Private Stream Search at Almost the Same Communication Cost as a Regular Search

Matthieu Finiasz[1] and Kannan Ramchandran[2]

[1] CryptoExperts
[2] UC Berkeley

Abstract. Private Stream Search allows users to perform keyword-based search queries on streaming data (or on a database) without revealing any information about the keywords they are searching. Using homomorphic encryption, Ostrovsky and Skeith proposed a solution to this problem in 2005. However, their solution requires the server to send an answer of size $O(mS \log m)$ bits when m documents of S bits match the query, while a regular (non-private) query only requires mS bits. Following this work, some improved schemes have been proposed with the aim of keeping the reply from the server *linear* in mS. In this work we propose two new *communication optimal* constructions: both allow communication linear in mS, but they also offer an expansion factor (compared to a non-private query) asymptotically equal to 1 when m and S increase. More precisely, our first scheme requires $m(S + O(\log t))$ bits (where t is the size of the database) and our second scheme $m(S + C)$ where C is a constant depending only on the chosen computational security level.

Keywords: privacy, keyword search, Reed-Solomon codes, LDPC codes.

1 Introduction

Internet search engines are able to gather a lot of information on users from the content of the search queries they make. Most users don't really care about this, but more and more users would prefer the servers not to learn *anything* from their query. This is the goal of Private Stream Search (PSS) algorithms: being able to perform a keyword-based search query on a server, without disclosing any information about the keywords in the query. However, even if the number of users having concerns about privacy is growing, the number of these users that are willing to trade efficiency for privacy is probably much smaller. Compared to standard non-private search, PSS should not have a much higher latency (response time) or bandwidth usage (response size) or be less reliable (miss some matching documents). This is the main focus of this article.

The first PSS algorithm was introduced in 2005 by Ostrovsky and Skeith [12,13] and makes clever use of homomorphic encryption to hide the content of the query while still allowing the server to perform some computations on it. This scheme requires the use of a public dictionary of possible keywords and is restricted to OR queries. These restrictions are not the main focus of our work, but as we discuss later, the use of a *fully homomorphic encryption* scheme could remove these restrictions from both the original Ostrovsky-Skeith construction and the new schemes we present here. Following Ostrovsky and Skeith's work, some improvements have been proposed independently by Bethencourt, Song and Waters [1,2] and by Danezis and Díaz [4,5]. These improvements have the same structure as the original scheme and are focused mainly on improving the size of the response from the server (one of the main issues in the original proposal) and the reliability of the scheme. However, they are suboptimal in some aspects.

The main contribution of this paper is the proposal of two new PSS algorithms combining the ideas of the Ostrovsky and Skeith construction with results from coding theory, thus allowing for state of the art results that improve significantly on current PSS schemes. Our first scheme uses Reed-Solomon codes [15] and allows for a zero-error guarantee, while offering optimal communication rates. It can however be computationally heavy at the server. Our second scheme is based on irregular LDPC codes [6,10] and is asymptotically optimal, thus interesting when a large number of documents (in practice, a few hundreds) match the query. We also propose an offline-online scheme, with a higher offline computational cost, but which allows the online step to be *as efficient* as a standard non-private search: the response suffers no latency and the communication overhead remains minimal.

This article is organized in 3 sections. Section 2 contains a description of the orignal Ostrovsky-Skeith PSS construction and of the Bethencourt *et al.* and Danezis and Díaz improvements. Then, in Section 3, we present our two new constructions and give some analysis of their performances. Finally, in Section 4, we detail some further improvements that apply to our schemes, but also to the previous PSS schemes.

2 Previous Constructions

2.1 Paillier's Encryption Scheme

All known private stream search constructions require the use of homomorphic encryption and the original Ostrovsky-Skeith construction relies on Paillier's cryptosystem [14]. The new schemes we present in Section 3 also rely on this scheme. Of course, any other homomorphic encryption scheme could be used instead, with only minor modifications to the PSS schemes.

Paillier's cryptosystem is a public key cryptosystem: in our PSS applications, a user and a server will communicate, and all encryptions will be done with respect to the user's key. We shall denote by $\mathcal{E} : \mathbb{Z}_N \mapsto \mathbb{Z}_{N^2}$ Paillier's encryption function and by $\mathcal{D} : \mathbb{Z}_{N^2} \mapsto \mathbb{Z}_N$ the associated decryption function. As the encryption is randomized, the same message can have different associated ciphertexts, decrypting to the same value. We thus introduce the notation $y \equiv y'$ which is equivalent to $\mathcal{D}(y) = \mathcal{D}(y')$: y and y' are encryptions of the same message. With these notations, the homomorphic property can be expressed as:

$$\mathcal{E}(x_1) \times \mathcal{E}(x_2) \equiv \mathcal{E}(x_1 + x_2) \quad \text{and} \quad \mathcal{E}(x)^c \equiv \mathcal{E}(c \times x) \text{ for any } c \in \mathbb{Z}_N.$$

Additionally, Paillier's cryptosystem is semantically secure, so the server cannot distinguish between $\mathcal{E}(0)$ and $\mathcal{E}(1)$. For simplicity, we will consider that N is 1024 bits long, but it should of course be chosen according to the required security level.

The Damgård-Jurik Extension. Semantic security requires a randomized encryption, which necessarily induces a message expansion: the ciphertext is larger than the associated message. Paillier's cryptosystem has a constant expansion factor of 2, for small messages of $\log N$ bits, but also for longer messages spanning several encrypted blocks.

To improve this, Damgård and Jurik [3] proposed a variant of Paillier's homomorphic encryption scheme which works very similarly but takes a message in $\mathbb{Z}_{N^s}$ (for any value of s) and outputs a message in $\mathbb{Z}_{N^{s+1}}$. For a message of $s \log N$ bits, the expansion factor is only $\frac{s+1}{s}$, which tends to 1 when the message size increases. However, the price to pay for this smaller expansion rate is a factor $O(s^2)$ on the cost of encryption/decryption.

Using the Damgård-Jurik encryption scheme with a modulus N of 1024 bits, the communication overhead is 1024 bits whatever the message size. More generally, this overhead is a constant C depending only on the required security level.

2.2 The Ostrovsky-Skeith Construction

A private stream search algorithm works in three steps: first the user builds a query and sends it to the server, then the server executes the query which outputs a result that it sends back to the user, finally the user extracts the queried documents from the result he received. Here is the description of these three algorithms for the original PSS scheme from Ostrovsky and Skeith [12].

Query Construction. Let $\Omega = \{w_1, w_2, \ldots, w_{|\Omega|}\}$ be the dictionary of possible keywords and $K \subseteq \Omega$ be the set of keywords the user wants to query. The query is $Q = \{q_1, q_2, \ldots, q_{|\Omega|}\}$, where $q_j = \mathcal{E}(\mathbf{1}_{w_j \in K})$ and $\mathbf{1}$ denotes the indicator function. The user thus sends an encrypted bit for each element in the dictionary: this bit is 1 if the keyword is part of the user's search, 0 otherwise. As each encryption is independently randomized and due to the semantic security of Paillier's cryptosystem, the server cannot tell which of these encrypted bits are 1 and 0.

As part of the query, the user also sends m, the expected number of matching documents, and γ, a reliability parameter (a larger γ gives a better probability of recovering all matching documents).

Query Execution. Upon receiving the query (Q, m, γ), the server first creates a buffer B of size $\ell = \gamma m$ and initializes each of its positions to the value $1 \equiv \mathcal{E}(0)$.

Let us assume the database contains t documents. Then, for each document $f_i \in \mathbb{Z}_N$ in the database, the server computes the set $W_i \subseteq \Omega$ of keywords in the dictionary that match document f_i. It then computes $F_i = \big(\prod_{w_j \in W_i} q_j\big)^{f_i} \equiv \prod_{w_j \in W_i} \mathcal{E}(\mathbf{1}_{w_j \in K})^{f_i}$. Thanks to the homomorphic property of $\mathcal{E}$, we also have: $F_i \equiv \mathcal{E}\big(f_i \sum_{w_j \in W_i} \mathbf{1}_{w_j \in K}\big)$. Denoting $c_i = \sum_{w_j \in W_i} \mathbf{1}_{w_j \in K}$ the number of keywords of K that match f_i, we then have $F_i = \mathcal{E}(c_i f_i)$. The server then select γ random positions $b_i = \{b_{i,1}, \ldots, b_{i,\gamma}\} \subset [1, m\gamma]$ of the buffer B and updates each of these γ positions by multiplying its current value by F_i.

After processing all the documents in the database, the j-th buffer position will be equal to $B_j = \prod_i F_i^{\mathbf{1}_{j \in b_i}} \equiv \mathcal{E}\big(\sum_i \mathbf{1}_{j \in b_i} c_i f_i\big)$, that is, the encryption of a linear combination of documents in the database. This linear combination is sparse if only few documents match the query K, meaning most of the c_i are equal to 0. The server then sends the buffer B back to the user.

In practice, everything happens as if the server had a random binary matrix H of size $\gamma m \times t$ with γ ones in each column and it was computing $B \equiv \mathcal{E}\big(H \times (c_i f_i)_{i \in [1,t]}\big)$.

Document Extraction. When receiving the encrypted buffer B, the user starts by decrypting each buffer position to get $\mathcal{D}(B_j) = \sum_i \mathbf{1}_{j \in b_i} c_i f_i$. He then scans the γm decrypted buffer positions for what we call *singletons*: buffer positions that contain only one file, that is, positions such that $\mathbf{1}_{j \in b_i} c_i = 0$ for all but one value of i. The user discards all buffer positions that are not singletons and extracts the value f_i of one document from each singleton.

Asymptotic Cost. The encrypted buffer that is sent back by the server to the user has size γm. In order for the user to recover the m matching documents with a high probability of success, γ must be of the order of $O(\log m)$. If documents are S bits long, using Paillier's cryptosystem, encrypted buffer positions should be $2S$ bits long and the answer is thus of order $2mSO(\log m)$. It can be reduced to $m(S + 1024)O(\log m)$ using the Damgård-Jurik extension (with a 1024-bits modulus N). The expansion factor is only logarithmic in m, but some improvements are needed to keep the buffer size *linear* in the number of matching documents.

2.3 The Danezis-Díaz Improvement

In [4,5], Danezis and Díaz propose to modify slightly Ostrovsky and Skeith's construction, allowing them to dramatically decrease the size of the buffer B. Their algorithm works exactly like the Ostrovsky-Skeith scheme, but with the following modifications:

- the query Q is the same, but the user chooses a target buffer size ℓ (typically $\ell = 2m$),
- before processing the query, the server embeds the index i of each document inside f_i,

- when processing the query, instead of adding the document to γ random buffer positions, the server now uses a public hash function/pseudo-random number generator h, seeded by the index i, to deterministically choose a set of d positions where the document f_i will be added. The integer d is a parameter of the system. In terms of matrices, H is such that its i-th column is $H_i = h(i)$, a binary vector of Hamming weight d.

The main change that Danezis and Díaz bring is a new document extraction algorithm. As in the Ostrovsky-Skeith construction, the user starts by looking for singletons and extracts the value of some documents from these singletons. However, now, each document also contains its index i, and from i, the user can generate $H_i = h(i)$ and know exactly where this document was added in the buffer. He can thus completely remove any document he recovers from the buffer, thereby uncovering new singletons. This simple iterative decoding algorithm allows for a much smaller buffer than in the original scheme, while maintaining a decoding cost linear in the buffer size.

This algorithm was only empirically tested by Danezis and Díaz, but their experimental results show that a buffer size of $\ell = 2m$ is sufficient to recover a high percentage of the m matching documents. However, they do not give any formal analysis.

2.4 The Bethencourt-Song-Waters Construction

In [1,2], Bethencourt, Song and Waters propose a completely different angle to improve on the Ostrovsky-Skeith scheme. Their approach is useful when documents are long (more than $\log N$ bits), which will be the case in many applications. In the previous schemes, the expansion ratio between the number of matching documents and the buffer size was independent of the document size. Meaning that even for large documents, the expansion would be $\log m$ for the Ostrovsky-Skeith scheme and 2 for the Danezis-Díaz variant.

Bethencourt *et al.* propose to use a similar scheme, but with 3 different buffers. The first buffer will contain the value of the documents (and will thus scale with the size of the documents), and the two other buffers are *independent* of the document size and are used to send the values of the c_i and the indexes i of the matching documents.

With this technique, Bethencourt *et al.* are able to achieve an asymptotic expansion rate of 1 (using the Damgård-Jurik cryptosystem), but their technique suffers a few drawbacks:

- recovering the indexes i of matching documents relies on Bloom filters and requires the reply to be of size $O(m \log \frac{t}{m})$ to have a good probability of success. This dependence in the size t of the database is not desirable and, as we will see, not necessary.
- recovering the document values requires to solve a linear system of size $m \times m$, which can be quite expensive compared to the rest of the document extraction process. Other algorithms have a linear cost in the buffer size.

3 Two New PSS Constructions

Building on the original Ostrovsky-Skeith scheme, we propose two new *communication optimal* constructions. The key idea is to consider that the job of the server is simply to compute a sparse encrypted vector $\mathcal{E}(c_i f_i)$ and then compress it *in the encrypted domain*. Thanks to the homomorphic property of Paillier's cryptosystem it is possible to compute the syndrome of this vector with respect to the parity check matrix H of an error correcting code. Using two optimal code constructions (Reed-Solomon codes [15] and irregular LDPC codes [6,10]) we obtain our two new PSS schemes. Of course, the idea of using a linear function to compress a sparse vector is not new, but homomorphic encryption allows to do this in the encrypted domain too. This is also what previous PSS schemes were doing, without explicitly stating it, and using sub-optimal linear compression.

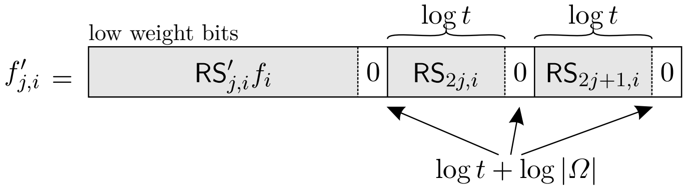


Fig. 1. Embedding of two Reed-Solomon codes and of f_i inside an element of $\mathbb{Z}_N$. $|\Omega|$ and t are the dictionary and the database sizes: the zero-paddings allow to avoid overflows when computing linear combinations of $f'_{j,i}$.

3.1 A Zero-Error Construction Using Reed-Solomon Codes

Apart from the communication overhead, one drawback of the existing PSS constructions is that they have a non-zero probability of failure. This is true even for very large expansion rates. In particular, when the number m of matching documents is small, the failure probability of all previous schemes becomes higher for a given expansion rate. We thus propose a deterministic algorithm that will guarantee that the document extraction will never fail if the number of matching documents is known. This algorithm uses Reed-Solomon codes and exploits their MDS (maximum distance separable) property in the following way:

- Reed-Solomon codes can correct up to m errors using $2m$ syndromes,
- they can also correct m erasures (errors at a known position) using only m syndromes.

Description of the Construction. A direct application of this would consist in replacing the matrix H in the PSS algorithm by the parity check matrix of a Reed-Solomon code over $\mathbb{Z}_N$. Then, recovering the value of any m documents would be equivalent to correcting m errors with the code and would only require $\ell = 2m$ buffer positions. However, it is possible to do better than this. Indeed, this straightforward application allows documents of $\log N$ bits, but also allows a database of up to N elements. In practice the database is much smaller than N (which for security reasons will be at least 2^{1024}), and using a Reed-Solomon code on $\mathbb{Z}_N$ is a waste. The server can encode the values of the c_i (and their positions) as errors in a smaller Reed-Solomon code, and then encode the documents as erasures, which can be efficiently recovered once the c_i are known. Here is how the algorithm works.

Query Construction. This step is the same as for the other algorithms (described in Section 2.2). Along with the query Q, the user sends an expected number of results m.

Query Execution. As in the previous constructions, for each document f_i, the server computes the encryption $\mathcal{E}(c_i)$ of the number of queried keywords matching f_i. Then, instead of simply computing $\mathcal{E}(c_i)^{f_i}$, the server will embed several values in an integer $f'_{j,i} \in \mathbb{Z}_N$ as shown in Fig. 1. This requires two different Reed-Solomon codes RS and RS'. The code RS will be used to recover the c_i and is defined on $\mathbb{Z}_{p_t}$ where p_t is the smallest prime greater than the database size t (it should also be greater than the dictionary size $|\Omega|$) and the coefficients of its parity check matrix are thus defined as $\mathsf{RS}_{j,i} = i^j \mod p_t$. Similarly, RS' will be used to recover the values of f_i and is defined over $\mathbb{Z}_{p_f}$ where p_f is the largest prime that can fit in the remaining bits of one plaintext (Paillier or Damgård-Jurik). We have $\mathsf{RS}'_{j,i} = i^j \mod p_f$. Thus, for each of the m positions of buffer B, the server multiplies B_j by $\mathcal{E}(c_i)^{f'_{j,i}}$. Once the whole database has been processed, $B_j \equiv \mathcal{E}\big(\sum_{i=1}^{t} c_i f'_{j,i}\big)$.

Document Extraction. As usual, the user starts by decrypting the buffer B. He then splits each plaintext he obtains in 3 parts: the first part corresponding to $\sum_i c_i \mathsf{RS}'_{j,i} f_i$, the second to $\sum_i c_i \mathsf{RS}_{2j,i}$ and the third to $\sum_i c_i \mathsf{RS}_{2j+1,i}$. These linear combinations have been computed in the encrypted domain by the server, with no reduction modulo p_t or p_f: this is why some zero-padding is required (see Fig. 1) to avoid overflows. The user thus starts by reducing the last two elements modulo p_t for each $j \in [1, m]$, and gets $2m$ syndrome positions in RS of the sparse vector (c_i). This is enough to recover the values and positions of the m non-zero c_i elements using the Reed-Solomon error correcting algorithm.

Then, the user reduces the first part of each plaintext modulo p_f to obtain m syndrome positions in RS' of the sparse vector $(c_i f_i)$. As the non-zero c_i elements are known, the positions of the non-zero $c_i f_i$ are also known, and the user has to solve an erasure problem. The m syndrome positions are enough to recover the values of $c_i f_i$, and thus also of f_i.

Computational Cost. Compared to the Ostrovsky-Skeith construction, the use of Reed-Solomon codes has an heavy impact on the server side computations. As all the lines of a Reed-Solomon parity check matrix are different, the server has to compute a modular exponentiation $\mathcal{E}(c_i)^{f'_{j,i}}$ for each coefficient in the matrix, that is mt exponentiations instead of t in the other algorithms.

However, on the user side, the computational cost remains similar and will be dominated by the buffer decryption step. Reed-Solomon decoding costs $O(m^2)$ multiplications in $\mathbb{Z}_{p_t}$ and $\mathbb{Z}_{p_f}$, which can be upper bounded by $O(m^2(\log N)^2)$. Decryption costs $O(m(\log N)^3)$, which will dominate as long as m is smaller than a few thousands. The document extraction process is thus no longer linear in m, but the quadratic component is negligible in practice.

Communication Cost. With this scheme, a buffer of size m is enough to recover m matching documents, which is optimal. However, part of each buffer position is reserved for the recovery of the c_i and for zero-padding. For each document, an overhead of $5 \log t + 3 \log |\Omega|$ bits has to be transmitted. The $3 \log t + 3 \log |\Omega|$ padding bits[3] are wasted bits that are due to the structure of the Paillier encryption scheme: using a different homomorphic encryption scheme could improve this. The remaining $2 \log t$ bits are however necessary to get a deterministic zero-error algorithm: the user has to solve an error correction problem, meaning he will have to learn both the value and *the position* of the errors, leading to an overhead of $O(\log t)$ bits per document. Overall, for m matching documents of S bits, this scheme requires $2m(S + O(\log t))$ bits using the original Paillier cryptosystem or $m(S + 1024 + O(\log t))$ using the Damgård-Jurik extension. Asymptotically, when m and S tend to infinity, this corresponds to an expansion ratio (compared to a non-private search) of 2 using Paillier and 1 using Damgård-Jurik.

Note that these ratios are valid only if the user knows in advance the number m of matches to expect. This can be the case in some scenarios, but not in a typical keyword search. In this case, the user will query for m positions and will receive a syndrome of size m: if less than m documents match the query he will be able to extract all of them with probability 1, but if more than m documents match the query he will not be able to decode and will not get *any* document. This gives another way of looking at the zero-error property: the user knows when he misses something, whereas in the Ostrovsky-Skeith scheme, the user will usually be able to extract at least a few documents, but has no information wether he got all the documents or not. With our Reed-Solomon scheme, it is possible to imagine an interactive protocol[4] where the user can query for additional syndrome

[3] In practice, this can be reduced to $3 \log m + 3 \log |K|$ bits, but having a dependency on m is not really convenient. It can however be easily reduced to $3 \log t + 3c$ for any c if queries are limited 2^c keywords at a time.

[4] Special care has to be taken when designing such an interactive protocol, so as not lose too much privacy by disclosing the actual number of matches to the server.

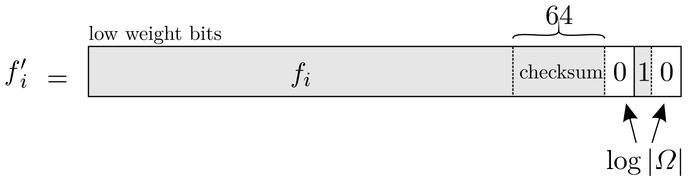


Fig. 2. Embedding of f_i, a padding and a checksum of f_i inside an element of $\mathbb{Z}_N$. $|\Omega|$ is the dictionary size (a bound on c_i) and the paddings avoid overflows when multiplying f_i' by c_i.

positions when the decoding fails, thus allowing him to choose a small m at first and still be sure to get all the documents in the end.

3.2 An Asymptotically Optimal Construction Using Irregular LDPC Codes

Description of the Construction. In order to improve the asymptotic communication cost and remove any dependency on the database size t, it is necessary to use a randomized scheme (thus with a non-zero probability of failure): in that case, it is well known that (irregular) LDPC codes can offer much better performance than Reed-Solomon codes. However, the error correction problem also has to be transformed into an erasure correction problem. This is possible by combining the following ideas (which is similar to what Danezis and Díaz proposed):

- instead of using a fix LDPC matrix, generate it from the documents f_i themselves,
- use a decoding algorithm similar to the erasure correction algorithm proposed by Luby and Mitzenmacher for verification codes [8].

Query Construction. This step is the same as for the Ostrovsky-Skeith construction. Instead of m and γ, the user sends the desired buffer length ℓ to the server.

Query Execution. The server first initializes a buffer B of size ℓ to $\mathcal{E}(0)$ in every position. Then, for each document f_i it proceeds as follows:

- compute f_i' from f_i as shown in Fig. 2, adding a padding (with a single 1 between some zeros) and a small non-linear checksum of f_i itself (a cryptographic hash of 64 bits can be used),
- compute $\mathcal{E}(c_i f_i')$ exactly as in the original scheme,
- using a pseudo-random number generator seeded by f_i, generate[5] an integer d following a given distribution (the best choice for this distribution is discussed at the end of this section) and generate a uniformly random binary column vector H_i of length ℓ and Hamming weight d,
- for every non-zero position in H_i, multiply the corresponding position in buffer B by $\mathcal{E}(c_i f_i')$.

In the end, $B = \mathcal{E}(H \times (c_i f_i'))$ contains the encrypted syndrome of the sparse $(c_i f_i')$ vector with respect to the parity check matrix H of an irregular LDPC code.

Document Extraction. When receiving buffer B, the user starts by decrypting it and looks for singletons. The detection of singletons is made easy by the structure of f_i' (see Fig. 2). The isolated 1 allows to read the value of c_i directly and divide $c_i f_i'$ by it: if it is indeed a singleton, the checksum will be valid and the value of f_i can be recovered, otherwise the checksum will not be valid (with high probability).

With every singleton the user gets the value of one document f_i and can now regenerate (using the same PRNG as the server) the corresponding column H_i. Knowing H_i (and c_i) it is possible to

[5] Note that Danezis and Díaz use the document index i to generate the column H_i, which requires to number documents. Using the document itself as the seed makes application to streaming data more natural.

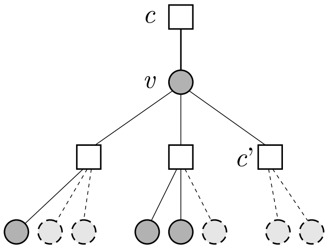


Fig. 3. Local view of the bipartite graph as a tree. The dashed lines correspond to nodes/edges removes at the end of step j. The edge between v and c will be removed at step $j+1$ as one son c' of v is a singleton (it has no sons remaining at the end of step j).

remove document f_i from the other syndrome positions where it has been added, thus uncovering new singletons, which in turn can reveal new documents f_i. This gives an iterative algorithm which we analyse asymptotically here.

Choosing an Optimal Column Weight Distribution. In order to analyze the decoding algorithm, the matrix H can be transformed into a bipartite graph. On the left of the graph are m information nodes (the non-zero $(c_i f'_i)$ elements) and on the right are ℓ parity nodes (the decrypted syndromes). These nodes are connected by edges: each 1 in H is an edge in the graph, linking a information node and a parity node. For each edge in the graph, its *left degree* is the number of edges connected to its information node and its *right degree* the number of edges connected to its parity node. Then the decoding algorithm consists in repeating the following steps:

- select all edges with right degree 1 (edges connected to singletons),
- remove these edges from the graph as well as the associated left and right nodes,
- remove all other edges that were connected to the left nodes (no other edges were connected to the right nodes).

Decoding is successful if, at the end, all the edges have been removed from the graph.

Studying the probability of success of this algorithm for given parameters m and ℓ is difficult, however, as proven in [9], if the left and right degree distribution of edges remains constant and m and ℓ tend to infinity, the asymptotic proportion of edges removed at each step can be computed.

Let $\lambda(x) = \sum_i \lambda_i x^{i-1}$ and $\rho(x) = \sum_i \rho_i x^{i-1}$, where λ_i (resp. ρ_i) denote the probability that an edge of the graph has left (resp. right) degree i. Also, let b_j denote the proportion of edges of the graph that are still present after step j of the algorithm. Then $b_0 = 1$ (all the edges are present before the algorithm starts) and:

$$b_{j+1} = \lambda(1 - \rho(1 - b_j)). \tag{1}$$

We will not prove this formula here (please refer to [9,11] for the complete proof), but the intuition is the following. During the decoding process, the neighborhood around the edge (v, c) between a message node v and a parity node c, can be seen as a tree (see Fig. 3). The decoding process then consists in letting information flow from the leaves of the tree to its root: the (v, c) edge will be removed from the graph at step $j+1$ if the value of v can be determined at step $j+1$ and it can thus send its value to c. As can be seen on Fig. 3, the value of the message v can be determined if it has a son c' that is a singleton at the end of step j of the algorithm, meaning one of the sons of v is a leaf. The probability this happens can easily be computed if the distributions λ and ρ are known. The parity node c' is connected to k edges with probability ρ_k and it is a

Table 1. Minimal expansion rates asymptotically allowing recovery of all the documents, for specific values of d, in the algorithm of Danezis and Díaz.

d	2	3	4	5	6	7	8	9
minimal $\frac{\ell}{m}$	2	1.2218	1.2949	1.4249	1.5697	1.7189	1.8692	2.0192

singleton at the end of step j if its $k-1$ sons in the Fig. 3 tree have been removed: this happens with probability $P_{singleton} = \sum_k \rho_k (1-b_j)^{k-1} = \rho(1-b_j)$.

Similarly, node v has k' neighbors (and thus $k'-1$ sons) with probability $\lambda_{k'}$, so one of them will be a singleton with probability $1 - \sum_{k'} \lambda'_k (1-P_{singleton})^{k'-1} = 1 - \lambda(1-P_{singleton})$. The edge (v, c) is thus removed at step $j+1$ with probability $1-\lambda(1-\rho(1-b_j))$, which leads to formula (1).

Asymptotically, the decoding algorithm is successful if $b_j \xrightarrow{j\to\infty} 0$, which will be the case if:

$$\forall x \in [0,1], \quad \lambda(1-\rho(1-x)) \leq x.$$

Of course, for finite values of m and ℓ, the algorithm can fail even if this condition is verified. The sequence of b_j represents the expected evolution of the number of edges in the graph, but the algorithm will deviate from the average from time to time.

Application to Danezis and Díaz's Algorithm. In [4], Danezis and Díaz propose to use a constant weight d for the columns of H. This means that $\lambda(x) = x^{d-1}$. The distribution ρ is a little more complicated as it is induced by λ. A row of H will have a weight following a binomial distribution: it is the sum of m random coins equal to 1 with probability $\frac{d}{\ell}$. We thus have $\rho_i = \frac{i\ell}{md}\binom{m}{i}\left(\frac{d}{\ell}\right)^i\left(1-\frac{d}{\ell}\right)^{m-i}$ and so, when m is large, $\rho(x) = \exp\big(-\frac{md}{\ell}(1-x)\big)$.

The best rate $\frac{\ell}{m}$ one can achieve for a given d is given in Table 1 and is obtained from the equation:

$$\forall x \in [0,1], \quad \Big(1-\exp(-\tfrac{md}{\ell}x)\Big)^{d-1} \leq x.$$

The best asymptotic choice is $d = 3$, which does not mean that for a given m and ℓ the best probability of success is always achieved for $d = 3$. This however proves that Danezis and Díaz's algorithm with constant column weight d can indeed achieve communications linear in m. However, it can never have a good probability of success for expansion rates smaller than 1.22.

The Harmonic Distribution. Using a constant column weight makes the description of the algorithm easy and gives very good results for some parameters (see Fig. 5), but it is not optimal.

The decoding problem we are facing is very close to the problem of decoding LT-codes [7] and, for LT-codes, it was proven that the optimal distribution choice is the so-called *Robust Soliton* distribution. However, in our context, using this distribution would correspond to fixing $\rho(x)$, that is, choosing a row weight distribution. This is not possible as each instance of our decoding problem corresponds to a set of m random columns of H, meaning we only have full control on the column weight distribution $\lambda(x)$. The row distribution will always be given by $\rho(x) = \exp\big(-\frac{m\tilde{d}}{\ell}(1-x)\big)$, where $\tilde{d} = 1/\sum_i \frac{\lambda_i}{i}$ is the average column weight of H. Thus, the constraint on $\lambda(x)$ is:

$$\forall x \in [0,1], \quad \lambda\big(1-\exp(-\tfrac{m\tilde{d}}{\ell}x)\big) \leq x. \tag{2}$$

With a change of variable $y = 1-\exp(-\frac{m\tilde{d}}{\ell}x)$, inequality (2) becomes:

$$\forall y \in \big[0, 1-\exp(-\tfrac{m\tilde{d}}{\ell})\big], \quad \lambda(y) \leq -\frac{\ell}{m\tilde{d}}\ln(1-y).$$

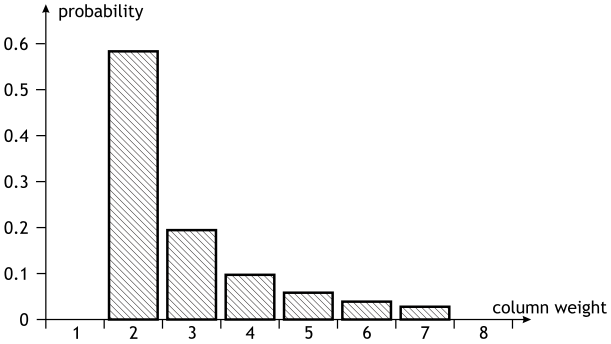


Fig. 4. Column weight distribution for the harmonic distribution of order 7.

The optimal choice is thus the harmonic distribution, consisting in a normalized Taylor series expansion of $-\ln(1-x)$ truncated at order D. It is given by:

$$\lambda_D(x) = \frac{1}{H(D)} \sum_{i=2}^{D} \frac{1}{i-1} x^{i-1} \quad \text{with} \quad H(D) = \sum_{i=2}^{D} \frac{1}{i-1}.$$

These $\lambda_D(x)$ distributions will satisfy inequality (2) if the expansion factor $\frac{\ell}{m}$ is greater than $1+\frac{1}{D}$. Any expansion ratio $\frac{\ell}{m} = 1+\epsilon$ can thus be chosen by the user, and using the harmonic distribution of order $\frac{1}{\epsilon}$ will asymptotically allow to recover most documents.

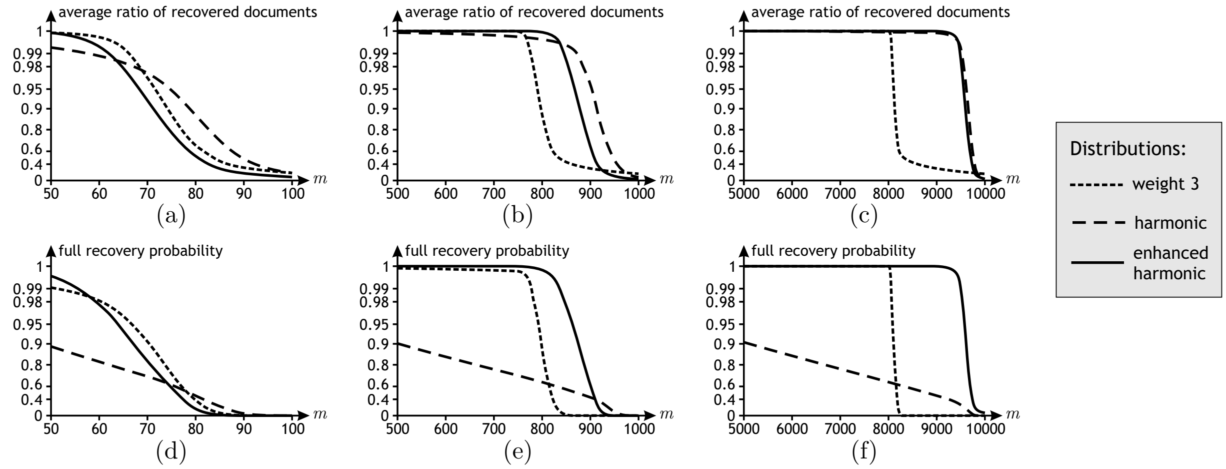
The Enhanced Harmonic Distribution. One problem with the harmonic distribution is the presence of columns of weight 2 (see Fig. 4). Any distribution containing columns of weight 2 will have a non-zero asymptotic probability of having identical columns[6]. Indeed, the collision probability P_{col} that 2 of the m matching columns of H are identical is $P_{\text{col}} \simeq \sum_i \binom{\frac{\tilde{d}\lambda_i}{i} m}{2} / \binom{\ell}{i}$. If the ratio $\frac{\ell}{m}$ is constant and m tends to infinity, all the terms in this sum tend to 0, except the term $i=2$. Asymptotically, $P_{\text{col}} \overset{m\to\infty}{\longrightarrow} \left(\frac{\tilde{d}\lambda_2 m}{2\ell}\right)^2$, which is not negligible for the harmonic distribution.

This means that, using the harmonic distribution, decoding will often end with a few ($O(1)$ asymptotically) unrecovered documents. In order to deal with this issue, one solution is to combine this distribution with a constant weight distribution to obtain what we call the *enhanced harmonic distribution.* Each column of H is the concatenation of a column of length $\ell - \ell_3$ following the harmonic distribution and a column of length ℓ_3 of constant weight 3.

In practice, the proportion of collisions tends to 0 when m grows, so the length ℓ_3 can be chosen such that $\frac{\ell_3}{m}$ tends to 0 and the probability of full recovery still tends to 1. For example, $l_3 = O(\sqrt{\ell})$ would be a reasonable choice. Asymptotically, these ℓ_3 additional rows in H do not increase the expansion factor. Using this enhanced harmonic distribution with an order D harmonic distribution, the probability of recovering all m documents tends to 1 when $\frac{\ell}{m} > 1 + \frac{1}{D}$ and m tends to infinity.

Communication Cost. We compared the different distributions through simulations and the results we obtained are shown in Fig. 5. One can clearly see the limitation of the weight 3 distribution: for $m > .8\ell \simeq \frac{\ell}{1.22}$ the probability of full recovery drops. However, the enhanced harmonic

[6] If two documents f_i generate the exact same column H_i, our decoding algorithm will be unable to recover any of them as no singleton can ever be obtained.



(a) (b) (c)

(d) (e) (f)

Fig. 5. Simulation results for different LDPC weight distributions. The curves represent the average ratio of recovered documents and the probability of recovering all m documents as a function of the number of matching documents m for different buffer sizes ℓ. In (a) and (d) $\ell = 100$ and $\ell_3 = 10$, in (b) and (e) $\ell = 1000$ and $\ell_3 = 35$, and in (c) and (f) $\ell = 10000$ and $\ell_3 = 100$.

distribution holds to our expectations, it behaves as expected with a probability of recovering all m matching documents close to 1 even for expansion rates smaller than 5%.

With this algorithm and the enhanced harmonic distribution, the asymptotic communication cost of private stream search with m matching documents of S bits is thus $2m(S + 64 + O(\log|\Omega|))$ using Paillier's cryptosystem, or $m(S + 1088 + O(\log|\Omega|))$ using the Damgård-Jurik extension, which corresponds to an expansion rate of 1 compared to a non-private search.

4 Further Improvements

An Offline-Online Construction. Using any of our two new schemes, it is possible to reduce the size of the reply from the server almost to the size of a non-private search result. However, the size of the query the user sends remains large. The size of a private query is linear in $|\Omega|$ whereas it is logarithmic for a non-private query. To improve this, we propose an offline-online scheme, where the linear query is sent offline and a logarithmic query is sent online.

A Single Keyword Scheme. We first focus on queries containing a single keyword. In this case, any query can be obtained as a cyclic shift of any other query. The scheme works as follows:

- offline, the user generates a query $Q_j = \{q_1, \ldots, q_{|\Omega|}\}$ where $q_j = \mathcal{E}(1)$ and $q_i = \mathcal{E}(0)$ otherwise (with j picked uniformly at random), and sends it to the server,
- offline, the server computes all possible cyclic shifts of Q_j by $i \in [0, [\Omega| - 1]$ positions and executes the corresponding queries. It stores each result in a separate buffer B_i.
- online, the user wants to query the server for the j'-th keywords and sends $j' - j \mod |\Omega|$ to the server,
- online, the server sends $B_{j'-j}$ to the user and discards the other B_i,
- the user decrypts/decodes buffer $B_{j'-j}$ normally.

With this scheme, the online work on the server side is a simple table lookup and the amount of online communication is very close to the non-private case: the query is only $\log|\Omega|$ bits long, and with the PSS schemes we have presented $B_{j'-j}$ can also be small.

The offline amount of communication is still the same as for the standard scheme, but the amount of computation on the server side is multiplied by $|\Omega|$. However, as this is offline work, it can easily be outsourced to distant server farm and does not have to be run on the "online" low-latency servers. Of course, if the amount of offline work is too high, it is also possible to treat the shifted query online as in the standard scheme: the amount of work the server has to do will then be the same as in the normal scheme, but most of the communication will be done offline.

Dealing with Multiple Keywords. To maintain privacy, the offline query the user sends has to be completely random and, at the same time, it should be possible to modify it into any other query the user might later want to ask. For a single keyword, cyclic shifts work well whatever the dictionary size. However, for several keywords (say k), the user should be able to transform any random query $Q_{j_1,\dots,j_k}$ into any chosen query $Q_{j'_1,\dots,j'_k}$, by simply giving the index of a permutation.

When $k = 2$, a solution is to transform each index j into $Aj + B \mod |\Omega|$, where $A \in [1, |\Omega| - 1]$ and $B \in [0, |\Omega| - 1]$ are the "permutation index" that the user will send to the server in the online phase. If $|\Omega|$ is prime, then any pair j_1, j_2 can be transformed into any pairs j'_1, j'_2 by choosing $A = \frac{j_2 - j_1}{j'_2 - j'_1} \mod |\Omega|$ and $B = Aj_1 - j'_1 \mod |\Omega|$.

Building such families of permutations for larger values of k is not always simple, and the number of permutations in the family will always have to be at least $\binom{|\Omega|}{k}$. This means that the offline work on the server side will be $O(|\Omega|^k)$ times more than in the standard online scheme. Values of k larger than 1 or 2 are therefore not very realistic, and for these values the solutions we presented work fine.

Using Fully Homomorphic Encryption. The PSS schemes we presented do not need fully homomorphic encryption (FHE) to work: computing a syndrome only requires addition and multiplication by a scalar. However, having an efficient FHE scheme would allow AND queries, whereas the current schemes are limited to OR queries. Also, the query size could be reduced to $O(\log |\Omega|)$ encrypted bits without having to use an offline-online scheme[7]. For instance, when querying for keyword w_i, the user could decompose the index i in $\log |\Omega|$ bits and send two ciphertexts $\mathcal{E}(0)$ and $\mathcal{E}(1)$ (or $\mathcal{E}(1)$ and $\mathcal{E}(0)$) for each of these bits. The server can then regenerate the whole query Q by homomorphically multiplying the encrypted bits corresponding to each index. Continuing on this idea, it would even be possible to get rid of the dictionary by simply querying a bit string: a query of $2n$ encrypted bits would be enough to search any pattern of n bits.

One interesting aspect of this application of FHE, is that the number of homomorphic multiplications that have to be done is independent of the database size: multiplications are required to reconstruct the query, but not for the stream search itself. Even if homomorphic multiplication is very expensive, or if the chosen scheme only allows for a few multiplications, it is still possible use it for very large databases: only the dictionary size is limited.

Also, one should note that the homomorphic encryption scheme used in PSS does not have to be a public key scheme: only the user encrypts and decrypts elements (the server simply has to be able to initialize the buffer to $\mathcal{E}(0)$, but the user could send him this initial value). A symmetric homomorphic encryption scheme with a limit on the number of possible multiplications could thus also be of interest if it allows a small message expansion rate.

Application to Set Reconciliation. An interesting aspect of the LDPC scheme we presented is that, even though it behaves like a randomized scheme, it is fully deterministic. What this means

[7] Not that a query of $O(\log |\Omega|)$ bits is probably not achievable in practice as current FHE schemes all require some important message expansion, but a poly-logarithmic query is possible.

is that if the user already knows a subset of the documents f_i that are going to match the query, he can compute a buffer B for the documents he knows in the exact same way as the server and "remove" this buffer from the reply he gets from the server.

This is the idea of set reconciliation: two users A and B know two sets S_A and S_B and want to learn the elements the other user knows with the minimal amount of communication. Classical results from set reconciliation show that the amount of communication required is only $|S_A|+|S_B|-2|S_A \cap S_B|$.

The same optimal result can be obtained with our construction: if m documents match the query, but the user already knows m' of these documents, a buffer of size $m-m'$ is (asymptotically) enough. This way, if a user repeats the same search every day (for example, to monitor changes to a database), the amount of communication required can be reduced even further by keeping a cache of the previous search results.

5 Conclusion

We presented two new private stream search constructions allowing minimal communication from the server to the user. Using Paillier's cryptosystem, compared to a standard non-private search, the response from the server in our schemes only suffers a factor 2 expansion. Using the Damgård-Jurik extension, this expansion factor can be made as close to one as required when the document size tends to infinity. Also, our Reed-Solomon based construction allows zero-error rate, meaning the user is guaranteed to get the documents he expects (or, if he chose m too small, he will know he missed some documents). This comes at a cost on the server side, but can be a very important feature for some applications.

The improvements we presented here do not solve all the problems of private stream search: the computational cost on the server side is still much more than for a regular search, and the size of the query the user has to send is also a problem. Still, our LDPC scheme offers better performances than all the previous private stream search constructions, without any additional computations.

References

1. John Bethencourt, Dawn X. Song, and Brent Waters. New constructions and practical applications for private stream searching (extended abstract). In *2006 IEEE Symposium on Security and Privacy*, pages 132–139. IEEE Computer Society, 2006.
2. John Bethencourt, Dawn X. Song, and Brent Waters. New techniques for private stream searching. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.
3. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, 2001.
4. George Danezis and Claudia Díaz. Improving the decoding efficiency of private search. In *Anonymous Communication and its Applications*, volume 05411 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2006.
5. George Danezis and Claudia Díaz. Space-efficient private search with applications to rateless codes. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography 2007*, volume 4886 of *LNCS*, pages 148–162. Springer, 2007.
6. Robert G. Gallager. Low-density parity-check codes. Cambridge, Massachussets: M.I.T. Press, 1963.
7. Michael G. Luby. LT codes. In *FOCS*, pages 271–280. IEEE, 2002.
8. Michael G. Luby and Michael Mitzenmacher. Verification codes. In *Proc. Allerton Conf. on Communication, Control, and Computing*, 2002.
9. Michael G. Luby, Michael Mitzenmacher, and M. Amin Shokrollahi. Analysis of random processes via and-or tree evaluation. In *SODA*, pages 364–373, 1998.
10. Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Analysis of low density codes and improved designs using irregular graphs. In *Proceedings of the 30th annual ACM Symposium on Theory of Computing*, STOC '98, pages 249–258. ACM, 1998.

11. Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569 –584, feb 2001.
12. Rafail Ostrovsky and William E. Skeith. Private searching on streaming data. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 223–240. Springer, 2005.
13. Rafail Ostrovsky and William E. Skeith. Private searching on streaming data. *Journal of Cryptology*, 20(4):397–430, 2007.
14. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.
15. Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the SIAM*, 8(2):300–304, June 1960.