Industrial Feasibility of Private Information Retrieval

Angela Jäschke[1], Björn Grohmann[2], Frederik Armknecht[1], and Andreas Schaad[2]

[1] University of Mannheim, Germany
{jaeschke, armknecht}@uni-mannheim.de
[2] Huawei German Research Institute

Abstract. A popular security problem in database management is how to guarantee to a querying party that the database owner will not learn anything about the data that is retrieved — a problem known as Private Information Retrieval (PIR). While a variety of PIR schemes are known, they are rarely considered for practical use cases yet. We investigate the feasibility of PIR in the telecommunications world to open up data of carriers to external parties. To this end, we first provide a comparative survey of the current PIR state of the art (including ORAM schemes as a generalized concept) as well as implementation and analysis of two PIR schemes for the considered use case. While an overall conclusion is that PIR techniques are not too far away from practical use in specific cases, we see ORAM as a more suitable candidate for further R&D investment.

1 Background and Motivation

The telecommunications world is undergoing a transition where carriers not only provide services such as telephony or internet access, but also attempt to monetize the huge amount of data associated with their subscribers' activity. Analyzing data such as call statistics or roaming behavior can be used to offer specifically tailored services and packages. The combination of such data with other data from 3rd parties can potentially result in even more value. As such, one direction is to open up the existing databases to subscribing external parties. In fact, it may very well be the case that two rivaling carriers allow each other to query their subscriber databases, e.g. for detecting faults in the network or to support detection of fraudulent activities. Another real-world scenario is that of answering to the demands of public authorities which may want to verify that a user has been making a call at a certain time or to assess whether a certain IMEI or IMSI is part of the carriers subscriber base.

An open practical problem is how to guarantee to the querying party that the database owner will not learn anything about the data that is retrieved — a problem known as Private Information Retrieval (PIR) [**?**]. Accordingly, we assessed the feasibility of PIR schemes to support such use cases, where the typical database consists of 400.000-800.000 entries of IMEIs and/or IMSIs. This paper provides a comparative survey of PIR schemes as part of Section **??**. We

then discuss two schemes in detail, i.e. a Trapdoor Group scheme in Section **??** and an ORAM approach in Section **??**. We provide detailed performance and runtime analysis data in Section **??**.

2 Overview and Comparison of Existing Schemes

The trivial solution for a user who wants to query a database without the database server learning about the query is for the server to simply send the entire database back to the user, who ignores all except the queried entries. Of course, this is very inefficient in terms of communication, but very efficient regarding computational effort because there is (almost) none. Thus, the incurred effort provides a good starting point in the sense that any new solution should have less communication than this trivial solution, often trading this for computational complexity in some form.

We split existing works that realize some form of private information retrieval into four main approaches. In a forthcoming paper, we present a detailed overview of the different schemes, here we only examine the high-level approaches and categorize the schemes into these approaches. Some of the mentioned schemes have also been presented in [**?**].

- **Homomorphic Approaches:** These protocols rely on the idea of the user masking (e.g., by homomorphically encrypting) the index which is being queried, and the server algebraically combining all indices with the database entries to obtain a masked version of only the queried entry. The user can then remove the mask to obtain the result. Publications which follow this general idea are [**?**,**?**,**?**] (Group Homomorphic), [**?**] (Trapdoor Group), [**?**,**?**,**?**] (Branching Programs), [**?**] (Lattice-based) and [**?**] (FHE-based).
- **ORAM Approaches:** ORAM comes from the field of software protection, but can also be used to protect privacy in databases as in this context. ORAM requires a slightly different setup in that the database must be encrypted and thus there must be some key management mechanism to implement PIR functionality. In contrast to pure PIR, ORAM offers the added option of writing, i.e., changing or adding entries. Publications based on ORAM are [**?**,**?**,**?**] (ORAM-Tree), [**?**] (Onion-ORAM), [**?**] (FHE-ORAM) and [**?**] (Parallel-Tree-ORAM).
- **Garbled Approaches:** Since PIR consists of two parties (the user and the server) trying to compute a function (the correct database entry) without the server learning the users input (the query index), it is natural to look to Multiparty-Computation, where two or more parties compute a function together without learning any input except their own, and the result of the computation. Publications involving this approach are [**?**,**?**,**?**].
- **Other Approaches:** The φ-Hiding Approach [**?**], the Trapdoor Permutation Approach [**?**], and the Sender Anonymity Approach [**?**].

Table **??** compares the schemes from the above approaches, indicating a particularly good value with a (light) green background and particularly unfavorable

aspects with a (darker) red background. The aspects considered are $\text{Comm}_{\mathcal{U}\to\mathcal{S}}$ (communication from the user to the server), $\text{Comm}_{\mathcal{S}\to\mathcal{U}}$ (communication from the server to the user), $\text{Comp}_{\mathcal{U}}$ (computational effort for the user) and $\text{Comp}_{\mathcal{S}}$ (computational effort for the server). The variables used are the following:

- n is the number of database elements
- B is the block size (i.e., in most cases the size of one database element)
- λ is the security parameter
- M (resp. C) is the message (resp. ciphertext) space of the encryption scheme
- m is a finite group order

The leftmost column denotes the underlying approach as presented above: H for homomorphic, O for ORAM, G for garbled and "-" if non apply.

3 Choosing and Optimizing

To test performance for the actual use cases as described in Section **??**, we implemented two different approaches — one homomorphic scheme and one ORAM-approach, as these differ greatly, yet can both solve our problem of private information retrieval. Concretely, we chose and modified a Trapdoor Group Scheme based on [**?**] and the Path-ORAM Scheme [**?**] for their conceptual simplicity.

3.1 The (Optimized) Trapdoor Group Scheme

The original scheme [**?**] only allows retrieval of an entire row (i.e., $\sqrt{n}$ out of n) of database entries, which we extend to allow single-entry-retrieval and minimize communication. To make this optimized scheme more easily accessible, we present it as a protocol:
Database Structure: n elements of $\mathbb{Z}_N$ arranged as a $\ln(n)$-dimensional array with entries $x_{i_1,\dots,i_{\ln(n)}}, i_j = 1,\dots,n^{1/\ln(n)}$ for $j = 1,\dots,\ln(n)$.
Prerequisites: We assume that we work in the group $(\mathbb{Z}_m, +)$ and that m and N are coprime.
Queries: Suppose the user wants to query the element $x_{i_1^*,\dots,i_{\ln(n)}^*}$.

1. The user selects m as the group order above depending on the required security level, but at least $m > N^{\lceil \ln(n) \rceil} \cdot n \cdot (N-1)$.
2. The user randomly selects $b^j \in \mathbb{Z}_m^*, j = 1,\dots,\ln(n)$ and $\ln(n) \cdot n^{1/\ln(n)}$ coefficients $e_{i,j}, i = 1,\dots,n^{1/\ln(n)}, j = 1,\dots,\ln(n)$, all of which are kept secret. There are three restrictions on the coefficients $e_{i,j}$:
   i. $e_{i,j} < \sqrt[\ln(n)]{\frac{m}{n\cdot(N-1)}}$ for all (i,j).
   ii. For $j = 1,\dots,\ln(n)$: If $i_j^* \neq i, e_{i,j}$ is a multiple of N (i.e., $e_i = a_i \cdot N$ for some a_i).
   iii. For $j = 1,\dots,\ln(n)$: If $i_j^* = i, e_{i,j}$ has the form $1 + a_l \cdot N$ for some a_l.
3. The user sends the $b_i^j = b^j \cdot e_{i,j} \bmod m$ to the database. This constitutes the query.

Idea	Scheme	$\text{Comm}_{\mathcal{U}\to\mathcal{S}}$	$\text{Comm}_{\mathcal{S}\to\mathcal{U}}$	$\text{Comp}_{\mathcal{U}}$	$\text{Comp}_{\mathcal{S}}$	Comments	Code
-	Trivial	1	$n \cdot B$	-	-	-	✗
H	[?,?,?] (Group Homomorphic)	$\sqrt{n} \cdot \log(\|C\|)$	$\sqrt{n} \cdot \log(\|C\|)$	$\sqrt{n}$ encryptions, 1 decryption	n scalar ciphertext multiplications, $\sqrt{n} \cdot \log(\sqrt{n})$ ciphertext additions	C is the ciphertext space, $\log(\|C\|)$ is the size of a ciphertext.	✗
H	[?] (Trapdoor Group)	$\sqrt{n} \cdot \log_2(m)$	$O(\sqrt{n} \cdot \log_2(m)+n)$	Generating m, $\sqrt{n}$ modular exponentiations/ multiplications $+\sqrt{n}$ discrete logs	n integer exponentiations/ multiplications $+\sqrt{n} \cdot \log(\sqrt{n})$ integer multiplications/ additions	Group order m can be chosen by user such that discrete log is efficient (e.g., additive group). Several queries can be sent at once, so amortized cost lower.	Not public
H	This paper (Optimized Trapdoor Group)	$\ln(n) \cdot n^{1/\ln(n)} \cdot \log_2(m)$	$O((\log_2(m))^{\ln(n)}+\sqrt{n})$	Generating m, $\ln(n) \cdot n^{1/\ln(n)}$ modular exponentiations/ multiplications $+\ln(n)$ discrete logs	$O(n)$ integer exponentiations/ multiplications $+O(n)$ integer multiplications/ additions	Group order m can be chosen by user such that discrete log is efficient (e.g., additive group).	Not public
H	[?,?,?] (Branching Programs)	$\log(n) \cdot \sqrt{n} \cdot \log(\|C\|)$	$\sqrt{n} \cdot \log(\|C\|)$	$\log(n)$ encryptions and decryptions	For k-ary branching program (optimal $k = 5$): n multiplications, $\frac{n}{k} \cdot \sqrt{k} \cdot \log(\sqrt{k})$ additions	C is the ciphertext space of the Damgård-Jurik cryptosystem.	✗
H	[?] (Lattice-based)	$O(n \cdot N^2 \cdot m)$	$N \cdot m$	$O(n \cdot N^x)$ where x depends on the matrix multiplication algorithm used, mostly a bit less than 3.	$2n \cdot N^2$ multiplications and $2N \cdot \log(n \cdot N)$ additions.	Recommended as $N = 50$.	✓ C++
H	[?] (FHE-based)	$\log(C)$	$\log(C)$	One encryption, one decryption	Depends on the concrete FHE scheme used, likely very expensive.	This is one extreme where the server does all the work and the user almost none.	✗
O	[?,?,?] (ORAM-Tree)	$O(\log(n)^3 + \log(n)^2 \cdot \log(\|C\|))$	$O(\log(n)^3 + \log(n)^2 \cdot \log(\|C\|))$	$\log(n)$ recryptions for each operation	-	Supports writing as well. Could be combined with FHE to reduce user communication and transfer computation to the server.	✓ Java
O	[?] (Onion-ORAM)	$O(\log(n))$	$O(B)$	$\tilde{O}(B \cdot \log^4(n))$	$\tilde{\omega}(B \cdot \log^4(n))$	The block size B needs to be very large ($\tilde{\Omega}(\log^5(n))$).	✗
O	[?] (FHE-ORAM)	$\log(\|C\|) \cdot \|op\|$, op = ORAM operation written as circuit	$\log(\|C\|)$	Convert operation into circuit, encrypt values, decrypt result.	Again depends on concrete FHE scheme, likely very expensive.	This seems worse than the trivial FHE approach above, but ORAM has a write-operation which pure PIR does not.	✗
O	[?] (Parallel-Tree-ORAM)	$\log(n)$	$O(B)$	-	$\log(n)$ recryptions for each operation, but parallelized.	Tree-ORAM outsourced to server using secure coprocessors (with which the user communicates in non-oblivious fashion).	Not public
G	Trivial Garbled Circuit	$>> n$	$O(B)$	Transform function into Boolean circuit, generate 4 keys for each gate, compute 2 MACS for each gate.	Evaluate the Boolean circuit with the garbled keys.	This is worse than the trivial solution in every aspect except server communication.	✗
G	[?,?,?] (Garbled RAM)	O(RAM-execution time of query)	$O(B)$	Garble the query (O(RAM-execution time of query))	Evaluate garbled query (O(RAM-execution time of query))	The user also has to garble and upload the database once in the beginning.	✗
-	[?] (φ-Hiding)	$\log(n) + \lambda$	λ	Effort of computing φ-hiding m plus 2 modular exponentiations	Hamming-weight(n) modular exponentiations	λ is logarithmic in n, with recommended settings total communication is $O(log^8(n))$.	Pseudo code
-	[?] (Trapdoor Permutation)	$O(B)$	$n - \frac{n}{2B}$ ($< O(n)$ while $n > B^2$)	$O(B)$	$O(n \cdot B)$	User computation depends on the trapdoor functions and hardcore predicates used, assumed $O(n)$.	✗
-	[?] (Sender Anonymity)	$(\lambda + 1) \cdot \sqrt{n}$	$(\lambda + 1) \cdot \sqrt{n}$	$O(\log(\lambda) \cdot \sqrt{n})$	$O(Q \cdot (\lambda + 1) \cdot \sqrt{n})$, Q is number of separate queries sent (> 1!)	Very likely insecure, as summing up all subqueries yields sum of separate query vectors.	Not public

Table 1. A comparison of different PIR solutions

Database Action: For $j = 1, \ldots, \ln(n)$: The database computes $x_{i_{1+j},\ldots,i_{\ln(n)}} := \sum_{k=1}^{n^{1/\ln(n)}} b_k^j \cdot x_{k,i_{1+j},\ldots,i_{\ln(n)}}$ and sends $x := x_{i_{\ln(n)+1}}$ to the user. Note that the operations in this step are done over the integers, as the database does not know the group order.
User Decoding: For $j = 1, \ldots, \ln(n)$, the user sets $x = x \cdot (b^j)^{-1} \bmod m$ and transforms the result to N-ary encoding. Then the least significant digit is the requested database item $x_{i_1^*,\ldots,i_{\ln(n)}^*}$.
Security: One notable aspect in this new protocol is that the size limit of the e_i's has changed from the original version. This requirement ensures getting the correct result without wrapping around mod m in the decoding phase. The security of the original scheme relies on the assumption that given the $\sqrt{n}$ PIR request elements $(b_1, \ldots, b_{\sqrt{n}})$, where $b_i = b \cdot e_i \bmod m$ and the e_i's are chosen according to the constraints detailed above (with the a_i's in constraints 2.ii. and 2.iii. selected uniformly at random), any computationally bounded adversary can output the correct m only with negligible probability. This assumption is called the Hidden Modular Group Order Assumption, and indicators for the hardness of the problem (i.e., how much information about the group order is leaked by the queries), are presented in the original paper [**?**], along with a reduction from the PIR protocol to this assumption. For our improved scheme, it can easily be verified that $\sqrt{n} > \ln(n) \cdot n^{1/\ln(n)}$ for $n \geq 213$. As databases are generally much larger than 213 elements, we can now base security on the security of the original scheme, since we will now be sending less query elements and thus leaking at most as much data as the original scheme.

3.2 The Path-ORAM Scheme

The second solution we chose to implement is the Path-ORAM scheme from [**?**] with non-recursive position map storage. We describe the scheme on a high level, but with some parameters as we implemented them instead of the generic version (e.g. encrypting with AES):
Database Structure: The database is held in a binary tree of height $L = \lceil \log_2(n) \rceil$ with 2^L leaves. Each leaf (called a "bucket") holds up to 5 database entries (and is filled with dummy entries if it contains less). Note that there are far more buckets than database elements. The bucket is encrypted by the user with AES in CBC mode, where in our use case each database entry consists of 1 or 2 AES-Blocks (128 bits each) depending on the chosen parameter setting (see Section **??**). Thus, each bucket contains 5 or 10 AES Blocks plus the IV, so 6 or 11 blocks in total. Also, the user maintains a local stash S (which acts as a temporary storage space) and a lookup table (called "position map") mapping database blocks to the leaves of the binary tree. We assume that the database is already initialized, as setup is rather tedious and must only be done once at the very beginning before the first query is made, making it irrelevant for performance comparisons.
Queries: To retrieve an entry, the user looks up what leaf of the tree the data

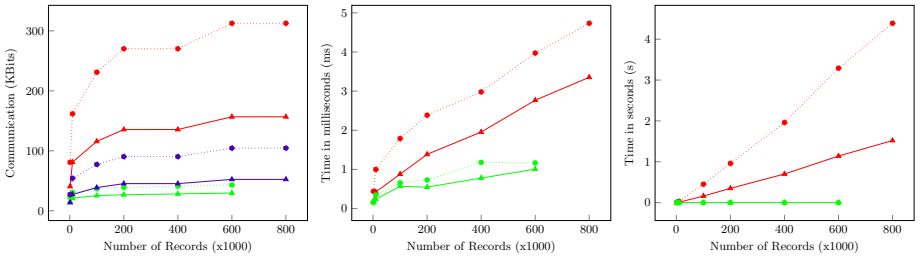
block is mapped to in the position map and reads the entire path from the root to the leaf into the local stash S (which may already contain some elements from previous queries), as the entry will be in some bucket on this path (or in the stash). The entry is mapped to a new leaf randomly and the data is replaced in case of a write operation. Then, all elements in the stash are reencrypted and written to the server in a bottom-up manner: Each entry in the stash is placed in a bucket on the path to the (old) leaf as far away from the root as possible, guaranteeing that the maximum amount of blocks can be placed into the tree. The bucket is filled up with dummy blocks and encrypted with AES in CBC-mode with a random IV. It can happen that some elements from the stash cannot be placed into the tree (e.g., when there are several elements whose paths to their leaves only intersect the current path at the root, and there are more of these elements than the root bucket can contain), these remain in the client stash so that they can be placed into a bucket in a future query. If too many of these elements accumulate and the stash cannot hold them in addition to the elements being read from a path during a query, we say that the ORAM has failed. We chose a stash size of 220 blocks.
Security: Since we implemented the scheme without changing it, the security analysis of the original paper holds.

4 Performance

We now present the performance of our two chosen schemes. Times were measured on an Intel Core i5-4570 CPU with 3.20GHz and constitute average values, and the number of database entries was derived from our use case (400.000 − 800.000 with some smaller numbers for scale). The entries are random numbers of lengths 256 (resp. 128) bits to simulate the IMEIs and (or) IMSIs. The chosen AES implementation in the ORAM scheme was wolfcrypt. Regarding memory requirements[1] for our use case, the Trapdoor Group scheme requires a group order m of at least 3756 bits. This implies a server memory of about 25.6MB for the database, and about 1.317GB for storing intermediate results, so roughly 1.345GB in total. The memory requirement on the user side is only about 34kB. In the ORAM-scheme, user memory is about 2.1MB, whereas server memory strongly depends on the number of entries and got so large that we could not supply data for 800.000 entries because our allotted memory limit was exceeded.

For the more traditional Private Information Retrieval metrics, we split the performance measure into three main components: Communication (Figure **??**), computational effort for the user (**??**) and computational effort for the server (**??**) and plot the effort for different numbers of database entries. In each diagram, the dotted plots represent entries of length 256 bits, and the solid plots are entries of length 128 bits. The green plots always correspond to the ORAM scheme, and the red plots to the Trapdoor-Group scheme. In the communication figure, there are three colors: Red represents the amount of data sent from the user to the server in the Trapdoor-Group scheme, blue represents the amount of data

[1] These are theoretical results and were not actually measured



(a) Communication. (b) User Comp. (c) DB Comp.

Fig. 1. Performance of the two approaches: Red (top 2 lines) is Trapdoor-Group (user to server in communication), green (bottom 2 lines) is ORAM, blue (middle 2 lines) is server to user communication (Trapdoor-Group). Dotted is 256 bit, solid is 128 bit database entries.

sent from the server to the user in the Trapdoor-Group scheme, and green is the amount of data sent from either the user to the server or vice versa (as these values are equal) in the ORAM scheme. User computation encompasses decrypting, reading and encrypting in the ORAM scheme, and the decoding step in the Trapdoor-Group scheme [2].

5 Conclusion and Future Work

As we can see, ORAM performs better in all these aspects, even though the Trapdoor-Group protocol actually performs worse in terms of user computation than Figure **??** implies (see Footnote **??**). Thus, for the use case upon which this paper is based, the ORAM-approach seems like the better solution — provided, of course, that the server has enough memory to store the tree. If, however, memory is the constraining factor rather than speed (which seems unlikely in today's world), the Trapdoor Group protocol would be the better choice both for user and for the server.

For future work, an interesting aspect in optimizing performance could be the "levels" observed in Figure **??** for the Trapdoor Group scheme (i.e., the

[2] Additionally, there is a user setup phase which incurs computational effort but was not included in Figure **??**. The reason is that m was computed as a prime by calling `nextprime()` from the GMP-library in our code and this function varies enormously in its runtime, dominating the total time. However, it seems that this could be easily circumvented in reality once the parameters of the database are set - e.g., by picking m randomly from a large list of primes, or choosing m as not prime and implementing constraints on the secret values instead. Either way, this is an additional cost that really needs to be added to the time in Figure **??**. The time for user setup without this prime generation can be seen in the figure on the right.

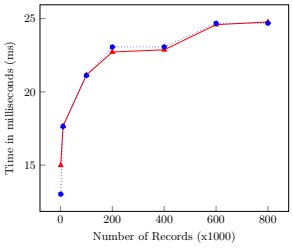


Fig. 2. User setup w/o prime generation for 128 and 256 bit inputs.

values for 200,000 and 400,000 entries seem similar, as do those for 600,000 and 800,000), which we suppose comes from the $\lceil \ln(n) \rceil$ exponent in the constraint for the size of the group order m (where n is the number of database entries).

References

1. Apon, D., Katz, J., Shi, E., Thiruvengadam, A.: Verifiable oblivious storage. In: PKC 2014
2. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: EUROCRYPT '99
3. Chang, Y.: Single database private information retrieval with logarithmic communication. In: ACISP 2004
4. Devadas, S., van Dijk, M., Fletcher, C.W., Ren, L., Shi, E., Wichs, D.: Onion ORAM: A constant bandwidth blowup oblivious RAM. In: TCC 2016-A
5. Doröz, Y., Sunar, B., Hammouri, G.: Bandwidth efficient PIR from NTRU. In: FC 2014 Workshops, BITCOIN and WAHC 2014
6. Gentry, C., Halevi, S., Lu, S., Ostrovsky, R., Raykova, M., Wichs, D.: Garbled RAM revisited. In: EUROCRYPT 2014
7. Gentry, C., Halevi, S., Raykova, M., Wichs, D.: Outsourcing private RAM computation. In: FOCS 2014
8. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM (1996)
9. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: TCC 2007
10. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Optimal rate private information retrieval from homomorphic encryption. PoPETs 2015
11. Kushilevitz, E., Ostrovsky, R.: One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In: EUROCRYPT 2000
12. Kushilevitz, E., Ostrovsky, R.: Replication is NOT needed: SINGLE database, computationally-private information retrieval. In: FOCS '97
13. Lipmaa, H.: First CPIR protocol with data-dependent computation. In: ICISC 2009
14. Lorch, J.R., Parno, B., Mickens, J.W., Raykova, M., Schiffman, J.: Shroud: ensuring private access to large-scale data in the data center. In: FAST 2013
15. Lu, S., Ostrovsky, R.: How to garble RAM programs. In: EUROCRYPT 2013
16. Ma, Q., Zhang, J., Peng, Y., Zhang, W., Qiao, D.: SE-ORAM: A storage-efficient oblivious RAM for privacy-preserving access to cloud storage. In: CSCloud 2016
17. Mayberry, T., Blass, E., Chan, A.H.: Efficient private file retrieval by combining ORAM and PIR. In: NDSS 2014
18. Melchor, C.A., Barrier, J., Fousse, L., Killijian, M.: XPIR : Private information retrieval for everyone. PoPETs 2016
19. Melchor, C.A., Gaborit, P.: A lattice-based computationally-efficient private information retrieval protocol. IACR Cryptology ePrint Archive 2007, 446 (2007)
20. Ostrovsky, R., III, W.E.S.: A survey of single database PIR: techniques and applications. IACR Cryptology ePrint Archive 2007, 59 (2007)
21. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C.W., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: CCS'13
22. Trostle, J.T., Parrish, A.: Efficient computationally private information retrieval from anonymity or trapdoor groups. In: ISC 2010