RAID-PIR: Practical Multi-Server PIR

Daniel Demmler
Engineering Cryptographic Protocols Group, EC SPRIDE
TU Darmstadt, Germany
daniel.demmler@ec-spride.de

Amir Herzberg
Computer Science Department
Bar-Ilan University, Israel
and TU Darmstadt, Germany
amir.herzberg@gmail.com

Thomas Schneider
Engineering Cryptographic Protocols Group, EC SPRIDE
TU Darmstadt, Germany
thomas.schneider@ec-spride.de

ABSTRACT

Private Information Retrieval (PIR) allows to privately request a block of data from a database such that no information about the queried block is revealed to the database owner. With the rapid rise of cloud computing, data is often shared across multiple servers, making multi-server PIR a promising privacy-enhancing technology.

In this paper, we introduce RAID-PIR, an efficient and simple multi-server PIR scheme, which has similar approach to RAID (Redundant Arrays of Inexpensive Disks) systems. Each server stores only a part of the database, its computational complexity depends only on this part, and multiple blocks can be queried efficiently in parallel. RAID-PIR improves efficiency over known PIR protocols, using only very efficient cryptographic primitives (pseudo-random generator). We demonstrate that RAID-PIR is practical and well-suited for cloud deployment as it reduces the communication as well as the computational workload per server.

Keywords

private information retrieval; RAID; redundant array of inexpensive disks; privacy enhancing technologies; cloud security; applied cryptography; secure storage

1. INTRODUCTION

Nowadays, the need for privately requesting files from the Internet is bigger than ever. Data can be encrypted during transport, thus effectively preventing a man-in-the-middle attacker from getting access to the data. However, a content provider needs to know what files a user requested, in order to deliver the content. Therefore, the user must trust the content provider to keep his request safe and to be protected against attacks. This trust cannot easily be established, especially because today many websites are not directly hosted by the content provider, but instead located in a Content Distribution Network (CDN) or on machines in the cloud. Furthermore, even if a content provider is trustworthy, he can be forced, e.g. by government agencies, to reveal information about the data that its users requested.

A solution to this problem is *Private Information Retrieval (PIR)*, which allows to protect the privacy of the users' queries. While PIR schemes with a single server exist, today's most efficient PIR schemes use multiple servers with the assumption that not all of them collude; our scheme further improves efficiency compared to known (multiple-server) PIR schemes. In particular, these servers could be run on different machines operated by different cloud providers, different administrators and/or in different regions or jurisdictions.

PIR Applications.

There is a multitude of applications for PIR schemes, with different reasons for hiding the identity of the item requested by the user. A typical motivation is to prevent disclosure of personal or business interest in specific information from a collection, e.g., or patents, medical articles, company evaluations, product descriptions, legal precedences or otherwise. For example, knowledge about patents requests may allow a competitor to identify directions of a company, and knowledge about requests for medical papers by an individual may expose an illness. Cappos [Cap13] applies PIR for a different motivation, namely, to hide identity of specific software update being retrieved, since knowing the requested update may allow an attacker to identify vulnerable system (i.e., a system using vulnerable, not yet updated, software). PIR can also be used to privately query messages from an encrypted mailbox [SCM05, BKOI07, MOT^{+}11] and is a building block in the recently proposed private presence service DP5 [BDG14]. A final motivation, from [GHS14], is to allow caching of encrypted (web) objects by an untrusted CDN, preventing the CDN from learning details by identifying the requested objects.

Redundant Array of Inexpensive Disks (RAID).

We use the name RAID-PIR, since our work shares design approach and ideas with RAID storage systems. RAID (Redundant Array of Inexpensive Disks), introduced in [PGK88], is a method to virtualize data storage that combines multiple storage units (e.g., disk drives), into one logical unit. The central idea of RAID is that by properly combining multiple units (drives), we can emulate a larger storage device, and/or achieve improvements in speed, reliability, etc., which would be infeasible or more expensive to build 'from scratch'. RAID defines multiple modes, providing a low-cost approach to improve different goals, mainly performance and/or protection against disk failures (reliability). All modes combine

CCSW'14, November 7, 2014, Scottsdale, Arizona, USA.
Copyright 2014 ACM 978-1-4503-3239-2/14/11 ...$15.00.
http://dx.doi.org/10.1145/2664168.2664181.

the results from the multiple disks; they do it in different ways, but the combining operation is always very simple and efficient. In particular, for improving performance, data is *striped* over multiple disks that can be accessed in parallel. To protect against disk failures, data is either *mirrored* on multiple disks or additional *parity* information is stored that allows data recovery. Further details on different RAID levels are given in Appendix A.

Our multi-server PIR scheme, RAID-PIR, has similar properties as a RAID system: the data is distributed to multiple servers (cf. RAID disks) for better performance, where each server stores only parts of the database. The data from the different servers is combined in a simple, efficient operation. The use of multiple server PIR implies that the entire system is trustworthy, even if some, but not all of the servers (cf. disks) are corrupted.

Our Contributions.

We present RAID-PIR, a family of multi-server PIR schemes that improve upon and generalize Chor et al.'s PIR protocol [CKGS95] and have several desirable properties. RAID-PIR has a *balanced workload* amongst all participating servers, *reduces the communication complexity*, and uses only highly efficient cryptographic primitives, namely a pseudo-random generator. RAID-PIR has several parameters that allow adaption to the specific deployment scenario and trust assumptions: the number of available servers and the maximum number of servers that can collude. In RAID-PIR, we *reduce the storage requirements for the servers* as each server is comparable to a disk in a striped RAID that stores only a part of the database and computes only a part of the answer to a user's query. The user can even *query multiple blocks of data in parallel* for higher efficiency. Communication and computation requirements are reduced – in particular the upstream communication from the client to the servers. We provide an open-source Python implementation[1] and performance benchmarks for our PIR schemes.

In Section 5, we discuss several deployment aspects, including preserving privacy when reading multiple-blocks concurrently, identifying the necessary blocks for a given query/file, and ensuring object integrity and availability in spite of server failures. Our approach, of building such features on top of RAID-PIR, allows us to maintain a very simple and extremely efficient PIR design, while providing practical solutions to failures, which other designs solve by non-trivial extensions and modifications to the PIR scheme, e.g., [DGH12, OG11, Gol07].

Outline.

We summarize preliminaries in §2. We detail Chor et al.'s PIR protocol and our RAID-PIR protocols in §3 and experimentally compare these protocols in §4. We discuss applications and deployment aspects of RAID-PIR in §5, give related work in §6, and conclude in §7.

2. PRELIMINARIES

In this section we give preliminaries on PIR, explain the multiple-servers PIR scenario (addressed in this work), and introduce notations used throughout this paper.

[1]https://github.com/dd23/RAID-PIR

2.1 Private Information Retrieval

Private Information Retrieval (PIR) is a term introduced by Chor et al. [CKGS95]. It refers to the privacy-preserving querying of data by a client from one or multiple data sources, such that this data sources cannot infer any information about the query. In contrast to the client's query, the available data in the database is public and does not need to be protected from the client. This allows for a trivial solution: Sending the entire database to the client, who then performs his query locally. However, this may be impractical or expensive, for large databases. PIR schemes allow clients to retrieve data without exposing their privacy, and requiring less communication (cf. to sending entire DB), albeit with computational overhead. We focus on *multiple-servers* PIR schemes, which have lower computation complexity.

Multiple Server PIR Scenario.

The content provider $\mathcal{CP}$ provides data and distributes it to k servers $\mathcal{S}_i$ with $1 \leq i \leq k$. This is common practice for distributing data over the Internet in a large scale for the purpose of load balancing and scalability. The direct communication between clients and $\mathcal{CP}$ should be kept to a minimum. A client $\mathcal{C}$ wants to retrieve data from $\mathcal{CP}$ and gets forwarded to several servers that deliver the requested data to him. The PIR protocols we consider here, are single-round protocols where $\mathcal{C}$ sends a query q_i to server $\mathcal{S}_i$ from whom he receives the answer a_i. An example setting with $k = 3$ servers is depicted in Fig. 1.

The properties a PIR scheme must satisfy are that the client $\mathcal{C}$ can recover his desired data (*correctness*), but neither the content provider $\mathcal{CP}$ nor any combination of less than r servers $\mathcal{S}_i$ learns anything about the data the client is interested in from the client's queries (*security*).

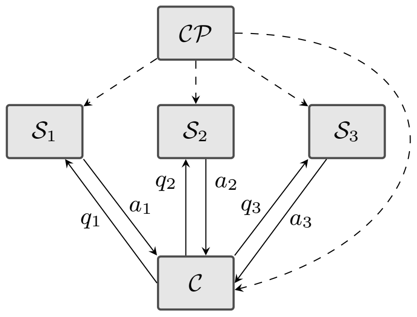


Figure 1: Example PIR Setting with servers $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$. Dashed lines represent offline communication (see §5.2).

2.2 Notations

We denote the content provider as $\mathcal{CP}$, the k servers as $\mathcal{S}_i$ with $1 \leq i \leq k$, and the client as $\mathcal{C}$. The database DB consists of B blocks of size b bits each. The block at position j is denoted as block_j. We partition the B blocks into k *chunks*, each containing $\frac{B}{k}$ blocks of the database.

The protocol makes operations over vectors of size B; in particular, for $c \in [1, B]$, the (B-bits) elementary vector e_c has a single bit set (to one) at position c, and all other bits are reset (zeros). For a given $B-$bit vector α, the notation $\alpha[i]$ refers to the i-th chunk of α, i.e., the $\frac{B}{k}$ bits beginning with $(i-1) \cdot \frac{B}{k}$; cf. §3.2.1.

From §3.2.2, our PIR schemes also use a *redundancy parameter* $2 \leq r \leq k$, s.t. the scheme is secure as long as the number of colluding servers is less than r.

Finally, in §3.2.3, we improve the performance using pseudo-random generator PRG, with security parameter σ. For simplicity, we use $\mathrm{PRG}(s, i)$ to denote the i^{th} 'block' of $\frac{B}{k}$ bits output by PRG for seed $s \in \{0,1\}^\sigma$.

3. PROTOCOL OVERVIEW

Our work is based on Chor et al.'s linear summation PIR system for multiple servers [CKGS95], which we denote as CKGS in the following. Following earlier work, e.g., [Cap13], we query blocks of data instead of single bits to improve efficiency. The overall goal is to allow privacy-preserving retrieval of data and outsourcing of workload from $\mathcal{CP}$ to the PIR servers $\mathcal{S}_i$. After introducing a slightly modified variant of the original CKGS scheme in §3.2.1, we present our improved PIR schemes.

3.1 Setup

Before clients can privately retrieve data, the content provider $\mathcal{CP}$ has to setup the database DB of his files and distribute it to the servers. This is realized by partitioning all available data into B blocks of size b bits and sending them to the servers. The mapping of actual files to blocks is discussed in §4.1.

3.2 Privately Requesting Files

The client $\mathcal{C}$ is interested in one or multiple blocks and wants to request them in a privacy-preserving fashion from the available servers.

3.2.1 A Variant of CKGS [CKGS95]

In the following we describe a slightly modified variant of the original CKGS scheme that has the same properties in terms of complexity, but with the structure of our improved protocols presented in the following sections. For completeness, we give the original CKGS protocol in Appendix B.

As in [CKGS95], the client $\mathcal{C}$ is interested in privately querying block_c and represents this plaintext query as the elementary vector e_c (i.e., a vector of B bits where the c-th bit is set to one and all other bits are zero). Also, the queries received by each server, appear random; only the XOR of all queries equals to the plaintext query e_c. However, differently from [CKGS95], in our variant all servers and queries are 'symmetrical', as follows.

As depicted in Fig. 2, we partition each query into k *chunks*, where k is the number of servers. A chunk, i.e., a column in Fig. 2, spans B/k adjacent *blocks* of the DB and hence contains B/k bits. As shown in Fig. 2, the query sent to server i contains random bits $rnd_i[x]$ for all the chunks $x \neq i$. Chunk i is denoted $flip_i$, and is computed to cancel out all the other (randomly chosen) chunks $\mathsf{rnd}_j[i]$ sent to other server j, leaving exactly $e_c[i]$, i.e.,

$$\mathsf{flip}_i \leftarrow e_c[i] \oplus \bigoplus_{j \in \{1,\dots,k\} \setminus i} \mathsf{rnd}_j[i],$$

where $e_c[i]$ is the i-th chunk of elementary vector e_c, cf. Fig. 2.

Each request q_i that $\mathcal{C}$ sends to server $\mathcal{S}_i$ for $i \in \{1, \dots, k\}$ contains one flip chunk and $k-1$ randomly chosen rnd chunks, and hence has a total length of B bits. The idea of distributing the flip chunks to multiple queries can be seen as analogous to RAID-5 (Rotating Parity), where parity information is distributed over all disks (cf. Appendix A).

As in [CKGS95], the servers' responses have length of b bits each, and are the XOR of all blocks that the user requested in his query, i.e. if the bit at index j was set in the client's query, the server XORs block_j into his response. When the client has received the replies from all k servers he calculates the XOR of all responses and gets block_c, as all other blocks are contained an even number of times and cancel out due to the XOR. See Fig. 2.

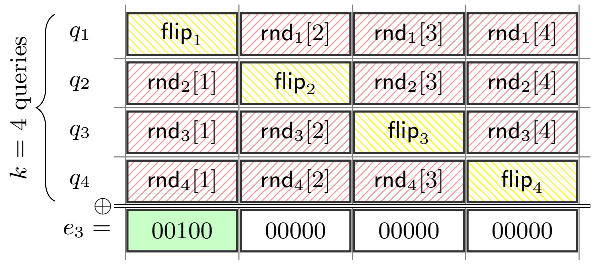


Figure 2: CKGS: The queries q_i sent by the client to servers $\mathcal{S}_i$ and their XOR. Here, $k = 4$ servers and $B = 20$ blocks. The block that the client is interested in has index 3, as the third bit is set to 1.

3.2.2 Using more than r servers

As our first optimization of Chor et al.'s protocol we introduce the redundancy parameter r with $2 \leq r \leq k$ which gives the minimum number of servers that need to collude in order to recover the block that is queried. In our protocol depicted in Fig. 3, the redundancy parameter specifies the number of chunks in each query and how often the chunks overlap throughout all queries, thus setting the storage requirements of the servers and the security of our scheme. Each of the k servers stores only $(r/k) \cdot B$ blocks of the DB, and each query now consists of r chunks, with a length of B/k bits each. A small r parameter allows to reduce the percentage of the DB each server has to store and the length of each query to a fraction of r/k, but also reduces the protection against colluding malicious servers. Hence, the redundancy parameter r allows a trade-off between storage/communication and security and can be chosen in accordance with the deployment scenario and trust assumptions. For $r = k$ we obtain exactly the variant of the original CKGS scheme described in §3.2.1; for better performance, use more servers (with fixed r).

Our idea stems from the striping technique used in RAID systems, where data is distributed over multiple disks in order to improve efficiency. However, to achieve security we have to also rely on mirroring, in order to protect against colluding servers. Our partially overlapping structure of chunks is comparable to a hybrid of RAID-0 (Striping) and RAID-1 (Mirroring), cf. Appendix A. An example of the partitioning into chunks and the use of the redundancy parameter r is depicted in Fig. 3.

3.2.3 SB: Single Block Queries with Seed Expansion

As the next optimization we make use of a pseudo random generator (PRG) to further improve the communication complexity by reducing the size of the queries. The first chunk of each query is chosen as before and sent as a bit

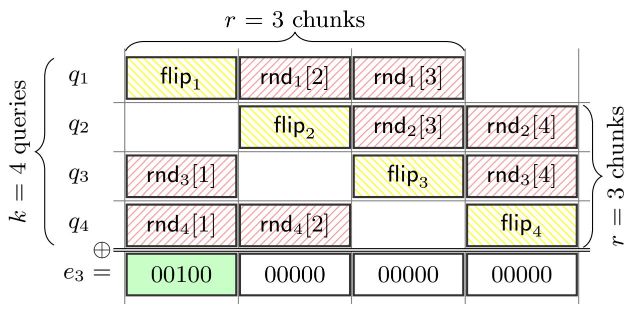


Figure 3: PIR with Redundancy Parameter r: Queries q_i sent by the client to server $\mathcal{S}_i$ and their XOR. Here, $k = 4$ servers, redundancy parameter $r = 3$, and $B = 20$ blocks. The block that the client is interested in has index 3, as the third bit is set to 1.

string, while the remaining $r - 1$ chunks are generated from a PRG and expanded from a seed s of length σ bits, where σ is the symmetric security parameter (set to 128 bit in our implementation). The seed expansion is depicted in Fig. 4.

This technique reduces the communication complexity, as soon as the symmetric security parameter σ is smaller than $B\,(r-1)\,/k$, at the cost of the evaluation of few symmetric cryptographic operations, and is very effective for large databases with a high number of blocks B. The efficiency of the PIR scheme is improved, since typically an end user's upstream is significantly slower than his downstream. Therefore, reducing the amount of data a user has to send to the servers will reduce the overall protocol runtime. We argue that, for large number of blocks B, the additional costs of evaluating a small number of symmetric cryptographic operations, e.g., by instantiating the PRG with AES, are very low compared to the bandwidth savings, especially due to the massive increase of computational power and the availability of the AES-NI instruction set for efficient evaluation of symmetric cryptographic operations. See results in §4.

The servers' replies are identical to the ones in the original protocol. The trick is to flip the bit at the beginning of the query, in the chunk that is not generated by a PRG. All queries start with such a chunk, and therefore all servers are equally likely to receive the query with the flipped bit.

We can combine this technique with the redundancy parameter to further increase efficiency. We refer to the scheme that uses chunks, the redundancy parameter r, and the seed expansion to query a one block as single block scheme SB. A formal description of SB is given in Algorithm 1.

From §3.2.3, our schemes use a *security parameter*

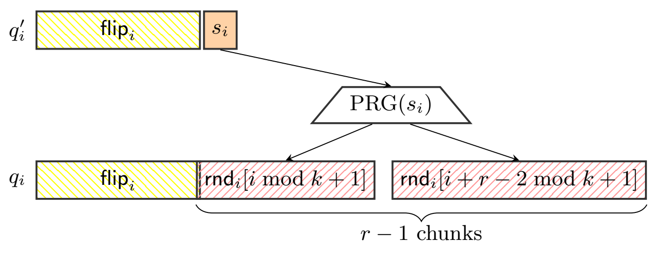


Figure 4: SB: Query expansion from seed s_i.

3.2.4 *MB: Multiple Block Queries*

Finally, we extend the protocol to allow $\mathcal{C}$ to request multiple blocks with a single query. For this, the servers reply with one block per query chunk and calculate the XOR of each response block only within each chunk. The size of the reply from each server to $\mathcal{C}$ is increased from b bits to $r \cdot b$ bits. This approach has the limitation, that the requested blocks must be located in different locations of the DB, as they must be queried in different chunks. Every chunk can contain at most one block of data, comparable to the original scheme, where every block has to be queried separately. However, we argue that the assumptions of blocks being located in different chunks is practical, especially for requests of a large amount of data. An example of the parallel query is depicted in Fig. 5.

We refer to the scheme, that incorporates all of our optimizations and that allows to query up to k blocks with one query as multi block scheme MB. A formal description of MB is given in Algorithm 1.

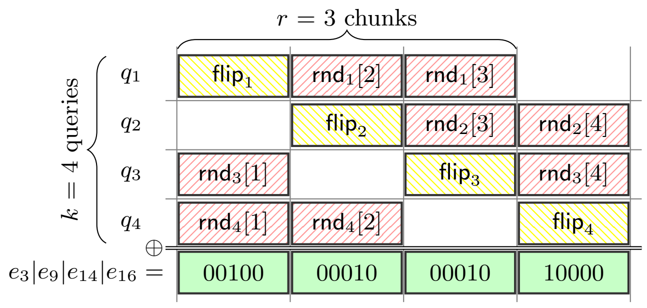


Figure 5: MB: The queries q_i sent by the client to server $\mathcal{S}_i$ and their XOR. Here, $k = 4$ servers, redundancy parameter $r = 3$, and $B = 20$ blocks. The client queries for the blocks 3, 9, 14, and 16 in parallel.

This optimization is again inspired by RAID, since blocks can be read from multiple disks in parallel. However, similar to RAID-5 (Rotating Parity) and RAID-0 (Striping), blocks can only be queried in parallel if they are located on different disks, cf. Appendix A.

3.3 Analysis

In the following we analyze the complexity, correctness, and security of our PIR schemes.

3.3.1 *Complexity and Efficiency*

The complexities and efficiency of the PIR schemes described in §3.2 are summarized in Tab. 1.

Our improved PIR protocols are well-suited for cloud-based applications where customers are charged for server computation and communication to/from the cloud. In contrast to the original Chor protocol [CKGS95], where multiple servers are used only to increase the security of the protocol, our protocols can use multiple servers for better efficiency, similar to the properties of a RAID system: for k servers, the upload communication from the client to each server is only about $1/k$-th of the original communication and when using redundancy parameter $r = 2$, each server loads and processes only about $1/k$-th of the blocks in the database.

We note that queries are sent to the servers concurrently with responses sent back from the servers; hence, the commu-

Algorithm 1 Description of our PIR schemes for retrieving a single block (SB) (cf. §3.2.3) or multiple blocks in parallel (MB) (cf. §3.2.4). See notations in §2.2.

```
Input: (SB) single block $\mathsf{block}_c$ or (MB) $k$ blocks $\mathsf{block}_c[i]$ for $i \in \{1, \ldots, k\}$

  1. $\mathcal{C}$ randomly picks the seeds $s_i \in_R \{0,1\}^\sigma$ for $i \in \{1, \ldots, k\}$.
  2. $\mathcal{C}$ expands each seed $s_i$ to generate the $r-1$ chunks $\mathsf{rnd}_i[j] \leftarrow \mathrm{PRG}(s_i, j)$ for $j \in \{(i \bmod k)+1, \ldots, (i+r-2 \bmod k)+1\}$.
  3. For each $i$, $\mathcal{C}$ sets the chunk $\mathsf{flip}_i$ as the XOR of the $r$ chunks $\mathsf{rnd}_j[i]$ in column $i$:
     $\mathsf{flip}_i \leftarrow \bigoplus_j \mathsf{rnd}_j[i]$ with $j = (i-1 \mod k)+1, (i-2 \mod k)+1, \ldots$.
  4. (SB) $\mathcal{C}$ identifies the $\mathsf{flip}$ chunk that contains $\mathsf{block}_c$ (the block $\mathcal{C}$ is interested in) and flips the bit at the corresponding position.
     (MB) $\mathcal{C}$ identifies all $\mathsf{flip}$ chunks that contain a $\mathsf{block}_c[i]$ he is interested in and flips the bit at the corresponding positions.
  5. $\mathcal{C}$ sends the queries $q'_i$ consisting of one $\mathsf{flip}$ chunk and one seed $s_i$ to the servers $\mathcal{S}_i$.
  6. $\mathcal{S}_i$ expands his seed $s_i$ to generate $r-1$ random chunks $\mathsf{rnd}_i[j] = \mathrm{PRG}(s_i, j)$ for $j \in \{(i \bmod k)+1, \ldots, (i+r-2 \bmod k)+1\}$ and gets his full query $q_i$, as depicted in Fig. 4.
  7. (SB) $\mathcal{S}_i$ calculates his answer $a_i \leftarrow \bigoplus_{x \in q_i} \mathsf{block}_x$ and sends answer $a_i$ to $\mathcal{C}$.
     (MB) For each of the $r$ chunks $j$, $\mathcal{S}_i$ calculates $a_i[j] \leftarrow \bigoplus_{x \in q_i[j]} \mathsf{block}_x$ and sends all $a_i[j]$ blocks to $\mathcal{C}$.
  8. (SB) $\mathcal{C}$ calculates the plaintext block by XORing the $k$ answers: $\mathsf{block}_c \leftarrow \bigoplus_{1 \le i \le k} a_i$.
     (MB) For each chunk $j$, $\mathcal{C}$ calculates the plaintext block by XORing the $r$ answers in column $j$ :
          $\mathsf{block}_c[j] \leftarrow \bigoplus_i a_i[j]$ with $i = (j-1 \mod k)+1, (j-2 \bmod k)+1, \ldots$.
```

Table 1: Comparison of complexity and efficiency of the original Chor PIR scheme with our optimizations from §3.2. k: #servers, r: redundancy parameter ($2 \le r \le k$), B: #blocks, b: block size, σ: security parameter.

	[CKGS95]	§3.2.2	SB §3.2.3	MB §3.2.4
Upload: Query length $\lvert q_i \rvert$ [bit]	B	$(r/k) \cdot B$	$(1/k) \cdot B + \sigma$	$(1/k) \cdot B + \sigma$
Download: Answer length $\lvert a_i \rvert$ [bit]	b	b	b	$r \cdot b$
Server Computation [#blocks to load and XOR]	$B/2$	$(r/k) \cdot B/2$	$(r/k) \cdot B/2$	$(r/k) \cdot B/2$
Server Storage [#blocks]	B	$(r/k) \cdot B$	$(r/k) \cdot B$	$(r/k) \cdot B$
Client Computation [#pseudorandom bits]	$(k-1) \cdot B$	$(r-1) \cdot B$	$(r-1) \cdot B$	$(r-1) \cdot B$
Result blocks per query	1	1	1	k
Security Level (max. #colluding servers)	$k-1$	$r-1$	$r-1$	$r-1$
Maximum communication efficiency	$b/(kB+kb)$	$b/(rB+kb)$	$b/(B+k\sigma+kb)$	$b/(B/k+\sigma+b)$

nication delay is essentially the maximum, rather than the sum, of the upload (query) and download (response) times.

3.3.2 *Correctness*

Our proposed PIR schemes are adaptations of the original CKGS scheme. The main difference is that we do not compute the XOR over the entire database, but over smaller chunks (column-wise). Therefore, correctness carries over from the original CKGS scheme [CKGS95].

3.3.3 *Security*

The security of the schemes of §3.2.1, §3.2.2 follows trivially using the same proof as of the original scheme, see [CKGS95]. Intuitively, for each column i, the flip_i chunk is the XOR of $r-1$ random (rnd) values. Therefore, these flip chunks can be seen as an r-out-of-r XOR-based secret sharing of either the zero-vector or an elementary vector with the single bit set to one in which the client is interested. From the security of the secret sharing scheme follows that a collusion of up to $r-1$ servers cannot gain any information about the block the client is interested in.

The security of the SB scheme, of §3.2.3, follows by standard reduction to the security of the PRG used. Namely, if the SB scheme is leaks information, then we can distinguish between the strings produced by the PRG vs. truly random strings of the same length, by running the protocol with the given bits and checking if the attacker can learn information (which is proven impossible using truly random bits, see above and in [CKGS95]).

The security of the MB scheme, of §3.2.4, also follows by a simple reduction argument, this time reducing to the SB scheme. Namely, assume an attacker can leak information from the MB scheme (but not from the SB scheme); for simplicity, assume this holds for the case shown in Fig. 5. Then the attacker runs four queries against the SB scheme, each time for block from a different chunk, and then XORs the views received by the corrupt servers; this is equivalent to a single run of the MB algorithm, so we can use the attack on MB to leak information - which contradicts the security of the SB scheme. Note that for convenience, we ignored here the fact that both SB and MB use pseudorandom strings (which can be easily dealt with as explained in previous paragraph).

Note that to additionally achieve security against external man-in-the-middle attackers, the client can connect to the servers via secure channels.

3.4 Further Improvements

The implementations of all PIR schemes described in §3.2 can be further improved as detailed next.

Precomputation at Servers.

It is possible to reduce the computational complexity for the servers at the cost of additional storage as proposed in [BIM00]. For this, each server precomputes all possible combinations of XOR for subsets of length ℓ of the data and stores the results on disk. This technique reduces each server's computational workload by a factor of ℓ while increasing its storage by a factor of $2^\ell/\ell$.

Memory Efficient PIR.

The PIR schemes can be implemented with low memory for the client. This makes them well-suited for resource-limited devices, such as web browsers, smartphones, embedded systems, or smartcards. For this, we instantiate the PRG with a block cipher in counter mode and iteratively compute the queries column-wise and chunk by chunk.

4. IMPLEMENTATION & BENCHMARKS

We base our implementations on the publicly available code of upPIR [Cap13] which implements the original protocol of Chor et al. [CKGS95]. We modified the existing Python code and implemented our improved PIR protocols described in §3.2: support for redundancy parameter r (§3.2.2), smaller upload by using a PRG for the queries (SB – §3.2.3), and enabling parallel requests (MB – §3.2.4). We set the security parameter to $\sigma = 128$ and instantiate the PRG with AES128-CTR. In §4.2 we compared performance of our different variants to that of [CKGS95], and in §4.3 we compared to the best-effcient robust PIR algorithm (to the best of our knowledge), [Gol07], using the implementation in [GDL$^+$14].

4.1 System Description

The upPIR system has two phases, a setup phase run between the content provider and the servers, and a PIR phase run between the client and the servers as described next.

Setup Phase.

The content provider $\mathcal{CP}$ sets up the database DB of his files by partitioning all available data into B blocks of size b. For this, $\mathcal{CP}$ creates a *manifest file* that maps every file to one or multiple blocks. Every block has a unique index and its content can optionally be hashed with a secure hash function in order to guarantee its integrity and thereby allowing to identify malicious servers. The manifest file is static and identical for all clients and servers. The partitioning is done in a compact way, such that files can start in the middle of a block, thus wasting no storage space. The files to be distributed and the manifest file are sent to all servers.

PIR Phase.

The client $\mathcal{C}$ requests the manifest file in order to determine what data is available and for creating his query to the servers. With the manifest file, he identifies the blocks he is interested in and runs the PIR protocol with the servers (cf. §3.2).

4.2 Benchmark Results

We benchmark our PIR schemes for different parameters and deployment scenarios. We deploy the k servers as `m3.large` instances on Amazon EC2. The client is a local desktop computer with 4 GB RAM and an AMD hexa-core CPU with 3.3 GHz. As DB we use a recent release of Ubuntu security updates, containing 964 updates adding up to a total size of 3.8 GB. The average file size of the DB is 4 MB, while the median file size is only 267 kB, as many small patches are contained. We run each experiment 5 times and give the average runtimes.

Varying Block Size.

In our first set of experiments, shown in Fig. 6, we fix the number of servers to $k = 3$ and the redundancy parameter to $r = 2$. We vary the block size b from 16 kB to 4 MB depicted on the x-axis and plot the total runtime for the different PIR schemes on the y-axis (CKGS §3.2.1, $r = 2$ §3.2.2, SB §3.2.3, and MB §3.2.4). We use two different network

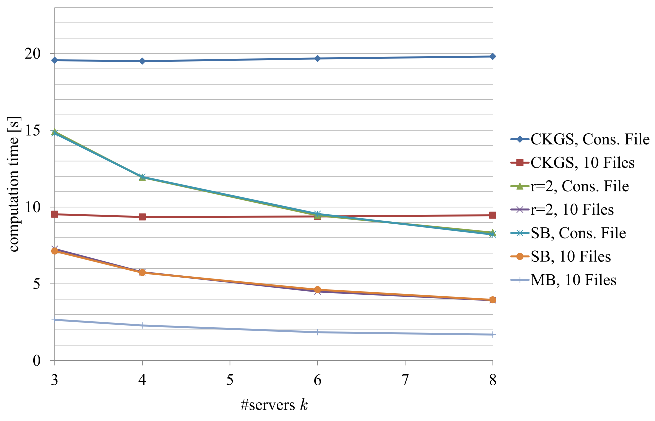


Figure 8: Server computation time [s] for varying number of servers k, redundancy parameter $r = 2$, and block size $b = 128$ kB.

settings between the client and EC2: a consumer grade DSL connection (1 MBit/s upstream and 16 MBit/s downstream) depicted in the left diagrams (Fig. 6(a) and Fig. 6(c)) and a WAN connection (350 MBit/s for both up- and downstream) depicted in the right diagrams (Fig. 6(b) and Fig. 6(d)). The latency averaged to 30 ms for both network settings. In the upper diagrams (Fig. 6(a) and Fig. 6(b)) we query for a large file that is stored consecutively in the DB (file size 8.5 MB) and in the lower diagrams (Fig. 6(c) and Fig. 6(d)) we retrieve 10 small files (total size of files 2.9 MB) that are distributed across different chunks of the database. Throughout all measurements the same 10 small files are used to achieve comparable results.

As can be seen from the diagrams, our improved PIR protocols result in larger runtime improvements when the upstream of the network connection is limited (left diagrams for DSL) as the client sends less data (cf. Upload in Tab. 1). Our multi-block PIR scheme MB described in §3.2.4 is beneficial when querying for multiple files distributed across multiple chunks (lower diagrams for 10 files).

The computation improvements for the servers cannot be seen clearly in these experiments as for parameters $r = 2$ and $k = 3$ the servers have in our protocols only 2/3 of the workload of the original Chor PIR scheme (cf. Server Computation in Tab. 1). Therefore, we vary the number of servers k in the following experiments.

Server Computation Workload.

Next we measure the total time that one server needs to respond to a client's query for either the large consecutive file or the 10 small files that are distributed throughout the DB. This server computation time includes expanding the seed (for SB and MB), reading the blocks, and calculating their XOR. The results are depicted in Fig. 8. Our schemes benefit from an increasing number of servers k as the computation time for each server decreases, whereas the original CKGS scheme has a constant server workload that is independent of k. The runtimes for $r = 2$ and SB are almost identical (for Cons. File and 10 Files, respectively), showing that the seed expansion with a PRG has negligible computation overhead.

Varying Number of Servers.

Finally, we vary the number of servers and measure the total runtime for the entire PIR protocol, including network communication. From the results depicted in Fig. 7 we observe that increasing k only improves the total runtime if the network bandwidth is high enough. For the DSL scenario the higher communication complexity, caused by the downstream from each server, also increases the overall runtime. For the WAN scenario the performance increase is clearly visible, but limited due to the increased communication and computation requirements for the client. The runtime for CKGS increases slightly, because the client has to wait for all k servers to respond and latency in the cloud can vary.

4.3 Comparison with [Gol07]

We evaluate the recent implementation of [GDL+14, Gol07] on the same machines and data that we used to benchmark our code and depict the results in Fig. 9. All results are for a WAN network connection, therefore the results in Fig. 9(b) and Fig. 6(d) are comparable. We show results for their implementation of CKGS, which achieves very similar performance as our CKGS implementation. We compare it with a version of their protocol that allows to query one block per query sequentially and a version that allows to query $k - 1$ blocks per query in parallel. The resulting runtimes are slower than the plain CKGS protocol. However their implementation scales very well with increasing k, and for large number of servers, may be competitive with ours; more significantly, [Gol07] ensure robustness against malicious servers, which we only address by the higher-layer mechanisms of §5.3.

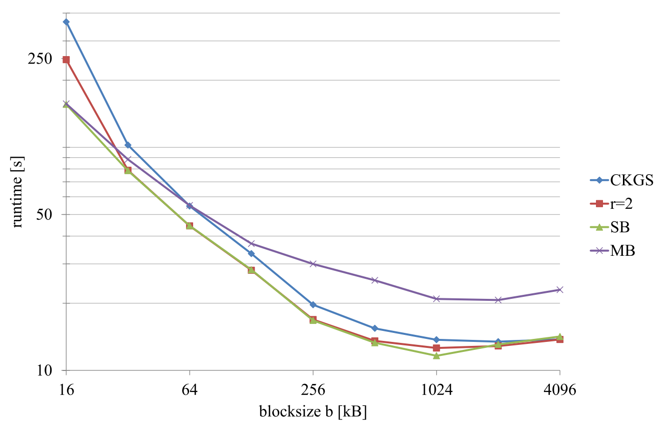
5. APPLYING RAID-PIR

PIR protocols can be used for different applications and scenarios, where clients wish to hide the storage locations they are reading. However, the applications may have additional requirements, beyond these provided by basic PIR schemes. Some of these requirements may require significant extensions to existing PIR schemes, or a new scheme; for example, several works study PIR schemes which are robust to benign and/or byzantine failures [Gol07, DGH12, DG14]. Addition of such properties to RAID-PIR is subject for further research.

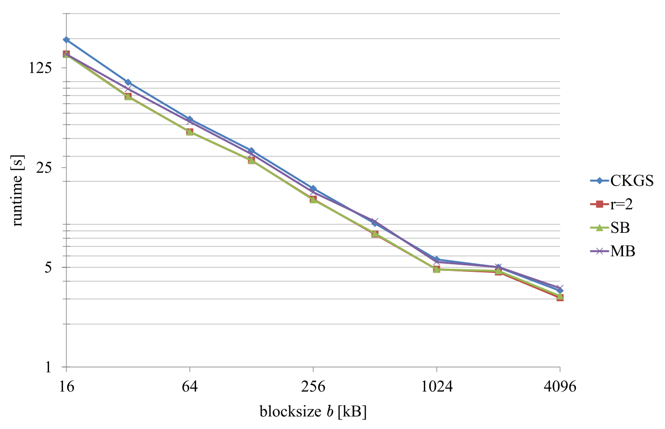
However, as we show below, some requirements can be achieved quite easily, by extending RAID-PIR or adding a layer on top of it. In the following, we discuss private retrieval of multi-block objects, PIR lookup mechanisms (identifying the block(s) necessary for a particular query/object), and finally object integrity, availability and accountability.

5.1 Private Multi-Block Object Retrieval

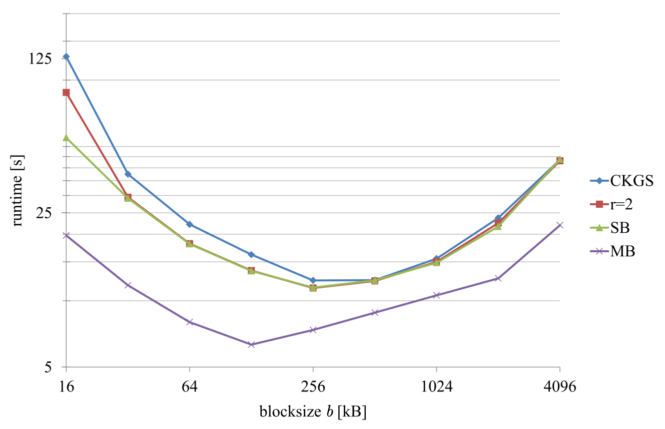
PIR schemes allow customers to retrieve an object, without allowing the servers to know which object was retrieved among the set of objects stored; in the case of multi-server PIR schemes such as RAID-PIR, this holds as long as the number of corrupt servers is less than r. However, many applications involve a collection of objects with significantly different lengths. In this case, using a block size equal to the object size involves significant communication and computation overhead. To reduce this overhead, we would normally use a smaller block size, s.t. retrieving a large object would imply retrieval of all the blocks in the objects. However, when using such a scheme, the number of blocks retrieved



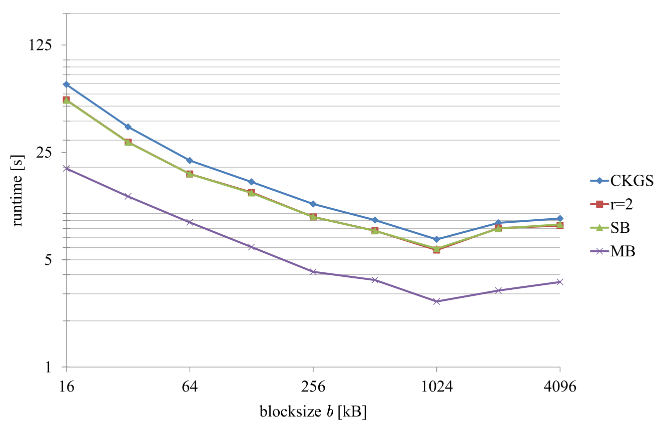
(a) DSL, Large Consecutive File



(b) WAN, Large Consecutive File

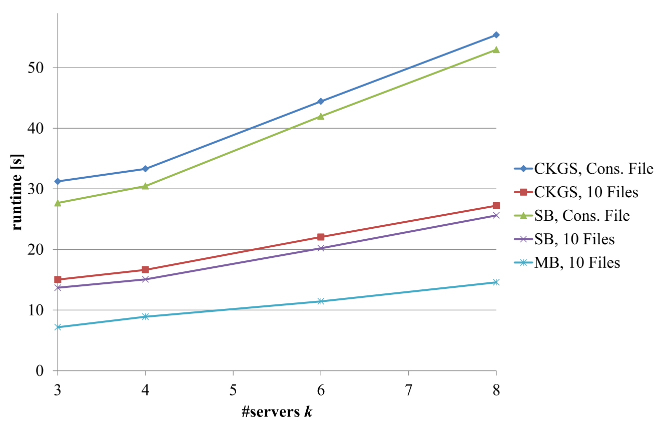


(c) DSL, 10 Small Files

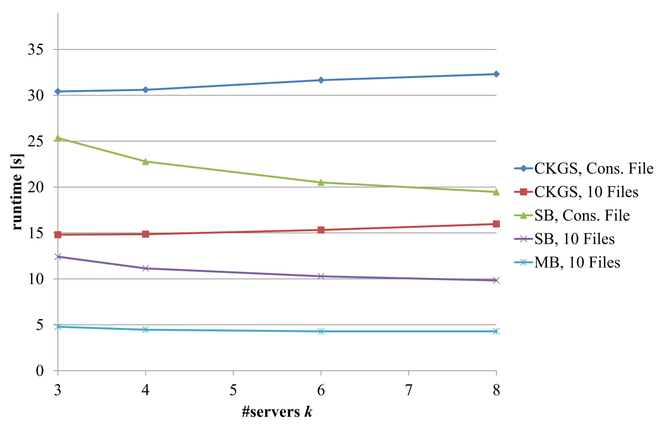


(d) WAN, 10 Small Files

Figure 6: Runtimes [s] for varying block sizes b [kB] with $k = 3$ servers and redundancy parameter $r = 2$.

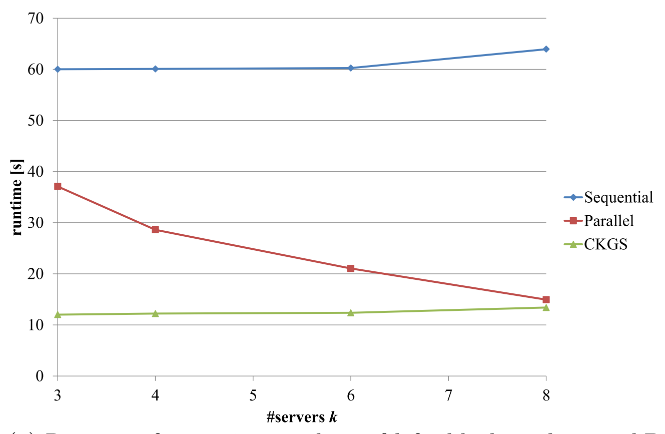


(a) DSL

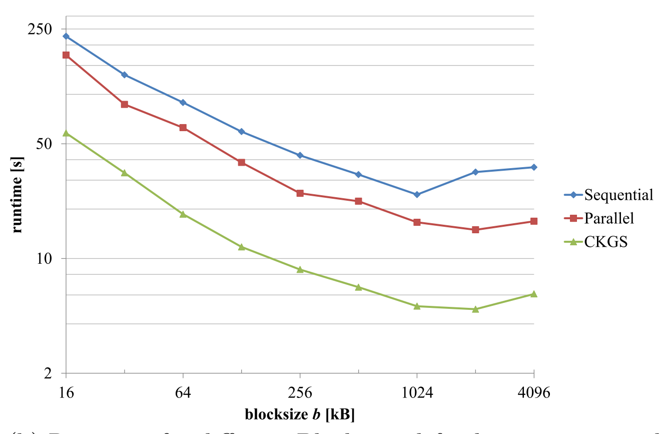


(b) WAN

Figure 7: Runtimes [s] for varying number of servers k, redundancy parameter $r = 2$, and block size $b = 128$ kB.



(a) Runtimes for varying numbers of k for blocksize $b = 128$ kB and the query of 10 random files.



(b) Runtimes for different Blocksizes b for $k = 3$ servers and the query of 10 random files.

Figure 9: Runtimes [s] for [Gol07] for varying number of servers k and different block sizes b.

may allow the attacker to identify the object (if it has unique number of blocks) or to know that the object is within the limited set of objects with the same number of blocks.

A trivial way to deal with this concern is to use very large blocks, such that every object will fit within a single block. Alternatively, clients may always retrieve a fixed number of blocks, including blocks they are not interested in, possibly using the parallel block query (§3.2.4) to reduce the overhead; by using a smaller block size, this design is less wasteful in storage, although it has the same communication cost as the use of blocks of the same total size. Hence, these solutions fix the privacy concern, at the cost of more communication.

In many scenarios, an alternative would be to address the privacy concern by increasing latency. Specifically, if a system has some known distribution of requests, say in rate of β blocks per second (taking into account the total number of blocks in requested objects), then a possible way to hide identity of requested objects would be to request objects at fixed rate of β blocks per second (or a bit higher, to avoid excessive latency), delaying requests if necessary and adding 'dummy' requests if none are present. Again, this combines well with the parallel block query (§3.2.4), allowing to retrieve k blocks with each request, improving request (upstream) bandwidth and processing overhead. Further optimizations may be possible, especially when the distribution of requests and of object-sizes is known or has certain properties.

5.2 PIR Lookup Mechanisms

PIR schemes allow clients to retrieve a block by specifying the block number, without exposing the block number to eavesdroppers or to the PIR servers. However, in many applications, clients do not necessarily know in advance the block number containing the information they need; instead, they have some higher-layer identifier such as a URL or file name. How can clients learn the block number(s) corresponding to the object identifier without exposing their interest in a particular object?

One solution is used in upPIR [Cap13]: the content provider maintains a *manifest file*, mapping all object identifiers to the corresponding block numbers. Clients always retrieve the entire manifest file (from the origin or any server, or even from another location). The manifest file can also allow to ensure integrity by including a collision-resistant hash of each object, and the origin signs the manifest file itself.

This use of such manifest file is fine, as long as its size is reasonable; it may be less appropriate to ensure privacy for queries to a huge collection of many (smaller) items, such as the Domain Name System, as proposed, e.g., in [ZHS07]; in such cases, the distribution of the manifest file may cause significant overhead, esp. if it has to be updated periodically.

In such cases, where distributing a manifest file causes significant overhead, it may be better to use other approaches. A simple approach is to store an object with identifier (URL) i, in block $h(i)$ where h is a hash function; this has obvious limitations, in particular, no control over placement. More elaborate schemes for object lookup in PIR based on keywords or identifiers were studied, e.g., [CGN98].

5.3 Object Integrity, Availability and Accountability

PIR schemes do not necessarily ensure integrity. Namely, a benign or malicious (byzantine) failure in one or more of the servers, may result in clients receiving corrupt data. Some PIR schemes are *robust*, i.e., designed to handle benign failures, or even byzantine failures, e.g., [DG14, DGH12]. Our RAID-PIR constructions, however, are not robust to server failures; it is an interesting challenge to adopt them to achieve robustness (with comparable performance). However, we note that clients may easily recover from failure of few servers, by using only the operative servers in the protocol. We now show how to take advantage of this and use RAID-PIR to ensure integrity, availability and accountability.

We first explain how to ensure object integrity: one natural solution is for the origin to sign every object, and to provide the signature as part of the object from the servers. Alternatively, when a manifest file is used, the origin can only sign the manifest file, and include therein a collision-resistant hash value for each object (cf. [Cap13]). In some applications, the content 'belongs' to a specific customer of the origin; in this case, the origin may use a Message Authentication Code (MAC), for improved efficiency (cf. to signature schemes).

The above mechanisms only allow the client to detect when it receives a corrupted object; this still allows one corrupt server to deny service to the PIR. However, we can efficiently deal with this threat, as follows: First, we assume that all the communication between client and each server is protected, and in particular authenticated (typically using a MAC). This allows the client to ignore servers to which there are repeated communication failures. We therefore focus on servers corruptions (and assume that all messages are received correctly as sent).

Second, we assume that the client uses some of the mechanisms above, e.g., signed objects, to detect corrupted objects. It remains to explain how we deal with this case, i.e., client receiving corrupted objects due to a server failure. In this case, the client proceeds with a *resolution protocol*, which identifies the corrupted server, as follows.

(A) The client simply re-sends the same query to the servers; however, this time it indicates that the responses should be *signed*. If the client detects that the server failed, by not returning the same response as before (or not returning a valid signature), then the client marks this server as 'bad' and continues the protocol with the remaining servers. The signature is over both the request received by the server and over the response.

(B) Once the client receives the signatures from all the (operative) servers, then it selects one or more of the servers, and sends the corresponding signed responses from these servers to the content origin or a trusted third party. The origin can validate that the response is valid; if not, it would blacklist that server so that all clients will stop using it.

If clients wish to preserve query privacy from the origin, then they should send less than r of the responses from the servers to the origin (each containing the corresponding request that the client sent to that server). However, surely, after few such interferences, a faulty server would certainly be detected.

6. RELATED WORK

Below we summarize related work for PIR with multiple servers or a single server, and oblivious RAMs.

Multi-server PIR.

The first work on PIR by Chor et al. [CKGS95] introduced information theoretically secure PIR in a setting with multiple servers. [Gol07] proposes multi-server PIR schemes with stronger robustness properties than our protocols, but these protocols are more complicated and use computationally more demanding cryptographic primitives (Shamir secret sharing or even homomorphic encryption). An experimental comparison of the multi-server PIR schemes of [CKGS95] and [Gol07] was given in [OG11]. [Cap13] implemented Chor et al.'s PIR scheme [CKGS95] for different block sizes in order to efficiently and privately retrieve security updates with similar performance as downloading the file via FTP. A robust multi-server PIR scheme that allows multi-block queries was introduced in [HHG13]. Efficiency of robust multi-server PIR was improved in [DGH12, DG14]. Multi-server PIR with verifiability was proposed in [ZSN14].

Single-Server PIR.

Private Information Retrieval with a single computationally bounded server was proposed in [KO97] and is often referred to as Computationally Private Information Retrieval (CPIR). Since then, several CPIR schemes have been proposed, e.g., with polylogarithmic communication [CMS99]; a survey of several CPIR schemes is given in [OI07]. In [CMO00] it was shown that CPIR implies Oblivious Transfer which gives strong evidence that CPIR cannot be constructed based on weak computational assumptions such as one-way functions. [SC07] show that non-trivial CPIR protocols implemented on standard PC hardware are orders of magnitude less time-efficient than trivially transferring the entire database. A lattice-based CPIR scheme was proposed in [MG08] and experiments in [OG11] demonstrate that this scheme can be more efficient than downloading the database. By using a trusted hardware token, the computational assumptions for CPIR can be circumvented and information theoretic security can be achieved, e.g., as shown in [WDDB06, YDDB08, DYDW10]. In [MBC13b] it was shown how to exploit the massive parallelism available in cloud computing to split the server's workload on multiple machines using MapReduce. A CPIR scheme with multiple queries was given in [GKL10]. [DG14] constructs a hybrid CPIR protocol that combines the multi-server PIR protocol of Goldberg [Gol07] with the single-server CPIR protocol of Melchor and Gaborit [MG08], for security even if *all* servers are corrupted.

Oblivious RAM.

Oblivious RAM (ORAM) [GO96] is more powerful than PIR as it allows not only private *retrieval* of data, but also private *write-access*. Burst ORAM [DSS14] allows efficient online requests through pre-computation. A combination of ORAM and PIR was presented recently in [MBC13a].

7. CONCLUSION AND FUTURE WORK

We presented RAID-PIR, a family of simple and practical multi-server PIR schemes, which are efficient in computation, storage and communication, especially upload from clients. Due to its simplicity and efficiency, RAID-PIR can be used by practical systems. We evaluated the efficiency of RAID-PIR, however, we plan to further extend the evaluation and report in full version of this work; in particular, use of larger database as well as of shorter blocks, may be more realistic, and also would show better the advantage of the PRG technique of §3.2.3.

RAID-PIR does not provide built-in robustness against faulty servers, however, we show (in §5.3) how to efficiently use it to handle erroneous or malicious servers. It would be an interesting challenge, to find another RAID-PIR mode, that will provide fault-tolerance, this may also help speed up results (by not waiting for response from slowest server). Of course, the challenge is to maintain the RAID-like simplicity and efficiency.

Acknowledgements.

We thank the anonymous reviewers of ACM CCSW 2014 for their helpful comments on our paper. We also thank the authors of [Cap13] for making their implementation publicly available and the authors of [GDL$^+$14] for sharing their code with us. Thanks also to Ian Goldberg for clarifying the right algorithm for our evaluation.

This work was supported by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, by the Hessian LOEWE excellence initiative within CASED, by the Israeli ministry of science, technology and space, by the Israeli Cyber-security department of prime minister office, by the Israel Science Foundation (grant 1354/11), and by the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE). Part of this work was done while the second author visited Saarland university.

8. REFERENCES

[BDG14] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. Technical Report 2014-10, Centre for Applied Cryptographic Research (CACR), University of Waterloo, May, 2014.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In *Advances in Cryptology – CRYPTO'00*, volume 1880 of *LNCS*, pages 55–73. Springer, 2000.

[BKOI07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Advances in Cryptology – CRYPTO'07*, volume 4622 of *LNCS*, pages 50–67. Springer, 2007.

[Cap13] Justin Cappos. Avoiding theoretical optimality to efficiently and privately retrieve security updates. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, pages 386–394. Springer, 2013.

[CGN98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Cryptology ePrint Archive, Report 1998/003, 1998. http://eprint.iacr.org/1998/003.

[CKGS95] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science (FOCS'95)*, pages 41–50. IEEE, 1995.

[CMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology – EUROCRYPT'00*, volume 1807 of *LNCS*, pages 122–138. Springer, 2000.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 402–414. Springer, 1999.

[DG14] Casey Devet and Ian Goldberg. The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency. In *Privacy Enhancing Technologies Symposium (PETS'14)*, volume 8555 of *LNCS*, pages 63–82. Springer, 2014.

[DGH12] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *USENIX Security'12*, pages 269–283. USENIX, 2012.

[DSS14] Jonathan Dautrich, Emil Stefanov, and Elaine Shi. Burst ORAM: Minimizing ORAM response times for bursty access patterns. In *USENIX Security'14*. USENIX, 2014.

[DYDW10] Xuhua Ding, Yanjiang Yang, Robert H. Deng, and Shuhong Wang. A new hardware-assisted PIR with O(n) shuffle cost. *International Journal of Information Security*, 9(4):237–252, 2010.

[GDL$^+$14] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ project on SourceForge. http://percy.sourceforge.net, 2014. Version 1.0.0. Pre-release version supplied by the authors.

[GHS14] Yossi Gilad, Amir Herzberg, and Michael Sudkovitch. CDN-on-Demand: Fighting DoS with Untrusted Clouds. Work in progress, 2014.

[GKL10] Jens Groth, Aggelos Kiayias, and Helger Lipmaa. Multi-query computationally-private information retrieval with constant communication rate. In *Public Key Cryptography (PKC'10)*, volume 6056 of *LNCS*, pages 107–123. Springer, 2010.

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

[Gol07] Ian Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy (S&P'07)*, pages 131–148. IEEE, 2007.

[HHG13] Ryan Henry, Yizhou Huang, and Ian Goldberg. One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In *Network and Distributed System Security (NDSS'13)*. The Internet Society, 2013.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science (FOCS'97)*, pages 364–373. IEEE, 1997.

[MBC13a] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Path-PIR: Lower worst-case bounds by combining ORAM and PIR. Cryptology ePrint Archive, Report 2013/086, 2013. http://eprint.iacr.org/2013/086.

[MBC13b] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: Efficient private information retrieval for MapReduce. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, pages 371–385. Springer, 2013.

[MG08] Carlos Aguilar Melchor and Philippe Gaborit. A fast private information retrieval protocol. In *IEEE International Symposium on Information Theory (ISIT'08)*, pages 1848–1852. IEEE, 2008.

[MOT$^+$11] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In *USENIX Security'11*. USENIX, 2011.

[OG11] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography and Data Security (FC'11)*, volume 7035 of *LNCS*, pages 158–172. Springer, 2011.

[OI07] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In *Public Key Cryptography (PKC'07)*, volume 4450 of *LNCS*, pages 393–411. Springer, 2007.

[PGK88] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM International Conference on Management of Data (SIGMOD'88)*, pages 109–116. ACM, 1988.

[SC07] Radu Sion and Bogdan Carbunar. On the practicality of private information retrieval. In *Network and Distributed System Security (NDSS'07)*. The Internet Society, 2007.

[SCM05] Len Sassaman, Bram Cohen, and Nick Mathewson. The pynchon gate: A secure method of pseudonymous mail retrieval. In *ACM Workshop on Privacy in the Electronic Society (WPES'05)*, pages 1–9. ACM, 2005.

[WDDB06] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In *European Symposium on Research in Computer Security (ESORICS'06)*, volume 4189 of *LNCS*, pages 49–64. Springer, 2006.

[YDDB08] Yanjiang Yang, Xuhua Ding, Robert H. Deng, and Feng Bao. An efficient PIR construction using trusted hardware. In *Information Security Conference (ISC'08)*, volume 5222 of *LNCS*, pages 64–79. Springer, 2008.

[ZHS07] Fangming Zhao, Yoshiaki Hori, and Kouichi Sakurai. Two-servers PIR based DNS query scheme with privacy-preserving. In *Intelligent Pervasive Computing, 2007. (IPC'07)*, pages 299–302. IEEE, 2007.

[ZSN14] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable multi-server private information retrieval. In *Applied Cryptography and Network Security (ACNS'14)*, volume 8479 of *LNCS*, pages 62–79. Springer, 2014.

APPENDIX

A. RAID

In the following, we summarize commonly used RAID levels from which we borrow ideas for our RAID-PIR schemes. The RAID levels are depicted in Fig. 10.

RAID-0 (Striping): This RAID level stripes blocks over multiple disks, e.g. in Fig. 10, block 0 is stored on disk 0, block 1 is stored on disk 1, etc.. This allows to improve read performance for parallel reads, e.g. in Fig. 10, a read for block 0 and block 1 can be processed in parallel by disk 0 and disk 1. However, if a disk fails, all blocks stored on it cannot be recovered.

RAID-1 (Mirroring): This RAID level mirrors blocks over multiple disks, i.e., each block is stored on multiple disks. This increases redundancy s.t. when one disk fails, data can be recovered from the other disk(s). It also improves read performance as blocks can be read from multiple disks.

RAID-4 (Parity): In this RAID level, one disk stores parity information about all other disks. Due to its simplicity XOR is usually used as parity scheme, e.g. in Fig. 10, $P_{01} = B_0 \oplus B_1$. In case of a failure of one disk the information can be recovered from the remaining disks. The performance is limited by the parity disk.

RAID-5 (Rotating Parity): This RAID level stripes data blocks across multiple disks and for each stripe of data blocks a parity block is stored on the disks in a rotating manner. The parity blocks are computed as the XOR of the blocks in one stripe. On failure of one of the disks, its data can be recovered from the data and parity blocks stored on the other disks.

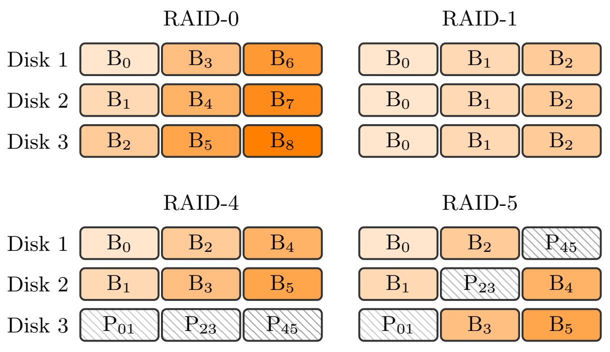


Figure 10: Distribution of blocks in RAID levels 0, 1, 4, and 5 with three disks. B_i denotes a data block, P_{ij} denotes a parity block.

B. THE CKGS SCHEME [CKGS95]

This is a description of the original linear summation PIR scheme by Chor et al. [CKGS95] that our ideas are based on. The client $\mathcal{C}$ is interested in privately querying block_c. The request q_i that $\mathcal{C}$ sends to server $\mathcal{S}_i$ is a randomly chosen string of B bits for $i \in \{1, \ldots, k-1\}$. The k-th request q_k corresponds to the XOR of all other requests except for one bit flipped at the index of block_c. The result of the XOR of all requests is the elementary vector e_c with length B bits that has a 1 in position c and 0 everywhere else. The servers' responses have a length b bits each and are the XOR of all blocks that the user requested in his query, i.e. if the bit at index j was set in the client's query, the server XORs the block_j into his response. When the client has received a reply from all servers he calculates the XOR of all k responses and gets block_i, as all other blocks are contained an even number of times and cancel out due to the XOR. An example of this scheme is depicted in Fig. 11.

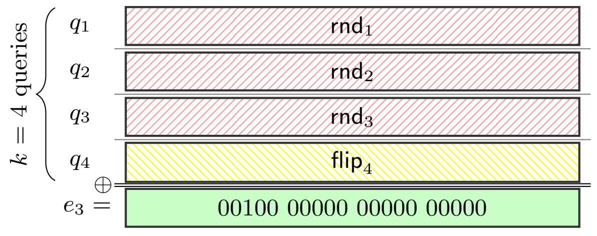


Figure 11: CKGS with $k = 4$ servers and $B = 20$ blocks.