Private Information Storage

Rafail Ostrovsky* Victor Shoup†

April 10, 1996
*** Extended Abstract ***

Abstract

We consider the setting of hiding information through the use of multiple databases that do not interact with one another. In this setting, there are $k \geq 2$ "databases" which can be accessed by some "users". Users do not keep any state information, but wish to access $O(n)$ bits of "data". Previously, in this setting solutions for *retrieval* of data in the efficient manner were given, where a user achieves this by interacting with all the databases. We consider the case of both *writing and reading*. While the case of reading was well studied before, the case of writing was previously completely open. In this paper, we show how to implement *both read and write* operations, with the following strong security guarantees: *all* the information about the read/write operation is information-theoretically hidden from all the databases (i.e. both the value of the bit and the address of the bit). As in the previous papers, we measure, as a function of k and n the amount of communication required between a user and all the databases for a single read/write operation, and achieve efficient read/write schemes. Moreover, we show a general *reduction* from reading database scheme to reading and writing database scheme, with the following guarantees: for any k, given a *retrieval only* k-database scheme with communication complexity $R(k, n)$ we show a $(k + 1)$ *reading and writing* database scheme with total communication complexity $O\left(R(k, n) \cdot (\log n)^{O(1)}\right)$. Our general reduction in combination with the paper of [Chor,Goldreich,Kushilevtiz,Sudan] yields:

- a 3-database scheme with read/write communication complexity of $O\left(n^{1/3} \cdot (\log n)^3\right)$;
- for all constants $k \geq 2$, a $(k + 1)$-database scheme with read/write communication complexity of $O\left(n^{1/k} \cdot (\log n)^3\right)$;
- $O(\log n)$-database scheme with read/write communication complexity of $O\left((\log n)^3\right)$.

It should be stressed that prior to the current paper no trivial (i.e. sub-linear) bounds for private information storage were known. Moreover, our result yields a solution to the problem of information-theoretically secure Oblivious RAM simulation with poly-log overhead in the above setting. Our result also implies that efficient instance-hiding schemes where the *state* can be altered are possible.

*Bellcore, e-mail: rafail@bellcore.com
†Bellcore, e-mail: shoup@bellcore.com

1 Introduction

1.1 The Problem

Consider a user who wants to access a database, performing both query (read) operations and update (write) operations. Furthermore, suppose that she wants to protect her privacy, keeping private the *address* of the data being accessed, as well as the data itself, in the case of updates. Under complexity-theoretic assumptions, and assuming that the user keeps small amount of state information herself, this is in fact possible with a single database [G-87, Ost-90, GO-96]. In this paper, we do not wish to rely on *any* complexity assumptions, wish to achieve information-theoretic security, and require that users do not keep *any* state information (which would allow different users to access the database). Can such very strong requirements be achieved? We show that this is in fact possible, when relying on several databases, instead of one.

While a single database (or a player) may not be trustworthy, the suggestion to rely on several databases (or players) to achieve greater security has been suggested in many different settings, including previous work on reading from multiple databases of [CGKS-95], U.S. government Clipper chip proposal [U.S.-93], Micali's"fair cryptosystems" [M-92], secret sharing schemes [B-79, S-79], and instance hiding schemes of [RAD-78, AFK-89, BF-90, BFKL-90]. The suggestion to rely on several, distributed agents is especially relevant due to the explosive growth of the internet and the advent of small, cheap "network computers". In this scenario, data will routinely be stored on remote servers, and accessed by the small, cheap computers via the internet.

Now, let us make the problem more precise. Suppose there are $k \geq 2$ databases which can store information. That is, we envision the actual database to be stored in a distributed manner consisting of two or more component databases that are not allowed to communicate with one another. The databases communicate with a user, responding to both "store(addr,value)" and "fetch(addr)" commands in the standard manner. The *view* of each *individual* constituent database is the transcript of all "store" and "fetch" commands issued to it. Databases are deterministic, but we assume that users are randomized and can flip coins. We stress that we do not assume that users have access to a random oracle, solely that users can flip coins as needed, and that users do not keep *any* state information. That is, a user can flip some small number of coins during read/write operation, but then a different user can perform another read/write operation without any interaction with the first user.

In its simplest form, a database is a bit-vector $D \in \mathbf{B}^n$, where $\mathbf{B} = \{0,1\}$. The database D is represented by keeping data in k constituent databases that do not talk to one another. A *private query* is an interactive protocol that allows any user to obtain a bit D_i for an address $i \in \{0,\ldots,n-1\}$ of the user's choice; the protocol is such that each constituent database's view of the interaction is *independent* of the address i. A *private update* is an interactive protocol that allows the user to set D_i to b, where the address $i \in \{0,\ldots,n-1\}$ and the bit $b \in \mathbf{B}$ are of the user's choice; the protocol is such that each individual database's view of the interaction is independent of the address i and the bit b. The user or users are not required to maintain any state information between database accesses. We allow queries and updates to be interleaved in an arbitrary manner and measure the total number of bits transmitted as a function of k (the total number of constituent databases) and n (the size of the actual database).

1.2 Our Contribution

We show two general reductions:

Theorem 1 For any $k \geq 2$, given any retrieval k-database scheme for n data bits with communication complexity $R(k,n)$ there exists storage and retrieval $(k+1)$-database scheme with communication complexity $O\left(R(k,n) \cdot k \cdot (\log n)^3\right)$, where each constituent database holds $O(kn)$ bits.

Theorem 2 For any $k \geq 2$, given any retrieval k-database scheme for n data bits with communication complexity $R(k,n)$ there exists storage and retrieval $(2k)$-database scheme with communication complexity $O\left(R(k,n) \cdot (\log n)^3\right)$, where each constituent database holds $O(n)$ bits.

REMARKS:

- For both theorems above, the communication complexity bounds are independent of the order of reading and writing. In fact, it is information-theoretically hidden from each database if this is a reading or writing command.
- Notice that the difference when going from $(k+1)$ databases to $(2k)$ databases in our two results is a k multiplicative factor in the communication complexity and the size of each constituent database. For the number of databases between $k+1$ and $2k$ simple tradeoffs can be achieved.
- For reading schemes where privately reading a contiguous block of l bits has smaller communication complexity them reading l single bits (as in [CGKS-95]), out reductions are more efficient, and the poly-log exponent can be further reduced. Moreover, the savings on reading schemes for blocks could be used to achieve savings in our schemes when reading/writing blocks of bits.
- Using secret sharing [S-79], our results could be adopted to the case where coalitions of databases are allowed to communicate. Moreover, we can adopt our solution to the malicious case, in the sense of certifying (with overwhelming probability) if data has been tampered with, as long as there exists at least one non-corrupted database.
- In case of writing, one can consider the number of bits not to be fixed, but grow as a function of time. In this case, we get poly-log overhead results as well. For example, we get an analog of information-theoretically secure Oblivious RAM simulation (see [G-87, Ost-90, GO-96]), where t steps of the original program can be simulated in an oblivious manner using $O(R(k,t) \cdot (\log t)^3)$ overhead per step using k databases. Moreover, all the solutions for the Oblivious RAM model are *amortized*, where as all our solutions are not.

Combining our general reductions with reading schemes of [CGKS-95], we get the following corollaries:

Corollary 3 There are private information storage and retrieval schemes for n bits of data, with the following parameters:

- a 3-database scheme with read/write communication complexity of $O\left(n^{1/3} \cdot (\log n)^3\right)$;
- for all constants $k \geq 2$, a $(k+1)$-database scheme with read/write communication complexity of $O\left(n^{1/k} \cdot (\log n)^3\right)$;
- $O(\log n)$-database scheme with read/write communication complexity of $O\left((\log n)^3\right)$.

REMARK: Notice that we show a general reduction from reading to reading and writing, with only poly-log overhead. Hence, due to the general nature of our reduction, any improvement in the efficiency of reading schemes would yield a more efficient reading and writing scheme as well.

1.3 Comparison with Previous Work

Closely related to the private query/update problem is the *oblivious RAM simulation problem*, studied in [G-87, Ost-90, GO-96]. The problem is to simulate a random-access machine (RAM) with another so that the memory contents and access patterns of the latter machine are independent of the input. In the oblivious RAM simulation problem, the central processing unit (CPU) plays the role of the user, and the main memory plays the role of the database. It is perhaps worth pointing out the technical differences between these two problems: Unlike the main memory of the RAM, the databases are distributed; Privacy in the database problem is required to be information theoretic, whereas in the oblivious RAM simulation problem, complexity-theoretic assumptions are used and only computational privacy is achieved (or an access to a random oracle is required, which we do not allow, making the problem much harder); Unlike the CPU, the user does not maintain any state, again making the problem harder.

The problem of performing private database queries with multiple databases which do not interact with one other was studied in two different settings: in instance hiding schemes of [BF-90, BFKL-90] and on private database queries of [CGKS-95]. In this paper we build on the work of [CGKS-95] which deals exclusively with private queries and show schemes which can perform both private queries *and* updates. Again, let us point out the technical differences with our work. In the instance-hiding schemes of [BF-90, BFKL-90] they deal with exponential-size databases and polynomial number of (trusted) oracles. In [CGKS-95] the approach is scaled down to liner copies of database and sub-linear, down to poly-log number of databases. We are dealing with the same setting. On the other hand, for both the instance hiding schemes of [BF-90, BFKL-90] and private information reading of [CGKS-95], they assume that all oracles/databases contain the same information. In our case, we relax this requirement and allow different databases to contain different bit-strings and allow individual databases to store more then n bits of data each, while still trying to keep this parameter as small as possible.

1.4 Organization

In §2 we show three elementary solutions, as means of illustrating several techniques used in our general reduction and also presenting relevant definitions. These elementary solutions are asymptotically inferior to our general reduction and are not used there, however we use them to preset some techniques. In particular, in subsection §2.1 we present a two-database trivial linear solution and in subsection §2.2 we show a 4-database solution where we separate reading from writing (i.e. writing still takes $O(n)$ steps but reading takes $O(n^{1/3})$ steps. Then, in subsection 2.3 we show the elementary 8-database scheme with writing communication complexity $O(n^{1/2})$ and reading communication complexity of $O(n^{1/3})$.
Having developed necessary definitions and techniques, we then show our first general reduction in section §3. Namely, we show how given a k database reading scheme with communication complexity $R(k,n)$ we show how to construct a reading and writing scheme with $2k$ databases and communication complexity $O\left(R(k,n) \cdot (\log n)^3\right)$ per read/write operation and $O(n)$ storage per database. In section §4 we show how to extend this solution in order to reduce the number of databases down to $k+1$ at the price of additional multiplicative k per read/write operation and additional multiplicative k in the size of each constituent database. Finally, in section §5 we present conclusions, additional remarks and open problems.

2 Elementary methods and definitions

2.1 Elementary linear solution

As a first simple example, one can build a distributed database consisting of two component databases that supports private queries and updates as follows. The idea is to use a very rudimentary form of *secret sharing*: a bit $b \in \mathbf{B}$ can be split into two *shares* r and $r \oplus b$, where $r \in \mathbf{B}$ is random. Each individual share is random and independent of b. At any time, for each $i \in \{0, \ldots, n-1\}$, each constituent database holds one share of D_i. To perform a private query, each database simply sends *all* of its shares to the user, who then combines the two shares of the bit he is actually interested in. To perform a private update, the user will sequentially read and re-write the entire database, changing only the bit of his choice. When writing, the user creates new random shares for each individual bit, sending these to the constituent databases. While this scheme is simple, it is obviously quite impractical for large databases, as it requires $O(n)$ bits of communication for every private access (i.e. for both reading and writing).

The above method of representing each bit as an *xor* of two bits in two different databases allows users to hide from each database the value of each bit that is being stored. Thus, one can think of this operation as "encrypting" data so that each database sees the access pattern (just scanning the database from left to right) but does not see the actual values of the bits, and hence does not know which bit was re-written. We define a write operation where the writing access pattern is visible to each constituent database but the value being written as not visible as a *semi-private* update.

2.2 Separating writing from reading

In this subsection we show how the above elementary scheme for writing could be augmented to have efficient reading with four databases. Recall that [CGKS-95] show that with two databases *which contain identical n bits of data*, it is possible to privately read a bit with $O(n^{1/3})$ communication complexity. Notice, however, that in the two-database scheme presented in the previous subsection, the two databases contain different data, since every bit is represented as an xor of two corresponding bits from two databases of subsection §2.1. The idea is very simple: using four databases, maintain two identical copies of each database of previous subsection §2.1. Now, writing still takes $O(n)$ steps, since we still must re-write the entire database, and in fact maintain two copies, but reading could be done in $O(n^{1/3})$ steps just by reading the appropriate bit from both identical pairs of databases using twice the reading scheme of [CGKS-95] and then just xoring these two bits.

2.3 Elementary Square-root solution

In this section, we present an elementary 8-database scheme for private queries and updates with a communication complexity of $O(n^{1/2})$ for writing and $O(n^{1/3})$ for reading. The idea will be an extension of the solution of the previous section, but with more efficient writing. Every data bit will be represented an an exclusive-or of four bits, from four different databases. Then we will duplicate each of these four databases in order to achieve efficient reading (so, using all together 8 databases). The advantage of this scheme will be that writing can also be done more efficiently. We now show how this can be done.

Assume we have 4 databases, D^{st} $(s, t \in \mathbf{B})$, each of which supports private queries (i.e. in which reading could be done efficiently). At any point in time, each bit in the database is represented as the exclusive-or of the four corresponding bits in the constituent databases. We now show how to privately toggle a particular bit in the database. Let $d = \lceil n^{1/2} \rceil$. For any address $i \in \{0, \ldots, n-1\}$, we can write

$$i = jd + k \qquad (0 \le j < d,\ 0 \le k < d).$$

To toggle bit i, the user generates two random bit-vectors $v, w \in \mathbf{B}^d$. To each component database D^{st}, the user sends vectors $v', w' \in \mathbf{B}^d$ where

$$v'_l = \begin{cases} v_l & \text{if } l \ne j, \\ v_l \oplus s & \text{if } l = j, \end{cases}$$

for $0 \le l < d$, and

$$w'_m = \begin{cases} w_m & \text{if } m \ne k, \\ w_m \oplus t & \text{if } m = k, \end{cases}$$

for $0 \le m < d$. Upon receiving vectors v', w', the database D^{st} toggles all bits whose address is of the form $ld + m$, where $v'_l = 1$ and $w'_m = 1$.

Consider the effect of this operation on an arbitrary bit in the database whose address is $ld + m$:

$$D_{ld+m} = D^{00}_{ld+m} \oplus D^{01}_{ld+m} \oplus D^{10}_{ld+m} \oplus D^{11}_{ld+m}. \tag{1}$$

Case 1. If $l = j$ and $m = k$, then exactly 1 term in (1) are toggled, effectively toggling the sum.
Case 2. If $l \ne j$ and $m \ne k$, then either none or all of the terms in (1) are toggled, leaving the sum unchanged.
Case 3. If $l \ne j$ or $m \ne k$, but not both, then either 0 or 2 of the terms in (1) are toggled, again leaving the sum unchanged.

From the above discussion, it is clear that this operation has the effect of toggling bit i in the database. Moreover, each constituent database receives two random bit-vectors that are independent of i. Thus, the update operation is private. The communication complexity is $O(n^{1/2})$. To complete the discussion, we observe that each of the four constituent databases, which support private queries, can be implemented using a pair of identical, ordinary databases. Using the results of [CGKS-95], a private query can then be implemented with communication complexity $O(n^{1/3})$. Putting all of this together, we get an 8-database scheme where private queries have a communication complexity of $O(n^{1/3})$, and private updates have a communication complexity of $O(n^{1/2})$.

REMARK: The above method could be naturally extended to higher dimensions, similar to [CGKS-95] approach for constant k. However, our general reductions in the next two sections yield asymptotically better results, and thus we do not present this simple extension.

3 Proof of theorem 2.

In this section, we present the proof of theorem 2. That is, we assume that we are given a retrieval k-database schemes with communication complexity $R(k, n)$ and we show a storage and retrieval $2k$ database scheme with communication complexity $O\left(R(k, n) \cdot (\log n)^3\right)$.

First, we reduce the problem of reading and writing to the problem of *oblivious writing only*, i.e. where reading is already private and writing is semi-private (recall that we sat that writing is semi-private if every constituent database sees which memory locations users writes into, but not there content).

3.1 Reduction to oblivious writing

We reduce the problem of private reading and writing to the problem of *oblivious writing*, where we need to hide the writing access pattern only. That is, we show how to construct a distributed database that supports private queries and semi-private updates (for the definition of semi-private updates see section §2.1).

The idea is to combine the secret-sharing, mentioned in subsection §2.1 and private reading schemes for k databases with complexity $R(k, n)$, similar to subsection §2.2. Thus, we utilize a k-database scheme for private queries. Each bit in the database is split into two random bits, or "shares," whose exclusive-or is the value of the bit. The database is partitioned into two components, and each share is stored in one component. Each component is then distributed and replicated k times, and the k-database scheme for private queries is then used for reading bits in one component. This gives rise to a $2k$-database scheme for private queries and semi-private updates whose communication complexity is bounded by a constant times that of the underlying k-database private query scheme. The size of each of the $2k$ databases is $O(n)$.

3.2 Solving oblivious writing via simulation

We assume that we have a database that supports private queries and *semi-private updates*, that is, updates such that the database's view of the interaction is independent of the data, but not necessarily the address, of the the update. Using a single such database, we show how to implement private queries and updates, where each private access requires $(\log n)^{O(1)}$ accesses to the underlying database. The underlying database itself stores $O(n)$ bits.

We make use of a variant of the memory-hierarchy idea used in [Ost-90, GO-96] for the oblivious RAM simulation problem. However, there are several obstacles which we must overcome:

- in the oblivious RAM simulation solution, the user (i.e. CPU) uses random oracle (or pseudo-random functions), where as in our case the user is allowed to flip coins, but he does not has access to a random oracle;
- in the oblivious RAM simulation solution, user has local storage, where as in our case user is completely memoryless (from one read/write operation to the next) and does not have any local storage;
- the solution presented in [Ost-90, GO-96] is amortized while here we do not allow any amortization.

On the other hand, in our case we assume that *reading* could be done privately already, and hence we have to hide the access pattern for the writing only. Hence, we show how using a somewhat similar data-structure to the one in [Ost-90, GO-96], and a new algorithm for this data-structure we can overcome all this difficulties for both reading and writing. We first give an overview of the algorithm, and then the details will follow.

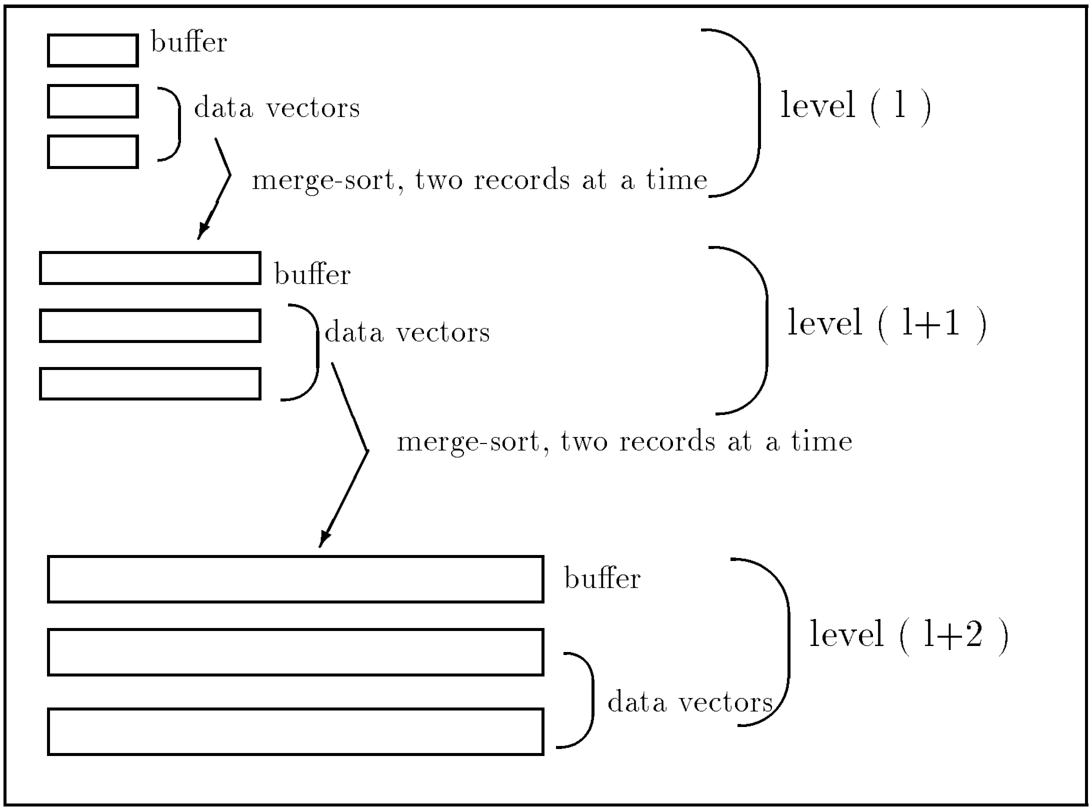
OVERVIEW OF THE ALGORITHM:

— The database is represented as a memory hierarchy with l levels $0, 1, \ldots, m$, where m is chosen so that $2^m \approx n/\log n$. At each level l, the database stores a set of address/value pairs (i, b), where $i \in \{0, \ldots, n-1\}$ and $b \in \mathbf{B}$. The number of such pairs stored at level l at any given time is either 0, 2^l, or 2^{l+1}. The pairs are stored in sorted order, by its actual address i. There is also a level $m+1$ which contains a bit vector of length n. The same address may appear in several levels, but the current value of a bit i is defined by the lowest (i.e. smallest) level containing i, if such a level exists; otherwise, the value is defined by location i of the bit vector at level $m+1$.

— To perform a private query, each level (from smallest to largest) is searched (using binary search) in turn until the desired address is found (dummy access are performed after this, to guarantee that the number of accesses is always constant.) Notice that since *reading* is private, binary search is implemented simply using $O\left((\log n)^2\right)$ calls to private reading. To update location i to the value b, a new pair (i, b) is inserted at level 0. As updates are performed, individual levels become full, and the data will flow from low levels to higher (i.e. bigger) levels. This is done in a gradual manner using merge-sort: for each level we perform two operations of the merge-sort, where we (privately) read two entries of the lower-level sorted data vectors and insert (in sorted order) into next level buffer.

Now the details. At a particular level l in the memory hierarchy, where $0 \leq l \leq m$, there will be three vectors, each of which stores 2^l address/value pairs, stored in sorted order of increasing address.

At any time, one of these vectors act as a "buffer," and the other two act as "data" vectors. Each of the two data vectors are either "empty" or "full." If both are full, one is designated as "primary." The roles of the vectors will change over time, which requires a small (constant) amount of state information, which is stored at this level as well.



To perform a private query, the user searches for the desired address in levels 0 through m in turn. The search is done by performing a binary search on the data vectors, searching in the primary data vector first. If the address is not found at any of these levels, the value is taken from the vector at level $m + 1$. Since the underlying database supports private queries, these query operations are private, provided the user makes a number of "dummy" queries on the underlying database so that the total number of such queries is always the same.

We now describe how private updates are performed. This will involve placing an address/value pair in the buffer at level 0. Now consider an arbitrary level l. By design, once every 2^l steps (update operations), the buffer at level l will become full, and level l will fill the buffer at level $l+1$ once every 2^{l+1} steps. Suppose level l's buffer has just become full. If both of the data vectors are full, then by design their contents has just been copied into the buffer at level $l+1$. At this point, the old buffer becomes a full data vector, one of the old data vectors becomes the buffer, and the other an empty data vector. Otherwise, if at least one of the data vectors is empty, the old buffer becomes a full data vector, and the empty data vector becomes the buffer. If now both data vectors are full, level l will have 2^l steps before its buffer gets filled again, and it uses these 2^l steps to merge and copy its two data vectors into the buffer at level $l+1$. This is done by executing *two* of the basic steps in the standard algorithm for merging sorted lists (as used in the merge-sort algorithm). This requires that three pointers are maintained at level l, two point to the current "front" of each of the two full data vectors, and one points to the "tail" of the partially full buffer at level $l+1$. An address that appears in both data vectors is easily detected, and only the address/value pair in the primary data vector is copied to the buffer—the other is effectively discarded (this requires that some "dummy" copies are performed to maintain privacy). Levels m and $m+1$ require special treatment. When both data vectors at level m are full, then with each step, about $\log n$ successive positions at level $m+1$ are updated. This is easily accomplished in a private manner using the list-merging idea above.

To see that these updates are private, recall that the value of the data stored in the underlying database is assumed private, and observe that the *order* in which locations in the database are updated is not data dependent, depending only on the number of private updates performed so far. One easily verifies that the underlying database stores $O(n)$ bits, and that each private query or update requires $O((\log n)^3)$ operations on the underlying database. This concludes the proof of theorem 2. ■

4 Proof of Theorem 1.

In this section, we present the proof of theorem 1. The approach presented in the previous section requires $2k$ databases to reduce reading to writing. Here, we show how to do this using only $k+1$ databases. Again suppose that we utilize a k-database scheme for private queries. To build our database for private queries and semi-private updates, we use $k+1$ databases. Each bit in the database is split into $k+1$ random bits, or "shares," whose exclusive-or is the value of the bit. Number the databases and shares 1 through $k+1$. Then share i is given to all databases except database i. Thus, each database contains only k of the shares, which keeps the value of the bit private. To read a bit from the database, one needs to obtain all $k+1$ shares. To obtain share i, one uses the k-database scheme for private queries on all databases other than database i. This gives rise to a $(k+1)$-database scheme for private queries and semi-private updates whose communication complexity is $O(k)$ times that of the underlying k-database scheme for private queries. Notice also that the sizes of each of the $k+1$ databases is $O(kn)$.

Combining the above observations with the results in the previous section, we obtain the results for private queries and updates for theorem 1. ■

5 Conclusion

We have given several constructions for distributed databases that support private queries and updates and essentially shown that private information storage is within poly-log factors from private information retrieval. One of our schemes achieves a bit complexity of $O\left(n^{1/3}(\log n)^{O(1)}\right)$ using just three databases. An open question is whether a sub-linear read/write communication complexity can be achieved with just two databases, the minimal number of databases for which private updates are at all possible.

References

[AFK-89] Abadai M., J. Feigenbaum, and J. Kilian "On Hiding Information from an Oracle" JCSS, Vol 39, No 1, 1989, pp.21-50.

[N-89] Adam N., and J. Wortmann ""security Control Methods for Statistical Databases': A Comparative Study" ACM Computing Surveys, Vp; 21. No. 4, pp. 515-555, 1989.

[BF-90] Beaver D., and J. Feigenbaum "Hiding Instances in Multioracle queries" Proc. of 7'th STACS, Springer-Verlag LNCS, Vol. 415, pp.37-48, 1990.

[BFKL-90] Beaver D., J. Feigenbaum, J. Kilian and P. Rogaway "Security with Low Communication Overhead" Crypto 1990.

[B-79] Blakley G. R. "Safeguarding Cryptographic Keys", Proc. NCC AFIPS 1979, pp. 313-317, 1979.

[CGKS-95] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, "Private Information Retrieval" Proc. of 36'th IEEE conference on the Foundations of Commuter Science (FOCS), pp. 41-50, October 1995. (Journal version submitted to JACM in January 1996)

[RAD-78] Rivest R.L., L. Adleman, and M.L. Dertouzos, "On Data Banks and Privacy Homomorshims", Foundations of Secure Computation (eds., R. DeMillo, D. Dobkin, A. Jones, and R. Lipton). Academic Press, 1978.

[G-87] Goldreich, O. "Towards a Theory of Software Protection and simulation by Oblivious RAMs" *STOC 87*.

[GO-92] Goldreich, O. and R. Ostrovsky "Comprehensive Software Protection System" U.S. Patent No. 5,123,045 (issued Jun. 16th 1992).

[GO-96] Goldreich, O. and R. Ostrovsky "Software Protection and Simulation by Oblivious RAMs" Manuscript, accepted to JACM (to appear in 1996).

[M-92] Micali, S. "Fair Public-Key Cryptosystems" Crypto 92, LNCS Vol 740, pp. 113-138.

[Ost-90] Ostrovsky, R. "Software Protection and Simulation on Oblivious RAMs" M.I.T. Ph.D. thesis in Computer Science, June 1992. Preliminary version in *STOC*, 1990.

[S-79] "How to Share a Secret", Comm. ACM, Vol 22, 1979, pp. 612-613.

[U.S.-93] Clipper Chip, U.S. Clinton administration government announcement, 1993.