Private Searching On Streaming Data*

Rafail Ostrovsky† William E. Skeith III‡

November 2, 2006

Abstract

In this paper, we consider the problem of private searching on streaming data, where we can efficiently implement searching for documents that satisfy a secret criteria (such as presence or absence of a hidden combination of hidden keywords) under various cryptographic assumptions. Our results can be viewed in a variety of ways: as a generalization of the notion of Private Information Retrieval (to more general queries and to a streaming environment); as positive results on privacy-preserving datamining; and as a delegation of hidden program computation to other machines.

KEYWORDS Code Obfuscation, Public Key Program Obfuscation, Program Obfuscation, Crypto-computing, Software security, Database security, Public-key Encryption with special properties, Private Information Retrieval, Privacy-Preserving Keyword Search, Secure Algorithms for Streaming Data, Privacy-Preserving Datamining, Secure Delegation of Computation, Searching with Privacy, Mobile code.

---

*Abridged version appeared at CRYPTO 2005. Patent pending.

†Department of Computer Science and Department of Mathematics, University of California, Los Angeles, 90095. E-mail: rafail@cs.ucla.edu. Supported in part by Intel equipment grant, NSF Cybertrust grant No. 0430254, OKAWA research award, B. John Garrick Foundation and Xerox Innovation group Award.

‡Department of Mathematics, University of California, Los Angeles, 90095. E-mail: wskeith@math.ucla.edu. Supported in part by NSF Cybertrust grant No. 0430254 and NSF VIGRE grant DMS-0502315.

1 Introduction

1.1 Data Filtering for the Intelligence Community

As our motivating example, we examine one the tasks of the intelligence community, which is to collect "potentially useful" information from huge streaming sources of data. The data sources are vast, and it is often impractical to keep all the data. Thus, streaming data is typically sieved from multiple data streams in an on-line fashion, one document/message/packet at a time. Most of the data is immediately dismissed and dropped to the ground, while only a small fraction of "potentially useful" data is retained. These streaming data sources, just to give a few examples, include things like packet traffic on network routers, on-line news feeds (such as Reuters.com), internet chat-rooms, or potentially terrorist-related blogs or web-cites. Again, most of the data is completely innocent and is immediately dismissed except for a small subset of the data that raises red flags which is collected for later analysis in a secure environment.

In almost all cases, what's "potentially useful" and raises a "red flag" is classified, and satisfies a secret criteria (i.e., a boolean decision whether to keep this document or throw it away). The classified "sieving" algorithm is typically written by intelligence community analysts. Keeping this sieving algorithm classified is clearly essential, since otherwise adversaries could easily prevent their messages from being collected by simply avoiding the criteria that is used to collect such documents in the first place. In order to keep the selection criteria classified, one possible solution (and in fact the one that is used in practice) is to collect *all* streaming data "on the inside", in a secure environment, and then filter the information according to classified rules, throwing away most of it and keeping only a small fraction of data-items that fit the secret criteria. Often, the criteria is simply a set of keywords that raise a red flag. While this method for collecting data certainly keeps the sieving information private, it requires *transferring* all streaming data to a classified environment, adding considerable cost in terms of communication and the risk of a delay or even loss of data, if the transfer to the classified environment is interrupted. Furthermore, it requires considerable cost of *storage* to hold this (un-sieved) data in case the transfer to the classified setting is delayed.

Clearly, a far more preferable solution, is to sieve all these data-streams at their sources (on the same computers or routers where the stream is generated or arrives in the first place). The issue, of course, is how can this be accomplished while keeping the sieving criteria classified, even if the computer where the sieving program executes falls into enemy's hands? Perhaps somewhat surprisingly, we show how to do just that while keeping the sieving criteria provably hidden from the adversary, even if the adversary is allowed to experiment with the sieving executable code and/or tries to reverse-engineer it. Put differently, we construct a "compiler" (i.e. of how to compile sieving rules) so that it is provably impossible to reverse-engineer the classified rules from the executable complied sieving code. In the following section, we state our results in more general terms that we believe are of independent interest.

1.2 Public-Key Program Obfuscation

Very informally, given a program f from a class $\mathcal{C}$ of programs, and a security parameter k, a *public-key program obfuscator* compiles f into (F, Dec), where F on any input computes an encryption of what f would compute on the same input. The decryption algorithm Dec decrypts the output of F. That is, for any input x, $Dec(F(x)) = f(x)$, but given code for F it is impossible to distinguish for any polynomial time adversary which f from the class $\mathcal{C}$ was used to produce F . We stress that in our definition, the program encoding length $|F|$ must polynomially depend only on $|f|$ and k, and the output length of $|F(x)|$ must polynomially depend only on $|f(x)|$ and k. It is easy to see that Single-Database Private Information Retrieval (including keyword search) can be viewed as a special case of public-key program obfuscation.

1.3 Obfuscating Searching on Streaming Data

We consider how to public-key program obfuscate keyword search algorithms on streaming data, where the size of the query (i.e. complied executable) must be *independent* of the size of stream (i.e., database), and that can be executed in an on-line environment, one document at a time. Our results also can be viewed as improvement and a speedup of the best previous results of single-round PIR with keyword search of Freedman, Ishai, Pinkas and Reingold [16]. In addition to the introduction of the streaming model, this paper also improves the previous work on keyword PIR by allowing for the simultaneous return of multiple documents that match a set of keywords, and also the ability to more efficiently perform different types of queries beyond just searching for a single keyword. For example, we show how to search for the disjunction of a set of keywords and several other functions.

1.4 Our Results

We consider a dictionary of finite size (e.g., an English dictionary) D that serves as the universe for our keywords. Additionally, we can also have keywords that must be absent from the document in order to match it. We describe the various properties of such filtering software below. A filtering program F stores up to some maximum number m of matching documents in an encrypted buffer B. We provide several methods for constructing such software F that saves up to m matching documents with overwhelming probability and saves non-matching documents with negligible probability (in most cases, this probability will be identically 0), all without F or its encrypted buffer B revealing any information about the query that F performs. The requirement that non-matching documents are not saved (or at worst with negligible probability) is motivated by the streaming model: in general the number of non-matching documents will be vast in comparison to those that do match, and hence, to use only small storage, we must guarantee that non-matching documents from the stream do not collect in our buffer. Among our results, we show how to execute queries that search for documents that match keywords in a disjunctive manner, i.e., queries that search for documents containing one or more keywords from a keyword set. Based on the

Paillier cryptosystem, [26], we provide a construction where the filtering software F runs in $O(l \cdot k^3)$ time to process a document, where k is a security parameter, and l is the length of a document, and stores results in a buffer bounded by $\mathcal{O}(m \cdot l \cdot k^2)$. We stress again that F processes documents one at a time in an online, streaming environment. The size of F in this case will be $O(k \cdot |D|)$ where $|D|$ is the size of the dictionary in words. Note that in the above construction, the program size is proportional to the dictionary size. We can in fact improve this as well: we have constructed a reduced program size model that depends on the Φ-*Hiding Assumption* [9]. The running time of the filtering software in this implementation is linear in the document size and is $O(k^3)$ in the security parameter k. The program size for this model is only $O(polylog(|D|))$. We also have an abstract construction based on any group homomorphic, semantically secure encryption scheme. Its performance depends on the performance of the underlying encryption scheme, but will generally perform similarly to the above constructions. As mentioned above, all of these constructions have size that is independent of the size of the data stream. Also, using the results of Boneh, Goh, and Nissim [6], we show how to execute queries that search for an "AND" of two sets of keywords (i.e., the query searches for documents that contain at least one word from K_1 *and* at least one word from K_2 for sets of keywords K_1, K_2), without asymptotically increasing the program size.

We also show several extensions: One has to do with buffer overflow, which we show how to recognize this with overwhelming probability. We also show that if you relax our definition of *correctness* (i.e. no longer require that the probability of non-matching documents being stored in the buffer is negligible), then the protocol can handle arbitrary bounded-length keywords that need not come from a finite dictionary.

To summarize, our contributions can be divided into three major areas: Introduction of the streaming model; having queries simultaneously return multiple results; and the ability to extend the semantics of queries beyond just matching a single keyword.

1.5 Comparison with Previous Work

A superficially related topic is that of "searching on encrypted data" (e.g., see [7] and the references therein). We note that this body of work is in fact not directly relevant, as the data (i.e. input stream) that is being searched is not encrypted in our setting.

The notion of obfuscation was considered by [3], but we stress that our setting is different, since our notion of public-key obfuscation allows the output to be encrypted, whereas the definition of [3] demands the output of the obfuscated code is given in the clear, making the original notion of obfuscation much more demanding.

Our notion is also superficially related to the notion of "crypto-computing" [27]. However, in this work we are concerned with programs that contain loops, and where we cannot afford to expand this program into circuits, as this will blow-up the program size.

Our work is most closely related to the notion of Single-database Private Information Retrieval (PIR), that was introduced by Kushilevitz and Ostrovsky [20] and has received a lot of subsequent attention in the literature [20, 9, 14, 24, 21, 8, 28, 22, 16]. (In the setting of multiple, non-communicating databases, the PIR notion was introduced in [11].)

In particular, the first PIR with poly-logarithmic overhead was shown by Cachin, Micali and Stadler [9], and their construction can be executed in the streaming environment. Thus the results of this paper can be viewed as a generalization of their work as well. The setting of single-database PIR keyword search was considered in [20, 10, 19] and more recently by Freedman, Ishai, Pinkas and Reingold [16]. The issue of multiple matches of a single keyword (in a somewhat different setting) was considered by Kurosawa and Ogata [19].

There are important differences between previous works and our work on single-database PIR keyword search: in the streaming model, the program size must be *independent* of the size of the stream, as the stream is assumed to be an arbitrarily large source of data and we do not need to know the size of the size of the stream when compiling the obfuscated query. In contrast, in all previous non-trivial PIR protocols, when creating the query, the user of the PIR protocol must know the upper bound on the database size while creating the PIR query. Also, as is necessary in the streaming model, the memory needed for our scheme is bounded by a value proportional to the size of a document as well as an upper bound on the total number of documents we wish to collect, but is independent of the size of the stream of documents. Finally, we have also extended the types of queries that can be performed. In previous work on keyword PIR, a single keyword was searched for in a database and a single result returned. If one wanted to query an "OR" of several keywords, this would require creating several PIR queries, and then sending each to the database. We however show how to intrinsically extend the types of queries that can be performed, without loss of efficiency or with multiple queries. In particular, all of our systems can efficiently perform an "OR" on a set of keywords and its negation (i.e. a document matches if certain keyword is absent from the document). In addition, we show how to privately execute a query that searches for an "AND" of two sets of keywords, meaning that a document will match if and only if it contains at least one word from each of the keyword sets without the increase in program (or dictionary) size.

1.6 Potential Applications

As mentioned above, there are many applications of this work for the purposes of intelligence gathering. One could monitor a vast number of internet transactions using this technology. For example, chat rooms, message boards, instant messaging, and online news feeds. It could also be run on search engine hosts to track the IP addresses of computers that search for certain suspicious phrases or words. Also, using an index of names or a phone book as a source for keywords, this work could enable a private search for locating an individual using an alias, without revealing one's knowledge of that alias. There are also several extensions of our techniques that we discuss in Section 8 that may also have practical significance. Another possible application outside of intelligence gathering, is to help facilitate a company or organization to perform an audit of its own branches without massive data transfers or commandeering actual hardware or machines.

1.7 Subsequent Work

Our definition of Public Key Program Obfuscation motivated further interest on public key obfuscation, for example see the work of Adida and Wickström on "obfuscated mixing" [1]. There have also been several follow-up works concerning practical considerations for private searching on streaming data, for example the work of Danezis and Diaz [13] and Bethencourt, Song and Waters [4, 5]. These works primarily concern the issue of reducing the buffer size of our construction by a logarithmic factor. However, this comes at the cost of making the running time for buffer decryption linear in the stream size. For further discussion see section 9.

1.8 Overview and Intuition of the Solution

Informally, our idea is to create a device that conditionally and obliviously creates encryptions (in some homomorphic encryption scheme) of documents based on the matching of keyword criteria, and then writes these encryptions to random locations in a buffer, using homomorphic properties of the encryption scheme. By "conditionally", what is meant is that if a document matches the query, the device will produce an encryption of the document itself. Otherwise, an encryption of the identity element will be produced. The key idea is that the encryption of the identity element that the software computes if the document does not match the secret criteria will be indistinguishable from the encryption of the matching document. This way, both matching and non-matching documents appear to be treated precisely the same way. The machine or anyone else who views the execution is totally unaware if the search condition is satisfied, as it is executed as a straight-line code (i.e., any branches that the program executes are independent of the search criteria), so that the conditions are never known unless the underlying encryption scheme is broken. Several probabilistic methods are then applied to ensure a strong sense of correctness. We demand that none of the documents that match the criteria are lost, and that no documents that do not satisfy the criteria are collected.

2 Definitions and Preliminaries

2.1 Basic Definitions

For a set V we denote the power set of V by $\mathcal{P}(V)$.

Definition 2.1 *Recall that a function $g : \mathbb{N} \longrightarrow \mathbb{R}^+$ is said to be* negligible *if for any $c \in \mathbb{N}$ there exists $N_c \in \mathbb{Z}$ such that $n \geq N_c \Longrightarrow g(n) \leq \frac{1}{n^c}$.*

Definition 2.2 *Let $\mathcal{C}$ be a class of programs, and let $f \in \mathcal{C}$. We define a* public key program obfuscator in the weak sense *to be an algorithm*

$$\mathsf{Compile}(f, r, 1^k) \mapsto \{F(\cdot,\cdot), \mathsf{Decrypt}(\cdot)\}$$

where r is randomness, k is a security parameter, and F and $\mathsf{Decrypt}$ *are algorithms with the following properties:*

- *(Correctness) F is a probabilistic function such that*

$$\forall x, \Pr_{R,R'}\Big[\mathsf{Decrypt}(F(x,R')) = f(x)\Big] \geq 1 - neg(k)$$

- *(Compiled Program Conciseness) There exists a constant c such that*

$$|f| \geq \frac{|F(\cdot,\cdot)|}{(|f|+k)^c}$$

- *(Output Conciseness) There exists a constant c such that For all x, R*

$$|f(x)| \geq \frac{|F(x,R)|}{k^c}$$

- *(Privacy) Consider the following game between an adversary A and a challenger C:*

  1. *On input of a security parameter k, A outputs two functions $f_1, f_2 \in \mathcal{C}$.*
  2. *C chooses a $b \in \{0,1\}$ at random and computes* $\mathsf{Compile}(f_b, r, k) = \{F, \mathsf{Decrypt}\}$ *and sends F back to A.*
  3. *A outputs a guess b'.*

  We say that the adversary wins if $b' = b$, and we define the adversary's advantage to be $\mathrm{Adv}_A(k) = |\Pr(b = b') - \frac{1}{2}|$. Finally we say that the system is secure if for all $A \in PPT$, $\mathrm{Adv}_A(k)$ is a negligible function in k.

We also define a stronger notion of this functionality, in which the decryption algorithm does not give any information about f beyond what can be learned from the output of the function alone.

Definition 2.3 *Let $\mathcal{C}$ be a class of programs, and let $f \in \mathcal{C}$. We define a* public key program obfuscator in the strong sense *to be a triple of algorithms* ($\mathsf{Key\text{-}Gen}, \mathsf{Compile}, \mathsf{Decrypt}$) *defined as follows:*

1. $\mathsf{Key\text{-}Gen}(k)$*: Takes a security parameter k and outputs a public key and a secret key $A_{public}, A_{private}$.*

2. $\mathsf{Compile}(f, r, A_{public}, A_{private})$*: Takes a program $f \in \mathcal{C}$, randomness r and the public and private keys, and outputs a program $F(\cdot,\cdot)$ that is subject to the same Correctness and conciseness properties as in Definition 2.2.*

3. $\mathsf{Decrypt}(F(x), A_{private})$*: Takes output of F and the private key and recovers $f(x)$, just as in the correctness of Definition 2.2.*

Privacy is also defined as in Definition 2.2, however in the first step the adversary A receives as an additional input A_{public} and we also require that Decrypt *reveals no information about f beyond what could be computed from $f(x)$: Formally, for all adversaries $A \in PPT$ and for all history functions h there exists a simulating program $B \in PPT$ that on input $f(x)$ and $h(x)$ is computationally indistinguishable from A on input* $(\mathsf{Decrypt}, F(x), h(x))$.

In what follows, we give instantiations of these general definitions for several classes of search programs $\mathcal{C}$. We consider a universe of words $W = \{0,1\}^*$, and a dictionary $D \subset W$ with $|D| = \alpha < \infty$. We think of a document as an ordered, finite sequence of words in W, however, it will often be convenient to look at the set of distinct words in a document, and also to look at some representation of a document as a single string in $\{0,1\}^*$. So, the term *document* will often have several meanings, depending on the context: if M is said to be a *document*, generally this will mean M is an ordered sequence in W, but at times, (e.g., when M appears in set theoretic formulas) *document* will mean (finite) element of $\mathcal{P}(W)$ (or possibly $\mathcal{P}(D)$), and at other times still, (say when one is talking of bit-wise encrypting a document) we'll view M as $M \in \{0,1\}^*$. We define a *set of keywords* to be any subset $K \subset D$. Finally, we define a *stream* of documents S simply to be any sequence of documents.

We will consider only a few types of queries in this work, however would like to state our definitions in generality. We think of a *query type*, $\mathcal{Q}$ as a class of logical expressions in $\wedge, \vee$, and $\neg$. For example, $\mathcal{Q}$ could be the class of expressions using only the operation $\wedge$. Given a query type, one can plug in the number of variables, call it α for an expression, and possibly other parameters as well, to select a specific boolean expression, Q in α variables from the class $\mathcal{Q}$. Then, given this logical expression, one can input $K \subset D$ where $K = \{k_i\}_{i=1}^{\alpha}$ and create a function, call it $Q_K : \mathcal{P}(D) \to \{0,1\}$ that takes documents, and returns 1 if and only if a document matches the criteria. $Q_K(M)$ is computed simply by evaluating Q on inputs of the form $(k_i \in M)$. We will call Q_K a *query over keywords K*.

We note again that our work does not show how to privately execute arbitrary queries, despite the generality of these definitions. In fact, extending the query semantics is an interesting open problem.

Definition 2.4 *For a query Q_K on a set of keywords K, and for a document M, we say that M* matches *query Q_K if and only if $Q_K(M) = 1$.*

Definition 2.5 *For a fixed query type $\mathcal{Q}$, a* private filter generator *consists of the following probabilistic polynomial time algorithms:*

1. $\mathsf{Key\text{-}Gen}(k)$*: Takes a security parameter k and generates public key A_{public}, and a private key $A_{private}$.*

2. $\mathsf{Filter\text{-}Gen}(D, Q_K, A_{public}, A_{private}, m, \gamma)$*: Takes a dictionary D, a query $Q_K \in \mathcal{Q}$ for the set of keywords K, along with the private key and generates a search program F. F searches any document stream S (processing one document at a time and updating B)*

collects up to m documents that match Q_K in B, outputting an encrypted buffer B that contains the query results, where $|B| = \mathcal{O}(\gamma)$ throughout the execution.

3. $\mathsf{Filter\text{-}Decrypt}(B, A_{private})$: *Decrypts an encrypted buffer B, produced by F as above, using the private key and produces output B^*, a collection of the matching documents from S.*

Definition 2.6 *(Correctness of a Private Filter Generator)*
Let $F = \mathsf{Filter\text{-}Gen}(D, Q_K, A_{public}, A_{private}, m, \gamma)$, where D is a dictionary, Q_K is a query for keywords K, $m, \gamma \in \mathbb{Z}^+$ and $(A_{public}, A_{private}) = \mathsf{Key\text{-}Gen}(k)$. We say that a private filter generator is correct *if the following holds:*

Let F run on any document stream S, and set $B = F(S)$.
Let $B^* = \mathsf{Filter\text{-}Decrypt}(B, A_{private})$. Then,

- *If $|\{M \in S \mid Q_K(M) = 1\}| \leq m$ then*

$$\Pr\Big[B^* = \{M \in S \mid Q_K(M) = 1\}\Big] > 1 - neg(\gamma)$$

- *If $|\{M \in S \mid Q_K(M) = 1\}| > m$ then*

$$\Pr\Big[(B^* \subset \{M \in S \mid Q_K(M) = 1\}) \vee (B^* = \bot)\Big] > 1 - neg(\gamma)$$

  where $\bot$ is a special symbol denoting buffer overflow, and the probabilities are taken over all coin-tosses of F, $\mathsf{Filter\text{-}Gen}$ *and of* $\mathsf{Key\text{-}Gen}$.

Definition 2.7 *(Privacy) Fix a dictionary D. Consider the following game between an adversary A, and a challenger C. The game consists of the following steps:*

1. *C first runs $\mathsf{Key\text{-}Gen}(k)$ to obtain $A_{public}, A_{private}$, and then sends A_{public} to A.*

2. *A chooses two queries for two sets of keywords, Q_{0K_0}, Q_{1K_1}, with $K_0, K_1 \subset D$ and sends them to C.*

3. *C chooses a random bit $b \in \{0, 1\}$ and executes $\mathsf{Filter\text{-}Gen}(D, Q_{bK_b}, A_{public}, A_{private}, m, \gamma)$ to create F_b, the filtering program for the query Q_{bK_b}, and then sends F_b back to A.*

4. *$A(F_b)$ can experiment with the code of F_b in an arbitrary non-black-box way, and finally outputs $b' \in \{0, 1\}$.*

The adversary wins the game if $b = b'$ and loses otherwise. We define the adversary A's advantage in this game to be

$$\mathrm{Adv}_A(k) = \left|\Pr(b = b') - \frac{1}{2}\right|$$

We say that a private filter generator is semantically secure *if for any adversary $A \in PPT$ we have that $\mathrm{Adv}_A(k)$ is a negligible function, where the probability is taken over coin-tosses of the challenger and the adversary.*

2.2 Combinatorial Lemmas

We require in our definitions that matching documents are saved with overwhelming probability in the buffer B (in terms of the size of B), while non-matching documents are not saved at all (at worst, with negligible probability). We accomplish this by the following method: If the document is of interest to us, we throw it at random γ times into the buffer. What we are able to guarantee is that if only one document lands in a certain location, and no other document lands in this location, we will be able to recover it. If there is a collision of one or more documents, we assume that all documents at this location are lost (and furthermore, we guarantee that we will detect such collisions with overwhelming probability). To amplify the probability that matching documents survive, we throw each γ times, and we make the total buffer size proportional to $2\gamma m$, where m is the upper bound on the number of documents we wish to save. Thus, we need to analyze the following combinatorial game, where each document corresponds to a ball of different color.

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 2.8 *The probability that the* color-survival game fails *is negligible in* γ*.*

Proof: We need to compute the probability that at least one of the m colors does not survive, i.e., all γ balls of one or more colors are destroyed, and show that this probability is negligible in γ. To begin, let us compute the probability that a single ball survives this process. Since the location of each ball is chosen uniformly at random, clearly these choices are independent of one another. Hence,

$$\Pr(survival) = \Big(\frac{2\gamma m - 1}{2\gamma m}\Big)^{\gamma m - 1}$$

Also recall that

$$lim_{x \to \infty}\Big(\frac{x-1}{x}\Big)^{x} = \frac{1}{e}$$

and hence

$$lim_{\gamma \to \infty}\Big(\frac{2\gamma m - 1}{2\gamma m}\Big)^{\gamma m - 1} = \frac{1}{\sqrt{e}} \approx .61$$

Furthermore, as γ increases, this function decreases to its limit, so we always have the probability of survival of a single ball is *greater* than $\frac{1}{\sqrt{e}}$ for any $\gamma > 0$.

Now, what is the probability of at least one out of the m colors having all of its γ balls destroyed by the process? First we compute the probability for just a single color. Let $\{E_j\}_{j=1}^{\gamma}$ be the events that the j-th ball of a certain color does not survive. Then the probability that all γ balls of this color do not survive is

$$\Pr\Big(\bigcap_{j=1}^{\gamma} E_j\Big) = \Pr(E_1)\Pr(E_2|E_1)\cdots\Pr(E_\gamma|E_{\gamma-1}E_{\gamma-2}\cdots E_1) < \Big(\frac{1}{2}\Big)^{\gamma}$$

We know the final inequality to be true since each of the probabilities in the right hand product are bounded above by $\frac{1}{2}$ as the probability of losing a particular ball was smaller than $(1 - \frac{1}{\sqrt{e}}) \approx .39 < 1/2$, regardless the choice of $\gamma > 0$, and given that collisions have already occurred only further reduces the probability that a ball will be lost. Now, by the union bound we have that the probability of losing all balls of at least one color is less than or equal to the sum of the probabilities of losing each color separately. So, we have

$$\Pr(\textit{at least one color does not survive}) \leq \sum_{i=1}^{m} \Pr(\textit{color}_i \textit{ does not survive}) < \frac{m}{2^\gamma}$$

which is clearly negligible in γ, which is what we wanted to show. ■

Another issue is how to distinguish valid documents in the buffer from collisions of two or more matching documents in the buffer. (In general it is unlikely that the sum of two messages in some language will look like another message in the same language, but we need to guarantee this fact.) This can also be accomplished by means of a simple probabilistic construction. We will append to each document k bits, partitioned into $k/3$ triples of bits, and then randomly set exactly one bit in each triple to 1, leaving the other two bits 0. When reading the buffer results, we will consider a document to be good if exactly one bit in each of the $k/3$ triples of appended bits is a 1. If a buffer collision occurs between two matching documents, the buffer at this location will store the sum of the messages, and the sum of 2 or more of the k-bit strings. [1] We need to analyze the probability that the sum of any number $n > 1$ of such k-bit strings *still* has exactly one bit in each of the $k/3$ triples set to 1, and show that this probability is negligible in k. We will assume that the strings add together bitwise, modulo 2 as this is the hardest case. [2] We first prove the following lemma.

Lemma 2.9 *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,*

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

Proof: We proceed by induction on n. For $n = 3$, the statement is easy to verify. Clearly there are 6 such sequences, as they are obviously in one to one correspondence with the set of all permutations of 3 items, and of course $|S_3| = 6$. Finally note that $6 = (3^3 - 3)/4$.

Now assume that for some odd integer n the statement is true. Note that the only possible sums for such sequences are $(1,0,0), (0,1,0), (0,0,1), (1,1,1)$ since the total number of bits equal to 1 in the sum must be odd since n is odd. Note also that by symmetry,

[1] If a document does not match, it will be encrypted as the 0 message, as will its appended string of k bits, so nothing will ever be marked as a collision with a non-matching document.

[2] In the general group homomorphic encryption setting, one will use a fixed non-identity element in place of 1 and the identity in place of zero, performing the same process. If the order of the non-identity element is 2, then this is the exact same experiment, and as the order increases, the strings add together more and more like a bitwise OR in which case this problem is trivial.

$(1,0,0),(0,1,0)$ and $(0,0,1)$ must all have the same number of sequences that sum to these values (since permuting coordinates induces a permutation of the set of all sequences). So, $T_n((1,0,0)) = T_n((0,1,0)) = T_n((0,0,1))$. Call this number R. Since the total number of sequences of length n is 3^n, and since they are partitioned by their sums, we have that

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

Now, we analyze the sums of the sequences of length $n+1$ from this data. For each sequence of length n that summed to $(1,0,0),(0,1,0)$ or $(0,0,1)$, there is exactly one sequence of length $n+1$ that sums to $(0,0,0)$. Hence, $T_{n+1}((0,0,0)) = 3R$. Then by symmetry again, we have that $T_{n+1}((0,1,1)) = T_{n+1}((1,0,1)) = T_{n+1}((1,1,0)) = 3R-1$. Again, we have the sequences partitioned by their sums, so using the same methods, we can compute $T_{n+2}((1,1,1))$. For each sequence of length $n+1$ that sums to $(0,1,1),(1,0,1)$ or $(1,1,0)$ there is exactly one sequence of length $n+2$ that sums to $(1,1,1)$. Hence

$$T_{n+2}((1,1,1)) = 3(3R-1) = 9\Big(\frac{3^n+1}{4}\Big) - 3 = \frac{3^{n+2}-3}{4}$$

This completes the proof. ■

Lemma 2.10 *Let H be a collection of k-bit strings, partitioned into $k/3$ triples of bits, chosen uniformly at random subject to the constraint that each triple contains exactly one bit that is set to 1. Then, if $|H| > 1$, the probability that the sum of all strings in H also satisfies the property that each triple has exactly one bit set to 1 is negligible in k.*

Proof: Let $n = |H|$. For n odd, this is an immediate corollary to Lemma 2.9. And of course if n is even, the probability is uniformly 0 since each triple would have an even number of bits set to 1 in this case. ■

2.3 Organization of the Rest of this Paper

In what follows, we will give several constructions of private filter generators, beginning with our most efficient construction using a variant of the Paillier Cryptosystem [26],[12]. We also show a construction with reduced program size using the Cachin-Micali-Stadler PIR protocol [9], then we give a construction based on any group homomorphic semantically secure encryption scheme, and finally a construction based on the work of Boneh, Goh, and Nissim [6] that extends the query semantics to include a single "AND" operation without increasing the program size.

3 Paillier-Based Construction

Definition 3.1 *Let $(G_1,\cdot),(G_2,*)$ be groups. Let $\mathcal{E}$ be the probabilistic encryption algorithm and $\mathcal{D}$ be the decryption algorithm of an encryption scheme with plaintext set G_1 and ciphertext set G_2. The encryption scheme is said to be* group homomorphic *if the encryption map*

$\mathcal{E} : G_1 \to G_2$ *has the following property:*

$$\forall\ a, b \in G_1,\ \mathcal{D}(\mathcal{E}(a \cdot b)) = \mathcal{D}(\mathcal{E}(a) * \mathcal{E}(b))$$

Note that since encryption is in general probabilistic, we have to phrase the homomorphic property using $\mathcal{D}$, instead of simply saying that $\mathcal{E}$ is a homomorphism. Equivalently, if $\mathcal{E}$ is onto G_2, one could say that the map $\mathcal{D}$ is a homomorphism of groups (in the usual sense), with each coset of $ker(\mathcal{D})$ corresponding to the set of all possible encryptions of an element of G_1. Also, as standard notation when working with homomorphic encryption as just defined, we will use id_{G_1}, id_{G_2} to be the identity elements of G_1, G_2, respectively.

3.1 Summary

We believe this construction to be our most practical and efficient solution. The running time is reasonable, and the program size is proportional to the size of the dictionary. In addition, the encrypted buffer can remain very small, due to the excellent plaintext-ciphertext ratio of the Damgård-Jurik extension to the Paillier system. This system can be used to perform queries consisting of any finite number of "OR" operations.

3.2 Brief Basics of the Paillier Cryptosystem

Recall that the plaintext and ciphertext in the Paillier cryptosystem are represented as elements of $\mathbb{Z}_n$ and $\mathbb{Z}^*_{n^2}$ respectively, where $n = pq$ is an RSA number such that $p < q$ and with the additional minor assumption that $p \nmid q - 1$. Recall also the extensions of Paillier by Damgård and Jurik in which the plaintext and ciphertext are represented as elements of $\mathbb{Z}_{n^s}$ and $\mathbb{Z}^*_{n^{s+1}}$ respectively for any $s > 0$. We will be using this extension in our work. Finally, recall that these cryptosystems are homomorphic, so in this case multiplying ciphertexts gives an encryption of the sum of the plaintexts.[3]

3.3 Private Filter Generator Construction

We now formally present the Key-Gen, Filter-Gen, and Buffer-Decrypt algorithms. The class $\mathcal{Q}$ of queries that can be executed is the class of all boolean expressions using only $\vee$. By doubling the program size, it is easy to handle an $\vee$ of both presence and absence of keywords. For simplicity of exposition, we describe how to detect collisions separately from the main algorithm.

Key-Gen(k)

Execute the key generation algorithm for the Paillier cryptosystem to find an appropriate RSA number, n and its factorization $n = pq$. We will make one additional assumption on

[3]For completeness, an exposition of the Paillier cryptosystem is provided in the appendix.

$n = pq$: we require that $|D| < \min\{p, q\}$. (We need to guarantee that any number $\leq |D|$ is a unit mod n^s.) Save n as A_{public}, and save the factorization as $A_{private}$.

Filter-Gen$(D, Q_K, A_{public}, A_{private}, m, \gamma)$

This algorithm outputs a search program F for the query $Q_K \in \mathcal{Q}$. So, $Q_K(M) = \bigvee_{w \in K}(w \in M)$.
We will use the Damgård-Jurik extension [12] to construct F as follows. Choose an integer $s > 0$ based upon the size of documents that you wish to store so that each document can be represented as a group element in $\mathbb{Z}_{n^s}$. Then F contains the following data:

- A buffer B consisting of $2\gamma m$ blocks with each the size of two elements of $\mathbb{Z}^*_{n^{s+1}}$ (so, we view each block of B as an ordered pair $(v_1, v_2) \in \mathbb{Z}^*_{n^{s+1}} \times \mathbb{Z}^*_{n^{s+1}}$). Furthermore, we will initialize every position to $(1, 1)$, two copies of the identity element.
- An array $\widehat{D} = \{\widehat{d_i}\}_{i=1}^{|D|}$ where each $\widehat{d_i} \in \mathbb{Z}^*_{n^{s+1}}$ such that $\widehat{d_i}$ is an encryption of $1 \in \mathbb{Z}_{n^s}$ if $d_i \in K$ and is an encryption of 0 otherwise. (Note: We of course use re-randomized encryptions of these values for each entry in the array.)

F then proceeds with the following steps upon receiving an input document M from the stream:

1. Construct a temporary collection $\widehat{M} = \{\widehat{d_i} \in \widehat{D} \mid d_i \in M\}$.
2. Compute
$$v = \prod_{\widehat{d_i} \in \widehat{M}} \widehat{d_i}$$
3. Compute v^M and multiply (v, v^M) into γ random locations in the buffer B, just as in our combinatorial game from section 2.2.

Note that the private key actually is not needed. The public key alone will suffice for the creation of F.

Buffer-Decrypt$(B, A_{private})$

First, this algorithm simply decrypts B one block at a time using the decryption algorithm for the Paillier system. Each decrypted block will contain the 0 message (i.e., $(0, 0)$) or a non-zero message, $(w_1, w_2) \in \mathbb{Z}_{n^s} \times \mathbb{Z}_{n^s}$. Blocks with the 0 message are discarded (collisions can easily be detected and discarded using Lemma 2.10, and Lemma 3.2). A non-zero message (w_1, w_2) will be of the form (c, cM') if no collisions have occurred at this location, where c is the number of distinct keywords from K that appear in M'. So to recover M', simply compute w_2/w_1 and add this to the array B^*. (We know that any non-zero w_1 will be a unit as we required that $|D| < min\{p, q\}$.) Finally, output B^*.

In general, the filter generation and buffer decryption algorithms will make use of Lemma 2.10, having the filtering software append a validation string to each message and having the buffer decryption algorithm save documents to the output B^* only when the validation string is valid. In any of our constructions, this can be accomplished by adding r extra blocks the size of the security parameter to an entry in the buffer to represent the bits of the validation string, however this will be undesirable in many settings where the plaintext group is large (e.g., our Paillier-based construction) as this would cause a significant increase in the size of the buffer. But of course, there will generally be efficient solutions in these cases, as shown below for the Paillier-based system.

Lemma 3.2 *With $\mathcal{O}(k)$ additional bits added to each block of B, we can detect all collisions of matching documents with probability $> 1 - neg(k)$.*

Proof: Since $log(|D|)$ will be much smaller than the security parameter k, we can encode the bits from Lemma 2.10 using $\mathcal{O}(k)$ bits via the following method. Let $t = log(|D|)$, which is certainly an upper bound for the number of bits of c, and will be considerably smaller than k. Let $r = k/t$. Let (v, v^M) be as described in the filter generation algorithm, so that v is an encryption of c, the number of keywords present in M. Pick a subset $T \subset \{0, 1, 2, ..., r-1\}$ of size $r/3$, uniformly at random in the format of Lemma 2.10 (so that exactly one of every three consecutive numbers is selected, i.e. among all $j \in \{0, 1, ..., r-1\}$ having the same quotient when divided by 3, only one such j will be in T). Then compute

$$x = \sum_{j \in T} 2^{tj} \text{ and } h = v^x$$

Now, h will encrypt a value that has exactly $r/3$ of the r, t-bit blocks containing non-zero bits as in Lemma 2.10. So, the filtering software would now write (v, v^M, h) to the buffer instead of just (v, v^M). The decryption of h can now be used as in Lemma 2.10 to distinguish collisions from valid documents, with only one more ciphertext per block. [4] Also, if one wishes to increase this security parameter r beyond k/t, then of course additional ciphertexts can be added to each block of the buffer, using them in the same manner. ■

3.4 Correctness

We need to show that if the number of matching documents is less than m, then

$$Pr\Big[B^* = \{M \in S \mid Q_K(M) = 1\}\Big] > 1 - neg(\gamma)$$

and otherwise, we have that B^* is a subset of the matching documents (or contains the overflow symbol, $\perp$). Provided that the buffer decryption algorithm can distinguish collisions in the buffer from valid documents (see above remark) this equates to showing that non-matching documents are saved with negligible probability in B and that matching documents are saved with overwhelming probability in B. These two facts are easy to show.

[4] This does not follow the form of Lemma 2.10 exactly, as exclusive OR is not the operation that is performed on the plaintext upon multiplying ciphertexts. However, having them added as they are here obviously further decreases the probability that a collision will look valid.

1. Are non-matching documents stored with negligible probability? Yes. In fact, they are stored with probability 0 since clearly a non-matching document M never affects the buffer: if M does not match, then v from step 2 will be an encryption of 0, as will be v^M. So, the private filter will multiply a encryptions of 0 into the buffer at various locations which by the homomorphic property of our encryption scheme has the effect of adding 0 to the plaintext corresponding to whatever encrypted value was in B. So clearly, non-matching documents are saved with probability 0. ■

2. Are all matching documents saved with overwhelming probability? If M does match, i.e., it contains $c > 0$ keywords from K, then v computed in step 2 will be an encryption of $c > 0$. So, v^M will be an encryption of cM. This encryption is then multiplied into the buffer just as in the color-survival game from 2.2, which we have proved saves all documents with overwhelming probability in γ. But we have written an encryption of cM and not of M in general. However, this will not be a problem as $c < \min\{p, q\}$ since $c \leq |K| < |D|$, and hence $c \in \mathbb{Z}_{n^s}^*$. So, the Buffer-Decrypt algorithm will be able to successfully divide by c and recover the message M. ■

3.5 Buffer Overflow Detection

For this construction, it is quite simple to create an overflow flag for the encrypted buffer. For a document M, define

$$v_M = \prod_{\widehat{d}_i \in \widehat{M}} \widehat{d}_i$$

just as above. Note that v_M encrypts the number of distinct keywords present in M. Then the value

$$V = \prod_{M \in S} v_M$$

will of course be an encryption of an upper bound on the number of matching documents that have been written to the buffer, where here S is the document stream. This encrypted value can be stored and maintained as a prefix of the buffer. If a reasonable estimate for the average number of keywords per matching document is available, then of course a more accurate detection value can be obtained. Note that although one may be tempted to use this value interactively to determine when to retrieve the buffer contents, this is potentially dangerous as this interaction between the parties could be abused to gain information about the keywords.

3.6 Efficiency in Time and Space

We compute now the efficiency of the software in relation to the security parameter k, the size of the dictionary D, the number of documents to be saved m, and the size of a document M.

1. **Time Efficiency**. For the software to process a given document it performs a number of multiplications proportional to the size of a document, followed by a single modular

exponentiation, and then followed by 2γ additional multiplications. Modular exponentiation takes $O(k^3)$ time which is clearly the dominating term since the multiplications take at worst quadratic time in k (using long multiplication) for a fixed document size. So we conclude that our private filter takes time $O(k^3)$ for fixed document size. If you instead fix the security parameter and analyze the filter based on document size, $|M|$, the running time will again be cubic as the modular exponentiation takes cubic time in the number of bits of a document. However, the running time could of course be changed to linear in the document length if you process documents in blocks, instead of as a whole. (I.e., compute v by examining the entire document, just as before, and then write the document to the buffer in smaller blocks.) So, the running time would be quadratic in k times linear in document length. Note: for $k = 1024$, modular exponentiation on a somewhat modern computer (2 GHz Pentium processor) can be accomplished in less than 0.03 seconds, so it seems that such a protocol could be practically implemented.

2. **Space Efficiency**. The largest part of the program is the array $\widehat{D}$. If you process documents in blocks, this array will take approximately $k \cdot |D|$ bits. However, if documents are processed as a whole, then the array will take $O(|M| \cdot |D|)$. The rest of the program size remains constant in terms of the variables we're studying, so these estimates hold for the size of the entire program. The size of the buffer, $B(\gamma)$ was set to be $4\gamma m$ times the size of a ciphertext value. However, since the ciphertext-plaintext size ratio approaches 1 as the message size increases (they differ by a constant number of bits) in the Damgård-Jurik system, this solution seems near optimal in terms of buffer size. A loose upper bound on the probabilities of losing a document is given by the proof of Lemma 2.8. The estimate it gives is that the probability of losing a single document is less than the total number of documents to be saved divided by 2^γ. Just using the basic Paillier scheme (ciphertext size to plaintext size ratio = 2), and dividing a message into message chunks, a buffer of approximately 50 times that of the *plaintext size* of the documents you expect to store (i.e., $\gamma = 13$) produces probabilities of success around .99 for $m = 100$, (again, m is the number of documents to be stored).

Theorem 3.3 *Assuming that the Paillier (and Damgård-Jurik) cryptosystems are semantically secure, then the private filter generator from the preceding construction is semantically secure according to Definition 2.7.*

Proof: Denote by $\mathcal{E}$ the encryption algorithm of the Paillier/Damgård-Jurik cryptosystem. Suppose that there exists an adversary A that can gain a non-negligible advantage ϵ in our semantic security game from Definition 2.7. Then A could be used to gain an advantage in breaking the semantic security of the Paillier encryption scheme as follows: Initiate the semantic security game for the Paillier encryption scheme with some challenger C. C will send us an integer n for the Paillier challenge. For messages m_0, m_1, we choose $m_0 = 0 \in \mathbb{Z}_{n^s}$ and choose $m_1 = 1 \in \mathbb{Z}_{n^s}$. After sending m_0, m_1 back to C, we will receive $e_b = \mathcal{E}(m_b)$, an encryption of one of these two values. Next we initiate the private filter generator semantic security game with A. A will give us two queries Q_0, Q_1 in $\mathcal{Q}$ for some sets of keywords K_0, K_1, respectively. We use the public key n to compute an encryption of 0, call it $e_0 = \mathcal{E}(0)$. Now we pick a random bit q, and construct filtering software for Q_q as follows: we proceed as described above, constructing the array $\widehat{D}$ by using re-randomized encryptions, $\mathcal{E}(0)$ of 0 for all words in $D \setminus K_q$, and for the elements of K_q, we use $\mathcal{E}(0)e_b$, which are randomized encryptions of m_b. Now we give this program back to A, and A returns a guess q'. With probability 1/2, e_b is an encryption of 0, and hence the program that we gave A does not search for anything at all, and in this event clearly A's guess is independent of q, and hence the probability that $q' = q$ is 1/2. However, with probability 1/2, $e_b = \mathcal{E}(1)$, hence the program we've sent A is filtering software that searches for Q_q, constructed exactly as in the Filter-Gen algorithm, and hence in this case with probability $1/2 + \epsilon$, A will guess q correctly, as our behavior here was indistinguishable from an actual challenger. We determine our guess b' as follows: if A guesses $q' = q$ correctly, then we will set $b' = 1$, and otherwise we will set $b' = 0$. Putting it all together, we can now compute the probability that our guess is correct:

$$\Pr(b' = b) = \frac{1}{2}\left(\frac{1}{2}\right) + \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}$$

and hence we have obtained a non-negligible advantage in the semantic security game for the Paillier system, a contradiction to our assumption. Therefore, our system is secure according to Definition 2.7. ■

4 Reducing Program Size Below Dictionary Size

In our other constructions, the program size is proportional to the size of the dictionary. By relaxing our definition slightly, we are able to provide a new construction using Cachin-Micali-Stadler PIR [9] which reduces the program size. Security of this system depends on the security of [9] which uses the Φ-*Hiding Assumption*.[5]

The basic idea is to have a standard dictionary D agreed upon ahead of time by all users, and then to replace the array $\widehat{M}$ in the filtering software with PIR queries that execute on a database consisting of the characteristic function of M with respect to D to determine if keywords are present or not. The return of the queries is then used to modify the buffer. This will reduce the size of the distributed filtering software. However, as mentioned above, we will need to relax our definition slightly and publish an upper bound U for $|K|$, the number of keywords used in a search.

4.1 Private Filter Generation

We now formally present the Key-Gen, Filter-Gen, and Buffer-Decrypt algorithms of our construction. The class $\mathcal{Q}$ of queries that can be executed by this protocol is again just the set of boolean expressions in only the operator $\vee$ over presence or absence of keywords, as

[5] It is an interesting open question how to reduce the program size under other cryptographic assumptions.

discussed above. Also, an important note: for this construction, it is necessary to know the set of keywords being used during key generation, and hence what we achieve here is only weak public key program obfuscation, as in Definition 2.2. For consistency of notation, we still present this as 3 algorithms, even though the key generation could be combined with the filter generation algorithm. For brevity, we omit the handling of collision detection, which is handled using Lemma 2.10.

Key-Gen(k, K, D)

The CMS algorithms are run to generate PIR queries, $\{q_j\}$ for the keywords K, and the resulting factorizations of the corresponding composite numbers $\{m_j\}$ are saved as the key, $A_{private}$, while A_{public} is set to $\{m_j\}$.

Filter-Gen($D, Q_K, A_{public}, A_{private}, m, \gamma$)

This algorithm constructs and outputs a private filter F for the query Q_K, using the PIR queries q_j that were generated in the Key-Gen(k, K, D) algorithm. It operates as follows. F contains the following data:

- The array of CMS PIR queries, $\{q_j\}_{j=1}^{U}$ from the first algorithm, which are designed to retrieve a bit from a database having size equal to the number of words in the agreed upon dictionary, D. Only $|K|$ of these queries will be meaningful. For each $w \in K$, there will be a meaningful query that retrieves the bit at index corresponding to w's index in the dictionary. Let $\{p_{j,l}\}_{l=1}^{|D|}$ be the primes generated by the information in q_j, and let m_j be composite number part of q_j. The leftover $U - |K|$ queries are set to retrieve random bits.
- An array of buffers $\{B_j\}_{j=1}^{U}$, each indexed by blocks the size of elements of $\mathbb{Z}_{m_j}^*$, with every position initialized to the identity element.

The program then proceeds with the following steps upon receiving an input document M:

1. Construct the complement of the characteristic vector for the words of M relative to the dictionary D. I.e., create an array of bits $\bar{D} = \{\bar{d}_i\}$ of size $|D|$, such that $\bar{d}_i = 0 \Leftrightarrow d_i \in M$. We'll use this array as our database for the PIR protocols.

   Next, for each $j \in \{1, 2, ..., U\}$, do the following steps:

2. Execute query q_j on $\bar{D}$ and store the result in r_j.
3. Bitwise encrypt M, using r_j to encrypt a 1 and using the identity of $\mathbb{Z}_{m_j}^*$ to encrypt a 0.
4. Take the jth encryption from step 3 and position-wise multiply it into a random location in buffer B_j γ-times, as described in our color-survival game from section 2.

Buffer-Decrypt($B, A_{private}$)

Simply decrypts each buffer B_j one block at a time by interpreting each group element with $p_{j,i}$th roots as a 0 and other elements as 1's, where i represents the index of the bit that is searched for by query q_j. All valid non-zero decryptions are stored in the output B^*.

4.2 Correctness of Private Filter

Since CMS PIR is not deterministic, it is possible that our queries will have the wrong answer at times. However, this probability is negligible in the security parameter. Again, as we've seen before, provided that the decryption algorithm can distinguish valid documents from collisions in buffer, correctness equates to storing non-matching documents in B with negligible probability and matching documents with overwhelming probability. These facts are easy to verify:

1. Are non-matching documents stored with negligible probability? Yes. With overwhelming probability, a non-matching document M will not affect any of the meaningful buffers. If M does not match, then the filtering software will (with very high probability) compute subgroup elements for all of the important r_j's. So, the encryption using these r_j's will actually be an encryption of the 0 message, and by our above remarks, will have no effect on the buffer.

2. Are matching documents saved with overwhelming probability? If M does match, i.e., it contains a keyword from K, then with very high probability, we will have at least one r_j that is not in the specified subgroup, and hence, the message will be properly encrypted and stored in the buffer. And since we used the method from our combinatorial game in section 2.2 to fill each buffer with documents, with overwhelming probability all matching documents will be saved.

4.3 Efficiency of Filtering Software in Time and Space

We compute now the efficiency of the software in relation to the security parameter k, the size of the dictionary D, the upper bound on the keywords U, and the number of documents to be saved n.

1. **Time Efficiency**. For the software to process a given document it needs to run U CMS PIR queries. To answer each query requires a number of modular exponentiations equal to the size of the dictionary, and each modular exponentiation takes about $O(k^3)$ time. This procedure is at worst linear in the number of words of a document (to construct the database for the PIR queries) so, we conclude that the running time is in fact $O(k^3)$.

2. **Space Efficiency**. The only variable-sized part of the program now is the PIR queries. Each CMS PIR query consists of only polylogarithmic bits in terms of the dictionary size, $|D|$. So, in general this could be an advantage.

Theorem 4.1 *Assuming that the Φ-Assumption holds, the Private Filter Generator from the preceding construction is semantically secure according to Definition 2.2.*

Proof: If an adversary can distinguish any two keyword sets, then the adversary can also distinguish between two fixed keywords, by a standard hybrid argument. This is precisely what it means to violate the privacy definition of [9], which is proven under the Φ-Assumption. ■

5 Eliminating the Probability of Error with Perfect Hash Functions

In this section, we present ways to reduce the probability of collisions in the buffer by using perfect hash functions. Recall the definition of perfect hash function. For a set $S \subset \{1, ..., m\}$, if a function $h : \{1, ..., m\} \to \{1, ..., n\}$ is such that $h|_S$ (the restriction of h to S) is injective, then h is called a *perfect hash function* for S. We will be concerned with families of such functions. We say that H is an (m, n, k)*-family of perfect hash functions* if $\forall S \subset \{1, ..., m\}$ with $|S| = k$, $\exists h \in H$ such that h is perfect for S.

We will apply these families in a very straightforward way. Namely, we define m to be the number of documents in the stream and k to be the number of documents we expect to save. Then, since there exist polynomial size (m, n, k)-families of perfect hash functions H, then our system could consist of $|H|$ buffers, each of size n documents, and our protocol would just write each (potential) encryption of a document to each of the $|H|$ buffers once, using the corresponding hash function from H to determine the index in the buffer. Then, no matter which of the $\binom{m}{k}$ documents were of interest, at least one of the functions in H would be injective on that set of indexes, and hence at least one of our buffers would be free of collisions.

We note that the current proven upper bounds on the sizes of such families do not necessarily improve our results; the purpose of this section is theoretical, the point being that we can *eliminate* the probability of losing a matching document in a non-trivial way.

In the work of Mehlhorn [23], the following upper bound for the size of perfect hash function families is proved, where H is an (m, n, k)-family as defined above:

$$|H| \leq \left\lceil \frac{\log\binom{m}{k}}{\log(n^k) - \log(n^k - k!\binom{n}{k})} \right\rceil$$

This result could be used in practice, but would generally not be as space efficient as our other models. However, if the lower bounds proved in [15] were achieved, then we could make such a system practical. For example, if one wanted to save 25 documents from a stream of tens of thousands of documents (say ≈ 60000), then 7 buffers the size of 250 documents each could be used to save 25 documents without any collisions in at least one of the buffers.

6 Construction Based On Any Homomorphic Encryption

We provide here an abstract construction based upon an arbitrary homomorphic, semantically secure public key encryption scheme. The class of queries $\mathcal{Q}$ that are considered here is again, all boolean expressions in only the operation $\vee$, over presence or absence of keywords, as discussed above. This construction is similar to the Paillier-based construction, except that since we encrypt bitwise, we incur an extra multiplicative factor of the security parameter k in the buffer size. However, both the proof and the construction are somewhat simpler and can be based on any homomorphic encryption.

6.1 Preliminaries

Throughout this section, let $\mathcal{PKE} = \{\mathcal{KG}, \mathcal{E}, \mathcal{D}\}$ be a public key encryption scheme. Here, $\mathcal{KG}, \mathcal{E}, \mathcal{D}$ are key generation, encryption, and decryption algorithms, respectively.

Semantically Secure Encryption

For an encryption scheme, we define semantic security in terms of the following game between an adversary A and a challenger C, consisting of the following steps:

1. C runs the key generation algorithm $\mathcal{KG}(k)$, and sends all public parameters to A.
2. A chooses two messages of equal length, M_0, M_1 and sends them to C.
3. C chooses a random bit $b \in \{0, 1\}$, computes $c = \mathcal{E}(M_b)$, an encryption of M_b, and sends this ciphertext c to A.
4. A outputs a guess $b' \in \{0, 1\}$.

We say that A wins the game if $b' = b$ and loses otherwise. We define the adversary A's advantage in this game to be

$$\mathrm{Adv}_A(k) = |\Pr(b = b') - \frac{1}{2}|$$

The encryption scheme is said to be *semantically secure* if for any adversary $A \in \mathrm{PPT}$ we have that $\mathrm{Adv}_A(k)$ is a negligible function.

6.2 Construction of Abstract Private Filter Generator

Let $\mathcal{PKE} = \{\mathcal{KG}, \mathcal{E}, \mathcal{D}\}$ be a group homomorphic, semantically secure, public key encryption scheme, satisfying Definition 3.1. We describe the Key-Gen, Filter-Gen, and Buffer-Decrypt algorithms. We will write the group operations of G_1 and G_2 multiplicatively. (As usual, G_1, G_2 come from a distribution of groups in some class depending on the security parameter, but to avoid confusion and unnecessary notation, we will always refer to them simply as G_1, G_2 where it is understood that they are actually sampled from some distribution based on k.)

Key-Gen(k)

Execute $\mathcal{KG}(k)$ and save the private key as $A_{private}$, and save the public parameters of $\mathcal{PKE}$ as A_{public}.

Filter-Gen($D, Q_K, A_{public}, A_{private}, m, \gamma$)

This algorithm constructs and outputs a filtering program F for Q_K, constructed as follows. F contains the following data:

- A buffer $B(\gamma)$ of size $2\gamma m$, indexed by blocks the size of an element of G_2 times the document size, with every position initialized to id_{G_2}.
- Fix an element $g \in G_1$ with $g \neq id_{G_1}$. The program contains an array $\widehat{D} = \{\widehat{d_i}\}_{i=1}^{|D|}$ where each $\widehat{d_i} \in G_2$ such that $\widehat{d_i}$ is set to $\mathcal{E}(g) \in G_1$ if $d_i \in K$ and it is set to $\mathcal{E}(id_{G_1})$ otherwise. (Note: we are of course re-applying $\mathcal{E}$ to compute each encryption, and not re-using the same encryption with the same randomness over and over.)

F then proceeds with the following steps upon receiving an input document M:

1. Construct a temporary collection $\widehat{M} = \{\widehat{d_i} \in \widehat{D} \mid d_i \in M\}$.
2. Choose a random subset $S \subset \widehat{M}$ of size $\lceil |\widehat{M}|/2 \rceil$ and compute
$$v = \prod_{s \in S} s$$
3. Bitwise encrypt M using encryptions of id_{G_1} for 0's and using v to encrypt 1's to create a vector of G_2 elements.
4. Choose a random location in B, take the encryption of step 3, and position-wise multiply these two vectors storing the result back in B at the same location.
5. Repeat steps 2-4 $(\frac{c}{c-1})\gamma$ times, where in general, c will be a constant approximately the size of G_1.

Buffer-Decrypt$(B, A_{private})$

Decrypts B one block at a time using the decryption algorithm $\mathcal{D}$ to decrypt the elements of G_2, and then interpreting non-identity elements of G_1 as 1's and id_{G_1} as 0, storing the non-zero, valid messages in the output B^*.

6.3 Correctness of Abstract Filtering Software

Again, provided that the decryption algorithm can distinguish valid documents from collisions in buffer, correctness equates to storing non-matching documents in B with negligible probability and matching documents with overwhelming probability, which can be seen as follows:

1. Are non-matching documents stored with negligible probability? Yes. In fact, they are stored with probability 0 since clearly if a document M does not match, then all of the values in $\widehat{M}$ will be encryptions of id_{G_1} and hence so will the value v. So, the buffer contents will be unaffected by the program executing on input M.
2. Are all matching documents saved with overwhelming probability? First of all, observe that if M contains at least one keyword, step 2 will compute v to be an encryption of a non-identity element of G_1 with probability at least 1/2, regardless of what G_1 is (as long as $|G_1| > 1$). So, by only repeating steps 2-4 a small number of times, the

probability that a matching document will be written at least once becomes exponentially close to 1. We will choose the number of times to repeat steps 2-4 so that the expected number of non-identity v's that we will compute will be equal to γ. Then, we will essentially be following the method in our "color-survival" game from section 2.2 for placing our documents in the buffer, and hence all documents will be saved with overwhelming probability in γ.

Theorem 6.1 *Assuming that the underlying encryption scheme is semantically secure, the Private Filter Generator from the preceding construction is semantically secure according to Definition 2.7.*

Proof: Suppose that there exists an adversary A that can gain a non-negligible advantage ϵ in our private data collection semantic security game. Then A could be used to gain an advantage in breaking the semantic security of $\mathcal{PKE}$ as follows: We initiate the semantic security game for $\mathcal{PKE}$ with some challenger C, and for the plaintext messages m_0, m_1 in this game, we choose $m_0 = id_{G_1}$ and choose m_1 to be $g \in G_1$, where $g \neq id_{G_1}$. After sending m_0, m_1 to our opponent C in the semantic security game, we will receive $e_b = \mathcal{E}(m_b)$, an encryption of one of these two values. Next we initiate the private data collection semantic security game with A, where we play the role of the challenger. A will give us two sets of keywords $K_0, K_1 \subset D$. We assume that we have access to $\mathcal{E}$ since the system was assumed to be public key, so we can compute $e_{id} = \mathcal{E}(id_{G_1})$.[6] Now we pick a random bit q, and construct filtering software for K_q as follows: we proceed as described above, constructing the array $\widehat{D}$ by using re-randomized encryptions $\mathcal{E}(id_{G_1})$ of the identity[7] for all words in $D \setminus K_q$, and for the elements of K_q, we use $\mathcal{E}(e_{id})e_b$, which will be a randomized encryption of m_b by our assumption that the system was homomorphic.[8] Now we give this program back to A, and A returns a guess q'. With probability 1/2, the program that we gave A does not search for anything at all, and in this event, clearly A's guess is independent of q, and hence the probability that $q' = q$ is 1/2. However, with 1/2 probability, the program we've sent A searches for K_q (and is in fact indistinguishable from programs that are actually created with the Filter-Gen algorithm), and hence in this case with probability $1/2 + \epsilon$, A will guess q correctly. We determine our guess b' as follows: if A guesses $q' = q$ correctly, then we will set $b' = 1$, and otherwise we will set $b' = 0$. Putting it all together, we can now compute the probability that our guess is correct:

$$\Pr(b' = b) = \frac{1}{2}\left(\frac{1}{2}\right) + \frac{1}{2}\left(\frac{1}{2} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}$$

and hence we have obtained a non-negligible advantage in the semantic security game for $\mathcal{PKE}$, a contradiction to our assumption. Therefore, our system is secure according to Definition 2.7. ■

[6] In most cases, just having an encryption of id_{G_1}, without access to $\mathcal{E}$ will suffice.

[7] Using e_{id}^r for random r would generally suffice

[8] Again, one could generally get away with using $e_{id}^r e_b$ if the group has simple enough (e.g., cyclic) structure. We just need to ensure that the distribution of encryptions we produce here is truly indistinguishable from the distributions created the Filter-Gen algorithm. This is the main reason why we required the underlying system to be public key- it in general will not be necessary, but at this level of abstraction, how else can one come up with uniform encryptions?

7 Construction For a Single AND

7.1 Handling Several AND Operations by Increasing Program Size

We note that there are several simple (and unsatisfactory) modifications that can be made to our basic system to compute an AND. For example a query consisting of at most a c AND operations can be performed simply by changing the dictionary D to a dictionary D' containing all $|D|^c$ c-tuples of words in D, which of course comes at a polynomial blow-up[9] of program size.[10] So, only constant, or logarithmic size keyword sets can be used in order to keep the program size polynomial.

7.2 Brief Basics of the Boneh, Goh, Nissim Cryptosystem

In [6], the authors make use of groups that support a bilinear map. In what follows, let $\mathbb{G}, \mathbb{G}_1$ be two cyclic groups of order $n = q_1 q_2$, a large composite number, and let g be a generator of $\mathbb{G}$. A map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_1$ is called a bilinear map if for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}$, we have that $e(u^a, v^b) = e(u, v)^{ab}$. Also, we require that $\langle e(g, g)\rangle = \mathbb{G}_1$ for any choice of a generator $g \in \mathbb{G}$. This bilinear map will serve as our multiplication operator for encrypted values, and hence only one such multiplication is possible.

The security of the system is based on a subgroup indistinguishability assumption, related to the difficulty of computing discrete logs in the groups $\mathbb{G}, \mathbb{G}_1$. More formally, it is as follows.

Let $\mathcal{G}(k)$ be an algorithm that returns $(q_1, q_2, \mathbb{G}, \mathbb{G}_1, e)$ as described above, where k is the number of bits of the primes q_1, q_2. Then the subgroup decision problem is simply to distinguish the distribution $(n, \mathbb{G}, \mathbb{G}_1, e, x)$ from the distribution $(n, \mathbb{G}, \mathbb{G}_1, e, x^{q_2})$, where x is uniformly random in $\mathbb{G}$, and the other variables come from a distribution determined by $\mathcal{G}$. Clearly this is a stronger assumption than the hardness of factoring, and it is also a stronger assumption than the hardness of discrete logs.[11] For an algorithm $A \in \mathrm{PPT}$, the hardness assumption is formalized by first defining the advantage of the adversary to be:

$$\mathrm{Adv}_A(k) = \left|\Pr\left[A(n, \mathbb{G}, \mathbb{G}_1, e, x) = 1\right] - \Pr\left[A(n, \mathbb{G}, \mathbb{G}_1, e, x^{q_2}) = 1\right]\right|$$

where the probabilities are taken over samples of $\mathcal{G}(k)$ to generate the $(n, \mathbb{G}, \mathbb{G}_1, e)$ and over x which was uniformly random in $\mathbb{G}$. One then says that $\mathcal{G}$ satisfies the subgroup decision problem if $\mathrm{Adv}_A(k)$ is negligible in k.

[9] Asymptotically, if we treat $|D|$ as a constant, the above observation allows a logarithmic number of AND operations with polynomial blow-up of program size. It is an interesting open problem to handle more than a logarithmic number of AND operations, keeping the program size polynomial.

[10] One suggestion that we received for an implementation of "AND" is to keep track of several buffers, one for each keyword or set of keywords, and then look for documents that appear in each buffer after the buffers are retrieved, however this will put many non-matching documents in the buffers, and hence is inappropriate for the streaming model. Furthermore, it really just amounts to searching for an OR and doing local processing to filter out the difference.

[11] One could just pick a generator g of $\mathbb{G}$, compute the log of the last parameter (x or x^{q_2}) with respect to the base g, and then compute the gcd with n to distinguish.

7.3 Executing AND Without Increasing Program Size

Using the results of Boneh, Goh, and Nissim [6], we can extend the types of queries that can be privately executed to include queries involving a single AND of an OR of two sets of keywords without increasing the program size. This construction is very similar to the abstract construction, and hence several details that would be redundant will be omitted from this section. The authors of [6] build an additively homomorphic public key cryptosystem that is semantically secure under this subgroup decision problem. The plaintext set of the system is $\mathbb{Z}_{q_2}$, and the ciphertext set can be either $\mathbb{G}$ or $\mathbb{G}_1$ (which are both isomorphic to $\mathbb{Z}_n$). However, the decryption algorithm requires one to compute discrete logs. Since there are no known algorithms for efficiently computing discrete logs in general, this system can only be used to encrypt small messages.[12] Using the bilinear map e, this system has the following homomorphic property. Let $F \in \mathbb{Z}_{q_2}[X_1, ..., X_u]$ be a multivariate polynomial *of total degree 2* and let $\{c_i\}_{i=1}^u$ be encryptions of $\{x_i\}_{i=1}^u$, $x_i \in \mathbb{Z}_{q_2}$. Then, one can compute an encryption c_F of the evaluation $F(x_1, ..., x_u)$ of F on the x_i with only the public key. This is done simply by using the bilinear map e in place of any multiplications in F, and then multiplying ciphertexts in the place of additions occurring in F. I.e., if $\mathcal{E}$ is the encryption map and if

$$F = \sum_{1 \leq i \leq j \leq u} a_{ij} X_i X_j$$

then from $\{c_l = \mathcal{E}(x_l)\}_{l=1}^u$, $x_l \in \mathbb{Z}_{q_2}$ we can compute

$$\mathcal{E}(F(x_1, ..., x_u)) = \prod_{1 \leq i \leq j \leq u} e(c_i, c_j)^{a_{ij}}$$

where all multiplications (and exponentiations) are in the group $\mathbb{G}_1$. Once again, since decryption is feasible only when the plaintext values are small, one must restrict the message space to be a small subset of $\mathbb{Z}_{q_2}$. (In our application, we will always have $x_i \in \{0, 1\}$.) Using this cryptosystem in our abstract construction, we can easily extend the types of queries that can be performed.

7.4 Construction of Private Filter Generator

More precisely, we can now perform queries of the following form, where M is a document and $K_1, K_2 \subset D$ are sets of keywords:

$$(M \cap K_1 \neq \varnothing) \wedge (M \cap K_2 \neq \varnothing)$$

We describe the Key-Gen, Filter-Gen, and Buffer-Decrypt algorithms below.

Key-Gen(k)

Execute the key generation algorithm of the BGN system to produce A_{public} =

[12] Small message size is clearly a fundamental limitation of the construction since efficiently computing arbitrary discrete logs would violate the security of the system.

$(n, \mathbb{G}, \mathbb{G}_1, e, g, h)$ where g is a generator, $n = q_1 q_2$, and h is a random element of order q_1. The private key, $A_{private}$ is the factorization of n. We make the additional assumption that $|D| < q_2$.

Filter-Gen$(D, Q_{K_1,K_2}, A_{public}, A_{private}, m, \gamma)$
This algorithm constructs and outputs a private filter F for the query Q_{K_1,K_2}, constructed as follows, where this query searches for all documents M such that $(M \cap K_1 \neq \varnothing) \wedge (M \cap K_2 \neq \varnothing)$.
F contains the following data:

- A buffer $B(\gamma)$ of size $2\gamma m$, indexed by blocks the size of an element of $\mathbb{G}_1$ times the document size, with every position initialized to the identity element of $\mathbb{G}_1$.
- Two arrays $\widehat{D_l} = \{\widehat{d_i^l}\}_{i=1}^{|D|}$ where each $\widehat{d_i^l} \in \mathbb{G}$, such that $\widehat{d_i^l}$ is an encryption of $1 \in \mathbb{Z}_n$ if $d_i \in K_l$ and an encryption of 0 otherwise.

F then proceeds with the following steps upon receiving an input document M:

1. Construct temporary collections $\widehat{M_l} = \{\widehat{d_i^l} \in \widehat{D_l} \mid d_i \in M\}$.
2. For $l = 1, 2$, compute
$$v_l = \prod_{\widehat{d_i^l} \in \widehat{M_l}} \widehat{d_i^l}$$
and
$$v = e(v_1, v_2) \in \mathbb{G}_1$$
3. Bitwise encrypt M using encryptions of 0 in $\mathbb{G}_1$ for 0's and using v to encrypt 1's to create a vector of $\mathbb{G}_1$ elements.
4. Choose γ random locations in B, take the encryption of step 3, and position-wise multiply these two vectors storing the result back in B at the same location.

Buffer-Decrypt$(B, A_{private})$
Decrypts B one block at a time using the decryption algorithm from the BGN system, interpreting non-identity elements of $\mathbb{Z}_{q_2}$ as 1's and 0 as 0, storing the non-zero, valid messages in the output B^*.[13]

7.5 Correctness of Filtering Software

As usual, we show the following two facts, which equate to correctness:

[13]See footnote 3.

1. Are non-matching documents stored with negligible probability? Yes. In fact, they are stored with probability 0 since clearly if a document M does not match, then it either did not match K_1 or it did not match K_2. Hence, all of the values in $\widehat{M_1}$ or $\widehat{M_2}$ will be encryptions of 0 and hence so will the value v. So, the buffer contents will be unaffected by the program executing on input M.

2. Are all matching documents saved with overwhelming probability? Clearly, if a document M satisfies $(M \cap K_1 \neq \varnothing) \wedge (M \cap K_2 \neq \varnothing)$, then v_1 and v_2 will be encryptions of non-zero elements of $\mathbb{Z}_{q_2}$ (as we ensured that $|D| < q_2$), and so will v, as $\mathbb{Z}_{q_2}$ is a domain. Then, we will be following the method in our "color-survival" game from section 2.2 for placing our documents in the buffer, and hence all documents will be saved with overwhelming probability in γ.

Theorem 7.1 *Assuming that the subgroup decision problem of [6] is hard, then the Private Filter Generator from the preceding construction is semantically secure according to Definition 2.7.*

Proof: Note that if an adversary can distinguish two queries, then the adversary has successfully distinguished one of the sets of keywords in the first query from the corresponding set in the second query. Now, it is a minor reduction to apply the abstract proof of Theorem 6.1, since this system is essentially the abstract construction built around the BGN cryptosystem. ■

8 Extensions

8.1 Detecting Buffer Overflow

We would like to take note of the fact that one can easily detect buffer overflow with overwhelming probability in the correctness parameter γ. In the work of Kamath, Motwani, Palem and Spirakis [18], a Chernoff-like bound is shown for the number of empty bins in the occupancy problem (where a number of balls are thrown uniformly and independently into n bins). I.e., as n increases, the probability that the number of empty bins after the process is a fixed proportion away from the mean is negligible in n. Hence, one could proceed as follows to detect overflow:

Let m be the maximum number of documents to save. Double the buffer size from $2\gamma m$ to $4\gamma m$. Let $n = 4\gamma m$. Let r be the number of matching documents written to the buffer. Overflow is defined as the condition $r > m$. Note that we can detect with probability 1 whether or not *any* documents have landed in a specific buffer location just by checking to see if it encrypts the identity or not. So, we can count the exact number of occupied bins. In the event that $m < r \leq 2m$, then by Lemma 2.8, we will in fact be able to recover at least one copy of all r documents, and hence be aware of an overflow. In the event that $r > 2m$, then we will throw more than $2\gamma m = n/2$ balls into our bins, and the expected number of occupied bins will be $\geq .4n$. Applying the results of [18], it will be negligibly likely that the number of occupied bins is less than $n/4$, which is always true if overflow has not occurred. So, if one modifies the filtering software to return overflow in the event that

1. more than m valid documents are recovered, or
2. the number of occupied bins is more than $n/4 = \gamma m$

then it will correctly detect overflow with overwhelming probability in the correctness parameter γ.

8.2 Keyword Search for Arbitrary Strings

We would like to point out a few straightforward extensions to this work which, while weakening our very strong notion of correctness, allows more functionality, which may be useful in practice.

One of the limitations of the strict model we presented is the fact that all keywords used in a search must come from a finite, public dictionary. In practice, we wish to extend this to be arbitrary words of finite length. This is easy to do, as long as we relax our definition of correctness to admit "false positives" into the buffer with small (but non-negligible) probability. In particular, in addition to the standard dictionary we can create a sufficiently large hash table, with a hash function that maps arbitrary strings into a smaller finite range, and use the output of the hash as additional keywords. The problem, of course, is that this ruins the strong notion of correctness that we have proved in our constructions. The probability of a false positive will then be proportional to the reciprocal of the size of the table, where as our definitions require this probability to be negligible. This may however, be useful in various practical applications. In [4, 5], the practicality and the consequences of this approach are examined in more detail.

8.3 Always Saving $\Omega(m)$ Documents in the Case of Overflow

We present here an elementary method which extends the buffer by an additional factor of κ, but has the property that even with buffer overflow that's exponential in this parameter, (say $2^\kappa m$ documents are written) the expected value of properly saved documents in the buffer will be at least $m/2$, where m is the designed buffer capacity. The method is as follows: replace the buffer B, by an array of identical buffers $\{B_i\}_{i=0}^{\kappa}$, each B_i having size $2\gamma m$. We will write to the buffers as follows: for each incoming document, write the document γ times (as in the original protocol) to buffer B_i with probability $\frac{1}{2^i}$. I.e., write to B_0 with probability 1, and for $i \in [1, \kappa]$ sample from a uniform distribution on $\{0, 1\}^i$ and write to B_i if the result is the 0 string. Now, as long as the total number of matching documents is less than $2^\kappa m$, then clearly there will be a buffer that has an expected number of matching documents between $m/2$ and m, and since each buffer is designed to store m documents with overwhelming probability, we will always have the expected number of recovered documents to be at least $m/2$ in this situation.

9 Open Questions and Further Work

In the follow-up work [4, 5], a different method for recovering documents from the buffer is presented. By recording extra information about the ordinal numbers of matching documents, and keeping track of the seed used for the random number generator to create the random locations at which documents are stored, they establish a system of linear equations that correspond exactly to the buffer contents. Now retrieving the documents amounts to solving the system of equations. The advantage of this, compared to our solution, is that buffer collisions are no longer necessarily lost, but in fact can often be recovered. Consequently, the buffer size does not need to be as large in order to maintain a high probability of recovering all documents. In fact, the buffer size becomes optimal. However, there is a drawback to this approach as well. To store the ordinal numbers of the documents that match, an encrypted Bloom filter is used, which is indeed a convenient device for storing set membership from a large universe without using much space. But although questions like "is $i \in S$?" can be answered efficiently using a Bloom filter, it is not easy to determine *all* elements of the universe that are in the set S without checking such statements for all possible values of i from the universe. So, the buffer recovery algorithm of [4, 5] now has a running-time proportional to the size of the data stream which may be undesirable, and does not fit the streaming model as proposed in this paper, where we insist that the buffer contents must be decrypted at the cost which is independent of the stream size. Again, such methods may be useful in various practical situations, and a more detailed analysis of such practical considerations can be found in [4, 5]. We also note that the idea of solving a system of linear equations to recover the buffer contents was also proposed in the [13], however, no formal arguments were given to support the method. It is an open question if one can reach buffer size as in [4, 5] without the computational decoding cost depending on the stream size.

Another limitation of our solution is of course the small variety of query types that can be performed. Using a multiplicative homomorphic encryption scheme can in a way perform an arbitrary AND query, however all such attempts have thus far failed to satisfy the correctness criteria. Extending the types of queries that can be executed, or proving that doing so is impossible under some general assumptions would be another interesting problem. In fact, some recent progress was made towards this end by Ostrovsky and Skeith [25]: they show that the general methods used here to create protocols for searching on streaming data (which are based essentially upon manipulating homomorphic encryption) cannot be extended to perform conjunctive queries beyond what has been accomplished in Sections 3, 7, 6, etc. More specifically, if one builds a protocol based on an abelian group homomorphic encryption (e.g. Sections 3 and 6) then no conjunctions (of more than one term) can be performed without increasing (super-linearly) the dictionary size. More generally, [25] shows that if the cryptosystem allows computation of polynomials of total degree t over any ring R, (as seen in Section 7 with $t = 2, R = \mathbb{Z}_p$) then the best one can hope for is a conjunction of t terms without increasing the dictionary size. So, the constructions we provide here meet a lower bound. It seems then, that to make progress in significantly extending the query semantics will likely require fundamentally different approaches to the problem (unless of course, major developments are made in the design of homomorphic encryption schemes, e.g. a scheme over a ring or non-abelian group).

Acknowledgements

We thank Martin Strauss for a useful suggestion on saving at least m documents in case of buffer overflow.

References

[1] B. Adida and D. Wikstrom Obfuscated Ciphertext Mixing In IACR Eprint archive number 394, 2005.

[2] L. Adleman, R. Rivest, A. Shamir. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems Communications of the ACM, 21(2):120-126,(February) 1978.

[3] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of software obfuscation. In *Crypto 2001*, pages 1–18, 2001. LNCS 2139.

[4] J. Bethencourt, D. Song, and B. Waters. New techniques for private stream searching. Technical Report CMU-CS-06-106, Carnegie Mellon University, March 2006.

[5] J. Bethencourt, D. Song, and B. Waters. New Constructions and Practical Applications for Private Stream Searching (Extended Abstract) Appeared at: http://www.cs.cmu.edu/ bethenco/searcheff.pdf July 2006.

[6] D. Boneh, E. Goh, K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. TCC 2005: 325-341

[7] D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano. Public Key Encryption with Keyword Search. EUROCRYPT 2004: 506-522

[8] Y. C. Chang. Single Database Private Information Retrieval with Logarithmic Communication. ACISP 2004

[9] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 1999.

[10] B. Chor, N. Gilboa, M. Naor. Private Information Retrieval by Keywords in Technical Report TR CS0917, Department of Computer Science, Technion, 1998.

[11] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of the 36th Annu. IEEE Symp. on Foundations of Computer Science*, pages 41–51, 1995. Journal version: *J. of the ACM*, 45:965–981, 1998.

[12] I. Damgård, M. Jurik. A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In Public Key Cryptography (PKC 2001)

[13] G. Danezis and C. Diaz Improving the Decoding Efficiency of Private Search In IACR Eprint archive number 024, 2006.

[14] G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology - EUROCRYPT 2000*, 2000.

[15] M. Fredman, J. Komlós. On the Size of Seperarting Systems and Families of Perfect Hash Functions. SIAM Journal on Algebraic and Discrete Methods. Vol. 5, No. 1, March 1984

[16] M. Freedman, Y. Ishai, B. Pinkas and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. To appear in 2nd Theory of Cryptography Conference (TCC '05) Cambridge, MA, Feb 2005.

[17] S. Goldwasser and S. Micali. Probabilistic encryption. In J. Comp. Sys. Sci, 28(1):270–299, 1984.

[18] A. Kamath, R. Motwani, K. Palem, P. Spirakis. Tail Bounds for Occupancy and the Satisfiability Threshold Conjecture. *Random Structures and Algorithms 7*, Pages: 59–80, 1995

[19] K. Kurosawa, W. Ogata. Oblivious Keyword Search. Journal of Complexity, Volume 20 , Issue 2-3 April/June 2004 Special issue on coding and cryptography Pages: 356–371

[20] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 364–373, 1997.

[21] E. Kushilevitz and R. Ostrovsky. One-way Trapdoor Permutations are Sufficient for Non-Trivial Single-Database Computationally-Private Information Retrieval. In *Proc. of EUROCRYPT '00*, 2000.

[22] H. Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. IACR ePrint Cryptology Archive 2004/063

[23] K. Mehlhorn. On the Program Size of Perfect and Universal Hash Functions. In Proc. 23rd annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 170-175.

[24] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. *Proc. 31st STOC*, pp. 245–254, 1999.

[25] R. Ostrovsky and W. Skeith. Algebraic Lower Bounds for Computing on Encrypted Data. *manuscript*, 2006.

[26] P. Paillier. Public Key Cryptosystems based on Composite Degree Residue Classes. Advances in Cryptology - EUROCRYPT '99, LNCS volume 1592, pp. 223-238. Springer Verlag, 1999.

[27] T. Sander, A. Young, M.Yung. Non-Interactive CryptoComputing For NC1 FOCS 1999: 554-567

[28] J.P. Stern. A New and Efficient All or Nothing Disclosure of Secrets Protocol Asiacrypt 1998 Proceedings, Springer Verlag.

10 Appendix

10.1 A Brief Review of the Paillier Cryptosystem

For the sake of completeness, we include a simple review of the Paillier Cryptosystem [26].

The Pailler system is based on an intractability assumption called the "Composite Residuosity Assumption", which as we will see before is something of a generalization of the hardness of distinguishing quadratic residues, and also can be reduced to the RSA problem [2]. This assumption (which we will abbreviate as CRA) is about distinguishing higher order residue classes. The Paillier system and its extensions (see [12]) are additively homomorphic, and have very low ciphertext to plaintext ratio.

10.1.1 Preliminaries

Let $n = pq$ be an RSA number, with $p < q$. We will make the additional minor assumption that $p \nmid q-1$, i.e., that $(n, \varphi(n)) = 1$. The plaintext for the Paillier system will be represented as elements of $\mathbb{Z}_n$ and the ciphertext will be elements of $\mathbb{Z}^*_{n^2}$. Note the following:

$$\mathbb{Z}^*_{n^2} \simeq \mathbb{Z}_n \times \mathbb{Z}^*_n$$

This can be proved using nothing more than elementary facts from number theory and group theory. (See Lemma 10.1 below and the corollary.) Given this structure of $\mathbb{Z}^*_{n^2}$, it is not hard to see that the factor of the direct product that is isomorphic to $\mathbb{Z}^*_n$ is in fact the *unique* subgroup of order $(p-1)(q-1)$. Let $H < \mathbb{Z}^*_{n^2}$ denote this subgroup of order $(p-1)(q-1)$. Now define G to be the quotient,

$$G = \mathbb{Z}^*_{n^2}/H$$

Then by our above remarks, we have the structure of G to be cyclic of order n: $G \simeq \mathbb{Z}_n$. We are now ready to state the Composite Residue Class Problem.

10.1.2 The Composite Residuosity Class Problem

Let $g \in \mathbb{Z}^*_{n^2}$ such that $\langle gH \rangle = G$ and let w be an arbitrary element in $\mathbb{Z}^*_{n^2}$. Then, since gH generates $G = \mathbb{Z}^*_{n^2}/H$, we have $w = g^i h$ for some $i \in \{0, 1, 2, ...n-1\}$ and $h \in H$. Given g and w, the Composite Residuosity Class Problem is simply to find i.

Note that there is also a decisional version of this problem: given w, g as above, and $x \in \{0, ..., n-1\}$, determine if $w = g^x h$ for some $h \in H$. This decision version of the problem is clearly equivalent to distinguishing n-th residues mod n^2 (which is the special case of $x = 0$) since H is exactly the subgroup of n-th residues. (Proof of this is given below- see Lemma 10.3.)

Note also that these problems have several random self-reducibility properties. Any instance of the problem can be converted to a uniformly random instance of the problem with respect to w (just by multiplying by $g^a b^n$, with $a \in \mathbb{Z}_n, b \in \mathbb{Z}^*_n$ and subtracting a from the answer). Also, the problem is self-reducible with respect to the generator g. In fact, one can show that any instance with generator g can be transformed into an instance with

generator g'. So, the choice of g has no effect on the hardness of this problem- if there are *any* easy instances, then *all* instances are easy.
Now that we have formalized the hardness assumptions, one can build a cryptosystem as follows:

10.1.3 The Cryptosystem

As mentioned before, there are several variants and extensions of this cryptosystem. Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the key generation, encryption, and decryption algorithms respectively. They are implemented as follows:

- $\mathcal{K}(s)$ This algorithm randomly selects an s-bit RSA number $n = pq$, with $p < q$ and the additional property that $p \nmid q-1$ (which is satisfied with overwhelming probability when p, q are randomly chosen). It outputs n as the public parameters, and saves the factorization as the private key.

- $\mathcal{E}(m)$ For a plaintext message $m < n$, choose $r \in \mathbb{Z}_n^*$ at random and set the ciphertext, c as follows:
$$c = (1+n)^m r^n \in \mathbb{Z}_{n^2}^*$$
Recovering m from c is precisely an instance of CRCP since r^n is a random element in the subgroup H, and the coset $(1+n)H$ will generate all of G. (See Lemma 10.5.)
Note 1: Due to the random self-reducibility of CRCP, $1+n$ is just as good of a choice of g as any other. Note 2: although it may seem more natural to choose $r \in \mathbb{Z}_{n^2}^*$, letting $r \in \mathbb{Z}_n^*$ is just as good. (See Lemma 10.4.)

- $\mathcal{D}(c)$ Let ciphertext $c = (1+n)^m r^n \ mod \ n^2$. To recover the message m, first look at this equation mod n rather than n^2:
$$c = (1+n)^m r^n \ mod \ n$$
becomes
$$c = r^n \ mod \ n$$
Now this equation is something familiar... finding r from c is an instance of the RSA problem (since we are given n which is relatively prime to $\varphi(n)$ and an exponentiation of $r \ mod \ n$). And since the factorization $n = pq$ is known to us, we can just use RSA decryption as a subroutine to recover r. Now that we have r, it is a simple process to obtain m.
To begin, compute $r^n \ mod \ n^2$ and divide c by this value:
$$\frac{c}{r^n} = (1+n)^m \ mod \ n^2$$
Now use the binomial theorem:
$$(1+n)^m = \sum_{i=0}^{m} \binom{m}{i} n^i$$

Reducing $mod\ n^2$ gives us

$$(1+n)^m = \sum_{i=0}^{1} \binom{m}{i} n^i = 1 + mn \ (mod\ n^2)$$

So finally, we have

$$m = \frac{\frac{c}{r^n} - 1}{n}$$

10.1.4 A few words about extensions to the system

Recently Mads Jurik and Ivan Damgård [12] made a very natural extension to the Paillier System that uses larger groups for its plaintext and ciphertext. This extension works for any $s \in \mathbb{Z}^+$. In the extended system, the plaintext is represented by an element in $\mathbb{Z}_{n^s}$, and the ciphertext is an element of $\mathbb{Z}^*_{n^{s+1}}$. There are two very appealing properties of this system: First, the ratio of plaintext length to ciphertext length approaches 1 as s tends to ∞. Second, just as in the original Paillier scheme, the public and private information can be simply n and its factorization, respectively. You need not share s ahead of time. In fact, the sender of a message can choose s to his/her liking based on the length of the message to be sent. Then, except with negligable probability, the receiver can deduce s from the length of the ciphertext. So, the public (and private) parameters remain extremely simple.

10.1.5 Lemmas and Proofs

Lemma 10.1 *Let $p \in \mathbb{Z}$ be a prime. Then $\mathbb{Z}^*_{p^2} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p$*

Proof: First note that $|\mathbb{Z}^*_{p^2}| = \varphi(p^2) = p(p-1)$ where p is prime and φ is the Euler phi-function. So, by Cauchy's Theorem, there is an element of order p inside $\mathbb{Z}^*_{p^2}$ (in fact, $p+1$ is such an element). So, there is a subgroup of order p in $\mathbb{Z}^*_{p^2}$. Call this subgroup H_p. Recall that $\mathbb{Z}^*_p$ is cyclic of order $p-1$, and let g be a generator of $\mathbb{Z}^*_p$. Notice that the order of g inside of $\mathbb{Z}^*_{p^2}$ is at least $p-1$ since equivalence mod p^2 implies equivalence mod p. (So, the first $p-1$ powers of g remain distinct mod p^2). But this severely limits the possibilities for the order of g inside of $\mathbb{Z}^*_{p^2}$. The only options that remain are $|g| = p-1$ or $|g| = p(p-1)$ since p is prime. In the first case, we have found a cyclic subgroup $\langle g \rangle$ of order $p-1$, and since $gcd(p, p-1) = 1$, we have

$$H_p \cap \langle g \rangle = \{1\}$$

and therefore,

$$\mathbb{Z}^*_{p^2} \simeq H_p \times \langle g \rangle \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p$$

which is exactly what we wanted. Or in the second case, $\langle g \rangle$ is all of $\mathbb{Z}^*_{p^2}$, hence

$$\mathbb{Z}^*_{p^2} \simeq \mathbb{Z}_{p(p-1)} \simeq \mathbb{Z}_p \times \mathbb{Z}_{p-1} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p$$

which is again, exactly what we wanted to prove. ■

Corollary 10.2 *Let $n = pq$, where $p, q \in \mathbb{Z}$ are primes. Then, $\mathbb{Z}^*_{n^2} \simeq \mathbb{Z}_n \times \mathbb{Z}^*_n$.*

Proof: First note that $\mathbb{Z}_{n^2} \simeq \mathbb{Z}_{p^2} \times \mathbb{Z}_{q^2}$ and hence $\mathbb{Z}^*_{n^2} \simeq \mathbb{Z}^*_{p^2} \times \mathbb{Z}^*_{q^2}$. Now applying Lemma 10.1, we have that

$$\mathbb{Z}^*_{n^2} \simeq \mathbb{Z}_p \times \mathbb{Z}^*_p \times \mathbb{Z}_q \times \mathbb{Z}^*_q$$
$$\simeq (\mathbb{Z}_p \times \mathbb{Z}_q) \times (\mathbb{Z}^*_p \times \mathbb{Z}^*_q) \simeq \mathbb{Z}_n \times \mathbb{Z}^*_n$$

which completes the proof. ■

Lemma 10.3 *The n-th residues mod n^2 are exactly the subgroup H.*

Proof: We would like to show that an element h of $\mathbb{Z}^*_{n^2}$ has an n-th root (i.e., can be written as $h = g^n \ mod \ n^2$ for some $g \in \mathbb{Z}^*_{n^2}$) if and only if $h \in H$. Define $\phi : \mathbb{Z}^*_{n^2} \to \mathbb{Z}^*_{n^2}$ by $x \mapsto x^n$. Certainly ϕ is a homomorphism: $\phi(ab) = (ab)^n = a^n b^n = \phi(a)\phi(b)$. Clearly, $im(\phi)$ is precisely the group of n-th residues, so hopefully we can show $im(\phi) = H$. What is $ker(\phi)$? Well, an element is in the kernel if and only if it has an order that divides n (i.e., elements of order $1, p, q, n$). Recall from the corollary that $\mathbb{Z}^*_{n^2} \simeq \mathbb{Z}_n \times \mathbb{Z}^*_n$. The $\mathbb{Z}_n$ component of this product consists of all of the elements of orders $1, p, q, n$ since we have that $(n, \varphi(n)) = 1$. So, $ker(\phi) \simeq \mathbb{Z}_n$ and hence $|im(\phi)| = |H|$, which is enough to show $im(\phi) = H$ as H is the unique subgroup of this order. ■

Lemma 10.4 *If $r \in \mathbb{Z}^*_n$ is chosen uniformly at random, then $r^n \ mod \ n^2$ is uniformly random in H.*

Proof: Let $\phi : \mathbb{Z}^*_n \to \mathbb{Z}^*_{n^2}$ be the n-th power map (composed first with the injection into $\mathbb{Z}^*_{n^2}$ if you like). Then of course, $im(\phi) \subset H$, given what we have already proved. But in fact $im(\phi) = H$ as ϕ is injective (and therefore surjective as $|\mathbb{Z}^*_n| = |H|$): recall that $(n, \varphi(n)) = 1$, so the n-th power map is 1 to 1 on $\mathbb{Z}^*_n$, and since equivalence mod n^2 implies equivalence mod n it must be that ϕ is also 1 to 1. So indeed, ϕ is a bijection of $\mathbb{Z}^*_n$ and H, so uniformly random in $\mathbb{Z}^*_n$ is uniformly random in H. ■

Lemma 10.5 *The coset $(1+n)H$ generates the factor group $G = \mathbb{Z}^*_{n^2}/H$.*

Proof: To see this, first look at the order of $1+n$ inside of $\mathbb{Z}^*_{n^2}$. Using the binomial theorem just as in our decryption specification, we have that $(1+n)^m = 1 + mn \ (mod \ n^2)$. So, clearly the order of $1+n$ is n, and hence $(1+n) \notin H$. Suppose for some $k \in \{2, ..., n\}$ that $(1+n)^k$ lies in H. Now, under any homomorphism, the order of the image of an element must divide the order of the element itself. Applying this to the homomorphism defined by raising elements to the k^{th} power, we would have that the order of $(1+n)^k$ must divide n. But $(1+n)^k \in H$ and $|H|$ is relatively prime to n, so this forces the order of $(1+n)^k$ to be 1, i.e., $k = n$. Hence $(1+n)H$ has order n in G as well. So, $\langle (1+n)H \rangle = G$. ■