Public-Key Encryption with Efficient Amortized Updates

Nishanth Chandran[*] Rafail Ostrovsky[†] William E. Skeith III[‡]

Abstract

Searching and modifying public-key encrypted data (without having the decryption key) has received a lot of attention in recent literature. In this paper we re-visit this important problem and achieve much better amortized communication-complexity bounds. Our solution resolves the main open question posed by Boneh at al., [BKOS07].

First, we consider the following much simpler to state problem (which turns out to be central for the above): A server holds a copy of Alice's database that has been encrypted under Alice's public key. Alice would like to allow other users in the system to replace a bit of their choice in the server's database by communicating directly with the server, despite other users not having Alice's private key. However, Alice requires that the server should not know which bit was modified. Additionally, she requires that the modification protocol should have "small" communication complexity (sub-linear in the database size). This task is referred to as private database modification, and is a central tool in building a more general protocol for modifying and searching over public-key encrypted data with small communication complexity. The problem was first considered by Boneh at al., [BKOS07]. The protocol of [BKOS07] to modify 1 bit of an N-bit database has communication complexity $\mathcal{O}(\sqrt{N})$. Naturally, one can ask if we can improve upon this. Unfortunately, [OS08] give evidence to the contrary, showing that using current algebraic techniques, this is not possible to do. In this paper, we ask the following question: what is the communication complexity when modifying L bits of an N-bit database? Of course, one can achieve naive communication complexity of $\mathcal{O}(L\sqrt{N})$ by simply repeating the protocol of [BKOS07], L times. Our main result is a private database modification protocol to modify L bits of an N-bit database that has communication complexity $\mathcal{O}(\sqrt{NL^{1+\alpha}}\text{poly-log } N)$, where $0 < \alpha < 1$ is a constant. (We remark that in contrast with recent work of Lipmaa [Lip08] on the same topic, our database size *does not grow* with every update, and stays exactly the same size.)

As sample corollaries to our main result, we obtain the following:

- First, we apply our private database modification protocol to answer the main open question of [BKOS07]. More specifically, we construct a public key encryption scheme supporting PIR queries that allows every message to have a non-constant number of keywords associated with it.
- Second, we show that one can apply our techniques to obtain more efficient communication complexity when parties wish to increment or decrement multiple cryptographic counters (formalized by Katz at al. [KMO01]).

We believe that "public-key encrypted" amortized database modification is an important cryptographic primitive in it's own right and will be a useful in other applications.

[*]Department of Computer Science, University of California, Los Angeles. Email: nishanth@cs.ucla.edu
[†]Department of Computer Science and Mathematics, University of California, Los Angeles. Email: rafail@cs.ucla.edu
[‡]Department of Computer Science, City College, CUNY. Email: wes@cs.ccny.cuny.edu

1 Introduction

The problem of private database modification was first studied in the context of public key encryption supporting private information retrieval (PIR) queries by Boneh et al. [BKOS07]. The private database modification protocol of [BKOS07] requires communication complexity $\mathcal{O}(\sqrt{N})$ to modify 1 bit of an N-bit database. Furthermore Ostrovsky and Skeith showed in [OS08], that using currently known algebraic techniques (which will not increase the database size after an update), one cannot obtain better communication complexity to modify a single bit. Hence, we turn to the question of modifying multiple bits of the database. Using repeated application of the protocol from [BKOS07], one can obtain a private database modification protocol to modify L bits with communication complexity $\mathcal{O}(L\sqrt{N})$.

Lipmaa, in [Lip08], also considered the question of amortizing the communication complexity of private database modification. However, his protocol has a significant drawback. In [Lip08], the size of the database increases with every update made and after only $\mathcal{O}(\sqrt[4]{N}/\log^4 N)$ bits have been updated in the database, Alice (the owner of the database) needs to download the entire database and re-send a new encrypted database, for the protocol to have efficient communication complexity thereafter. We shall see a little later that in applications of the private database modification protocol, this drawback is significant.

Main Result

Our main contribution in this paper is a pair of amortized protocols that have communication complexity $\mathcal{O}(\sqrt{L^{1+\alpha}N}\text{poly-log } N)$ when modifying L bits of the database (where $0 < \alpha < 1$ is a constant), *without ever increasing the size of the database, regardless of the number of updates.* The two protocols have slightly different properties and play a different role in applications.

The first protocol we present gives an amortization of the communication complexity when modifying a 0 bit into a 1 bit as well as vice-versa, where as (a somewhat simpler) second protocol gives an amortization of the communication complexity only when modifying a 0 bit into a 1 bit. Although we believe the amortized protocols for database modification to be of independent interest, we also describe two results that we obtain as corollaries of our main result.

Applications

Our first application is an answer to the main open question of [BKOS07] (resolving the main drawback of their solution.) Recall that in [BKOS07], a private database modification protocol was used to construct a public key encryption scheme supporting PIR queries. An illustrative example of this concept is that of web-based email. Suppose that Alice stores her email on the server of a storage provider Bob (as is the case for a Yahoo! or Hotmail email account, for example). Bob must provide Alice with the ability to collect, retrieve, search and delete emails but at the same time learn nothing about the contents of the email nor the search criteria used by Alice. For example, a user might send an encrypted email to Alice (via Bob) that is marked with the keyword "Urgent". Bob should store this email without knowing that a user sent an email marked as urgent. Later, Alice should have the ability to retrieve all email messages marked with "Urgent" without Bob knowing what search criteria Alice used. The goal was to obtain communication efficient protocols for this task. This goal was accomplished by making use of a protocol for private database modification in order to mark emails as containing certain keywords. However, in order to keep the communication complexity of the protocol sub-linear, the authors of [BKOS07] put constraints on the number of keywords that could be attached to a single message. In particular, this number was forced to be a constant. We can directly apply our batch update protocol to remove this constraint, allowing for non-constant numbers of keywords to be associated to a single message (which may often be the case for large messages). Note, that if we were to use a modification protocol in which the size of the database increased with every update (such as the one in [Lip08]), then Alice needs to frequently download her entire email and send an updated database back to Bob, so that further executions of the modification protocol can have efficient communication complexity. This defeats the entire purpose of having an email system supporting oblivious collect, retrieve, search and delete queries. Alice could simply achieve all these queries in an oblivious manner when she downloads the entire database. Furthermore, [Lip08] requires that message senders be aware of a certain aspect of the state of the email database (in particular, the number of layers of encryption) before they send a message and perform updates to the database to mark the message with keywords. This would appear to force an interactive protocol for message sending, which seems undesirable and need not be the case (as shown in [BKOS07]).

A second use of our main result is a protocol for amortized updates of cryptographic counters. Cryptographic counters, formalized by [KMO01], allow a group of participants to increment and decrement a public-key encrypted cryptographic representation of a hidden numeric value, stored on a server, in such a way that the server can not observe the value of the counter. The value of the counter can then be determined only by someone with a private key. Cryptographic counters can be used in several electronic voting protocols [CF85, BY86, CFSY96, CGS97, Sch99, DJ00]. One can imagine a situation in which parties would like to modify not one cryptographic counter but several such counters. For example, there could be a total of N counters and every party could wish to modify at most L of them. This could be seen in situations where every voter must vote for at most L out of N candidates and possibly also specify a ranking of the L selected candidates. We show how to obtain a communication efficient protocol for batch cryptographic counters (better than the batch protocol that is obtained through the trivial repetition of existing protocols). Once again, we cannot use a modification protocol in which the size of the database increases with every update. This is because, after every party updates the counters (or casts his or her votes), a party holding the private key, must get the value of the counter and send a new "updated" value to the server. Clearly, privacy of the protocol would be lost here.

Our Techniques and high-level outline of our constructions

Our starting point are the techniques from Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS04] on batch codes. Before we explain the main ideas of our construction (and in particular why batch codes do not apply directly) we need to give short background on batch codes.

Recall that batch codes of [IKOS04] are used for encoding an N-bit database on M different servers such that a user could read L bits of the N-bit database, with the following properties:

- The user reads the same number of bits from every server when retrieving L bits.
- The total storage of the M servers is minimized. In other words, the size of the encoding is minimized.

At a high level, Ishai et al., make D copies of every bit and store each copy on a different server. To decide the server on which the r^{th} copy of a bit should be stored, they use an expander graph for the encoding.

We note that we cannot apply the construction of Ishai et al., in our setting: in the setting of Ishai et al., the problem is to read a bit of the (static) database, and reading any of the D copies of the bit gives the correct value. However, while writing a bit of the database, modifying one copy out of D copies does not yield a correct solution and modifying all eliminates all the efficiency savings.

We provide two approaches to solving this tantalizing problem. In our first solution, we use a different encoding of a bit of the database. We encode every bit of the database through D bits such that the *majority* of the D bits decodes to the bit in the database. The D copies of the bit will again be on M different "virtual" databases. However, now in order to modify a bit in the database, one needs to modify a majority of the copies of the bit. Unfortunately, using an expander (as in [IKOS04]) as the encoding *does not* allow us to enjoy the property that a user reads the same number of bits from every virtual database. However, it turns out that careful use of *lossless* expanders for our encoding achieves the desired saving. This requires us to prove that for every set of L bits, one can modify a majority of each of the bits in the encoding by modifying the same (small) number of bits on each virtual database.

The solution that we then obtain gives us an amortization on the communication complexity when we modify a 0 bit into a 1 as well as vice-versa.

Our second approach builds on the techniques of [IKOS04] as follows. We change the base-level representation of bits as follows. We do not store D copies of every bit, but instead encode every bit through a set of D bits. Through a careful choice of the encoding, we can eliminate the problems that arise when we try to apply the techniques of [IKOS04] directly. We encode a 0 through a set of D zeroes (i.e. all D zeroes) on the "virtual databases" and a 1 through **any other** set of D bits. We then change the construction of [IKOS04] to use an explicit unbalanced expander from [GUV07]. Now, to modify an encoding of 0 to an encoding of 1, we only need to modify any single bit of the encoding which allows us to carry analysis similar to [IKOS04]. However, this idea prevents us from modifying a 1 bit back into a 0, as one might need to modify **all** bits of an encoding. This prevents us from obtaining an amortization on the communication complexity in this case. However, in certain applications (though not all) just modifying bits from 0 to 1 is sufficient, by designing additional machinery on top of the "crippled" database modification protocol. We note, however, that our second construction is strictly weaker in its properties than the first one that uses lossless expanders.

Organization of the paper. We begin with a description of the private database modification protocol of [BKOS07] in §2. In §3, we describe our main result, the private database modification protocol for modifying L bits with amortized communication complexity. In §4, we show how to use the amortized private database modification protocol to obtain a public key encryption scheme supporting PIR queries with non-constant number of keywords associated with each message. Finally, in §5, we show how to apply the amortized modification protocol to obtain an amortization of cryptographic counters.

2 Background: Private Database Modification with Sub-linear Communication

Consider the following problem. A server is holding a database of Alice's, which has been encrypted under her public key. Alice would like to allow her friends to use her public key to update a bit (of their choice) in the database by communicating directly with the server. However, she requires the following conditions:

1. The details of each modification are hidden from the server. That is, without the private key, each transaction for an update is computationally indistinguishable from any another.

2. Her friends need only a "small" amount of communication (sub-linear in the database size) in order to perform an update of a single bit.

For a database of size N, we'll call a protocol for privately updating a single bit $\mathsf{Update}(N, 1)$. The first such protocol which satisfied the above requirements was developed in [BKOS07]. The [BKOS07] protocol made use of a homomorphic cryptosystem that allows computation of polynomials of total degree 2 on ciphertexts (due to Boneh et al. [BGN05]). That protocol has communication complexity $\mathcal{O}(\sqrt{N})$ for updating a single bit.

If one of Alice's friends wishes to modify L bits of the database, then he or she can simply run $\mathsf{Update}(N, 1)$ L times sequentially, which will of course come at a $\mathcal{O}(L\sqrt{N})$ cost in communication complexity. Hence, if $\Omega(\sqrt{N})$ updates are to be made at once, the total communication becomes $\Omega(N)$, making the scheme no better than communicating a new encrypted database in its entirety.

In this work, we present an oblivious database modification protocol that amortizes the communication complexity of modifying L bits of the database. Our protocol, which we will denote by $\mathsf{Update}(N, L)$ has communication complexity $\mathcal{O}(\sqrt{NL^{1+\alpha}}\text{ poly-log}(N))$, where $0 < \alpha < 1$ is a constant. We begin, however, with a brief summary of the $\mathsf{Update}(N, 1)$ protocol from [BKOS07].

The Database Modification Protocol of [BKOS07]

The work of [BKOS07] provides a simple solution to the problem of low-communication, privacy-preserving modification of an encrypted database. The main ingredient of the protocol was the cryptosystem of [BGN05]. This cryptosystem has a set of integers as its plaintext set, and has the interesting homomorphic property that arbitrary multivariate polynomials of *total degree 2* can be computed on ciphertext. That is, given an array of u ciphertext values, call them $c_i = \mathcal{E}(r_i)$, and a u-variate polynomial f of total degree two, a party with only public information can compute some function $\widetilde{f}(c_1, ..., c_u)$ (in polynomial time) such that $\mathcal{D}(\widetilde{f}(c_1, ..., c_u)) = f(r_1, ..., r_u)$. The following paragraph outlines an $\mathsf{Update}(N, 1)$ protocol with $\mathcal{O}(\sqrt{N})$ communication complexity.

To begin, note that for matrices A, B, C, one can express the function $AB + C$ as a polynomial of total degree two in the matrix entries (provided of course that the dimensions match). Next, represent the database as a two dimensional square matrix (adding padding if needed). In order to specify the modification of a particular bit, send two vectors of ciphertext, A and B of length $\sqrt{N}$ which represent an encryption of the characteristic vectors of the coordinates of the bit in the database that is to be updated. Treating these vectors of ciphertext as $\sqrt{N} \times 1$ and $1 \times \sqrt{N}$ matrices, respectively, we can multiply them together and produce a $\sqrt{N} \times \sqrt{N}$ matrix that is an encryption of the identity element in every position except for one: the location to be updated. Now, this matrix of ciphertext can be added to the original database (which we'll call C), and exactly one of the N underlying plaintext values will be modified, yet only $2\sqrt{N}$ ciphertext values were transmitted. Since this computation requires nothing more than the evaluation of a polynomial of total degree 2 on ciphertext (we just computed $AB + C$ for the matrices described above), it can be accomplished using the cryptosystem of [BGN05] in a black-box way, just manipulating the homomorphic properties. Essentially, this completes the description of $\mathsf{Update}(N, 1)$.

3 Private Database Modification with Batches

In this section, we describe how one can amortize the communication complexity when running a private database modification protocol to modify L bits. In other words, let $\mathsf{Update}(N, 1)$ denote a protocol for private database modification to modify 1 bit of an N-bit database and let the communication complexity of $\mathsf{Update}(N, 1)$ be denoted by $C_{N,1}$. Let $\mathsf{Update}(N, L)$ denote a private database modification protocol to modify L bits of an N-bit database and let the communication complexity of $\mathsf{Update}(N, L)$ be denoted by $C_{N,L}$. We construct two protocols for $\mathsf{Update}(N, L)$, such that $C_{N,L} < LC_{N,1}$ for sufficiently large values of L.

We first begin with the description of our security game for privacy. We assume a semi-honest adversary $\mathcal{A}$ that runs in probabilistic polynomial time (PPT). Informally, a semi-honest adversarial server should not have any knowledge about which L bits a user modified in the database and also what values these bits were modified to.

The security game for privacy is defined through the two experiments (0 and 1) below. Let W_b denote the probability with which $\mathcal{A}$ outputs 1 in Experiment b for $b = 0, 1$.

1. In both experiments,
   (a) The challenger picks the public key of the encryption scheme pk and sends it to $\mathcal{A}$.
   (b) $\mathcal{A}$ sends an N-bit string denoting the database to the challenger.
   (c) The challenger encrypts these N bits using pk and sends it to $\mathcal{A}$.
   (d) $\mathcal{A}$ picks a set $S \subseteq [N]$ of size L. For every index in S, $\mathcal{A}$ denotes the value to which the bit in the index must be modified to. The challenger runs $\mathsf{Update}(N, L)$ with $\mathcal{A}$ using S as input. This step may be repeated polynomially many times.
   (e) $\mathcal{A}$ picks 2 sets S_0 and $S_1 \subseteq [N]$, such that $|S_0| = |S_1| = L$. Also, for every index in S_0 and S_1, $\mathcal{A}$ denotes the value to which the bit in the index must be modified to ($\mathcal{A}$ may also specify that this bit not be changed).

2. In Experiment b, the challenger runs $\mathsf{Update}(N, L)$ with $\mathcal{A}$ using S_b as input.

3. $\mathcal{A}$ outputs a bit 0 or 1.

Definition 1 *We say that* $\mathsf{Update}(N, L)$ *is* L*-private, if for all semi-honest PPT adversaries* $\mathcal{A}$*, we have* $|W_0 - W_1|$ *is negligible.*

Theorem 1 [BKOS07] $\mathsf{Update}(N, 1)$*, described in §2, is* 1*-private.*

Let $\mathsf{Update}^*(N, 1)$ denote a private database modification protocol in which a user runs the modification protocol but does not modify any bit of the database. Theorem 1 also implies that $\mathcal{A}$ cannot distinguish an execution of $\mathsf{Update}^*(N, 1)$ from an execution of $\mathsf{Update}(N, 1)$.

We first begin with some background on expanders in §3.1. In §3.2, we describe our first solution and in §3.3, we describe our second solution.

3.1 Expander graphs

Expanders are graphs that are sparse but highly connected. Expanders have had several applications in computer science (see for example the survey of [HLW06]).

Definition 2 *A bipartite multi-graph with* $N = 2^n$ *left vertices and* $M = 2^m$ *right vertices, where every left vertex has degree* $D = 2^d$*, can be specified by a function* $\Gamma : [N] \times [D] \rightarrow [M]$*, where* $\Gamma(u, r)$ *denotes the* r^{th} *neighbor of vertex* $u \in [N]$*. For a set* $S \subseteq [N]$*, we write* $\Gamma(S)$ *to denote the set of neighbors of* S*. That is,* $\Gamma(S) = \{\Gamma(x, y) : x \in S, y \in [D]\}$*. Let* $|\Gamma(S)|$ *denote the size of the set* $\Gamma(S)$*.*

Definition 3 *A bipartite graph* $\Gamma : [N] \times [D] \rightarrow [M]$ *is a* (L, A) *expander, if for every set* $S \subseteq [N]$*, with* $|S| = L$*, we have* $|\Gamma(S)| \geq A \cdot L$*.* Γ *is a* $(\leq L_{max}, A)$ *expander if it is a* (L, A) *expander for every* $L \leq L_{max}$*.*

An expander is unbalanced if $M << N$.

Definition 4 *A* $(\leq L_{max}, A)$ *expander* $\Gamma : [N] \times [D] \rightarrow [M]$ *is a* (L_{max}, ϵ) *lossless expander if* $A = (1 - \epsilon)D$*.*

We refer the reader to [CRVW02, HLW06, TSUZ07, GUV07] for further details on lossless expanders.

3.2 Protocol using unbalanced lossless expanders

Our solution, uses techniques from the work of Ishai et al. [IKOS04] on batch codes and their applications. Ishai et al. considered the problem of encoding an N-bit database on M different servers such that a user could read L bits of the N-bit database, with the following properties:

- Equalizing the number of bits that the user reads from each server.
- Minimizing the total storage of the M servers. In other words, minimizing the size of the encoding.

The idea in Ishai et al. is as follows. Make D copies of every bit in the N-bit database. The parameters D and M are picked such that $\Gamma : [N] \times [D] \rightarrow [M]$ is a $(\leq L_{\max}, A)$ expander for some $A > 1$. Now, the ND bits are distributed among the M servers according to the expander graph. In other words, the r^{th} copy of bit $i \in [N]$ of the database is stored in database $\Gamma(i, r)$. Now one can show that to read any L bits of the N-bit database (with $L \leq L_{\max}$), one only needs to read at most 1 bit from each of the M servers. So, by reading 1 bit from each of the M servers, the first property above can be satisfied. The bound on the total storage of the M servers is obtained through the expansion property of Γ, thus satisfying the second property above.

Note that [IKOS04], do not consider the problem of modifying bits of a database. The encoding in [IKOS04] works because in order to read a bit from the N-bit database, one only needs to read any copy of that bit. The encoding does not directly apply in our setting as modifying 1 bit out of the D bits that encode a bit does not result in a correct modification.

At a high level, our protocol for private database modification to modify $L_{\max}$ bits of an N-bit database is as follows. We encode every bit of the database through D bits. The majority value of these D bits decodes to the original bit in the database. The resulting ND bits from the encoding are distributed into M "virtual" databases according to a (L_{max}, ϵ) *lossless* expander graph Γ. Let the number of bits in each of the M virtual databases be denoted by $a_1, a_2, \cdots, a_M$.

We will then show that to modify a majority of each of the bits in any set of $L_{\max}$ bits of the N-bit database, one only needs to modify at most 1 bit from each of the M virtual databases. One can modify 1 bit from each of the M virtual databases using $\mathsf{Update}(a_i, 1)$ for all $1 \leq i \leq M$. The bound on the communication complexity of the protocol will be obtained through the lossless expansion property of Γ.

While reading a bit from the N-bit database, one reads all D bits that encode this bit from the M virtual databases and takes the majority value. We first describe how to create the virtual databases.

Creating virtual databases

Consider a (L_{max}, ϵ) lossless expander $\Gamma : [N] \times [D] \rightarrow [M]$ as defined in §3.1. Let $L = 2^l$ and $L_{max} = 2^{l_{max}}$.

Every node $u \in [N]$ represents a bit in the database and the $D = 2^d$ neighbors of the node u are the encoded bits of u. For bit $u \in [N]$, the r^{th} bit of the encoding of u is present in database $\Gamma(u, r)$. To read the value of bit $u \in [N]$, one reads all D bits of the encoding of u and takes the majority of these values as the value of u. Hence, note that to modify bit u, one has to modify a majority of the bits that encode u. Below, we show that this can be done by modifying at most 1 bit in every virtual database.

Note that for all sets $S \subseteq [N]$ with $|S| = L \leq L_{\max}$, we have $\Gamma(S) \geq (1-\epsilon)LD$. To modify bits from a set S, we show that there is a strategy to modify at least $(1-\epsilon)D$ bits of the encodings of all the bits in S by modifying at most 1 bit in each of the virtual databases.

Lemma 3.1 *Let Γ be a lossless expander as above. Then for every subset $S \subseteq [N]$ where $|S| = L \leq L_{max}$, the number of nodes $v \in [M]$ that have exactly one neighbor in S (v is then called a* unique neighbor node *with respect to S) is at least $(1-\epsilon)LD$.*

Proof: Suppose there exists a set $S \subseteq [N]$ where $|S| = L \leq L_{max}$, such that the number of nodes $v \in [M]$ that have exactly one neighbor in S is less than $(1-\epsilon)LD$. This means, that at least ϵLD nodes in $[M]$ have two or more neighbors in S. This means that $\Gamma(S) < \epsilon LD + LD - 2\epsilon LD = LD(1-\epsilon)$ which is a contradiction to the property of expansion of lossless expanders. Hence, the lemma. $\square$

Lemma 3.2 *Fix any set $S \subseteq [N]$ where $|S| = L \leq L_{max}$. Let $g_S(v)$ for all $v \in [M]$, be a function such that $g_S(v) = \text{NIL}$ or $g_S(v) = u$ such that $u \in S$ and there exists $r \in [D]$ such that $\Gamma(u, r) = v$. In other words, $g_S(v)$ is either NIL or a neighbor of v in S. Let $h_S(u) = |g_S^{-1}(u)|$ for all $u \in S$. That is, $h_S(u)$ is the number of $v \in [M]$ such that $g_S(v) = u$. There exists polynomial time computable function $g_S(\cdot)$, such that $h_S(u) \geq (1-\epsilon)D$ for all $u \in S$.*

Proof: We shall construct $g_S(\cdot)$ as follows:

1. Let $S' = S$. A node $u \in S'$ is *satisfied* if $h_S(u) \geq (1-\epsilon)D$.

2. For every node $v \in [M]$, let $g_S(v) = u$ if u is the only neighbor of v in S'. Let H denote the set of nodes in S' that are satisfied.

3. Set $S' = S - H$. If S is not empty, repeat Step 2, otherwise halt, setting $g_S(v)$, for all unassigned nodes v, to NIL.

We will prove that at every iteration of the algorithm, a constant number of the nodes in S' are satisfied. This means the algorithm will halt in time $\mathcal{O}(\log L)$. In this case, every node $u \in S$ is satisfied and hence $h_S(u) \geq (1-\epsilon)D$ for all $u \in S$.

Let $|S'| = L'$. We will show that $|H| \geq \frac{L'}{\epsilon D+1}$. Let $|H| = h$. Let l be the number of unique neighbor nodes with respect to S' in $[M]$. We have that $l \geq (1-\epsilon)L'D$ (By Lemma 3.1).

Now, consider a *satisfied* node $u \in S'$. The number of unique neighbor nodes with respect to S' in $[M]$ that have their unique neighbor as u can be at most D. This is because the degree of every node in U is at most D.

Consider a node $u \in S'$ that is not satisfied. The number of unique neighbor nodes with respect to S' in $[M]$ that have their unique neighbor as u can be at most $(1-\epsilon)D - 1$. Otherwise, u would be satisfied. (This is because at no stage of the algorithm did we assign $g_S(v)$ to be u when v was also a neighbor of a node $u' \in S$ that was not already satisfied.)

Hence we have $l \leq hD + (L-h)((1-\epsilon)D-1)$. Since $l \geq (1-\epsilon)L'D$, we have that $hD + (L-h)((1-\epsilon)D-1) \geq (1-\epsilon)L'D$, which means $h \geq \frac{L'}{\epsilon D+1}$. □

We note that the above proof is similar in flavor to the proof of error correction in linear time encodable/decodable expander codes (Refer [SS96, CRVW02] for further details.)

Our protocol uses the specific lossless expander explicit construction from [GUV07]. We state the theorem below.

Theorem 2 [GUV07] *For all constants $\alpha > 0$, every $N \in \mathbb{N}$, $L_{max} \leq N$, and $\epsilon > 0$, there is an explicit $(\leq L_{max}, (1-\epsilon)D)$ expander $\Gamma : [N] \times [D] \to [M]$ with degree $D = \mathcal{O}((\log N)(\log L_{max})/\epsilon)^{1+1/\alpha}$ and $M \leq D^2 \cdot L_{max}^{1+\alpha}$. Moreover D is a power of 2.*

Protocol Description

We now describe the private database modification protocol $\mathsf{Update}(N, L_{\max})$.

1. Create M smaller databases according to lossless expander Γ from Theorem 2 and encode the bits of the database into the M smaller databases as described earlier. Let size of database $v \in [M]$ be denoted by a_v.

2. To modify a set $S \subseteq [N]$ of bits of the database with $|S| = L_{max}$, create $h_S(v)$ as described in Lemma 3.2.

3. Run $\mathsf{Update}(a_v, 1)$ to modify bit $h_S(v)$ in database v for all databases $v \in [M]$. If $h_S(v) = \text{NIL}$, then run $\mathsf{Update}^*(a_v, 1)$ with database v.

Protocol Correctness, Privacy and Communication Complexity

The correctness of the protocol $\mathsf{Update}(N, L_{\max})$ follows trivially from Lemma 3.2 and from Theorem 1.

Theorem 3 $\mathsf{Update}(N, L_{\max})$ *is $L_{\max}$-private.*

Proof: Lemma 3.2 shows that the number of bits we modify in each of the M virtual databases is independent of the subset of $L_{\max}$ bits we wished to modify in the original database. In particular, in each virtual database, we either modify 1 bit by running $\mathsf{Update}(a_v, 1)$ or do not modify any bits by running $\mathsf{Update}^*(a_v, 1)$. Now, by Theorem 1, $\mathsf{Update}(a_v, 1)$ is 1-private. Hence, no adversary can distinguish between the case when we run $\mathsf{Update}(a_v, 1)$ and modify a bit and when we run $\mathsf{Update}^*(a_v, 1)$. Hence, it follows that $\mathsf{Update}(N, L_{\max})$ is $L_{\max}$-private. □

We now analyze the communication complexity. Note that if a_v is the number of bits in the

v^{th} smaller database, then the communication complexity of $\mathsf{Update}(N, L_{\max})$, $C_{N,L_{\max}} = \sum_{i=1}^{M} C_{a_i,1}$. If we use the private database modification protocol to modify 1 bit from Boneh et al. [BKOS07], we have $C_{a_i,1} = \mathcal{O}(\sqrt{a_i})$. We also have $\sum_{i=1}^{M} a_i = ND$. Hence by Cauchy-Schwartz inequality, it follows that $C_{N,L_{\max}} = \mathcal{O}(\sqrt{NDM})$. Setting the parameters according to Theorem 2, we get the communication complexity to be $\mathcal{O}(\sqrt{NMD}) = \mathcal{O}(D^{3/2}\sqrt{NL_{max}^{1+\alpha}})$, where D is poly-log in N and $0 < \alpha < 1$ is a constant. This is an improvement upon the repeated application of the protocol of [BKOS07], when $L_{\max}$ is Ω(poly-log N).

Maintaining consistencies over different values of L_{max}

Note that if we run $\mathsf{Update}(N, L_{\max})$ when we wish to modify L bits in the database with $L < L_{\max}$, then the communication complexity is not optimal as the communication complexity depends only on L_{max} and not on L. For example, if we have $L_{max} = \mathcal{O}(\sqrt{N})$, then running $\mathsf{Update}(N, L_{\max})$ when we want to modify $L = \mathcal{O}(\sqrt[4]{N})$ bits, will not be optimal. $\mathsf{Update}(N, L_{\max})$ will then have communication complexity $\mathcal{O}(D^{3/2}\sqrt[3]{N^{4+4\alpha}})$, which is more than the communication complexity when running $\mathsf{Update}(N, 1)$, $\mathcal{O}(\sqrt[4]{N})$ times.

Now, if we use different lossless expanders for the encoding depending on the number of bits we wish to modify, then the encoding of each bit of the original database will not be consistent. More specifically, since the degree of the graphs are different in different cases, we may not modify "enough" copies of a particular bit.

To overcome this difficulty, we pick $W = \mathcal{O}((\log N)^2/\epsilon)^{1+1/\alpha}$ copies of every bit and store them. Now, for all values of $L_{max} \leq N$, the corresponding value of D is $\leq W$. When we wish to modify $L_{\max}$ bits of the original database, we use the corresponding lossless expander with degree D. We repeat this protocol $\lceil \frac{W}{D} \rceil$ times using a different set of D copies of every bit in each iteration. Now, since in each execution we modify at least $(1-\epsilon)D$ copies of a bit, in total we will modify at least $(1-\epsilon)W$ copies of every bit that we modify and hence the decoding of majority still works. Furthermore, we increase the communication complexity by only a factor of $\lceil \frac{W}{D} \rceil$, which is still poly-log in N.

3.3 Protocol using unbalanced expanders

In this section, we outline a second protocol that also obtains an amortization on the communication complexity when modifying $L_{\max}$ bits of an N-bit database.

This solution is also based on the work of Ishai et al. [IKOS04] on batch codes. We modify our encoding of every bit of the database. We again use D bits to represent a single bit in the N-bit database and will store the ND bits in M virtual databases. Now a set of D 0's in the M virtual databases decode into a 0 in the N-bit database and any other set of D bits decode into a 1 in the N-bit database.

Now, note that in order to modify a 0 bit into a 1, the user only needs to modify at most 1 out of the D bits that encode the bit. Following work from Ishai et al. [IKOS04], it follows that one can modify 1 copy of each bit in a set of $L_{\max}$ bits of the N-bit database by modifying at most 1 bit in each of the M virtual databases. Again using the explicit unbalanced expander from Guruswami et al. [GUV07], one can obtain the same communication complexity as in the protocol described in §3.2. Here, we note that we do not require the expander to be lossless, but only that it is unbalanced.

Again, in order to use the protocol with different values of $L_{\max}$, we store more copies of each bit and use the same solution as described earlier in §3.2.

An important remark: Suppose, we know the contents of the database, and wish to modify L bits that are all 0 into 1s. Then both protocols (described in §3.2 and §3.3) can be used to achieve an amortization of the communication complexity as described earlier. Now, suppose we know the contents of the database (including that of the encoding) and wish to modify L bits, that maybe either 0 or 1. Then, only the protocol in §3.2 gives us an amortization on the communication complexity. This is because of the following reason. In the solution described in §3.2, each bit is encoded through D bits and the majority of the D bits decode to the bit in the N-bit database. Now, irrespective of whether we are changing a 0 bit into a 1, or vice-versa, we always need to only modify at most a majority of the D copies of the bit. Hence, the protocol in §3.2 works. However, if we were to use the protocol in §3.3, then modifying a bit that is 1 into a 0 would require the modification of all D copies. This is because a 0 bit is encoded as D 0s and a 1 bit by all other strings. In order to ensure correctness, if a 1 bit is encoded by a string of D 1s, then we would have to modify all the D bits of the encoding to modify this bit to 0. Since, we do not want to reveal which bit of the database we are modifying, we would then have to modify all D bits in the encoding for all the L bits that we want to modify. This ends up having the same communication complexity as running $\mathsf{Update}(N, 1)$ sequentially L times. Hence, the protocol in §3.2 is strictly more powerful than the one described in §3.3.

4 Applications of Batch Protocols for Database Modification to [BKOS07]

The protocol of [BKOS07] applies to a scenario that models a somewhat ideal internet-based email service: all email messages are encrypted under the user's public key, yet the user can still perform the common tasks of searching for and retrieving messages via keywords, erasing messages, etc., without revealing any information to the service provider about the messages nor the keywords being searched for. Furthermore, this can be done with "small" (i.e. sub-linear) communication.

The protocols will typically involve a message sender, receiver, and a storage provider. We'll use the following notational conventions to represent the various parties:

$\mathcal{X}$ – refers to a message sending party

$\mathcal{Y}$ – refers to the message receiving party (owner of the private key)

$\mathcal{S}$ – refers to the server/storage provider

The protocol of [BKOS07] accomplished the basic task outlined above, but in order to maintain sub-linear communication complexity as well as to preserve the correctness of the protocol, several limitations were enforced. The most prominent conditions needed were as follows:

1. The number of messages associated to a single keyword must be bounded by a constant.
2. The number of keywords in use at any given time must be proportional to the number of messages.
3. The number of keywords associated to a particular message must be bounded by a constant.

We still enforce conditions 1 and 2 (which apply for the same technical reasons regarding correctness) however using batch protocols for private database modification, we show how to relax the third condition and allow non-constant numbers of keywords to be associated with a single message. Clearly the protocol of [BKOS07] cannot have this capability for a keyword set of size $\Omega(\sqrt{N})$: The expected number of bits one is required to update would similarly be $\Omega(\sqrt{N})$, and $\sqrt{N}$ executions of $\mathsf{Update}(N, 1)$ from [BKOS07] will yield $\Omega(N)$ communication complexity for sending this single message, violating the requirement of maintaining sub-linear communication.

Protocol Description

The details of the protocol are fairly straightforward. Let $\mathcal{K}, \mathcal{E}, \mathcal{D}$ represent the key generation, encryption and decryption algorithms, respectively, of a public key cryptosystem that allows for the evaluation of polynomials of total degree 2 on ciphertext (e.g., [BGN05]). Adopting the notation of [BKOS07], we'll denote the maximum number of keywords that can be associated to a single message by θ. The protocol of [BKOS07] requires that this value in fact be constant. We will make no such assumption on θ and demonstrate a protocol that satisfies the same definitions of correctness and of privacy. Some additional background and definitions from [BKOS07] can be found in Appendix A.1. The work of [BKOS07] presents a public key storage with keyword search that is (N, λ, θ)-correct, where N is the number of messages in the email database, as well as the maximum number of distinct keywords that may be in use at any given time, and λ, θ are constants with θ representing the maximum number of keywords that may be attached to a single message (even if the message is of non-constant size). Below, we extend this protocol to maintain communication efficiency in the case of non-constant θ. In order to simplify the description, we will present the protocol at a high level and refer the reader to the work of [BKOS07] for details when needed. The protocol consists of the following three algorithms.

$\mathsf{KeyGen}(s)$ — Run the key generation algorithm $\mathcal{K}$ of the underlying cryptosystem to produce a public and private pair of keys.

$\mathsf{Send}_{\mathcal{X},\mathcal{S}}(M, W)$ — Sender $\mathcal{X}$ wants to send message M marked with the set of keywords W to $\mathcal{Y}$ via $\mathcal{S}$. $\mathcal{X}$ encrypts M and the keywords and then proceeds as in [BKOS07] in order to update the keyword-message association structure. However, rather than repeatedly applying $\mathsf{Update}(N, 1)$, $\mathcal{X}$ will use the $\mathsf{Update}(N, \theta)$ protocol described in §3.2 to efficiently perform the updates as a batch. Note that in order to mark a message as having a only single keyword, $\mathcal{X}$ is required to update $\Omega(\log^2 N)$ bits of the Bloom filter structure that holds the keyword-message associations.

$\mathsf{Retrieve}_{\mathcal{Y},\mathcal{S}}(w)$ — $\mathcal{Y}$ wishes to retrieve all messages associated with the keyword w and optionally erase them from the server. This protocol consists of steps similar to [BKOS07] in order to decrypt the locations of matching messages and subsequently download and decrypt. However in the case where message erasure is also performed, we will have $\mathcal{Y}$ execute $\mathsf{Update}(N, \theta)$ from §3.2 with $\mathcal{S}$, as opposed to repeated usage of $\mathsf{Update}(N, 1)$ (as found in [BKOS07]) which allows us to handle non-constant numbers of keywords to be associated with a single message.

Theorem 4.1 *The Public-Key Storage with Keyword Search from the preceding construction is (n, λ, θ)-correct according to definition A.8.*

Proof: This follows in much the same way as that of [BKOS07]. However, there is a need for one remark on this subject. Recall that in [BKOS07] it was required that only a constant number of messages were associated to a particular keyword, since fixed-length buffers were needed to represent sets. As mentioned, we have adopted this same requirement for much the same reason. However, we would like to note that in a practical implementation of our protocol this may be harder to achieve since the increased number of keywords per message will naturally lead to more messages being associated with a particular keyword. That is, if θ is large, it will generally not be possible to have *every* message associated to θ keywords without exceeding λ messages associated to some particular keyword. None the less, we emphasize that our protocol conforms to the very same definitions for correctness as that of [BKOS07]; it is just that the antecedent will perhaps not come as easily. $\square$

Theorem 4.2 *Assuming CPA-security of the underlying cryptosystem, the Public Key Storage with Keyword Search from the above construction is sender-private according to definition A.9 as well as receiver-private according to definition A.10.*

Proof: This follows almost immediately from Theorem 1, Theorem 3 and the analogous theorem from [BKOS07]. $\square$

5 Application of Batch Protocols for Database Modification to Cryptographic Counters

Cryptographic counters, formalized by Katz et al. [KMO01], allow a group of participants to increment and decrement a cryptographic representation of a hidden numeric value privately. The value of the counter can then be determined by a specific party only.

More formally, there are a set of R parties, $\{P_1, P_2, \cdots, P_R\}$. These parties wish to increment and decrement the value of a specific counter C which is stored by a party T, assumed to be semi-honest. After they have incremented/decremented the counter C, T must reveal the value of the counter to a specific party denoted by P. T is semi-honest and is trusted not to collude with P. At the same time, parties wish only the output of the counter to be revealed to P. One can implement this protocol in the following way. Let P pick a public/private key pair (pk_P, sk_P) of an additively homomorphic encryption scheme over $\mathbb{Z}_n$ with n larger than the maximum value of the counter. Let P_i hold input x_i. Let $\mathcal{E}(pk, m)$ denote the encryption of message m with public key pk. Now, P_i sends $\mathcal{E}(pk_P, x_i)$ to T. Using the additive homomorphic property, T computes $\mathcal{E}(pk_P, \Sigma_{i=1}^R x_i)$ and sends the result to P who can then compute $\Sigma_{i=1}^R x_i$.

Now, suppose there are N such counters $C_1, C_2, \cdots, C_N$ that the parties wish to update. Furthermore, assume that each user updates no more than L of these N counters. No user P_i wishes to reveal to anyone, which of the counters he/she modified. An example of this situation would be a voting protocol which has several candidates (N candidates). Voters have to select L out of these N candidates and rank them. Candidates are then selected according to a weighted sum of their votes.

Now, let $x_i[1], \cdots, x_i[N]$ denote the inputs (or weighted votes) held by P_i (only at most L out of these values are non-zero). Using the solution described above, P_i can send $\mathcal{E}(pk_P, x_i[1]), \cdots, \mathcal{E}(pk_P, x_i[N])$ to T. Using the additive homomorphic property, T can compute $\mathcal{E}(pk_P, \Sigma_{i=1}^R x_i[1]), \cdots, \mathcal{E}(pk_P, \Sigma_{i=1}^R x_i[N])$ and send the result to P. However, this protocol has communication complexity $\mathcal{O}(N)$ for every user P_i.

Let a cryptographic counter protocol between a user and server T, where there are N counters and every party modifies at most L out of the N counters, be denoted by $\mathsf{Counter}(N, L)$. Let the protocol to transfer an encrypted value of the final counter value from the server T and user P be denoted by $\mathsf{Transfer}(x, y)$, where x and y are inputs of T and P respectively. The privacy game for $\mathsf{Counter}(N, L)$ with respect to server T and the privacy game for $\mathsf{Transfer}(x, y)$ with respect to party P and server T is given in Appendix A.2. We do not focus on the other security requirements such as universal verifiability and robustness ([KMO01]). Universal verifiability informally means that any party (including third parties) can be convinced that all votes were cast correctly and that the tally was made correctly. Robustness informally means that the final output can be computed even in the presence of a few faulty parties.

We now describe below two solutions for $\mathsf{Counter}(N, L)$ that have communication complexity $\mathcal{O}(\text{poly-log } N\sqrt{L^{1+\alpha}N})$.

5.1 Protocol using Lossless Unbalanced Expanders

This protocol, uses the private database modification protocol for modifying L bits of an N-bit database from §3.2. We first note that the additively homomorphic encryption scheme of [BGN05] can be used to encrypt messages from a polynomially large message space (of size n). Choose n such that n is greater than the maximum value of the counter. Now, since the encryption scheme of [BGN05] is additively homomorphic, we can use this scheme in order to encrypt the value of the counter. Every user P_i can update the counter by sending an encryption of their input x_i to T.

Now, we describe below how we can amortize the communication complexity of this protocol. We use the $(\leq L_{max}, (1-\epsilon)D)$ expander Γ from Theorem 2. The protocol requires the number of parties R to be less than $\frac{1}{2\epsilon}$. The protocol $\mathsf{Counter}(N, L)$ is described below.

1. Encode every counter through D different counters. To decode, the simple majority value of all values held in these D counters is the value of the counter. Initially these D counters all hold an encryption of 0 under P's public key according to the encryption scheme of [BGN05].

2. Now, using the protocol from §3.2, each party P_i modifies $(1-\epsilon)$ copies of each of the L counters that he/she wishes to update.

$\mathsf{Transfer}(x, y)$: For each counter, T runs an efficient two-party computation [Yao82, Gol04, IKOS08] with P in order to compute the simple majority value present in these counters and returns the value to P. T's input x is all the D encrypted values of a counter and P's input y is the secret key of the homomorphic encryption scheme.

Protocol correctness, security and communication complexity

We have $R < \frac{1}{2\epsilon}$. We note that since each party modifies at least $(1-\epsilon)D$ copies of every counter, after the first modification to a counter, at least $(1-\epsilon)D$ copies of every counter hold the correct value. Now, after the second modification to the counter at least $(1-2\epsilon)D$ copies of the counter hold the correct value and so on. Since $R < \frac{1}{2\epsilon}$, after all parties have modified the counters, a majority of the counters still hold the correct value of the counter and hence when evaluating the simple majority of the value held in the counter (via the two-party computation protocol between P and T), the output obtained will be correct.

Theorem 4 $\mathsf{Counter}(N, L)$ *is L-private with respect to T according to the definition given in §A.2.*

Proof: This theorem follows from Theorem 3, that guarantees that T cannot tell which of the L counters were modified. $\square$

Theorem 5 $\mathsf{Transfer}(x, y)$ *is private with respect to P and T according to the definition given in §A.2.*

Proof: This theorem follows from the security of the two-party computation protocol (that guarantees that P learns only the output value of the counter) and that T does not learn the value of the counter. $\square$

The communication complexity of $\mathsf{Counter}(N, L)$ for every user is the same as that in §3.2, that is $\mathcal{O}(\sqrt{NL^{1+\alpha}}\text{poly-log } N)$ for some constant $0 < \alpha < 1$. Since, the server can run an efficient two-party computation protocol with P, with inputs of size $\mathcal{O}(\text{poly-log } N)$ to compute the majority value of each counter, the communication complexity of $\mathsf{Transfer}(x, y)$ is $\mathcal{O}(N\text{poly}D)$ which is $\mathcal{O}(N\text{poly-log } N)$.

5.2 Protocol using Unbalanced Expanders

This protocol, uses the private database modification protocol for modifying L bits of an N-bit database from §3.3. $\mathsf{Counter}(N, L)$ is describe below.

1. Encode every counter as D different counters. To decode, compute the sum of all these counters to obtain the value of the counter. Initially these D counters all hold an encryption of 0 under P's public key according to [BGN05].

2. Using the protocol from §3.3, each party P_i modifies any 1 copy of each of the L counters.

$\mathsf{Transfer}(x, y)$: For each counter, the server (using the additive homomorphism property of [BGN05]) computes the decoding of the counter and returns the encrypted value of the counter to P.

Protocol correctness, security and communication complexity

The correctness and privacy of the protocol trivially follows from the correctness and privacy of the private database modification protocol from §3.3. The communication complexity of each user is the same as that in §3.3.

References

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC'05*, pages 325–341, 2005.

[BKOS07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith. Public key encryption that allows PIR queries. In *CRYPTO'07*, pages 50–67, 2007.

[Blo70] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[BSW06] John Bethencourt, Dawn Song, and Brent Waters. New techniques for private stream searching. Technical Report CMU-CS-06-106, Carnegie Mellon University, March 2006.

[BY86] Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 52–62. ACM, 1986.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme. *Symposium on Foundations of Computer Science*, pages 372–382, 1985.

[CFSY96] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In *EUROCRYPT'96*, pages 72–83, 1996.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT'97*, pages 103–118, 1997.

[CRVW02] Michael R. Capalbo, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *IEEE Conference on Computational Complexity*, page 15, 2002.

[DJ00] Ivan Damgard and Mads Jurik. Efficient protocols based on probabilistic encryption using composite degree residue classes, 2000.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA, 2004.

[GUV07] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. In *IEEE Conference on Computational Complexity*, pages 96–108, 2007.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *STOC'04*, pages 262–271, 2004.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *STOC'08*, pages 433–442, 2008.

[KMO01] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. Cryptographic counters and applications to electronic voting. In *EUROCRYPT'01*, pages 78–92, 2001.

[Lip08] Helger Lipmaa. Private branching programs: On communication-efficient cryptocomputing. Cryptology ePrint Archive, Report 2008/107, 2008. `http://eprint.iacr.org/2008/107`.

[OS05] Rafail Ostrovsky and William E. Skeith. Private searching on streaming data. In *CRYPTO'05*, pages 223–240, 2005.

[OS08] Rafail Ostrovsky and William E. Skeith. Communication complexity in algebraic two-party protocols. In *CRYPTO'08*, pages 379–396, 2008.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO'99*, pages 148–164, 1999.

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[TSUZ07] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS'82*, pages 160–164, 1982.

A Appendix

A.1 Background from [BKOS07]

For completeness, we provide some basic background from [BKOS07]. In particular, we briefly describe

- Probabilistic methods for oblivious buffer-writing
- Bloom filters and extensions
- Basic definitions of correctness and privacy

A.1.1 Oblivious Buffer-Writing

Tools like homomorphic encryption allow a party with only public information to modify encrypted data in a meaningful way, even though the underlying data itself remains unreadable. Now consider the problem of multiple users repeatedly writing data into an encrypted buffer in this way. One natural problem is that since the buffer is encrypted, one party may not be aware of what parts of the buffer are empty and what parts are available without some synchronization. Furthermore, such synchronization may reveal information that the parties would prefer to be kept secret. The work of [OS05] proposes a way for parties to write into a reasonably sized buffer ($\mathcal{O}(N \log N)$, where N is the total size of information written) in a way that requires no synchronization amongst the writers, allows the writers to remain completely oblivious as to the existing buffer contents, yet still ensures that with overwhelming probability exactly the data written will be recoverable from the buffer. We'll outline the probabilistic methods from [OS05] and state the required lemmas without proof.

Essentially, the method of [OS05] is to simply write documents into the buffer at uniformly random locations, and employ a few probabilistic methods to overcome the difficulties that arise when documents collide (which is inevitable, given that the buffer size should remain close to the total size of written documents). It is assumed that nothing is recoverable from a collision of documents in the buffer (although with additional work, this is not necessarily the case; see [BSW06] for details). Given this strong assumption, clearly collisions must somehow be avoided. To remedy this problem, multiple copies of each document are placed at uniformly random locations in the buffer. As this number of copies grows, the probability of all copies of a document being unrecoverable diminishes exponentially. This idea is formalized as follows.

Color-survival game: Let $m, \gamma \in \mathbb{Z}^+$, and suppose we have m different colors, call them $\{color_i\}_{i=1}^{m}$, and γ balls of each color. We throw the γm balls uniformly at random into $2\gamma m$ bins, call them $\{bin_j\}_{j=1}^{2\gamma m}$. We say that a ball "survives" in bin_j, if no other ball (of any color) lands in bin_j. We say that $color_i$ "survives" if at least one ball of color $color_i$ survives. We say that the game *succeeds* if *all* m colors survive, otherwise we say that it *fails*.

Lemma A.1 *The probability that the* color-survival game fails *is negligible in γ.*

Again, refer to [OS05] for proof of this lemma. Another slightly more subtle problem that arises from collisions in the buffer is that they must be detected in the first place. Otherwise, it may be the case that the result of multiple writes to the same buffer location appears to be a valid document, even though this fabricated document was never written by any user. One simple approach from [OS05] to address this issue is to append a "collision detection string" to each document. These strings are randomly sampled from a distribution that has the property that sums of one or more such strings do not belong to the distribution with overwhelming probability. One solution from [OS05] for addition modulo 2 is given here.

Lemma A.2 *Let $\{e_i\}_{i=1}^3$ be the three unit vectors in $\mathbb{Z}_2^3$, i.e., $(e_i)_j = \delta_{ij}$. Let n be an odd integer, $n > 1$. For $v \in \mathbb{Z}_2^3$, denote by $T_n(v)$ the number of n-element sequences $\{v_j\}_{j=1}^n$ in the e_i's, such that $\sum_{j=1}^n v_j = v$. Then,*

$$T_n((1,1,1)) = \frac{3^n - 3}{4}$$

It readily follows from this lemma that if one concatenates strings of randomly chosen e_i, then with overwhelming probability in the length of the concatenation, the sum of such strings (modulo 2) will not result in another, which yields an effective method for detecting collisions.

A.1.2 Bloom Filters

Bloom filters [Blo70] provide a way to probabilistically encode set membership using a small amount of space, even when the universe set is large, for example $\{0,1\}^*$. The basic idea is as follows:

Let $A = \{a_i\}_{i=1}^l$ be a finite subset of $\{0,1\}^*$. We will represent set membership information for A by a single element $T \in \{0,1\}^m$ for some $m \in \mathbb{Z}^+$. First, we choose an independent set of hash functions $\{h_i\}_{i=1}^k$, where each function $h_i : \{0,1\}^* \longrightarrow [m]$. We'll define $T = \{t_i\}_{i=1}^m$ by

$$t_i = \begin{cases} 1 & \text{if } \exists j \in [k], j' \in [l] \text{ such that } h_j(a_{j'}) = i \\ 0 & \text{otherwise} \end{cases}$$

That is, the ith component of T is set to 1 $\iff$ $i \in \bigcup_{j \in [k]} h_j(A)$. Now to test the validity of a statement like "$a \in A$", one simply verifies that $t_{h_i(a)} = 1, \forall i \in [k]$. If this does not hold, then certainly $a \notin A$. If the statement does hold, then there is still some probability that $a \notin A$, however this can be shown to be negligible in k when m and k are proportional. For details see [BKOS07, Blo70].

This work will use a variation of a Bloom filter, as we require more functionality. We would like our Bloom filters to not just store whether or not a certain element is in a set, but also to store some values $v \in V$ which are associated to the elements in the set (and to preserve those associations).

Definition A.3 *Let V be a finite set. A (k,m)-*Bloom Filter with Storage *is a collection $\{h_i\}_{i=1}^k$ of functions, with $h_i : \{0,1\}^* \longrightarrow [m]$ for all i, together with a collection of sets, $\{B_j\}_{j=1}^m$, where $B_j \subseteq V$. If $a \in \{0,1\}^*$ and $v \in V$, then to insert a pair (a,v) into this structure, v is added to $B_{h_i(a)}$ for all $i \in [k]$. Then, to determine whether or not $a \in S$, one examines all of the sets $B_{h_i(a)}$ and returns true if all are non-empty. The set of values associated with $a \in S$ is simply $\bigcap_{i \in [k]} B_{h_i(a)}$.*

Note: every inserted value is assumed to have at least one associated value.

For our purposes, communication is at a premium, so we'd like to know the amount of space that a (k,m)-Bloom Filter with Storage requires. The work of [BKOS07] proves the following claim, using uniform randomness to model the behavior of the hash functions (for the purposes of probabilistic analysis of correctness).

Claim A.4 *Let* $(\{h_i\}_{i=1}^k, \{B_j\}_{j=1}^m)$ *be a* (k,m)*-Bloom filter with storage as described in Definition A.3. Suppose the filter has been initialized to store some set* S *of size* n *and associated values. Suppose also that* $m = \lceil cnk \rceil$ *where* $c > 1$ *is a constant. Denote the (binary) relation of element-value associations by* $R(\cdot,\cdot)$*. Then, for any* $a \in \{0,1\}^*$*, the following statements hold true with probability* $1 - \mathrm{neg}(k)$*, where the probability is over the uniform randomness used to model the* h_i*:*

1. $(a \in S) \iff (B_{h_i(a)} \neq \varnothing \quad \forall i \in [k])$
2. $\bigcap_{i\in[k]} B_{h_i(a)} = \{v \mid R(a,v) = 1\}$

Note that the B_j are modelled abstractly as sets, but in practice, there must be some physical structure to store the information that they contain. For our applications, just as in the protocol of [BKOS07], we would like to *obliviously update* such a structure, which leads us to use fixed-length buffers to represent the B_j. Dynamic structures (e.g., linked lists) cannot be effectively used, as their change in size would reveal information about the updates made. The work of [BKOS07] proves the following claim, which analyzes the storage required for using fixed length buffers to store the sets B_j. Naturally, some uniformity will be required regarding the number of values associated to a particular element $a \in S$.

Claim A.5 *Let* $(\{h_i\}_{i=1}^k, \{B_j\}_{j=1}^m)$ *be a* (k,m)*-Bloom filter with storage as described in Definition A.3. Suppose the filter has been initialized to store some set* S *of size* n *and associated values. Again, suppose that* $m = \lceil cnk \rceil$ *where* $c > 1$ *is a constant, and denote the relation of element-value associations by* $R(\cdot,\cdot)$*. Let* $\lambda > 0$ *be any constant. If for every* $a \in S$ *we have that* $|\{v \mid R(a,v) = 1\}| \leq \lambda$ *then*

$$\Pr\Big[\max_{j\in[m]}\{|B_j|\} > \sigma\Big] < \mathrm{neg}(\sigma)$$

Again, the probability is over the uniform randomness used to model the h_i*.*

Combining these results with those of §A.1.1, we see that we can obliviously update an encrypted Bloom filter with storage, just using homomorphic encryption. Updates can be performed by many different, independent users holding only public information. Also, note that these users need not know anything about the current contents of the encrypted data structure. Finally, given our work in §2, we can furthermore accomplish this with sub-linear communication. These are the basic ingredients used by [BKOS07] to create a public key encryption scheme that supports PIR queries by keywords.

A.1.3 Some Definitions from [BKOS07]

In the following definitions, these notational conventions are employed:

- $\mathcal{X}$ – refers to a message sending party
- $\mathcal{Y}$ – refers to the message receiving party (owner of the private key)
- $\mathcal{S}$ – refers to the server/storage provider

Definition A.6 *A* Public Key Storage with Keyword Search *consists of the following probabilistic polynomial time algorithms and protocols:*

- $\mathsf{KeyGen}(1^s)$*: Outputs public and private keys,* A_{public} *and* $A_{private}$ *of length* s*.*
- $\mathsf{Send}_{\mathcal{X},\mathcal{S}}(M, K, A_{public})$ *This is either a non-interactive or interactive two-party protocol that allows* $\mathcal{X}$ *to send the message* M *to a server* $\mathcal{S}$*, encrypted under some public key* A_{public}*, and also associates* M *with each keyword in the set* K*. The values* M, K *are private inputs that only the message-sending party* $\mathcal{X}$ *holds.*

- $\mathsf{Retrieve}_{\mathcal{Y},\mathcal{S}}(w, A_{private})$: *This is a two party protocol between a user $\mathcal{Y}$ and a server $\mathcal{S}$ that retrieves all messages associated with the keyword w for the user $\mathcal{Y}$. The inputs $w, A_{private}$ are private inputs held only by $\mathcal{Y}$. This protocol also removes the retrieved messages from the server and properly maintains the keyword references.*

Correctness is defined as follows:

Definition A.7 *Let $\mathcal{Y}$ be a user, $\mathcal{X}$ be a message sender and let $\mathcal{S}$ be a server/storage provider. Let $A_{public}, A_{private} \longleftarrow \mathsf{KeyGen}(1^s)$. Fix a finite sequence of messages and keyword sets:*

$$\{(M_i, K_i)\}_{i=1}^{m} \ .$$

Suppose that, for all $i \in [m]$, the protocol $\mathsf{Send}_{\mathcal{X},\mathcal{S}}(M_i, K_i, A_{public})$ is executed by $\mathcal{X}$ and $\mathcal{S}$. Denote by R_w the set of messages that $\mathcal{Y}$ receives after the execution of $\mathsf{Retrieve}_{\mathcal{Y},\mathcal{S}}(w, A_{private})$. Then, a Public Key Storage with Keyword Search is said to be correct on the sequence $\{(M_i, K_i)\}_{i=1}^{m}$ *if*

$$\Pr\Big[R_w = \{M_i \ | \ w \in K_i\}\Big] > 1 - \mathrm{neg}(1^s)$$

for every w, where the probability is taken over all internal randomness used in the protocols Send *and* Retrieve*. A Public Key Storage with Keyword Search is said to be* correct *if it is correct on all such finite sequences.*

Definition A.8 *A Public Key Storage with Keyword Search is said to be (n, λ, θ)-*correct *if whenever $\{(M_i, K_i)\}_{i=1}^{m}$ is a sequence such that*

- *$m \le n$*
- *$|K_i| < \theta$, for every $i \in [m]$, and*
- *for every $w \in \bigcup_{i\in[m]} K_i$, at most λ messages are associated with w*

then, it is correct on $\{(M_i, K_i)\}_{i=1}^{m}$ in the sense of Definition A.7.

Finally, privacy for the sender and receiver are defined in the following ways:

Definition A.9 *We define* Sender-Privacy *in terms of the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{A}$ will play the role of the storage provider and $\mathcal{C}$ will play the role of a message sender. The game consists of the following steps:*

1. *$\mathsf{KeyGen}(1^s)$ is executed by $\mathcal{C}$ who sends the output A_{public} to $\mathcal{A}$.*
2. *$\mathcal{A}$ asks queries of the form (M, K) where M is a message string and K is a set of keywords, and $\mathcal{C}$ answers by executing the protocol $\mathsf{Send}(M, K, A_{public})$ with $\mathcal{A}$.*
3. *$\mathcal{A}$ now chooses two pairs $(M_0, K_0), (M_1, K_1)$ and sends this to $\mathcal{C}$, where both the messages and keyword sets are of equal size, the latter being measured by set cardinality.*
4. *$\mathcal{C}$ picks a bit $b \in \{0, 1\}$ at random and executes the protocol $\mathsf{Send}(M_b, K_b, A_{public})$ with $\mathcal{A}$.*
5. *$\mathcal{A}$ may ask more queries of the form (M, K) and $\mathcal{C}$ responds by executing $\mathsf{Send}(M, K, A_{public})$ with $\mathcal{A}$.*
6. *$\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.*

We define the adversary's advantage as

$$\mathrm{Adv}_{\mathcal{A}}(1^s) = \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

We say that a Public-Key Storage with Keyword Search is CPA-Sender-Private *if, for all $\mathcal{A} \in$ PPT, we have that $\mathrm{Adv}_{\mathcal{A}}(1^s)$ is a negligible function.*[1]

For the message receiver, privacy is defined as

Definition A.10 *We define* Receiver-Privacy *in terms of the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{A}$ will again play the role of the storage provider, and $\mathcal{C}$ will play the role of a message receiver. The game consists of the following steps:*

1. $\mathsf{KeyGen}(1^s)$ *is executed by $\mathcal{C}$ who sends the output A_{public} to $\mathcal{A}$.*
2. *$\mathcal{A}$ asks queries of the form w, where w is a keyword, and $\mathcal{C}$ answers by executing the protocol* $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}(w, A_{private})$ *with $\mathcal{A}$.*
3. *$\mathcal{A}$ now chooses two keywords w_0, w_1 and sends both to $\mathcal{C}$.*
4. *$\mathcal{C}$ picks a bit $b \in \{0, 1\}$ at random and executes the protocol* $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}(w_b, A_{private})$ *with $\mathcal{A}$.*
5. *$\mathcal{A}$ may ask more keyword queries w and $\mathcal{C}$ responds by executing* $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}(w, A_{private})$ *with $\mathcal{A}$.*
6. *$\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.*

We define the adversary's advantage as

$$\mathrm{Adv}_{\mathcal{A}}(1^s) = \left|\Pr[b = b'] - \frac{1}{2}\right|.$$

We say that a Public Key Storage with Keyword Search is CPA-Receiver-Private *if, for all $\mathcal{A} \in$ PPT, we have that $\mathrm{Adv}_{\mathcal{A}}(1^s)$ is a negligible function.*

A.2 Privacy game for batch cryptographic counters

The privacy for batch cryptographic counters with respect to server T (modified from [KMO01]) is given through the following two experiments 0 and 1.

Let the batch cryptographic counter protocol with N counters, where every party modifies at most L counters, be denoted by $\mathsf{Counter}(N, L)$. Let W_b denote the probability with which $\mathcal{A}$ outputs 1 in Experiment b for $b = 0, 1$.

1. In both experiments,
   (a) The challenger picks the public key of the encryption scheme *pk* and sends it to $\mathcal{A}$. $\mathcal{A}$ plays the role of the server T here.
   (b) $\mathcal{A}$ sends N values denoting the initial values of the N cryptographic counters to the challenger.
   (c) The challenger encrypts these N values using *pk* and sends it to $\mathcal{A}$.
   (d) $\mathcal{A}$ picks a set $S \subseteq [N]$ of size L. For every index in S, $\mathcal{A}$ denotes the value by which the counter in the index must be incremented or decremented. The challenger runs $\mathsf{Counter}(N, L)$ with $\mathcal{A}$ using S as input. This step may be repeated polynomially many times.

[1] "PPT" stands for *Probabilistic Polynomial Time.* We use the notation $\mathcal{A} \in$ PPT to denote that $\mathcal{A}$ is a probabilistic polynomial-time algorithm.

(e) $\mathcal{A}$ picks 2 sets S_0 and $S_1 \subseteq [N]$, such that $|S_0| = |S_1| = L$. Also, for every index in S_0 and S_1, $\mathcal{A}$ denotes the value by which the counter in the index must be incremented or decremented ($\mathcal{A}$ may also specify that this counter not be changed).

2. In Experiment b, the challenger runs $\mathsf{Counter}(N, L)$ with $\mathcal{A}$ using S_b as input.

3. $\mathcal{A}$ outputs a bit 0 or 1.

Definition 5 *We say that* $\mathsf{Counter}(N, L)$ *is* L*-private with respect to server* T *(run by* $\mathcal{A}$*), if for all semi-honest PPT adversaries* $\mathcal{A}$*, we have* $|W_0 - W_1|$ *is negligible.*

The privacy of $\mathsf{Transfer}(x, y)$ with respect to party P and server T is defined via standard definitions of two-party computation which guarantees that P does not learn anything other than the final value of the counter and that T does not learn anything.