Public Key Encryption That Allows PIR Queries

Dan Boneh*, Eyal Kushilevitz**,
Rafail Ostrovsky***, and William E. Skeith III†

dabo@cs.stanford.edu,eyalk@cs.technion.ac.il,
rafail@cs.ucla.edu,wskeith@math.ucla.edu

Abstract. Consider the following problem: Alice wishes to maintain her email using a storage-provider Bob (such as a Yahoo! or hotmail e-mail account). This storage-provider should provide for Alice the ability to collect, retrieve, search and delete emails but, at the same time, should learn neither the content of messages sent from the senders to Alice (with Bob as an intermediary), nor the search criteria used by Alice. A trivial solution is that messages will be sent to Bob in encrypted form and Alice, whenever she wants to search for some message, will ask Bob to send her a copy of the entire database of encrypted emails. This however is highly inefficient. We will be interested in solutions that are communication-efficient and, at the same time, respect the privacy of Alice. In this paper, we show how to create a public-key encryption scheme for Alice that allows PIR searching over encrypted documents. Our solution is the first to reveal no partial information regarding the user's search (including the access pattern) in the public-key setting and with non-trivially small communication complexity. This provides a theoretical solution to a problem posed by Boneh, DiCrescenzo, Ostrovsky and Persiano on "Public-key Encryption with Keyword Search." The main technique of our solution also allows for Single-Database PIR writing with sub-linear communication complexity, which we consider of independent interest.

keywords: Searching on encrypted data, Database security, Public-key Encryption with special properties, Private Information Retrieval.

1 Introduction

Problem Overview. Consider the following problem: Alice wishes to maintain her email using a storage-provider Bob (such as Yahoo! or hotmail e-mail account). She

* Stanford Department of Computer Science. Supported by NSF and the Packard foundation.

** Department of Computer Science, Technion. Partially supported by BSF grant 2002-354 and by Israel Science Foundation grant 36/03.

*** Computer Science Department and Department of Mathematics, University of California, Los Angeles, CA 90095. Research partially done while visiting IPAM, and supported in part by IBM Faculty Award, Xerox Innovation Group Award, NSF Cybertrust grant no. 0430254, and U.C. MICRO grant.

† Department of Mathematics, University of California, Los Angeles. Research done in part at IPAM, and supported in part by U.C. Chancellor's Presidential Dissertation Fellowship 2006-2007.

A. Menezes (Ed.): CRYPTO 2007, LNCS 4622, pp. 50–67, 2007.

publishes a public-key for a semantically-secure Public-Key Encryption scheme, and asks all people to send their e-mails, encrypted under her public-key, to the intermediary Bob. Bob (the storage-provider) should allow Alice to collect, retrieve, search and delete emails at her leisure. In known implementations of such services, either the content of the emails is known to the storage-provider Bob (and then the privacy of both Alice and the senders is lost) or the senders can encrypt their messages to Alice, in which case privacy is maintained, but sophisticated services (such as search by keyword) cannot be easily performed and, more importantly leak information, such as Alice's access pattern, to Bob. Of course, Alice can always ask Bob, the storage-provider, to send her a copy of the entire database of emails. This however is highly inefficient in terms of communication, which will be a main focus in this work. In all that follows, we will denote the number of encrypted documents that Bob stores for Alice by the variable n.

We will be interested in solutions that are communication-efficient and, at the same time, respect the complete privacy of Alice. A seemingly related concept is that of *Private Information Retrieval (PIR)* (e.g., [13,23,10], or [27] for a survey). However, existing PIR solutions either allow only for retrieving a (plain or encrypted) record of the database by address, or allow for search by keyword [12,23,25] in a non-encrypted data. The challenge of creating a public-key encryption that allows for keyword search, where keywords are encrypted in a probabilistic manner, remained an open problem prior to this paper.

In our solution, Alice creates a public key that allows arbitrary senders to send her encrypted e-mail messages. Each such message M is accompanied by an "encoded" list of keywords in response to which M should be retrieved. These email messages are collected for Alice by Bob, along with the "encoded" keywords. When Alice wishes to search in the database maintained by Bob for e-mail messages containing certain keywords, she is able to do so in a communication-efficient way and does not allow Bob to learn *anything* about the messages that she wishes to read, download or erase. In particular, Alice is not willing to reveal what particular messages she downloads from the mail database, from which senders these emails are originating and/or what is the search criterion, including the access pattern.

Furthermore, our solution allows the communication from any sender to Bob to be *non-interactive* (i.e. just a single message from the sender to Bob), and allow a single round of communication from Alice to Bob and back to Alice, with total communication complexity sub-linear in n. Furthermore, we show a simple extension that allows honest-but-curious Bob to tolerate malicious senders, who try to corrupt messages that do not belong to them in Bob's database, and reject all such messages with overwhelming probability.

Comparison with Related Work. Recently, there was a lot of work on *searching on encrypted data* (see [7,6] and references therein). However, all previous solutions either revealed some partial information about the data or about the search criterion, or work only in *private-key* settings. In such settings, only entities who have access to the private key can do useful operations; thus, it is inappropriate for our setting, where both the storage-provider and the senders

of e-mail messages for Alice have no information on her private key. We emphasize that, in settings that include only a user Alice and a storage-provider, the problem is already solved; for example, one can apply results of [17,29,9,7]. However, the involvement of the senders who are also allowed to *encrypt* data for Alice (but are not allowed to decrypt data encrypted by other senders) requires using public-key encryption. In contrast to the above work, we show how to search, in a communication-efficient manner, on encrypted data in a *public-key setting*, where those who store data (encrypted with a public key of Alice) do not need to know the private key under which this data is encrypted. The only previous results for such a scenario in the public-key setting, is due to Boneh et al. [6] and Abddalla et al. [1] who deal with the same storage-provider setting we describe above; however, their solution *reveals* partial information; namely, the particular keyword that Alice is searching for is given by her, in the clear, to Bob (i.e., only the content of the email messages is kept private while the information that Alice is after is revealed). This, in particular, reveals the *access pattern* of the user. The biggest problem left was creating a scheme that hides the access pattern as well. This is exactly what we achieve in this paper. That is, we show how to hide *all* information in a semantically-secure way.

As mentioned, private information retrieval (PIR) is a related problem that is concerned with communication-efficient retrieval of *public* (i.e., plain) data. Extensions of the basic PIR primitive (such as [12,23], mentioned above, and, more recently, [22,15,25]) allow more powerful keyword search *un-encrypted* data. Therefore, none of those can directly be used to solve the current problem.

It should also be noted that our paper is in some ways only a partial solution to the problem. Specifically, we put the following constraint in our model: the number of total messages associated to each keyword is bounded by a constant. It is an interesting question as to whether this condition can be relaxed, while keeping communication non-trivially small and maintaining the strict notions of security presented here.

Our Techniques. We give a short overview of some of the tools that we use. The right combination of these tools is what allows for our protocol to work.

As a starting point, we examine *Bloom filters* (see Section 2.1 for a definition). Bloom filters allow us to use space which is not proportional to the number of all potential keywords (which is typically huge) but rather to the maximal number of keywords which are in use at any given time (which is typically much smaller). That is, the general approach of our protocols is that the senders will store in the database of the storage-provider some extra information (in encrypted form) that will later allow the efficient search by Alice. *Bloom filters* allow us to keep the space that is used to store this extra information "small". The approach is somewhat similar to Goh's use of Bloom filters [16]; the important difference is that in our case we are looking for a public-key solution, whereas Goh [16] gives a private-key solution. This makes our problem more challenging, and our use of Bloom filter is somewhat different. Furthermore, we require the Bloom filters in our application to encode significantly more information than just set membership. We modify the standard definitions of Bloom filters to accommodate the additional functionality.

Recall that the use of Bloom filters requires the ability to flip bits in the array of extra information. However, the identity of the positions that are flipped should be kept secret from the storage-provider (as they give information about the keywords). This brings us to an important technical challenge: we need a way to specify an encrypted length-n unit vector e_i (i.e., a length n vector with 1 in its i-th position and 0's elsewhere) while keeping the value i secret, and having a representation that is short enough to get communication-efficiency beyond that of the trivial solution. We show that a recent public-key homomorphic-encryption scheme, due to Boneh, Goh and Nissim [5], that supports additions and one multiplication on ciphertexts, allows us to obtain just that. For example, one can specify such a length-n unit vector using communication complexity which is $\sqrt{n}$ times a security parameter. Also, as shown in [26], this is optimal, from an algebraic point of view.

Finally, for Alice to read information from the array of extra information, she applies efficient PIR schemes, e.g. [23,10], that, again, allow keeping the keywords that Alice is after secret.

We emphasize that the communication in the protocol is sub-linear in n. This includes both the communication from the senders to the storage-provider Bob (when sending email messages) and the communication from Alice to Bob (when she retrieves/searches for messages). Furthermore, we allow Alice to *delete* messages from Bob's storage in a way that hides from Bob which messages have been deleted. Our main theorem is as follows:

MAIN THEOREM (informal): There exists Public-Key Encryption schemes that support sending, reading and writing into remote server (honest-but-curious Bob) with the following communication complexity:

- $\mathcal{O}(\sqrt{n \log^3 n})$ for sending a message from any honest-but-curious Sender to Bob. In case the sender is malicious, the communication complexity for sending a message becomes $\mathcal{O}(\sqrt{n \log n} \cdot \text{polylog}(n))$
- $\mathcal{O}(\text{polylog}(n))$ for reading by Alice from Bob's (encrypted) memory.
- $\mathcal{O}(\sqrt{n \log^3 n})$ for deleting messages by Alice from Bob's memory.

Organization: In Section 2, we explain and develop the tools needed for our solutions. Section 3 defines the properties we want our protocols to satisfy. Finally, Section 4 gives the construction and its analysis.

1.1 Reference Table of Notation

For the reader's convenience, we provide a table of the most frequently used notation in this work.

- n – size of e-mail database
- s – a security parameter
- k – number of hash functions used in Bloom filter
- m – size of Bloom filter hash table
- $\{h_i\}_{i=1}^{k}$ – Bloom filter hash functions

- H_w – set of hash images for a word $w \in \{0,1\}^*$, i.e. $\{h_i(w) \mid i \in [k]\}$
- B_j – a buffer in a Bloom filter with storage (so, $j \in [m]$)
- σ – size of fixed length buffers in a Bloom filter with storage
- l – size of the associated values in a Bloom filter with storage
- $\mathcal{X}$ – a message sender
- $\mathcal{Y}$ – message receiver (owner of public key)
- $\mathcal{S}$ – owner of remote storage (mail server)
- K – a set of keywords
- M – a message
- $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ – key generation, encryption and decryption, respectively
- c – a constant, greater than 1
- λ – maximum number of messages associated to a specific keyword
- θ – maximum size of a keyword set associated to a specific message

2 Ingredients

We will make use of several basic tools, some of which are being introduced for the first time in this paper. In this section, we define (and create, if needed) these tools, as well as outline their utility in our protocol.

2.1 Bloom Filters

Bloom filters [4] provide a way to probabilistically encode set membership using a small amount of space, even when the universe set is large. The basic idea is as follows. Choose an independent set of hash functions $\{h_i\}_{i=1}^k$, where each function $h_i : \{0,1\}^* \longrightarrow [m]$. Suppose $S = \{a_i\}_{i=1}^l \subset \{0,1\}^*$. We set an array $T = \{t_i\}_{i=1}^m$ such that $t_i = 1 \iff \exists j \in [k]$ and $j' \in [l]$ such that $h_j(a_{j'}) = i$. Now to test the validity of a statement like "$a \in S$", one simply verifies that $t_{h_i(a)} = 1, \forall i \in [k]$. If this does not hold, then certainly $a \notin S$. If the statement does hold, then there is still some probability that $a \notin S$, however this can be shown to be small. Optimal results are obtained by having m proportional to k; in this case, it can be shown that the probability of an inaccurate positive result is negligible as k increases, as will be thoroughly demonstrated in what follows.

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 the set, but also to store some values $v \in V$ which are associated with the elements in the set (and to preserve those associations).

Definition 1. *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 $B_j \subseteq V$. To insert a pair (a, v) into this structure, where $a \in \{0,1\}^*$ and $v \in V$, 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.)*

Next, we analyze the total size of a (k, m)-Bloom filter with storage. For the purpose of analysis, the functions h_i will, as usual, be modeled as uniform, independent randomness. For $w \in \{0,1\}^*$, define $H_w = \{h_i(w) \mid i \in [k]\}$.

Claim. Let $(\{h_i\}_{i=1}^k, \{B_j\}_{j=1}^m)$ be a (k, m)-Bloom filter with storage. 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 $1 - \text{neg}(k)$, 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. $\bigcap_{i \in [k]} B_{h_i(a)} = \{v \mid R(a, v) = 1\}$

Proof. (1., $\Rightarrow$) Certainly if $B_{h_i(a)} = \varnothing$ for some $i \in [k]$, then a was never inserted into the filter, and $a \notin S$. (1., $\Leftarrow$) Suppose that $B_{h_i(a)} \neq \varnothing$ for every $i \in [k]$. We'd like to compute the probability that for an arbitrary $a \in \{0,1\}^*$, $a \notin S$, we have $H_a \subset \bigcup_{w \in S} H_w$; i.e., that such an element will appear to be in S by our criteria. Recall that we model each evaluation of the functions h_i as independent and uniform randomness. Therefore, a total of nk (not necessarily distinct) random sets are modified to insert the n values of S into the filter. So, we only need to compute the probability that all k functions place a in this subset of the B_j's. By assumption, there are a total of $m = \lceil cnk \rceil$ sets where $c > 1$ is a constant. Let $X_{k,k'}$ denote the random variable that models the experiment of throwing k balls into m bins and counting the number that land in the first k' bins. For a fixed insertion of the elements of S into our filter and letting k' be the number of distinct bins occupied, $X_{k,k'}$ represents how close a random element appears to being in S according to our Bloom filter. More precisely, $\Pr[X_{k,k'} = k]$ is the probability that a random element will appear to be in S for this specific situation. Note that $X_{k,k'}$ is a sum of independent (by assumption) Bernoulli trials, and hence is distributed as a binomial random variable with parameters, $(k, \frac{k'}{cnk})$, where $k' \leq nk$. Hence, $\Pr[X_{k,k'} = k] = \left(\frac{k'}{cnk}\right)^k \leq \left(\frac{1}{c}\right)^k$. So, we've obtained a bound that is negligible in k, independently of k'. Hence, if we let Y_k be the experiment of sampling k' by throwing nk balls into $\lceil cnk \rceil$ bins and counting the distinct number of bins, then taking a random sample from the variable $X_{k,k'}$ and returning 1 if and only if $X_{k,k'} = k$, then Y_k is distributed identically to the variable that describes whether or not a random $a \in \{0,1\}^*$ will appear to be in S according to our filter. Since we have $\Pr[X_{k,k'} = k] < \text{neg}(k)$ and the bound was independent of k', it is easy to see that $\Pr[Y_k = 1] < \text{neg}(k)$, as needed.

(2.) The argument is quite similar to part 1. ($\supseteq$) If $R(a, v) = 1$, then the value v has been inserted and associated with a and by definition, $v \in B_{h_i(a)}$ for every $i \in [k]$. ($\subseteq$) Suppose $a \in S$ and $v \in B_{h_i(a)}$ for every $i \in [k]$. The probability of this event randomly happening independent of the relation R is maximized if every other element in S is associated with the same value. In this case, the problem reduces to a false positive for set membership with $(n-1)k$ writes if $a \in S$, or the usual nk if $a \notin S$. This has already been shown to be negligible in part 1.

In practice, we will need some data structure to model the sets of our Bloom filter with storage, e.g. a linked list. However, in this work we will be interested in *oblivious* writing to the Bloom filter, in which case a linked list seems quite inappropriate as the dynamic size of the structure would leak information about the writing. So, we would like to briefly analyze the total space required for a Bloom filter with storage if it is implemented with fixed-length buffers to represent the sets. Making some needed assumptions about uniformity of value associations, we can show that with overwhelming probability (exponentially close to 1 as a function of the size of our structure) no buffer will overflow.

Claim. Let $(\{h_i\}_{i=1}^{k}, \{B_j\}_{j=1}^{m})$ be a (k, m)-Bloom filter with storage. 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 $|\{v \mid R(a, v) = 1\}| \leq \lambda$, then for $\sigma \in \mathbb{N}$ we have that as σ increases, $\Pr\left[\max_{j \in [m]}\{|B_j|\} > \sigma\right] < \text{neg}(\sigma)$. Again, the probability is over the uniform randomness used to model the h_i.

Proof. First, let us analyze the case $\lambda = 1$, so there will be a total of nk values placed randomly into the $\lceil cnk \rceil$ buffers. Let X_j be a random variable that counts the size of B_j after the nk values are randomly placed. X_j has a binomial distribution with parameters $(nk, \frac{1}{cnk})$. Hence $E[X_j] = (1/c)$. If $(1 + \delta) > 2e$, we can apply a Chernoff bound to obtain the following estimation: $\Pr[X_j > (1 + \delta)/c] < 2^{-\delta/c}$. Now, for a given σ we'd like to compute $\Pr[X_j > \sigma]$. So, set $(1 + \delta)/c = \sigma$ and hence $\delta/c = \sigma - 1/c$. Then, $\Pr[X_j > \sigma] < 2^{-\sigma+1/c} = 2^{-\sigma}2^{(1/c)} = \text{neg}(\sigma)$. By the union bound, the probability that *any* X_j is larger than σ is also negligible in σ.

Now, if $\lambda > 1$, what has changed? Our analysis above treated the functions as uniform randomness, but to associate additional values to a specific element of $a \in S$ the same subset of buffers (H_a in our notation) will be written to repeatedly- there is no more randomness to analyze. Each buffer will have at most a factor of λ additional elements in it, so our above bound becomes $\text{neg}(\sigma/\lambda)$ which is still $\text{neg}(\sigma)$ as λ is an independent constant.

So, we can implement a (k, m)-Bloom filter with storage using fixed-length buffers. However, the needed length of such buffers depends on the maximum number of values that could be associated to a specific $a \in S$. A priori, this is bounded only by $|V|$, the size of the value universe: for it could be the case that all values are associated to a particular $a \in S$, and hence the buffers of H_a would need to be as large as this universe. But, since we want to fix the buffers length *ahead of time*, we will enforce a "uniformity" constraint; namely, that the number of values associated to each word is bounded by a constant. We summarize with the following observation.

Observation 1. *One can implement a (k, m)-Bloom filter with storage by using fixed-length arrays to store the sets B_j, with the probability of losing an associated value negligible in the length of the arrays. The total size of such a structure is*

linear in n, k, σ, l and c where n is the maximum number of elements that the filter is designed to store, k is the number of functions (h_i) used (which serves as a correctness parameter), σ is the size of the buffer arrays (which serves as a correctness parameter; note that σ should be chosen to exceed λ, the maximum number of values associated to any single element of the set), l is the storage size of an associated value, and c is any constant greater than 1.

So, for our application of public-key storage with keyword search, if we assume that there are as many keywords as there are messages, then we have created a structure of size $\mathcal{O}(n \cdot l) = \mathcal{O}(n \log n)$ to hold the keyword set and the message references. However, the correctness parameter σ has logarithmic dependence on n, leaving us with $\mathcal{O}(n \log^2 n)$.

2.2 Oblivious Modification

For our application, we will need message senders to update the contents of a Bloom filter with storage. However, all data is encrypted under a key which neither they, nor the storage provider have. So, they must write to the buffers in an "oblivious" way- they will not (and cannot) know what areas of the buffer are already occupied, as this will reveal information about the user's data, and the message-keyword associations. One model for such a writing protocol has been explored by Ostrovsky and Skeith [25]. They provide a method for obliviously writing to a buffer which, with overwhelming probability in independent correctness parameters, is completely correct: i.e., there is a method for extracting documents from the buffer which outputs exactly the set of documents which were put into it.

In [25], the method for oblivious buffer writing is simply to write messages at uniformly random addresses in a buffer, except to ensure that data is recoverable with very high probability, messages are written repeatedly to an appropriately sized buffer, which has linear dependence on a correctness parameter. To ensure that no additional documents arise from collisions, a "collision detection string" is appended to each document from a special distribution which is designed to not be closed under sums. We can apply the same methods here, which will allow senders to update an encrypted Bloom filter with storage, without knowing anything about what is already contained in the encrypted buffers. For more details on this approach, see [25], or the full version of this work. Another approach to this situation was presented by Bethencourt, Song, and Waters [3], who solve a system of linear equations to recover buffer contents. These methods may also be applicable, but require additional interaction to evaluate a pseudo-random function on appropriate input. So, with an added factor of a correctness parameter to the buffer lengths, one can implement and *obliviously update* an encrypted Bloom filter with storage, using the probabilistic methods of [25], or [3].

As a final note on our Bloom filters with storage, we mention that, in practice, we can replace the functions h_i with pseudo-random functions; in this case our claims about correctness are still valid, only with a computational assumption

in place of the assumption about the h_i being truly random, provided that the participating parties are non-adaptive[1].

By now, we have an amicable data structure to work with, but there is a piece of the puzzle missing: this data structure will be held by a central storage provider that we'd like to keep in the dark regarding all operations performed on the data. Next, we give message senders a way to update this data structure without revealing to the storage provider any information about the update, *and using small communication.*

2.3 Modifying Encrypted Data in a Communication-Efficient Way

Our next tool is that of encrypted database modification. This will allow us to privately manipulate our Bloom filters. The situation is as follows:

- A database owner Bob holds an array of ciphertexts $\{c_i\}_{i=1}^n$ where each $c_i = \mathcal{E}(x_i)$ is encrypted using a public-key for which Bob does not have the private key.
- A user would like to modify one plaintext value x_i, without revealing to Bob which value was modified, or how it was modified.

Furthermore, we would like to minimize the communication between the parties beyond the trivial $\mathcal{O}(n)$ solution which could be based on any group homomorphic encryption. Using the cryptosystem of Boneh, Goh, and Nissim [5], we can accomplish this with communication $\mathcal{O}(\sqrt{n})$. The important property of the cryptosystem of [5], for our purposes, is its additional homomorphic property; specifically, in their system, one can compute multivariate polynomials of total degree 2 on ciphertexts; i.e., if $\mathcal{E}$ is the encryption map (and $\mathcal{D}$ is the corresponding decryption) and if $F(X_1, \ldots, X_n) = \sum_{1 \le i \le j \le n} a_{ij} X_i X_j$, then there exists some function $\widetilde{F}$ on ciphertexts (which can be computed using public information alone) such that, for any array of ciphertexts $\{c_l = \mathcal{E}(x_l)\}_{l=1}^n$, it holds that $\mathcal{D}(\widetilde{F}(c_1, ..., c_n)) = F(x_1, ..., x_n)$.

Applying such a cryptosystem to encrypted database modification is simple. Suppose $\{x_{ij}\}_{i,j=1}^{\sqrt{n}}$ is our database (not encrypted). Then, to increment the value of a particular element at position (i^*, j^*) by some value α, we proceed as

[1] In the case of malicious senders, we cannot reveal the seeds for the random functions and still guarantee correctness; however, we can entrust the storage provider (Bob) with the seeds, and have the senders execute a secure two-party computation protocol with Bob to learn the value of the functions. This can be accomplished without Bob learning anything, and with the sender learning only $h_i(w)$ and nothing else. Examples of such a protocol can be found in the work of Katz and Ostrovsky [20], if we disallow concurrency, and the work of Canetti et al. [11], to allow concurrency. Here, the common reference string can be provided as part of the public key. These solutions require additional rounds of communication between the senders and the storage provider Bob, and additional communication. However, the size of the communication is proportional to the security parameter and is independent of the size of the database. We defer this and other extensions to the full version of the paper.

follows: Create two vectors v, w of length $\sqrt{n}$ where, $v_i = \delta_{ii^*}$ and $w_j = \alpha\delta_{jj^*}$ (here $\delta_{k\ell} = 1$ when $k = \ell$ and 0 otherwise). Thus, $v_i w_j = \alpha$ if $(i = i^* \wedge j = j^*)$ and 0 otherwise. Now, we wish to add the value $v_i w_j$ to the (i, j) position of the database. Note that, for each i, j, we are just evaluating a simple polynomial of total degree two on v_i, w_j and the data element x_{ij}. So, if we are given any cryptosystem that allows us to compute multivariate polynomials of total degree two on ciphertexts, then we can simply encrypt every input (the database, and the vectors v, w) and perform the same computation which will give us a private database modification protocol with communication complexity $\mathcal{O}(\sqrt{n})$.

More formally, suppose $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a CPA-secure public-key encryption scheme that allows polynomials of total degree two to be computed on ciphertexts, as described above. Suppose also that an array of ciphertexts $\{c_l = \mathcal{E}(x_l)\}_{l=1}^{n}$ is held by a party $\mathcal{S}$, which have been encrypted under some public key, A_{public}. Suppose that n is a square (if not, it can always be padded by $< 2\sqrt{n}+1$ extra elements to make it a square). Define $F(X, Y, Z) = X + YZ$. Then by our assumption, there exists some $\widetilde{F}$ such that $\mathcal{D}(\widetilde{F}(\mathcal{E}(x), \mathcal{E}(y), \mathcal{E}(z))) = F(x, y, z)$ for any plaintext values x, y, z. We define a two party protocol $\mathsf{Modify}_{\mathcal{U},\mathcal{S}}(l, \alpha)$ by the following steps, where l and α are private inputs to $\mathcal{U}$:

1. $\mathcal{U}$ computes i^*, j^* as the coordinates of l (i.e., i^* and j^* are the quotient and remainder of l/n, respectively).
2. $\mathcal{U}$ sends $\{\overline{v}_i = \mathcal{E}(\delta_{ii^*})\}_{i=1}^{\sqrt{n}}, \{\overline{w}_j = \mathcal{E}(\alpha\delta_{jj^*})\}_{j=1}^{\sqrt{n}}$ to $\mathcal{S}$ where all values are encrypted under A_{public}.
3. $\mathcal{S}$ computes $\widetilde{F}(c_{ij}, \overline{v}_i, \overline{w}_j)$ for all $i, j \in [\sqrt{n}]$, and replaces each c_{ij} with the corresponding resulting ciphertext.

By our remarks above, this will be a correct database modification protocol. It is also easy to see that it is private, in that it resists a chosen plaintext attack. In a chosen plaintext attack, an adversary would ask many queries consisting of requests for the challenger to execute the protocol to modify positions of the adversary's choice. But all that is exchanged during these protocols is arrays of ciphertexts for which the plaintext is known to the adversary. Distinguishing two different modifications is precisely the problem of distinguishing two finite arrays of ciphertexts, which is easily seen to be infeasible assuming the CPA-security of the underlying cryptosystem and then using a standard hybrid argument.

3 Definitions

In what follows, we will denote message sending parties by $\mathcal{X}$, a message receiving party will be denoted by $\mathcal{Y}$, and a server/storage provider will be denoted by $\mathcal{S}$.

Definition 2. *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.
- $\mathsf{Send}_{\mathcal{X},\mathcal{S}}(M, K, A_{public})$ *is (an interactive or non-interactive) two-party*

protocol that allows $\mathcal{X}$ to send the message M to server $\mathcal{S}$, encrypted under 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.

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

We now describe correctness and privacy for such a system.

Definition 3. *Let $\mathcal{Y}$ be a user, $\mathcal{X}$ be a message sender and $\mathcal{S}$ 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* $\Pr\Big[R_w = \{M_i \mid w \in K_i\}\Big] > 1 - \mathrm{neg}(1^s)$, *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 4. *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 (1) $m \leq n$, (2) $|K_i| < \theta$, for every $i \in [m]$, and (3) 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 3.*

For privacy, there are several parties involved, and hence there will be several definitional components.

Definition 5. *For* sender-privacy, *consider 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 and K is a set of keywords; $\mathcal{C}$ answers by executing the protocol $\mathsf{Send}(M, K, A_{public})$ with $\mathcal{A}$.*
3. *$\mathcal{A}$ 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.*
4. *$\mathcal{C}$ picks a random bit $b \in_R \{0, 1\}$ and executes $\mathsf{Send}(M_b, K_b, A_{public})$ with $\mathcal{A}$.*
5. *$\mathcal{A}$ asks more queries of the form (M, K) and $\mathcal{C}$ responds by executing protocol $\mathsf{Send}(M, K, A_{public})$ 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.*[2]

[2] "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.

Definition 6. *For* receiver-privacy*, consider the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathcal{A}$ again plays the role of the storage provider, and $\mathcal{C}$ plays the role of a message receiver. The game proceeds as follows:*

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; $\mathcal{C}$ answers by executing the protocol $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}(w, A_{private})$ with $\mathcal{A}$.*
3. *$\mathcal{A}$ chooses two keywords w_0, w_1 and sends both to $\mathcal{C}$.*
4. *$\mathcal{C}$ picks a random bit $b \in_R \{0,1\}$ and executes the protocol $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}(w_b, A_{private})$ with $\mathcal{A}$.*
5. *$\mathcal{A}$ asks more keyword queries w and $\mathcal{C}$ responds by executing the protocol $\mathsf{Retrieve}_{\mathcal{C},\mathcal{A}}$ $(w, A_{private})$ 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.*

Remark: Note that we could have also included a separate protocol for erasing items. At present, the implementation erases messages as they are retrieved. These processes need not be tied together. We have done so to increase the simplicity of our definitions and exposition.

3.1 Extensions

The reader may have noted that this protocol deviates from the usual view of sending mail in that the process requires interaction between a message sender and a server. For simplicity, this point is not addressed in the main portion of the paper, however, it is quite easy to remedy. The source of the problem is that the mail server must communicate the internal address of the new message back to the sender so that the sender can update the Bloom filter with storage to contain this address at the appropriate locations. However, once again, using probabilistic methods from [25], we can solve this problem. As long as the address space is known (which just requires knowledge of the database size, which could be published) the mail sender can simply instruct the server to write the message to a number of random locations, and simultaneously send modification data which would update the Bloom filter accordingly. There are of course, prices to pay for this, but they will not be so significant. The Bloom filter with storage now has addresses of size $\log^2(n)$, since there will be a logarithmic number of addresses instead of just one and, furthermore, to ensure correctness, the database must also grow by a logarithmic factor. A detailed analysis is given in the full version of this work.

Another potential objection to our construction is that mail senders are somewhat free to access and modify the keyword-message associations. Hence, a malicious message sender could invalidate the message-keyword associations, which is another way that this protocol differs from what one may expect from a mail system. (We stress, however, that a sender has no means of modifying other

senders' mail data - only the keyword association data can be manipulated.) However, this too can be solved by using "off the shelf" protocols; namely, non-interactive efficient zero knowledge proof systems of Groth, Ostrovksy and Sahai [18]. In particular, the receiver publishes a common reference string as in [18] (based on the same cryptographic assumption that we already use in this paper; i.e., [5]). The sender is now required to include a NIZK proof that the data for updating the Bloom filter is correct according to the protocol specification. The main observation is that the theorem size is $O(\sqrt{n \log n})$ and the circuit that generates it (and its witness) are $O(\sqrt{n \log n} \cdot \text{polylog}(n))$. The [18] NIZK size is proportional to the circuit size times the security parameter. Thus, assuming poly-logarithmic security parameter, the result follows.

4 Main Construction

We present a construction of a public-key storage with keyword search that is (n, λ, θ)-correct, where the maximum number of messages to store is n, and the total number of distinct keywords that may be in use at a given time is also n (however, the keyword universe consists of arbitrary strings of bounded length, say proportional to the security parameter). Correctness will be proved under a computational assumption in a "semi-honest" model, and privacy will be proved based only on a computational assumption. In our context, the term "semi-honest" refers to a party that correctly executes the protocol, but may collect information during the protocol's execution. We assume the existence of a semantically secure public-key encryption scheme with homomorphic properties that allow the computation of polynomials of total degree two on ciphertexts, e.g., [5]. The key generation, encryption and decryption algorithms of the system will be denoted by $\mathcal{K}, \mathcal{E}$, and $\mathcal{D}$ respectively. We define the required algorithms and sub-protocols below. First, let us describe our assumptions about the parties involved: $\mathcal{X}, \mathcal{Y}$ and $\mathcal{S}$. Recall that $\mathcal{X}$ will always denote a message sender. In general, there could be many senders but, for the purposes of describing the protocol, we need only to name one. Sender $\mathcal{X}$ is assumed to hold a message, keyword(s) and the public key. Receiver $\mathcal{Y}$ holds the private key. $\mathcal{S}$ has a storage buffer for n encrypted messages, and it also has a (k, m)-Bloom filter with storage, as defined in Definition 1, implemented with fixed-length buffers and encrypted under the public key distributed by $\mathcal{Y}$. As before, $m = \lceil cnk \rceil$, where $c > 1$ is a constant; the functions and buffers are denoted by $\{h_i\}_{i=1}^{k}$ and $\{B_j\}_{j=1}^{m}$. The buffers $\{B_j\}$ will be initialized to 0 in every location. $\mathcal{S}$ maintains in its storage space encryptions of the buffers, and not the buffers themselves. We denote these encryptions $\{\widehat{B_j}\}_{j=1}^{m}$. The functions h_i are implemented by pseudo-random functions, which can be published by $\mathcal{Y}$. Recall that for $w \in \{0,1\}^*$, we defined $H_w = \{h_i(w) \mid i \in [k]\}$.

KeyGen(k): Run $\mathcal{K}(1^s)$, the key generation algorithm of the underlying cryptosystem, to create public and private keys, A_{public} and $A_{private}$ respectively. Private and public parameters for a PIR scheme are also generated by this algorithm.

$\mathsf{Send}_{\mathcal{X},\mathcal{S}}(M, K, A_{public})$: Sender $\mathcal{X}$ holds a message M, keywords K and A_{public} and wishes to send the message to $\mathcal{Y}$ via the server $\mathcal{S}$. The protocol consists of the following steps:

1. $\mathcal{X}$ modifies M to have K appended to it, and then sends $\mathcal{E}(M)$, an encryption of the modified M to $\mathcal{S}$.
2. $\mathcal{S}$ receives $\mathcal{E}(M)$, and stores it at an available address ρ in its message buffer. $\mathcal{S}$ then sends ρ back to $\mathcal{X}$.
3. For every $j \in \bigcup_{w \in K} H_w$, sender $\mathcal{X}$ writes γ copies of the address ρ to $\widehat{B_j}$, using the methods of [25]. However, the information of which buffers were written needs to be hidden from $\mathcal{S}$. For this, $\mathcal{X}$ repeatedly executes protocol $\mathsf{Modify}_{\mathcal{X},\mathcal{S}}(x, \alpha)$ for appropriate (x, α), in order to update the Bloom filter buffers. Writing a single address may take several executions of Modify depending on the size of the plaintext set in the underlying cryptosystem. Also, if $|\bigcup_{w \in K} H_w| < k|K|$, then $\mathcal{X}$ executes additional $\mathsf{Modify}(r, 0)$ invocations (for any random r) so that the total number of times that Modify is invoked is uniform among all keyword sets of equal size.

$\mathsf{Retrieve}_{\mathcal{Y},\mathcal{S}}(w, A_{private})$: $\mathcal{Y}$ wishes to retrieve all messages associated with the keyword w, and erase them from the server. The protocol proceeds as follows:

1. $\mathcal{Y}$ repeatedly executes an efficient PIR protocol (e.g., [23,10]) with $\mathcal{S}$ to retrieve the encrypted buffers $\{\widehat{B_j}\}_{j \in H_w}$ which are the Bloom filter contents corresponding to w. If $|H_w| < k$, then $\mathcal{Y}$ executes additional PIR protocols for random locations and discards the results so that the same number of executions are invoked regardless of the keyword w. Recall that $\mathcal{Y}$ possesses the seeds used for the pseudo-random functions h_i, and hence can compute H_w without interacting with $\mathcal{S}$.
2. $\mathcal{Y}$ decrypts the answers for the PIR queries to obtain $\{B_j\}_{j \in H_w}$, using the key $A_{private}$. Receiver $\mathcal{Y}$ then computes $L = \bigcap_{j \in H_w} B_j$, a list of addresses corresponding to w, and then executes PIR protocols again with $\mathcal{S}$ to retrieve the encrypted messages at each address in L. Recall that we have bounded the maximum number of messages associated with a keyword. We refer to this value as λ. Receiver $\mathcal{Y}$ will, as usual, invoke additional random PIR executions so that it appears as if every word has λ messages associated to it. After decrypting the messages, $\mathcal{Y}$ will obtain any other keywords associated to the message(s) (recall that the keywords were appended to the message during the Send protocol). Denote this set of keywords $\overline{K}$.
3. $\mathcal{Y}$ first retrieves the additional buffers $\{\widehat{B_j}\}$, for all $j \in \bigcup_{w' \neq w \in \overline{K}} H_{w'}$, using PIR queries with $\mathcal{S}$. The number of additional buffers is bounded by the constant $\theta \cdot t$. Once again, $\mathcal{Y}$ invokes additional PIR executions with $\mathcal{S}$ so that the number of PIR queries in this step of the protocol is uniform for every w. Next, $\mathcal{Y}$ modifies these buffers, removing any occurrences of any address in L. This is accomplished via repeated execution of $\mathsf{Modify}_{\mathcal{Y},\mathcal{S}}(x, \alpha)$ for appropriate x and α. Additional Modify protocols are invoked to correspond to the maximum $\theta \cdot k$ buffers.

Remark: If one wishes to separate the processes of message retrieval and message erasure, simply modify the retrieval protocol to skip the last step, and then use the current retrieval protocol as the message erasure procedure.

Theorem 2. *The Public-Key Storage with Keyword Search from the preceding construction is (n, λ, θ)-correct according to Definition 4, under the assumption that the functions h_i are pseudo-random.*

Proof sketch: This is a consequence of Claim 2.1, Claim 2.1, and Observation 1. The preceding claims were all proved under the assumption that the functions h_i were uniformly random. In our protocol, they were replaced with pseudo-random functions, but since we are dealing with non-adaptive adversaries, the keywords are chosen before the seeds are generated. Hence they are independent, and if any of the preceding claims failed to be true with pseudo-random functions in place of the h_i, our protocol could be used to distinguish the h_i from the uniform distribution without knowledge of the random seed, violating the assumption of pseudo-randomness. As we mentioned before, we can easily handle adaptive adversaries, by implementing h_i using PRF's, where the seeds are kept by the service provider, and users executing secure two-party computation protocols to get $h_i(w)$ for any w using [20] or, in the case of concurrent users, using [11] and having the common random string required by [11] being part of the public key. □

We also note that in a model with potentially malicious parties, we can apply additional machinery to force "malicious" behavior using [18] as discussed above.

Theorem 3. *Assuming CPA-security of the underlying cryptosystem*[3] *(and therefore the security of our* Modify *protocol as well), the Public Key Storage with Keyword Search from the above construction is sender private, according to Definition 5.*

Proof sketch: Suppose that there exists an adversary $\mathcal{A} \in$ PPT that can succeed in breaking the security game, from Definition 5, with some non-negligible advantage. So, under those conditions, $\mathcal{A}$ can distinguish the distribution of $\mathsf{Send}(M_0, K_0)$ from the distribution of $\mathsf{Send}(M_1, K_1)$, where the word "distribution" refers to the distribution of the transcript of the interaction between the parties. A transcript of $\mathsf{Send}(M, K)$ essentially consists of just $\mathcal{E}(M)$ and a transcript of several Modify protocols that update locations of buffers based on K. Label the sequence of Modify protocols used to update the buffer locations for K_i by $\{\mathsf{Modify}(x_{i,j}, \alpha_{i,j})\}_{j=1}^{\nu}$. Note that by our design, if $|K_0| = |K_1|$, then it will take the same number of Modify protocols to update the buffers, so the variable ν does not depend on i in this case. Now consider the following sequence of distributions:

$$\begin{array}{|llll|}
\hline
\mathcal{E}(M_0) & \mathsf{Modify}(x_{0,0}, \alpha_{0,0}) & \cdots & \mathsf{Modify}(x_{0,\nu}, \alpha_{0,\nu}) \\
\mathcal{E}(M_0) & \mathsf{Modify}(x_{0,0}, \alpha_{0,0}) & \cdots & \mathsf{Modify}(x_{1,\nu}, \alpha_{1,\nu}) \\
\vdots & \vdots & \vdots & \vdots \\
\mathcal{E}(M_0) & \mathsf{Modify}(x_{1,0}, \alpha_{1,0}) & \cdots & \mathsf{Modify}(x_{1,\nu}, \alpha_{1,\nu}) \\
\mathcal{E}(M_1) & \mathsf{Modify}(x_{1,0}, \alpha_{1,0}) & \cdots & \mathsf{Modify}(x_{1,\nu}, \alpha_{1,\nu}) \\
\hline
\end{array}$$

The first line of distributions in the sequence is the transcript distribution for $\mathsf{Send}(M_0, K_0)$ and the last line of distributions is the transcript distribution for $\mathsf{Send}(M_1, K_1)$. We assumed that there exists an adversary $\mathcal{A}$ that can distinguish

[3] For concreteness, this may be implemented using the cryptosystem of [5], in which case security relies on the subgroup decision problem (see [5]).

these two distributions. Hence, not all of the adjacent intermediate distributions can be computationally indistinguishable since computational indistinguishability is transitive. So, there exists an adversary $\mathcal{A}' \in$ PPT that can distinguish between two adjacent rows in the sequence. If $\mathcal{A}'$ distinguishes within the first $\nu + 1$ rows, then it has distinguished $\mathsf{Modify}(x_{0,j}, \alpha_{0,j})$ from $\mathsf{Modify}(x_{1,j}, \alpha_{1,j})$ for some $j \in [\nu]$ which violates our assumption of the security of Modify. And if $\mathcal{A}'$ distinguishes the last two rows, then it has distinguished $\mathcal{E}(M_0)$ from $\mathcal{E}(M_1)$ which violates our assumption on the security of the underlying cryptosystem. Either way, a contradiction. So we conclude that no such $\mathcal{A}$ exists in the first place, and hence the system is secure according to Definition 5. □

Theorem 4. *Assuming CPA-security of the underlying cryptosystem (and therefore the security of our* Modify *protocol as well), and assuming that our PIR protocol is semantically secure, the Public Key Storage with Keyword Search from the above construction is receiver private, according to Definition 6.*

Proof sketch: Again, assume that there exists $\mathcal{A} \in$ PPT that can gain a non-negligible advantage in Definition 6. Then, $\mathcal{A}$ can distinguish $\mathsf{Retrieve}(w_0)$ from $\mathsf{Retrieve}(w_1)$ with non-negligible advantage. The transcript of a Retrieve protocol consists a sequence of PIR protocols from steps 1, 2, and 3, followed by a number of Modify protocols. For a keyword w_i, denote the sequence of PIR protocols that occur in $\mathsf{Retrieve}(w_i)$ by $\{\mathrm{PIR}(z_{i,j})\}_{j=1}^{\zeta}$, and denote the sequence of Modify protocols by $\{\mathsf{Modify}(x_{i,j}, \alpha_{i,j})\}_{j=1}^{\eta}$. Note that by the design of the Retrieve protocol, there will be equal numbers of these PIR queries and Modify protocols regardless of the keyword w, and hence ζ and η are independent of i. Consider the following sequence of distributions:

$$
\begin{array}{|cccccccc|}
\hline
\mathrm{PIR}(z_{0,0}) & \cdots & \mathrm{PIR}(z_{0,\zeta}) & \mathsf{Modify}(x_{0,0}, \alpha_{0,0}) & \cdots & \mathsf{Modify}(x_{0,\eta}, \alpha_{0,\eta}) \\
\mathrm{PIR}(z_{1,0}) & \cdots & \mathrm{PIR}(z_{0,\zeta}) & \mathsf{Modify}(x_{0,0}, \alpha_{0,0}) & \cdots & \mathsf{Modify}(x_{0,\eta}, \alpha_{0,\eta}) \\
\vdots & \ddots & \vdots & \vdots & & \vdots \\
\mathrm{PIR}(z_{1,0}) & \cdots & \mathrm{PIR}(z_{1,\zeta}) & \mathsf{Modify}(x_{0,0}, \alpha_{0,0}) & \cdots & \mathsf{Modify}(x_{0,\eta}, \alpha_{0,\eta}) \\
\mathrm{PIR}(z_{1,0}) & \cdots & \mathrm{PIR}(z_{1,\zeta}) & \mathsf{Modify}(x_{1,0}, \alpha_{1,0}) & \cdots & \mathsf{Modify}(x_{0,\eta}, \alpha_{0,\eta}) \\
\vdots & & \vdots & \vdots & \ddots & \vdots \\
\mathrm{PIR}(z_{1,0}) & \cdots & \mathrm{PIR}(z_{1,\zeta}) & \mathsf{Modify}(x_{1,0}, \alpha_{1,0}) & \cdots & \mathsf{Modify}(x_{1,\eta}, \alpha_{1,\eta}) \\
\hline
\end{array}
$$

The first line is the transcript distribution of $\mathsf{Retrieve}(w_0)$ and the last line is the transcript distribution of $\mathsf{Retrieve}(w_1)$. Since there exists $\mathcal{A} \in$ PPT that can distinguish the first distribution from the last, then there must exist an adversary $\mathcal{A}' \in$ PPT that can distinguish a pair of adjacent distributions in the above sequence, due to the transitivity of computational indistinguishability. Therefore, for some $j \in [\zeta]$ or $j' \in [\eta]$ we have that $\mathcal{A}'$ can distinguish $\mathrm{PIR}(z_{0,j})$ from $\mathrm{PIR}(z_{1,j})$ or $\mathsf{Modify}(x_{0,j'}, \alpha_{0,j'})$ from $\mathsf{Modify}(x_{1,j'}, \alpha_{1,j'})$. In both cases, a contradiction of our initial assumption. Therefore, no such $\mathcal{A} \in$ PPT exists, and hence our construction is secure according to Definition 6. □

Theorem 5. *(Communication Complexity) The Public-Key Storage with Keyword Search from the preceding construction has sub-linear communication complexity in n, the number of documents held by the storage provider $\mathcal{S}$.*

Proof. From Observation 1, a (k, m)-Bloom filter with storage that is designed to store n different keywords is of linear size in n (the maximum number of elements that the filter is designed to store), k (the number of functions h_i used, which serves as a correctness parameter), σ (the size of the buffer arrays, which serves as a correctness parameter; note that σ should be chosen to exceed λ, the maximum number of values associated to any single element of the set), $l = \log n$ (the storage size of an associated value), and c (any constant greater than 1).

However, all the buffers in our construction have been encrypted, giving an extra factor of s, the security parameter. Additionally, there is another correctness parameter, γ coming from our use of the methods of [25], which writes a constant number copies of each document into the buffer. Examining the proof of Theorem 2.1, we see that the parameters k and c are indeed independent of n. However, $\{s, l, \gamma\}$ should have logarithmic dependence on n. So, the total size of the encrypted Bloom filter with storage is $\mathcal{O}(n \cdot k \cdot \sigma \cdot l \cdot c \cdot s \cdot \gamma) = \mathcal{O}(n \log^3 n)$, as all other parameters are constants or correctness parameters independent of n (i.e., their value in preserving correctness does not deteriorate as n grows).

Therefore the communication complexity of the protocol is $\mathcal{O}(\sqrt{n \log^3 n})$ for sending a message assuming honest-but-curious sender; $\mathcal{O}(\sqrt{n \log^3 n} \cdot \text{polylog}(n))$ for any malicious poly-time bounded sender; $\mathcal{O}(\text{polylog}(n))$ for reading using any polylog(n) PIR protocol, e.g. [8,10,24]; and $\mathcal{O}(\sqrt{n \log^3 n})$ for deleting messages.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable Encryption Revisited: Consistency Properties, Relation to Anonymous IBE, and Extensions. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 205–222. Springer, Heidelberg (2005)
2. Barak, B., Goldreich, O.: Universal Arguments and their Applications. In: IEEE Conference on Computational Complexity, pp. 194–203 (2002)
3. Bethencourt, J., Song, D., Waters, B.: New techniques for private stream searching. Technical Report CMU-CS-06-106, Carnegie Mellon University (March 2006)
4. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
5. Boneh, D., Goh, E., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: TCC, 325–341 (2005)
6. Boneh, D., Crescenzo, G., Ostrovsky, R., Persiano, G.: Public Key Encryption with Keyword Search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
7. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Proc. of CCS-2006, pp. 79–88 (2006)
8. Chang, Y.C.: Single Database Private Information Retrieval with Logarithmic Communication. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, Springer, Heidelberg (2004)
9. Chang, Y.C., Mitzenmacher, M.: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005)

10. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999)
11. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proc. of the thiry-fourth annual ACM symposium on Theory of computing, pp. 494–503. ACM Press, New York (2002)
12. Chor, B., Gilboa, N., Naor, M.: Private Information Retrieval by Keywords in Technical Report TR CS0917, Department of Computer Science, Technion (1998)
13. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proc. of the 36th Annu. IEEE Symp. on Foundations of Computer Science, pp. 41–51 (1995). Journal version: J. of the ACM, 45 965–981 (1998)
14. Di Crescenzo, G., Malkin, T., Ostrovsky, R.: Single-database private information retrieval implies oblivious transfer. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, Springer, Heidelberg (2000)
15. Freedman, M., Ishai, Y., Pinkas, B., Reingold, O.: Keyword Search and Oblivious Pseudorandom Functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, Springer, Heidelberg (2005)
16. Goh, E.J.: Secure indexes (2003), available at `http://eprint.iacr.org/2003/216`
17. Goldreich, O., Ostrovsky, R.: Software Protection and Simulation on Oblivious RAMs. In J. ACM 43(3), 431–473 (1996)
18. Groth, J., Ostrovsky, R., Sahai, A.: Perfect Non-interactive Zero Knowledge for NP. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 339–358. Springer, Heidelberg (2006)
19. Goldwasser, S., Micali, S.: Probabilistic encryption. In J. Comp. Sys. Sci. 28(1), 270–299 (1984)
20. Katz, J., Ostrovsky, R.: Round-Optimal Secure Two-Party Computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (2004)
21. Kilian, J.: A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In: Proc. of STOC 1992, pp. 723–732 (1992)
22. Kurosawa, K., Ogata, W.: Oblivious Keyword Search. Journal of Complexity (Special issue on coding and cryptography) 20(2-3), 356–371 (2004)
23. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science, pp. 364–373. IEEE Computer Society Press, Los Alamitos (1997)
24. Lipmaa, H.: An Oblivious Transfer Protocal with Log-Squared Communication. IACR ePrint Cryptology Archive 2004/063
25. Ostrovsky, R., Skeith, W.: Private Searching on Streaming Data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, Springer, Heidelberg (2005)
26. Ostrovsky, R., Skeith, W.: Algebraic Lower Bounds for Computing on Encrypted Data. In: Electronic Colloquium on Computational Complexity, ECCC TR07-22
27. Ostrovsky, R., Skeith, W.: A Survey of Single Database PIR: Techniques and Applications. In: Proceedings of Public Key Cryptology (PKC-2007). LNCS, Springer-Verlag/IACR, Heidelberg (2007)
28. Sander, T., Young, A., Yung, M.: Non-Interactive CryptoComputing For NC1. In: FOCS 1999, pp. 554–567 (1999)
29. Song, D.X., Wagner, D., Perrig, A.: Practical Techniques for Searches on Encrypted Data. In: Proc. of IEEE Symposium on Security and Privacy, pp. 44–55. IEEE Computer Society Press, Los Alamitos (2000)