server IT-PIR protocols also need to make some assumptions to guarantee privacy; a commonly used assumption is that at most some threshold of the servers are colluding to discover the contents of a client's query.

2.2.1 Chor et al.'s Protocol

A basic multiple-server IT-PIR protocol developed by Chor et al. requires that there are $\ell \geq 2$ database servers, each with a copy of the same database. This protocol requires that all servers respond and do so correctly for the client to be able to accurately recover the desired database record. We will use C95 to denote this protocol.

Algorithm 2.4 C95 Query Generation

```
Input: Desired record index: $i_0$
1: Create $n$-length elementary vector
       $\mathbf{e_{i_0}} = \langle 0, 0, \ldots, 1, \ldots, 0 \rangle$
   where all terms of $\mathbf{e_{i_0}}$ are zero except the term at index $i_0$ is a one.
2: For $j \in \{1, \ldots, \ell - 1\}$ generate a uniform random $n$-length binary vector $\mathbf{v_j} \in \{0,1\}^n$.
3: Compute
       $\mathbf{v}_\ell = \mathbf{v_1} \oplus \mathbf{v_2} \oplus \cdots \oplus \mathbf{v_{\ell-1}} \oplus \mathbf{e_{i_0}}$.
4: For each $j \in \{1, \ldots, \ell\}$, send $\mathbf{v_j}$ to server $j$.
```

Algorithm 2.5 C95 Server Computation

```
Input: Query from client: $\mathbf{v_j}$
1: Compute response vector
       $\mathbf{r_j} = \mathbf{v_j} \cdot D \pmod{2}$.
2: Send $\mathbf{r_j}$ to the client.
```

Notation

For this protocol we add the following notation:

- ℓ is the number of servers.

Protocol

To query the database for record i_0, a client creates an elementary vector $\mathbf{e_{i_0}}$ for index i_0 (See Algorithm 2.4). She then creates a uniform random query vector $\mathbf{v_j}$ for each server j such that when all such vectors are XOR'd, we get the elementary vector $\mathbf{e_{i_0}}$.

The communication cost from the client to each server is n bits.

For this protocol, the servers treat the database as a binary matrix D whose rows D_i are the records. Each server simply multiples their query vector by the database to get a response $\mathbf{r_j}$, which is sent to the client (see Algorithm 2.5).

Algorithm 2.6 C95 Response Decoding

```
Input: Responses from the servers: $\mathbf{r_1}, \ldots, \mathbf{r_\ell}$
1: Recover database record $D_{i_0}$ by computing
       $D_{i_0} = \mathbf{r_1} \oplus \cdots \oplus \mathbf{r_\ell}.$
```

The communication cost from each server to the client is s bits.

Finally, after the client receives all of the responses, she simply XORs them to get the database record D_{i_0}. This is correct because of the distributive property of the XOR operation.

This protocol is private as long as not all of the servers are colluding. This is because without all query vectors $\mathbf{v_j}$ we are not able to find out any information about $\mathbf{e_{i_0}}$. Therefore, for this protocol we assume that at most $t = \ell - 1$ servers are allowed to collude.

Advantages and Disadvantages

The main advantages of this protocol are that it is very fast, has very low communication cost and is easy to implement. It is fast because the server-side computation can be implemented as the XOR of a subset of the database records, which is an inexpensive operation.

This protocol can also handle a very high level of collusion between servers. As long as not all servers are colluding together the privacy of the client's query will hold. This privacy level is high compared to other multiple-server IT-PIR protocols [Gol07].

The downfall of this protocol is that it is not robust: if any of the servers do not respond to the query, the client would not have enough information to recover the database record. Similarly, if any of the servers respond to the query incorrectly (either maliciously or erroneously), the client will incorrectly decode the database record and she will have no clear way of knowing that what she decoded is incorrect or which server was responsible for the failure. This deficiency has been addressed in more recent protocols, including one by Goldberg [Gol07].

2.2.2 Goldberg's Protocol

In 2007, Goldberg introduced a multiple-server IT-PIR protocol that was both efficient and provided for greater robustness than previous schemes. The idea is to use Shamir secret sharing [Sha79] to split the client's query across multiple servers, and error-correcting codes to combine the responses. We denote this protocol by G07.

Notation

For this protocol we add the following notation:

- The database is laid out as an $n \times m$ matrix of w_G-bit words. Each record is one row of this matrix, and $m = \left\lceil \frac{s}{w_G} \right\rceil$.
- ℓ is the number of servers.
- k is the number of servers that respond to the query.
- t is the privacy level—no coalition of t or fewer servers can learn the query.
- v is the number of Byzantine servers—these are servers that may give incorrect responses.
- $\mathbb{F}$ is the field used for arithmetic ($|\mathbb{F}| \geq 2^{w_G}$). All vectors and matrices in the protocol are over $\mathbb{F}$.

Typically, $w_G = 8$, $\mathbb{F} = GF(2^8)$, and records are an integer number of bytes, so that s is a multiple of 8, and $m \cdot w_G = s$ exactly.

Protocol

To query the server for record i_0, a client creates the elementary vector $\mathbf{e_{i_0}}$ with a 1 in the ${i_0}^{\text{th}}$ place, and 0 everywhere else. She then creates ℓ Shamir secret shares $\mathbf{v_1}, \ldots, \mathbf{v_\ell}$ for $\mathbf{e_{i_0}}$ in the field $\mathbb{F}$. Algorithm 2.7 details how these shares are created.

Each server is then sent one of these shares. The communication cost from the client to each server is then $n \cdot w_G$ bits.

The server simply multiplies their query vector $\mathbf{v_j}$ by the database D to get a response vector $\mathbf{r_j}$, and sends it back to the client. This makes the communication cost from each server to the client $mw_G = s$ bits.

In this protocol, we assume that some number of servers $k \leq \ell$ respond to the query. Even if $k \neq \ell$, meaning that not all servers responded, the client may still be able to recover

Algorithm 2.7 G07 Query Generation

```
Input: Desired record index: $i_0$
1: Choose $\ell$ distinct non-zero elements
       $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}$.
2: Generate $n$ uniform random degree-$t$ polynomials
       $f_1, \ldots, f_n \in \mathbb{F}[x]$
   such that $f_{i_0}(0) = 1$ and $f_i(0) = 0$ for all $i \neq i_0$.
3: For each $j \in \{1, \ldots, \ell\}$ compute
       $\mathbf{v_j} = \langle f_1(\alpha_j), \ldots, f_n(\alpha_j) \rangle$.
4: For each $j \in \{1, \ldots, \ell\}$, send $\mathbf{v_j}$ to server $j$.
```

Algorithm 2.8 G07 Server Computation

```
Input: Query from client: vector $\mathbf{v_j}$
1: Compute (over $\mathbb{F}$) the response vector
       $\mathbf{r_j} = \mathbf{v_j} \cdot D$.
2: Send vector $\mathbf{r_j}$ to the client.
```

the database record. This is because the use of Shamir secret sharing in the query makes the server responses Shamir secret shares for the database record D_{i_0}. This implies that the client only needs $k > t$ of the responses (where $t < \ell$) to successfully recover the record.

Similarly, we also do not need to assume that all of the servers are behaving correctly. The client can treat the responses as Reed-Solomon error correction codewords and use a Reed-Solomon decoding algorithm to recover the database record D_{i_0}. As shown by Devet et al. [DGH12], the client can decode the database record in polynomial time as long as the number of Byzantine servers v is bounded by $v < k - t - 1$; this result holds in the PIR setting because queries can be randomized, and clients can make multiple queries. They also show that this bound is the optimal bound on the number of tolerable Byzantine servers.

The Shamir secret shares are generated from a degree-t polynomial where $t < k$. By the properties of Shamir secret sharing, any coalition of at most t servers will not gain any

Algorithm 2.9 G07 Response Decoding

```
Input: Responses from the servers: $\mathbf{r_1}, \ldots, \mathbf{r_k}$
1: For $h \in \{1, \ldots, m\}$, form the codeword
       $$\mathbf{c_h} = \langle \mathbf{r_1}[h], \ldots, \mathbf{r_k}[h] \rangle .$$
2: Perform Reed-Solomon decoding on each codeword $\mathbf{c_1}, \ldots, \mathbf{c_m}$ to yield polynomials $F_1, \ldots, F_m$.
3: Output the database record $D_{i_0} = \langle F_1(0), \ldots, F_m(0) \rangle$.
```

information about the secret $\mathbf{e_{i_0}}$. However, if at least $t+1$ of the servers collude, they will be able to discover $\mathbf{e_{i_0}}$; that is, the query is information-theoretically private assuming that at most t servers are allowed to collude. We note that there is a trade off between the level of robustness and the privacy level—the client can chose a value of t to provide the wanted privacy up to and including $t = \ell - 1$ (all but one of the servers colluding), but then there is no robustness.

Advantages and Disadvantages

As discussed above, the main advantage of the G07 protocol over other protocols is that is it robust and can handle missing and/or incorrect server responses. This allows us to combat some stronger adversarial servers that maliciously alter their responses in an attempt to block the client from recovering the database record. We note that the AG07 single-server CPIR scheme has no robustness since there is only one server and missing or incorrect responses from that server can not be overcome.

The G07 protocol has low communication cost and computation time. It is also very simple to implement on the server side. A series of works since 2011 have shown that Goldberg's protocol is faster than the trivial protocol [OG11] and have added improvements to the performance by using distribution of computation [Dev13] and advanced error-correction algorithms [DGH12].

This protocol sacrifices some level of privacy to gain robustness. Because of this we need to assume that there is no collusion between some number of servers. In some settings, it is unclear how this requirement can be enforced or detected. This uncertainty may make this protocol less desirable than others with different privacy guarantees.

Adaptation for Hybrid Security

When he introduced his IT-PIR scheme in 2007 [Gol07], Goldberg proposed an extension to create a scheme whose privacy relied on a hybrid of information-theoretic and computational primitives. This extended scheme provides information-theoretic protection of the query as long as no more than t servers collude, but retains computational protection when any number of the servers collude.

This is accomplished by encrypting the query with an additive homomorphic cryptosystem—G07 used the Pailler cryptosystem. The client will encrypt the query before it is sent to the servers. When the servers receive the query, they multiply it by the database, but use the homomorphic property. In the case of the Pailler cryptosystem, the server would use multiplication in the place of addition and exponentiation in the place of scalar multiplication. The response that the client receives is decrypted before the regular G07 decoding operations are performed.

Though this hybrid scheme relies on two assumptions for privacy (the information-theoretic assumption that no more than t servers collude and the assumption that adversaries do not have the computational resources to break the additive homomorphic cryptosystem used), as long as one of them holds, the protocol still guarantees perfect privacy of the query.

This added protection comes at an extreme cost, however: the hybrid version of G07 is *3–4 orders of magnitude slower* [Gol07] than the pure information-theoretic version. In the next section, we will introduce a new approach to hybrid PIR that combines the benefits of CPIR and IT-PIR without the overhead of previous proposals.

Chapter 3

Hybrid PIR

3.1 Our Hybrid PIR Protocol

In this chapter, we propose a hybrid protocol that combines a multiple-server IT-PIR protocol with a single-server CPIR protocol. Our goal is to incorporate the positive aspects of each protocol into our hybrid protocol, while mitigating the negative aspects of each. In particular, we want to join the low communication and computation cost of multiple-server IT-PIR schemes with the recursion of single-server CPIR schemes to improve the communication cost of PIR queries relative to both classes of protocols.

Our scheme will use a recursive depth of d as in the AG07 CPIR scheme. However, the first layer of recursion will be performed using the chosen multiple-server IT-PIR protocol. On each server, the remainder of the recursive steps will be done on the result of each previous step using the chosen recursive single-server CPIR scheme.

3.1.1 Notation

Our hybrid protocol will use the notation outlined in Sections 1.2, 2.1.1, and 2.2.2 as well as:

- Ψ is the multiple-server IT-PIR protocol being used.
- Φ is the single-server recursive CPIR protocol being used.

- d is the recursive depth. We will do one iteration of Ψ followed by $d-1$ iterations of Φ.
- γ_u $(1 \leq u \leq d)$ is the number of virtual records that the database is split into for the u^{th} step of recursion of the hybrid scheme. It is required that $n \leq \prod_{u=1}^{d} \gamma_u$.
- δ_u $(1 \leq u \leq d)$ is the number of actual records in each virtual record at the u^{th} step of recursion of the hybrid scheme. If the database does not evenly split, dummy records are appended to the end of the database to make each virtual record the same size.
- π_u $(1 \leq u \leq d)$ is the index of the virtual record that the client's desired actual record i_0 is in at the u^{th} step of recursion.
- i_u $(1 \leq u \leq d)$ is the index of record i_0 in the result of iteration u.

We show how to calculate π_u and i_u in Section 3.1.2 and we outline how to optimally choose the values for γ_u and δ_u in Section 3.2.2.

3.1.2 Protocol

Our protocol is generalized to use the implementer's choice of inner protocols. We use Ψ to denote the multiple-server IT-PIR inner protocol and use Φ to denote the single-server recursive CPIR inner protocol. We use this notation because our protocol is very well suited for a modular implementation. That is, an implementation of this scheme could easily swap inner protocols for other suitable protocols.

Algorithm 3.1 Hybrid Query Generation

```
Input: Desired record index: $i_0$
1: For each recursive step $u \in \{1, \ldots, d\}$ find the index of the virtual record $\pi_u$ that record
   $i_0$ belongs to.
2: Generate a multiple-server $\Psi$-query $Q_1$ for index $\pi_1$ and send each server its part of
   the query.
3: for $u = 2 \rightarrow d$ do
4:     Generate a single-server $\Phi$-query $Q_u$ for index $\pi_u$ and send each server a copy of the
       query.
```

Algorithm 3.2 Hybrid Server Computation

```
Input: Query from client: $Q_1, \dots, Q_d$
1: Split the database $D$ into $D^{(1)}$, a virtual database of $\gamma_1$ consecutive virtual records,
   each containing $\delta_1$ actual records.
2: Apply the $\Psi$-query $Q_1$ to database $D^{(1)}$ using the $\Psi$ server computation algorithm.
   The result is $R_1$ which will be used as the database for the next recursive step.
3: for $u = 2 \to d$ do
4:     Split the result $R_{u-1}$ into $D^{(u)}$, a virtual database of $\gamma_u$ consecutive virtual records,
       each containing the encoding of $\delta_u$ actual records
5:     Apply the $\Phi$-query $Q_u$ to database $D^{(u)}$ using the $\Phi$ server computation algorithm
       to get result $R_u$.
6: Send the final result $R_d$ to the client.
```

Algorithm 3.1 outlines how to generate a query for this protocol. To query the database servers, the client must determine the index π_u of the virtual record that her desired record i_0 is contained in, at each step u of the recursion. This is done for $u = \{1, \dots, d\}$ using the mutually recursive formulas

$$\begin{aligned} \pi_u &= \left\lfloor \tfrac{i_{u-1}}{\delta_u} \right\rfloor \\ i_u &= i_{u-1} \mod \delta_u \end{aligned}$$

where i_0 is the desired record and the δ_u are predetermined constants (see 3.1.1 for more details). She then creates a multiple-server IT-PIR Ψ-query for index π_1 and sends each server its part of the query. Then for each remaining recursive step $u \in \{2, \dots, d\}$, she creates single-server CPIR Φ-queries for index π_u and sends this same query to each of the servers.

Algorithm 3.2 outlines the server-side computations for this protocol. In each recursive step u, the server splits the database into γ_u virtual records, each containing δ_u actual records. For the first step, the server uses the IT-PIR Ψ server computation algorithm. For the remainder of the steps, the server uses the CPIR Φ server computation algorithm. The result of the last recursive step is sent back to the client.

We note that we can somewhat improve the performance of this scheme by starting the server-side computations for each recursive step before reading the queries for subsequent recursive steps, thus overlapping computation and communication.

Algorithm 3.3 Hybrid Response Decoding

```
Input: Responses from the servers: $X_1^{(d)}, \ldots, X_k^{(d)}$
1: for $u = d \to 2$ do
2:     for $j = 1 \to k$ do
3:         Decode $X_j^{(u)}$ from server $j$ using the $\Phi$ single-server decoding algorithm to get
           result $X_j^{(u-1)}$.
4: Decode $X_1^{(1)}, \ldots, X_k^{(1)}$ simultaneously using the $\Psi$ multiple-server decoding algorithm
   to recover the database record $D_{i_0}$.
```

When the client receives the servers' responses, she applies the corresponding decoding algorithms using the information stored during query generation in reverse order. That is, she first uses the information from query Q_d to decode the received responses. Treating the results of that decoding as virtual responses themselves, she uses information from Q_{d-1} to decode those, and so on until she uses information from Q_1 to decode the final step. This yields the desired record. The procedure for decoding server responses for this protocol is outlined in Algorithm 3.3. We note that for all but the last step of decoding, the result from each server must be decoded separately using the single-server decoding algorithm for protocol Φ. In the last step of decoding, all server results are decoded simultaneously using the multiple-server decoding algorithm for protocol Ψ.

3.2 Analytical Evaluation

3.2.1 Inner Protocols

Our hybrid scheme is very well suited to a modular implementation. That is, an implementation of this protocol could could easily swap the inner protocols with other suitable PIR schemes. Our protocol can be implemented simply as a wrapper to other non-recursive PIR implementations.

This evaluation of the hybrid scheme uses the G07 IT-PIR scheme for Ψ and the AG07 CPIR scheme for Φ.

Why G07?

We chose to evaluate our hybrid scheme using the G07 IT-PIR scheme as Ψ because it has very good communication and computation costs that are used to improve an iteration of the recursion. The Goldberg scheme is also robust and its robustness properties are completely inherited by our hybrid scheme.

One alternative to the G07 scheme is the C95 scheme presented in Section 2.2.1. The main difference between it and G07 is that C95 does not have the same robustness properties. However, it is faster and has less communication than the Goldberg scheme. Therefore using C95 as the first iteration of the hybrid protocol would be more beneficial in situations where servers are not expected to be down or where there is little threat of servers maliciously altering their responses.

Why AG07?

The choice of the AG07 CPIR protocol for Φ is because of previous work that has shown that it outperforms the trivial protocol [OG11], a rare quality for CPIR schemes. It should be noted that the security of AG07 is not well understood and further work needs to be done to prove that this protocol is indeed as private as advertised.

3.2.2 Communication

The communication cost of the response from the server to the client is simply

$$C_{down} = 6^{d-1}s.$$

If we combine the communication costs for the queries at each recursive step, we get the following total cost for the query (in bits) from the client to each server:

$$C_{up} = \gamma_1 w_G + \sum_{u=2}^{d} \left(6N^2 w_{AG} \cdot \gamma_u\right).$$

To optimize C_{up}, we first find the optimal choices for the γ_u values for any given d.

After the first recursive step the result will encode $\delta_1 = \lceil \frac{n}{\gamma_1} \rceil$ records. We can optimize the CPIR query sizes by splitting the database at each remaining step into $\gamma_u = \sqrt[(d-1)]{\delta_1}$

virtual records. The cost becomes:

$$\begin{aligned} C_{up} &= \gamma_1 w_G + \sum_{u=2}^{d} \left(6N^2 w_{AG} \cdot \left(\frac{n}{\gamma_1}\right)^{\frac{1}{d-1}} \right) \\ &= \gamma_1 w_G + (d-1) \cdot 6N^2 w_{AG} \cdot \left(\frac{n}{\gamma_1}\right)^{\frac{1}{d-1}} \end{aligned} .$$

We then find the value of γ_1 that minimizes C_{up} to be:

$$\gamma_1 = \left(\frac{6N^2 w_{AG}}{w_G} \sqrt[(d-1)]{n} \right)^{\frac{d-1}{d}} .$$

Therefore, at recursive step u, we split the database as follows:

$$\gamma_u = \begin{cases} \left(\frac{6N^2 w_{AG}}{w_G} \sqrt[(d-1)]{n} \right)^{\frac{d-1}{d}} & : u = 1 \\ \left(\frac{n}{\gamma_1}\right)^{\frac{1}{d-1}} & : 2 \le u \le d \end{cases}$$

$$\delta_u = \left(\frac{n}{\gamma_1}\right)^{\frac{d-u}{d-1}}$$

With these values, our query communication cost simplifies to:

$$C_{up} = d \left(6N^2 w_{AG}\right)^{\frac{d-1}{d}} \sqrt[d]{w_G} \sqrt[d]{n}.$$

We observe that both the query and response cost functions (C_{up} and C_{down}) are concave up in d. Therefore, the combined communication cost can be minimized for some depth d. Since d is an integer, we evaluate the cost functions at each d starting at 1 and incrementing until we find a value for d such that the cost at d is less than the cost at $d+1$. This value of d is our optimal depth.

Note that the combined cost function should ideally, if such information is available, take the bandwidths of both directions of our connection into account and that the different directions may have different bandwidths. This is accomplished with a simple linear weighting, such as $4C_{down} + C_{up}$ if the downstream bandwidth is 4 times that of the upstream.

Table 3.1 shows a comparison of the communication cost for each of the protocols in this thesis.

Table 3.1: A comparison of the communication costs (in bits) for the PIR protocols discussed in this thesis.

Protocol	Query Cost	Response Cost
AG07 [AG07]	$(6N^2 w_{AG})\, n$	$6s$
Recursive AG07 [AG07]	$d\,(6N^2 w_{AG})\, \sqrt[d]{n}$	$6^d s$
G07 [Gol07]	$\ell w_G n$	ℓs
Our hybrid (with AG07 and G07)	$\ell d\,(6N^2 w_{AG})^{\frac{d-1}{d}}\, \sqrt[d]{w_G}\sqrt[d]{n}$	$\ell 6^{d-1} s$

If we use our hybrid protocol with a depth of $d = 1$, then we are simply using the G07 protocol (with no CPIR component) and so will clearly have the same amount of communication as the G07 protocol. Since we choose the value of d that minimizes the communication cost for our hybrid protocol, we only use $d > 1$ if doing so results in a lower communication cost. Hence when we use a depth of $d > 1$, we will have a lower communication cost than the G07 scheme. Therefore, our hybrid scheme will not have a higher communication cost than G07 for any depth.

Consider the difference between the per-server cost of our hybrid protocol and G07:

$$\Delta(d) = d(6N^2 w_{AG})^{\frac{d-1}{d}} \sqrt[d]{w_G n} + 6^{d-1}s - w_G n - s.$$

Note that $\Delta(1) = 0$ since the hybrid protocol with depth $d = 1$ is the same as G07. By finding the second derivative, we can conclude that Δ is concave up for $d > 1$ since

$$\Delta''(d) = \frac{1}{d^3}(6N^2 w_{AG})^{\frac{d-1}{d}} \sqrt[d]{w_G n}(\ln(C) - \ln(w_G n))^2 + s\ln(6)^2 6^{d-1} > 0$$

on this range. Therefore, for some n and s, either $\Delta(2) \geq 0$ and so $\Delta(d) \geq 0$ for all integers $d \geq 2$, or $\Delta(2) < 0$, which occurs if and only if there exists an integer $d \geq 2$ such that our hybrid protocol with depth d costs less than G07. Therefore, our protocol outperforms G07 if and only if

$$\begin{array}{rrcl} & d(2) = 2\sqrt{6N^2 w_{AG} w_G}\sqrt{n} + 6s - w_G n - s & < & 0 \\ \iff & w_G n - 2\sqrt{6N^2 w_{AG} w_G}\sqrt{n} - 5s & > & 0, \end{array}$$

which occurs when

$$\begin{array}{rcl} \sqrt{n} & > & \frac{2\sqrt{6N^2 w_{AG} w_G} + \sqrt{4\cdot 6N^2 w_{AG} w_G + 4\cdot 5s}}{2w_G} \\ & = & \frac{\sqrt{6N^2 w_{AG}} + \sqrt{6N^2 w_{AG} + 5s}}{\sqrt{w_G}}, \end{array}$$

Table 3.2: The minimum number of blocks needed to make using the hybrid protocol more communication efficient than G07 for the given block sizes and the typical values for our security parameters.

Block Size	Minimum Blocks for Hybrid to Outperform G07
1 KB	159,844 blocks
10 KB	239,565 blocks
100 KB	858,946 blocks
1 MB	5,944,267 blocks

or

$$n > \frac{12N^2 w_{AG} + 5s + 2\sqrt{36N^4 {w_{AG}}^2 + 30N^2 w_{AG} s}}{w_G}.$$

Table 3.2 shows the minimum number of blocks needed to make using the hybrid protocol more communication efficient than G07 for a given block size when using the typical values for N, w_{AG} and w_G. Notice that the minimum number of blocks grows more slowly than the block size.

Comparing the formulas in Table 3.1, we see that the upstream cost of our hybrid protocol is no worse than that of Recursive AG07 when $\ell^d \leq \frac{6N^2 w_{AG}}{w_G}$ (= 37500 for the recommended parameters), and similarly for the downstream cost when $\ell \leq 6$. For many reasonable PIR setups, these inequalities are easily satisfied. Even if they are not, however, the computational savings of our scheme over Recursive AG07 (see below) more than make up for the difference. A slight complication in the analysis arises in cases in which the optimal recursive depth d differs between the Recursive AG07 scheme and our hybrid scheme; however, we will see in Section 3.3 that our scheme nonetheless outperforms the Recursive AG07 scheme.

3.2.3 Computation

Unlike our analysis of communication, we do not have simple expressions for our computation costs. In this section we reason about the computational cost of our protocol compared to others; in Section 3.3.2, below, we directly measure the computation costs of our scheme using empirical experimentation. The key observation, however, is that the slower CPIR protocol is being performed over a *much smaller database* than the original.

The protocol effectively consists of IT-PIR over the whole database of n records, followed by recursive CPIR over a sub-database of δ_1 records.

Query Encoding

AG07 is expensive when generating the query because it involves matrix multiplications. However, G07 is relatively cheap because it is essentially just generating random values and evaluating polynomials. We expect the hybrid scheme will be better than AG07 for this step because it replaces one iteration with the cheap G07 scheme encoding. Our hybrid scheme may also be faster in this stage than G07 because of the addition of recursion. As when recursion is added to the AG07 protocol, we change the request from one large (size n) query into d much smaller (size proportional to $\sqrt[d]{n}$) queries.

Server Computation

The AG07 scheme is also expensive compared to the G07 scheme for server-side computation. This is because AG07 uses matrix-by-matrix multiplication for the bulk of its work, whereas G07 uses vector-by-matrix multiplication. Our hybrid scheme will use the relatively cheap server computation of G07 for the first iteration where the database is its full size. The subsequent iterations will use a much smaller subset of the database, so using AG07's server computation will not add much additional expense.

Response Decoding

The last recursive step of decoding for our hybrid scheme will take the same amount of computation as the G07 scheme. Since we have $d-1$ steps of AG07 decoding as well, our hybrid protocol will not outperform G07 in the decoding step. Our hybrid protocol will also need to do any AG07 decoding once for every server at every recursive step. However, the response being decoded at each recursive step is smaller than that of the recursive AG07 protocol by a factor of 6 in our hybrid scheme. Therefore, when $d > 1$, the decoding for our hybrid protocol will be comparable to that of recursive AG07.

If there are a significant number of Byzantine servers—those that attempt to maliciously alter the result of the query—then the decoding time will be increased for the G07 iteration of our hybrid scheme, though this increase will not be very significant compared to the server computation of the G07 scheme [DGH12].

3.2.4 Privacy

The AG07 scheme keeps the client's query private as long as the servers are computationally bounded and as long as the Hidden Lattice Problem and the Differential Hidden Lattice Problem are indeed hard to solve. Our hybrid scheme also relies on these assumptions for perfect privacy.

The G07 scheme keeps the query private as long as no more than t servers are colluding to find the contents of the query. Our hybrid scheme also relies on this non-collusion assumption for perfect privacy.

One advantage of our hybrid scheme is that if the privacy assumptions for one of the inner protocols is broken, then the query will still be *partially* private as long as we use depth $d > 1$. For example, if the G07 non-collusion assumption is broken, then the colluding servers will be able to find out a subset of the database that the desired record is in, but they will not find out which record in that subset is the wanted one as long as the AG07 assumptions still hold. We similarly have partial privacy if the AG07 computational assumptions are broken and the G07 non-collusion assumption still holds.

This "defence in depth" is a benefit because it may dull some of the fears about using a scheme that a user thinks does not adequately enough guarantee privacy. For example, if someone does not feel that the non-collusion assumption is adequate enough for the G07 scheme, they may be more comfortable using this hybrid scheme because they know that even in the event that too many servers collude, they will still maintain some privacy.

Unlike the hybrid protection extension to G07 (Section 2.2.2), our protocol does not provide perfect privacy if one of the two privacy assumptions fails. The advantages of using our protocol over hybrid G07 are a significant reduction (more than 3 orders of magnitude) in computation time and, as will be illustrated in Section 3.3.1, improved communication cost.

3.2.5 Robustness

As stated previously, the G07 scheme has the ability to correct for servers not responding or responding incorrectly. The single-server AG07 scheme, however, does not have any robustness.

In 2012, Devet et al. [DGH12] observed that the G07 protocol can be slightly modified to be able to withstand up to $v < k-t-1$ misbehaving servers, with no extra computation or communication cost over the original protocol, in a typical setting where clients aim to fetch multiple records from the database. (The original G07 bound [Gol07] is $v < k - \sqrt{kt}$ when only one record is retrieved.)

An advantage of using G07 as the first iteration of our scheme is that the hybrid protocol retains exactly the same robustness properties as G07: any misbehaviour will be detected and corrected for at the Ψ IT-PIR multiple-server decoding step.

3.3 Implementation and Empirical Evaluation

We have implemented these protocols as an extension to the Percy++ [GDHH13] library, an implementation of Goldberg's scheme from Section 2.2.2. We incorporated both the AG07 CPIR scheme and our hybrid scheme. Our implementation will be available in the next release of Percy++.

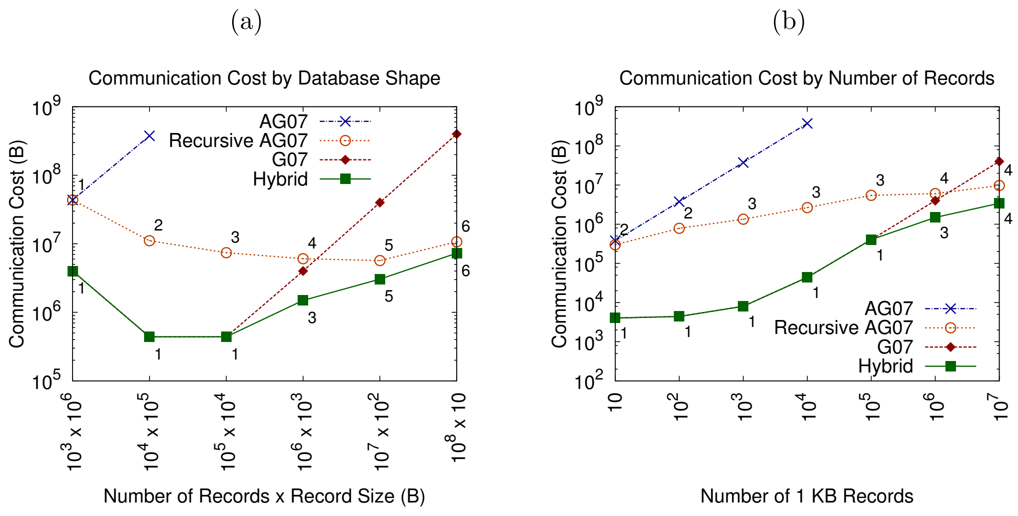
The implementation of our hybrid PIR system combines the implementations of the two inner protocols (G07 and AG07), using them as black boxes. Given all of the other parameters, our implementation will find the optimal depth (d) and the best way to split the database for the first (IT-PIR) iteration of the scheme (γ_1) to minimize the communication cost.

All of our multi-server queries were run on $\ell = 4$ servers and we used $t = 1$ for the G07 privacy parameter. For our client machine, we used a 2.4 GHz Intel Xeon 8870, and each server machine was a 2.0 GHz Intel Xeon E5-2650. All computations reported here were done in a single thread, so that the reported times reflect total CPU time. However, all of the computations are almost completely parallelizable [Dev13], and as we will show in Chapter 4, using multiple cores would greatly reduce end-to-end latency, though not total CPU time.

3.3.1 Communication

The plots in Figure 3.1 illustrate the amount of communication needed for our hybrid scheme and the schemes of which it is comprised. We see that our hybrid protocol uses less communication than that of AG07, and no more than that of G07, verifying the analysis in Section 3.2.2, above.

Figure 3.1: Comparison of communication used by each scheme. Plot (a) shows the communication used for queries on a 1 GB database for different database shapes. In plot (b), the record size is fixed at 1 KB and we see the communication for different numbers of records. Non-recursive AG07 imposes a limit of approximately 10,000 records so we do not have data points for larger numbers of records. Error bars are present for all data points, but may be too small to see. The datapoint labels for the Hybrid and Recursive AG07 schemes indicate the recursive depth used.



3.3.2 Computation

Query Encoding

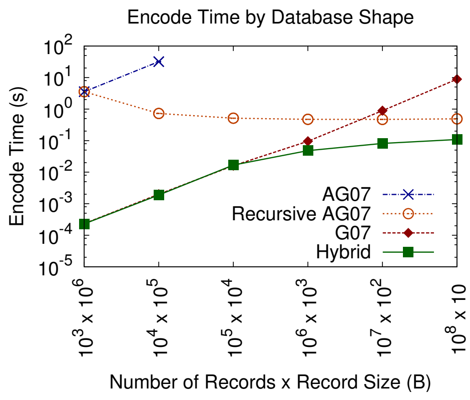
Our experimental results show us that the encoding time is very much related to the size of a PIR request. This is evidenced by how similar Figure 3.2b is to Figure 3.1b (the communication associated with the same tests). The query encoding time for G07 is linear in the number of records. On the other hand, the query encoding time for the AG07 scheme is dominated by the d^{th} root of the number of records. Because of this, for larger numbers of records, the hybrid protocol encodes queries faster than G07.

Server Computation

As expected, Figure 3.3 shows that the server computation time of our hybrid PIR system is very comparable to that of the G07 protocol. The figures also show that our hybrid

Figure 3.2: Comparison of the time used by the client to encode the request for each protocol. Plot (a) shows the encoding time for queries on a 1 GB database for different database shapes. In plot (b), the record size is fixed at 1 KB and we see the computation time for different numbers of records. Non-recursive AG07 imposes a limit of approximately 10,000 records so we do not have data points for larger numbers of records. Error bars are present for all data points, but may be too small to see.

(a)



(b)

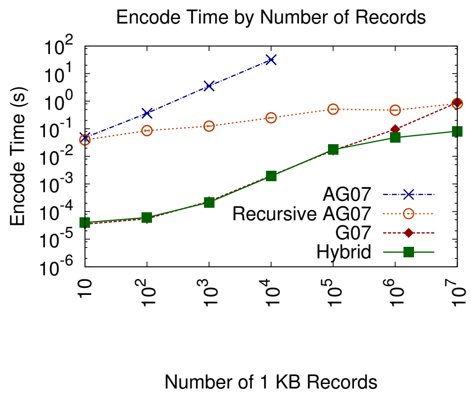
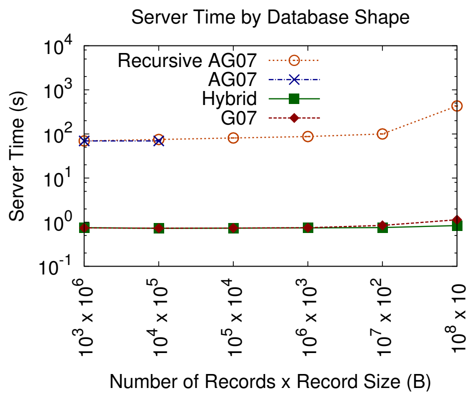


Figure 3.3: Comparison of the time used by the server(s) to process the client request for each protocol. For further details see the caption for Figure 3.2.

(a)



(b)

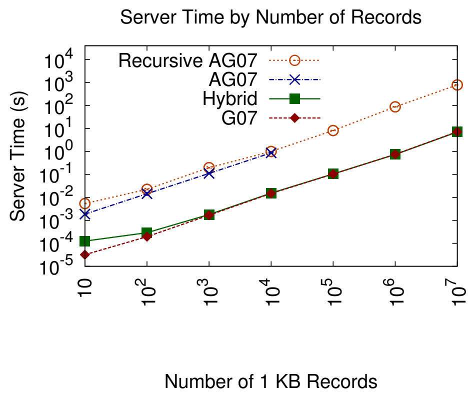
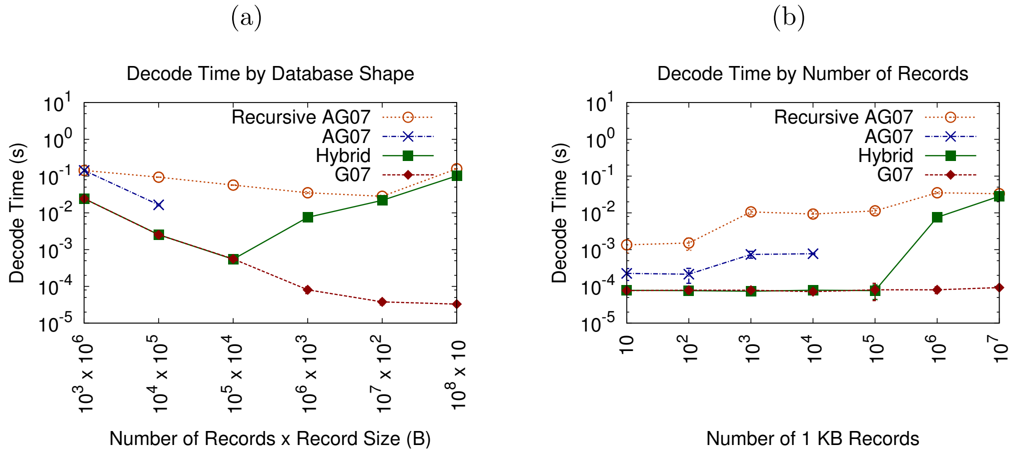


Figure 3.4: Comparison of the time used by the client to decode the server response for each protocol. For further details see the caption for Figure 3.2.



system performs its server computation approximately 2 orders of magnitude faster than Recursive AG07. As noted above, this time is also highly parallelizable; the times reported in the figure use only a single thread, and so represent total CPU time.

Response Decoding

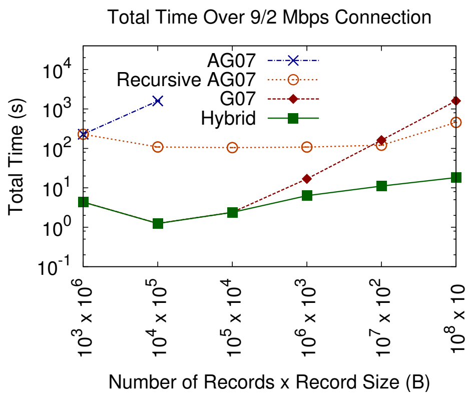
Figure 3.4 shows us that when we have a depth of at least 2 for the hybrid PIR system (i.e., we have at least one iteration of AG07) , the decoding time approaches that of recursive AG07. This is because, unlike the server computation where the cheap G07 computation is being done on the first iteration when the database is large, the cheap G07 decoding is happening on the last iteration, when the response has been reduced in size by $d-1$ iterations of AG07 decoding. For this reason, the decoding step of our hybrid PIR system is comparable to that of recursive AG07 and not the quicker G07. Even so, we note that the time of the decoding step is quite insignificant compared to the server computation step of a query.

3.3.3 Total Query Time

In Figures 3.5, 3.6, and 3.7 we plot the total time for a query on our hybrid PIR system as well as its component protocols. We show the total time for three different connection

Figure 3.5: Comparison of total query time for each scheme over a connection with 9 Mbps download bandwidth and 2 Mbps upload bandwidth. Plot (a) shows the time for a 1 GB database with different database shapes. In plot (b), the record size is fixed at 1 KB and we see the time for different numbers of records. Non-recursive AG07 imposes a limit of approximately 10,000 records so we do not have data for larger numbers of records. Error bars are present for all data points, but may be too small to see.

(a)



(b)

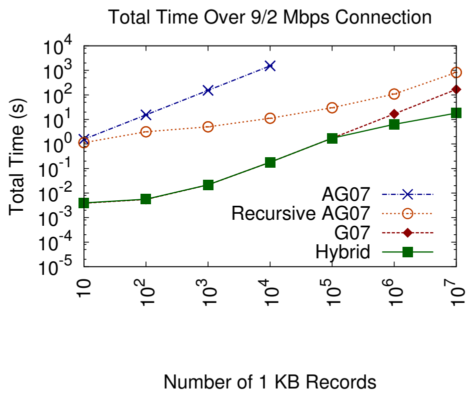
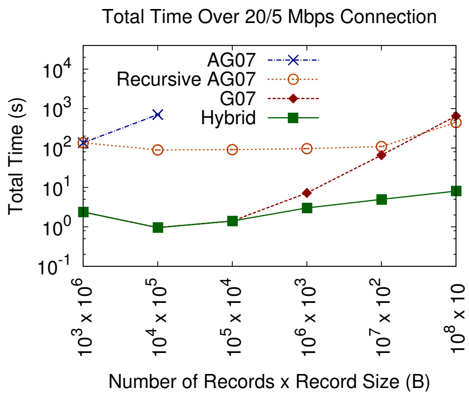


Figure 3.6: Comparison of total query time for each scheme over a connection with 20 Mbps download bandwidth and 5 Mbps upload bandwidth. For further details see the caption for Figure 3.5.

(a)



(b)

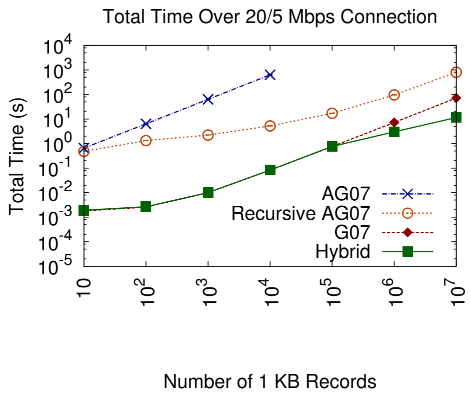
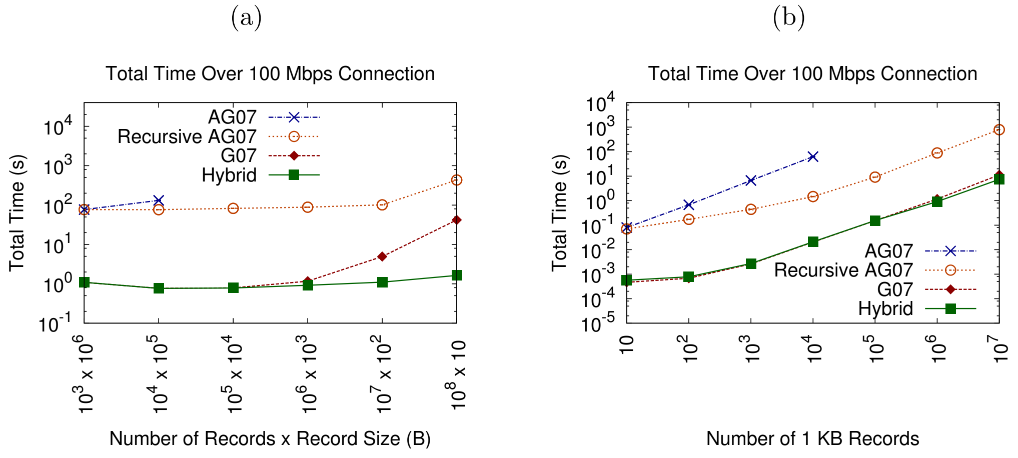


Figure 3.7: Comparison of total query time for each scheme over a connection with 100 Mbps download bandwidth and 100 Mbps upload bandwidth. For further details see the caption for Figure 3.5.



speeds between the client and server(s). Figure 3.5 shows tests using a connection with 9 Mbps download and 2 Mbps upload. This connection was used by Olumofin and Goldberg [OG11] to represent a home user's connection in 2010. Using the same source [Ook14], we represent a home user in 2014 in Canada or the U.S. with 20 Mbps download and 5 Mbps upload in Figure 3.6. Figure 3.7 models a connection over 100 Mbps Ethernet.

Our results show us that the total query time needed for our hybrid PIR system is similar or better than that of G07. We also see that the total query time of recursive AG07 is approximately 2 orders of magnitude larger than that of our system.

These plots also illustrate that our hybrid PIR system does not use much communication time. This is because the total query time of the hybrid system does not improve much when the network capacity is increased. Contrast this with G07 when there are a large number of records—in this case we see a significant improvement in total query time as the network capacity increases.

3.4 Summary

In this chapter, we introduced a hybrid PIR protocol that combines the recursive property of CPIR protocols with the communication and computation efficiency of IT-PIR protocols.

We showed that our hybrid protocol is more communication efficient than the CPIR and IT-PIR protocols of which it is composed and that it inherits other desirable properties including robustness. We have implemented our hybrid protocol as an extension to the Percy++ [GDHH13] library and show empirical evidence of these benefits.

Chapter 4

Parallelized Server Computation

4.1 Parallelizing PIR

For each of these protocols, the computation required of each server is a significant portion of the total query time. A common approach to decreasing the time needed for computation is parallelization: splitting the job into parts that are not dependent on each other, computing these parts at the same time on different resources and putting the results together to form the result of the entire job. As shown in the sections above, the server-side computation for all of these PIR protocols is a matrix multiplication operation, which lends itself quite naturally to parallelization.

Previous work [Dev13] has shown that the G07 protocol can be parallelized effectively. We expand on this work by parallelizing all of the PIR protocols that we have outlined and providing different methods of parallelization to suit the resources available to those using our system. We have implemented this work as part of Percy++ [GDHH13] and present the results produced by our implementation.

Please note that for any of the multiserver protocols, it is likely that the client will not experience a speedup from parallelization of server-side computation unless all servers (or at least the slowest servers) use parallelization.

4.1.1 More Than One Way to Split Computation

To perform parallelization, we need to have the ability to process multiple parts of a job at once. We outline two methods of parallel execution and discuss when each is appropriate.

Multithreading

If a machine has multiple processors, we can use multithreading to perform different parts of the matrix multiplication at once over the machine's cores. The advantage of using multithreading is that worker threads use shared memory and will not need to use significant resources to communicate to each other.

Distributed Computation

Another method of parallel execution is to distribute computation to a group of worker processes on different machines. We note that these processes could be on the same machine, however if that is the case, using multithreading would be preferable. The main process will need to send part of the request to each of the workers and, after doing their portion of the work, the workers send their results back to the main process.

A Combination

We could also use a combination of both methods: Our main processes splits the computation up between workers on separate machines. Each of these workers will then use multithreading to split their portion computation between the processors on their machine.

Our implementation provides all of these methods of parallel execution. From this point we will use the term *worker* to refer to either a worker thread or worker process.

4.1.2 More Than One Way to Split a Request

To simplify things, we will first consider a non-recursive PIR scheme (i.e. G07, C95 or non-recursive AG07). In all of our protocols, the servers are performing a matrix multiplication of two matrices: a request matrix and a database matrix. Figure 4.1 illustrates the server-side computation for the G07 and C95 algorithms. Note that the given dimensions are the dimensions of the matrices in the field $\mathbb{F}$ used for computation. In G07 and C95, each row of the request matrix is a query for one record, each row of the database matrix is a record and each row of the response matrix is the server response for one record. We will use m to denote the number of words in $\mathbb{F}$ each record is made up of and use Q to denote the number of records the client has requested.

Figure 4.1: Server-side computation for G07 and C95.

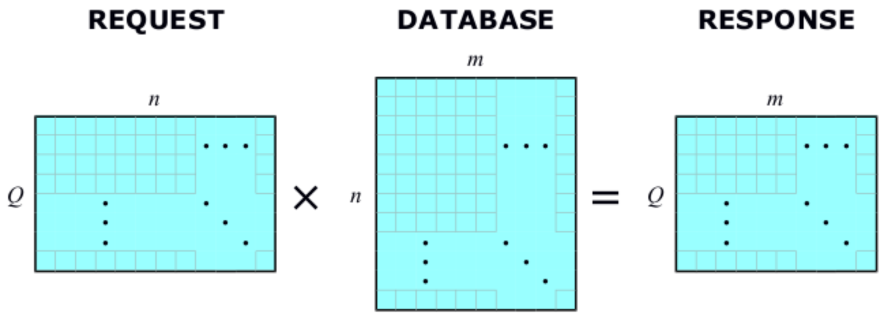
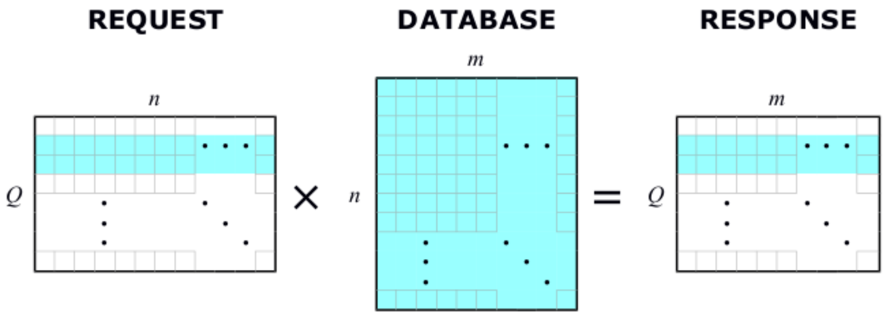


Figure 4.2: Server-side computation for G07 and C95 when partitioning the record queries.

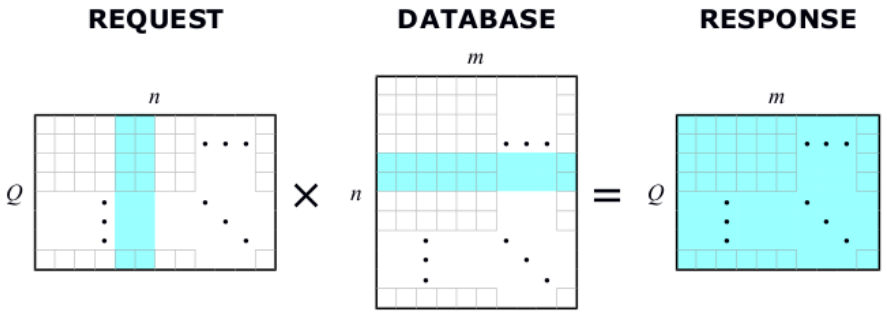


Partitioning the Queries

One approach to parallelizing the server-side computation is that each worker performs the computation for a subset of the record queries that the client has made. Figure 4.2 illustrates the part of the computation for one worker for G07 or C95. A worker gets a portion of the request matrix that corresponds to a subset of the requested records. It applies its part of the request matrix to the entire database and comes up with a portion of the response matrix. To complete the computation, all of the worker responses must be concatenated together to get the full response.

An advantage to this approach is that there is very little overhead involved in partitioning the request or combining the worker responses. However, the parallelization is not effective unless the client queries for multiple records.

Figure 4.3: Server-side computation for G07 and C95 when partitioning the records of the database. Note that the responses from the workers must be added together to get the full response for the client.



Partitioning the Database Records

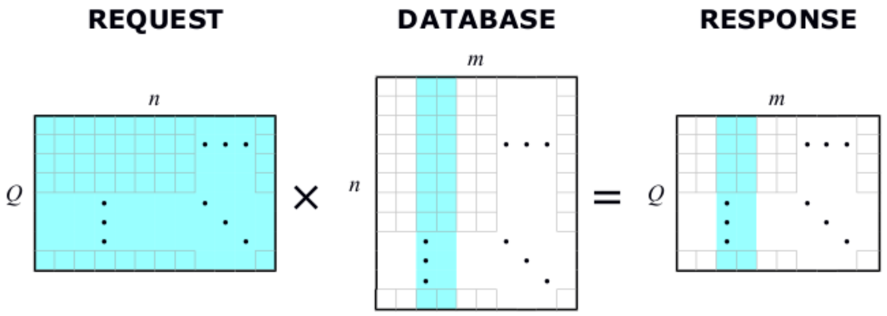
Another approach is to give each worker a portion of the database. That is, we partition the records and assign some to each worker. Figure 4.3 illustrates the computation for one worker using this approach. The request matrix is partitioned so that each worker gets the part of every query that corresponds to its part of the database. It applies its part of the request matrix to its part of the database matrix and comes up with a response matrix. To complete the computation, all of the worker responses must be added together to get the full response to send to the client.

This approach is particularly useful for the case where the database is distributed between a set of machines in a cluster as each machine can do the computation for the part of the database which it possesses and the parts of the database will not need to be sent across the network for another machine to process. The main disadvantage is that the matrix addition of worker responses involves more overhead than the other approaches use for their combination step.

Partitioning the Words of Each Record

One more approach to parallelizing the computation is to partition the database matrix so that a worker is assigned a portion of each record. Figure 4.4 illustrates the computation for one worker. The worker is given the entire request matrix and applies it to its portion of the database matrix. To complete the computation, the worker responses must be joined together as shown in the figure.

Figure 4.4: Server-side computation for G07 and C95 when partitioning the words of each record of the database.



It is important that our partition assigns whole columns of the database matrix to each worker (or whole rows in the case of AG07; see Figure 4.8). If the partitioning is done incorrectly the response-combining step is not as straightforward as concatenating parts of the worker responses.

This approach is similar to partitioning the queries in that there is very little overhead in the combination step. A disadvantage is that, in the case of distributed worker processes, more communication is necessary since each worker needs the entire request matrix.

The first two approaches have been implemented as part of Percy++ [GDHH13], however this third approach has not been implemented. This may be part of a future release of Percy++.

A Combination

A final possibility is to combine two, or even all three, of the above approaches to parallelizing the server-side computation.

* * *

Figures 4.5, 4.6, 4.7 and 4.8 illustrate the approaches to parallel server-side computation for AG07. Note that the database matrix is on the left of the multiplication operation and the request matrix is on the right (the opposite of the order of G07 and C95). The database matrix is made up of n side-by-side submatrices of size $L \times N$ where each submatrix is a record. The request matrix is made up of Q (number of records requested) side-by-side submatrices of size $N \cdot n \times 2 \cdot N$ where each submatrix is the query for one record.

Figure 4.5: Server-side computation for AG07.

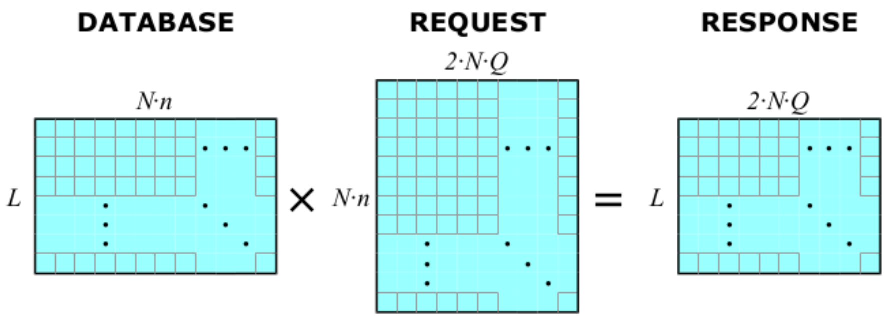
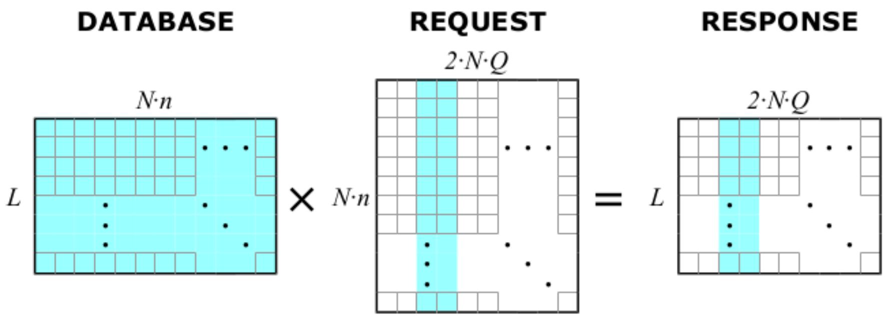


Figure 4.6: Server-side computation for AG07 when partitioning the record queries.



4.2 Parallelizing Recursive PIR

We will now apply parallelization to our recursive PIR protocols: recursive AG07 and our hybrid protocol.

Partitioning the Queries

Since the queries for each record are independent, we can simply partition the recursive query requests. That is, if some worker is assigned query i, the response will be correct as long as that worker computes all iterations of query i.

Figure 4.7: Server-side computation for AG07 when partitioning the records of the database. Note that the responses from the workers must be added together to get the full response for the client.

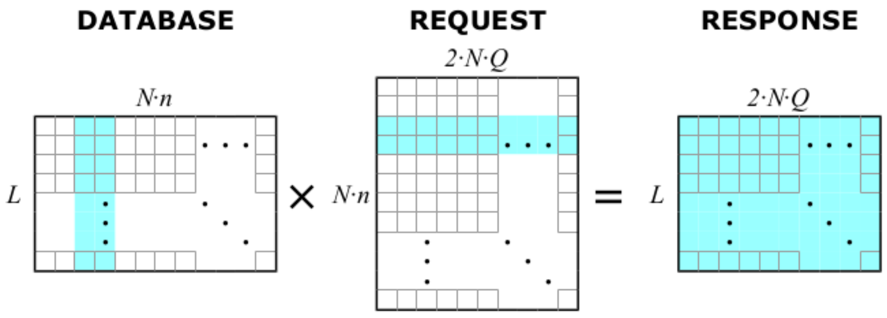
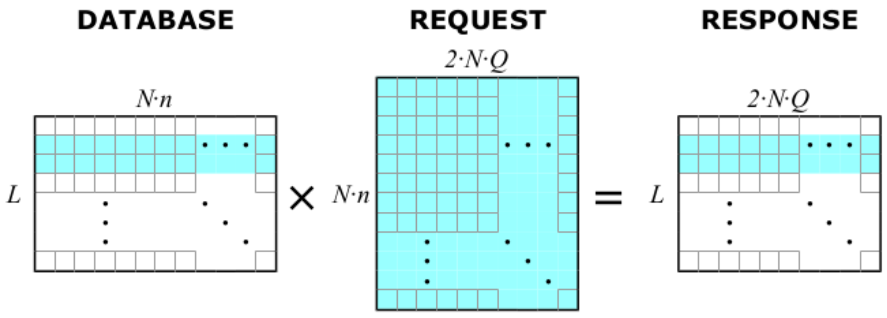


Figure 4.8: Server-side computation for AG07 when partitioning the bytes of each record of the database.



Partitioning the Database Records

It is not so easy applying parallelization to recursive PIR when we assign a subset of the records to each worker. Consider a case where we are using recursive AG07 with a depth of at least two. We can split the records of the original database among our workers and have them do the computation for the first iteration of the protocol. After one iteration of the protocol, each word of the result will be split into six pieces, each of which is a word in the subdatabase for iteration two. However, if we do not combine (add) the workers' responses before this splitting operation, we will not get the correct subdatabase needed for iteration two. That is, the matrix addition modulo p does not commute with the database preparation operations between iterations one and two. So we can only parallelize in this way if we do the worker response combining step between each iteration, adding more overhead.

This is especially troublesome when the work is being distributed over multiple worker processes because they will need to send their result back to the master server between each iteration and then wait for the new subdatabase to be sent back to them. This adds a significant amount of overhead and may not be worth the hassle. One way to mitigate this is to do distributed computation for the first iteration and allow the master server to compute all the following iterations. This way, the largest iteration, the first, will be parallelized and the amount of extra communication between workers is reduced. Our implementation includes an option to do distributed computation for only the first iteration when each worker gets a subset of the database records.

4.3 Empirical Evaluation

We have implemented parallelized server computation for all of the PIR protocols discussed in this thesis as an extension to the Percy++ [GDHH13] library. Our implementation will be available in the next release of Percy++.

All of our servers and worker servers were run on a 2.0 GHz Intel Xeon E5-2650 which has 16 cores and is able to provide 32 simultaneous threads using hyperthreading. Our measurements are of just the server computation time; we do not include the time the server takes reading a request or sending a response.

Each test queried the server(s) for 50 records. The plots show the number of records that can be retrieved per second based on the number of threads and/or worker processes being used for the computation. In these plots, highly parallelized computation is evidenced by a near-linear relation.

4.3.1 Multithreading

G07 and our Hybrid Protocol

Figures 4.9 and 4.10 show the results of using multithreading with G07 and our hybrid scheme. Note that Figure 4.9 only shows results for G07 because with this size of database, our hybrid protocol will simply be running G07 and has the same results.

The near-linear relations seen for G07 confirm that its server computations are highly parallizable. We see that as the number of threads becomes very large, the effects of

Figure 4.9: The number of queries possible per second given the number of threads. The 1 GB database used was a square 32768 records each of 32768 B.

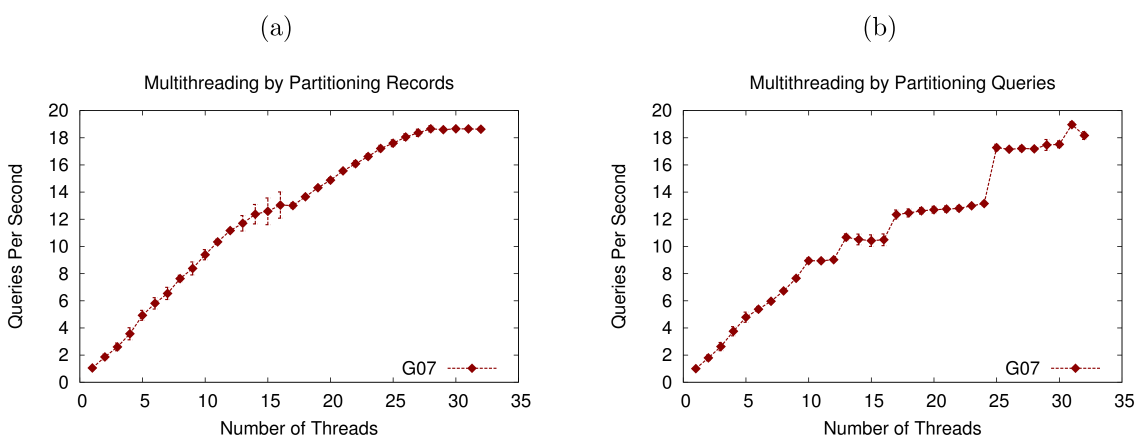
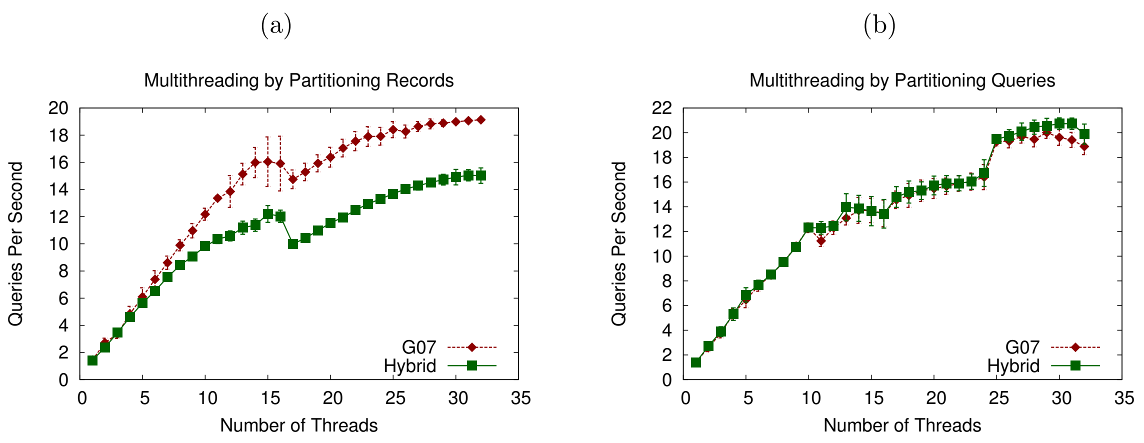


Figure 4.10: The number of queries possible per second given the number of threads. The 1 GB database used was 1048576 records each of 1024 B.



the overhead of parallelization start be seen. Notice that when partitioning records (Figure 4.10a), our hybrid protocol is affected by overhead much more than G07. This is because we need to combine the results from each thread at the end of every iteration (in this case $d = 3$).

Also when partitioning the records (Figures 4.9a and 4.10a), we see a blip in the performance at around 16 threads. We suspect this is because the servers can handle 32 threads at a time using hyperthreading, but they only have 16 actual cores.

Note that there are a few plateaus in the data when partitioning the queries. This is because the tests only requested 50 records. For example, for all $t \in [17, 24]$, if there are t threads, then some thread will need to do the computation for three of the fifty queries, and so for these values of t we expect the computation time to be approximately the same. If we query for more than 50 records, we expect these plateaus to smooth.

4.3.2 Distributed Computation

G07 and our Hybrid Protocol

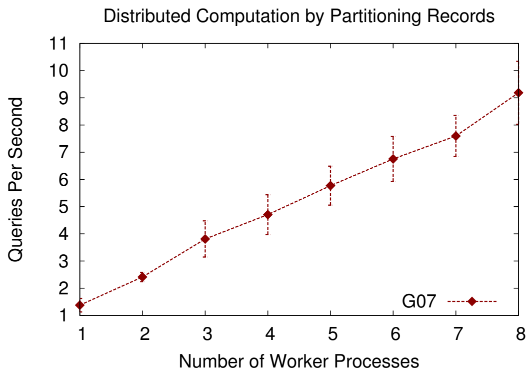
Figures 4.11 and 4.12 show the number of queries that can be processed per second given the number of worker processes for G07 and our hybrid protocol. We can see that parallelization is having the desired effect: the number of queries possible is increasing by nearly a factor of the number of workers.

When partitioning records, as in Figure 4.12a, we see that overhead is affecting our hybrid protocol more than G07. As with the same case using multithreading, we need to combine worker results between each iteration. Since we are using distributed worker processes, this means that the master server needs to receive all of the worker responses between each iteration, add the responses and send the subdatabase for the next iteration to all of the workers. The workers cannot continue their computation until they receive this new subdatabase.

Figure 4.12a includes tests where only the first iteration of the hybrid protocol is distributed and the other iterations are computed by the master. We see that there is some improvement in the computation time, however, it is not very significant. This is likely because the hybrid protocol does not have very large databases after iteration one and this means there is not much computation after iteration one.

Figure 4.11: The number of queries possible per second given the number of worker processes. The 1 GB database used was a square 32768 records each of 32768 B.

(a)



(b)

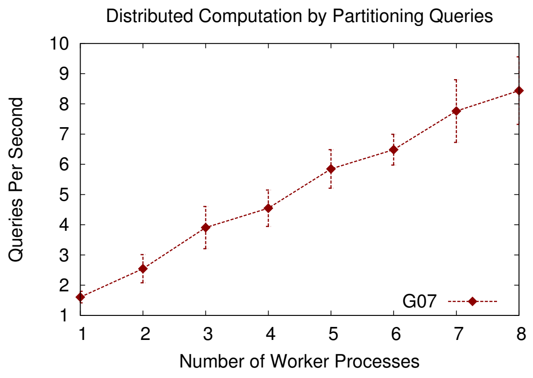
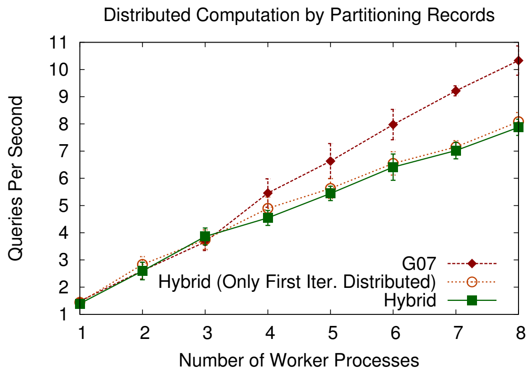


Figure 4.12: The number of queries possible per second given the number of worker processes. The 1 GB database used was 1048576 records each of 1024 B. Plot (a) contains results for Hybrid when only the first iteration is distributed (and the rest are computed on the master server).

(a)



(b)

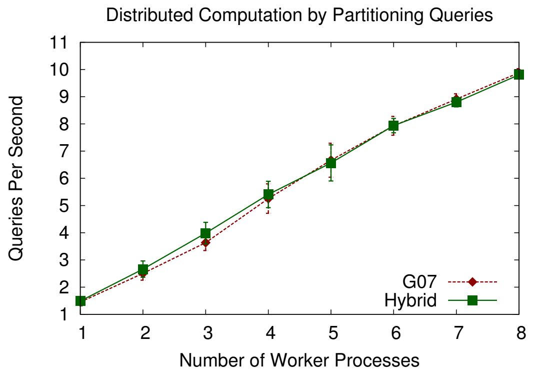
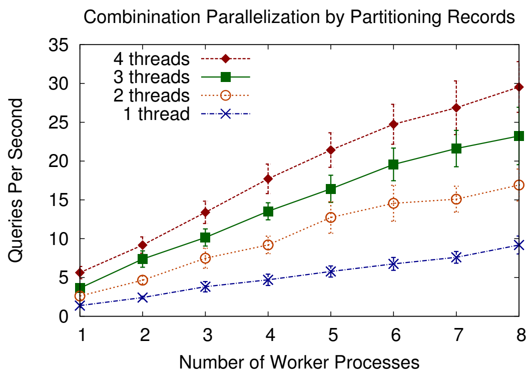


Figure 4.13: The number of queries possible per second given the number of worker processes using G07. Each line corresponds to the number of threads used on each worker. The 1 GB database used was 32768 records each of 32768 B. In both plots the workers' records are split between worker threads.

(a)



(b)

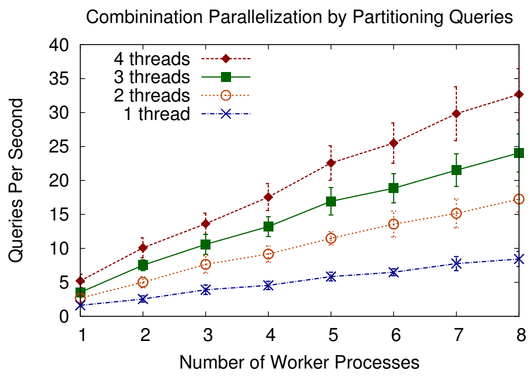
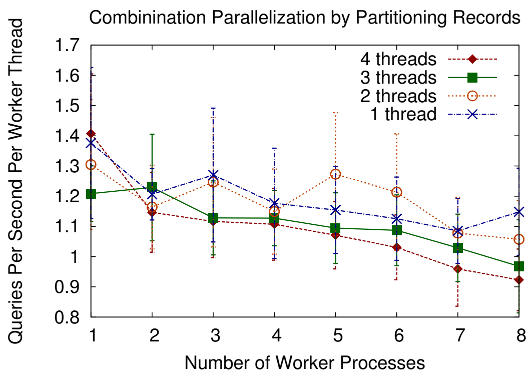
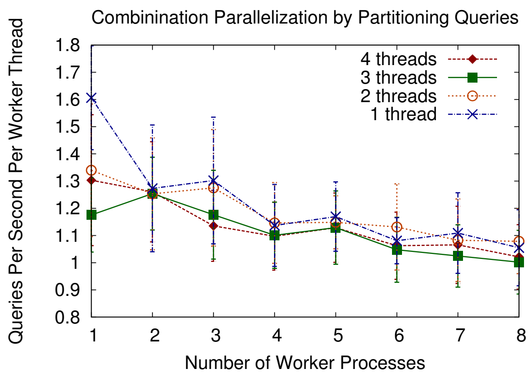


Figure 4.14: The number of queries possible per second per worker thread for a given number of worker processes using G07. Each line corresponds to the number of threads used on each worker. The 1 GB database used was 32768 records each of 32768 B. In both plots the workers' records are split between worker threads.

(a)



(b)



4.3.3 A Combination

The tests in Figure 4.13 combine the use of distributed workers and multithreading for G07. There are a number of worker servers and each server is doing computation over a number of threads. Each line in the plot corresponds to a certain number of threads. In Figure 4.14 we show the queries per second per worker thread for the same tests. Our lines have a slight downward slope, signifying that as we add workers there is more effect from the overhead of splitting the request between workers and combining the results. There is a similar trend as the number of threads increases. However, the effects of overhead are not very significant when error and the scale of the plot are taken into consideration. As expected, the improvement in performance is approximately the number of workers multiplied by the number of threads on each worker.

Using parallel server computation makes it possible to use PIR on very large databases. To show this, we tested G07 on a 1 TB database using 8 worker machines, each running 16 threads. The database records were split evenly between workers and on each worker the database records were again split evenly between the threads. The server computation for PIR queries of this database took 7.9 ± 0.8 s. We also ran G07 with no parallelization on a database of size 7.82 GB, approximately the size of the subdatabases that the threads owned in the pervious test, and the server computation took 7.3 ± 0.8 s. This shows that, even at such a large scale, parallel server computation is effective. We conclude that, through parallel computation, PIR is not just feasible on large databases, but practical.

4.4 Summary

In this chapter, we showed how we can use multithreading and/or distributed computation to parallelize the server-side computation of a variety of PIR protocols, including a CPIR protocol by Aguilar Melchor and Gaborit [AG07], an IT-PIR protocol by Goldberg [Gol07], and the hybrid PIR protocol introduced in Chapter 3 of this thesis. We have implemented parallelization for these protocols as an extension to the Percy++ [GDHH13] library and showed that these protocols are indeed highly parallelizable. We showed that by using parallelization, PIR is possible on large-scale databases and that private queries of such databases can be performed in a reasonable amount of time.

Chapter 5

Conclusion

We introduce a hybrid Private Information Retrieval protocol that combines the low communication and computation costs of multiple-server IT-PIR protocols with the ability of single-server CPIR protocols to do recursion. We show that our protocol inherits several positive aspects of both types of protocols and mitigates the negative aspects. In particular, our protocol maintains partial privacy of client query information if the security assumptions made by one of the inner protocols is broken.

We have implemented this hybrid protocol as part of the open-source Percy++ library for PIR, and using this implementation, demonstrated that our protocol performs as well or better than PIR schemes by Aguilar Melchor and Gaborit and by Goldberg. Our hybrid scheme is particularly effective when the number of records in a database is large relative to the size of each record—a situation that arises naturally in a number of network scenarios, including TLS certificate checking, private LDAP lookups, sensor networks, and more.

We have also implemented parallelized server-side computation for a number of PIR protocols as an extension to the Percy++ library. We show that by using multithreading or distributed worker processes we can greatly reduce the server computation time, allowing requests to be served much more quickly. In most cases, we can increase the number of queries possible in some time period by a factor of the number of worker threads or processes, achieving perfect scalability.

5.1 Future Work

AG07 using GPUs

Aguilar Melchor et al. [ACG+08] demostrate how the AG07 scheme can be made much faster by implementing the server-side computations on GPUs instead of CPUs. Our implementation does not include this feature, but we plan on implementing it in the future and investigating how much this will speed up our hybrid protocol.

Security of AG07

AG07's privacy guarantees rely on the hardness of the Hidden Lattice Problem and the Differential Hidden Lattice Problem, as specified by Aguilar Melchor and Gaborit [AG07]. According to Aguilar Melchor et al. [ACG+08] and Olumofin et al. [OG11] the security of this scheme is not well understood. Future work could involve investigating the security of the scheme and either developing a security proof or altering the scheme to make it provably secure.

References

[ACG$^+$08] Carlos Aguilar Melchor, Benoît Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre Rousseau. High-Speed Private Information Retrieval Computation on GPU. In *SECURWARE*, pages 263–272, 2008.

[AG07] Carlos Aguilar Melchor and Philippe Gaborit. A Lattice-Based Computationally-Efficient Private Information Retrieval Protocol. In *WEWORC 2007*, July 2007.

[Aso01] Dmitri Asonov. Private Information Retrieval: An overview and current trends. In *ECDPvA Workshop*, 2001.

[BIM04] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. *J. Cryptology*, 17(2):125–151, 2004.

[CCC$^+$09] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[CGN97] Benny Chor, Niv Gilboa, and Moni Naor. Private Information Retrieval by Keywords. Technical Report TR CS0917, Department of Computer Science, Technion, Israel, 1997.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *IEEE Symposium on Security and Privacy*, pages 2–15, 2003.

[Dev13] Casey Devet. Evaluating Private Information Retrieval on the Cloud. Technical Report 2013-05, CACR, 2013. http://cacr.uwaterloo.ca/techreports/2013/cacr2013-05.pdf.

[DGH12] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally Robust Private Information Retrieval. In *21st USENIX Security Symposium*, 2012.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*, 2004.

[GDHH13] Ian Goldberg, Casey Devet, Paul Hendry, and Ryan Henry. Percy++ project on SourceForge. http://percy.sourceforge.net, 2013. Version 0.9.0. Accessed February 2014.

[GGM98] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A Random Server Model for Private Information Retrieval. In *2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 200–217, 1998.

[Gol07] Ian Goldberg. Improving the Robustness of Private Information Retrieval. In *2007 IEEE Symposium on Security and Privacy*, pages 131–148, 2007.

[HOG11] Ryan Henry, Femi G. Olumofin, and Ian Goldberg. Practical PIR for Electronic Commerce. In *ACM Conference on Computer and Communications Security*, pages 677–690, 2011.

[Kik04] Hiroaki Kikuchi. Private Revocation Test using Oblivious Membership Evaluation Protocol. In *3rd Annual PKI R&D Workshop*, 2004.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In *FOCS*, pages 364–373, 1997.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, June 2013.

[McC90] Kevin S. McCurley. Odds and Ends from Cryptology and Computational Number Theory. In *Cryptology and Computational Number Theory*, volume 42 of *Proceedings of Symposia in Applied Mathematics*, pages 145–166, 1990.

[MOT$^+$11] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *20th USENIX Security Symposium*, pages 475–490, 2011.

[OG10] Femi Olumofin and Ian Goldberg. Privacy-preserving Queries over Relational Databases. In *10th International Privacy Enhancing Technologies Symposium*, pages 75–92, 2010.

[OG11] Femi Olumofin and Ian Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *15th International Conference on Financial Cryptography and Data Security*, pages 158–172, 2011.

[Ook14] Ookla. Net Metrics for Canada and the United States. http://www.netindex.com, 2014. Accessed February 2014.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt à Voter with Re-encryption Mixes. In *ESORICS*, pages 313–326, 2006.

[SC05] Len Sassaman and Bram Cohen. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *In Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2005)*, pages 1–9, 2005.

[SC07] Radu Sion and Bogdan Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2007.

[Ser06] Jim Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511, June 2006.

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

[SMA$^+$13] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.

[Xiv14] Xively. Public Cloud for the Internet of Things. http://www.xively.com, 2014. Accessed February 2014.