Private Stream Search at the Same Communication Cost as a Regular Search: Role of LDPC Codes

Matthieu Finiasz
CryptoExperts

Kannan Ramchandran
UC Berkeley

***Abstract*—Private Stream Search allows users to perform keyword-based queries to a database without revealing any information about the keywords they are searching. Using homomorphic encryption, Ostrovsky and Skeith proposed a computationally secure 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 non-private query only requires mS bits. In this work we propose two new *communication optimal* constructions, both allowing a communication 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 on the chosen computational security level.**

I. Introduction

The goal of Private Stream Search (PSS) is to be able to perform keyword-based search queries to a server, without disclosing any information about the keywords in the queries. For the moment, PSS is not widely used in practice, but it could play a key role in privacy protection. Compared to a standard non-private search, there is no fundamental reason for PSS to have a much higher latency (response time), bandwidth usage (response size) or to be less reliable (miss some matching documents). The main focus of this article is to show that, apart from the cost of the required cryptographic operations, PSS can indeed be as efficient as a normal search.

Previous Works: The first private stream search algorithm was introduced in 2005 by Ostrovsky and Skeith [1] and makes clever use of homomorphic encryption to hide the content of the query. This scheme requires the use of a public dictionary of possible keywords and is restricted to OR queries. We do not address these restrictions here, but any improvement to the original Ostrovsky-Skeith scheme in these domains will almost certainly also apply to the new schemes presented here. Following Ostrovsky and Skeith's work, some improvements have been proposed independently by Bethencourt, Song and Waters [2] and by Danezis and Díaz [3]. These improved schemes 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.

Our Work: The main contribution of this paper is the proposal of two new private stream search algorithms that are centered on the Ostrovsky and Skeith construction. Both constructions are founded on coding theory, resulting in a fundamental decomposition of the communication component of the PSS problem, thus allowing for state of the art results that improve significantly on current schemes. Our first scheme uses Reed-Solomon codes [4] 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 [5], [6] 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.

II. Description of the Original Scheme

A. Homomorphic Encryption: the Paillier Cryptosystem

All known PSS schemes require the use of homomorphic encryption, that is, encryption schemes for which some computations can be performed on ciphertexts and have a meaning for the corresponding plaintext. The most famous example of homomorphic encryption is the Paillier cryptosystem [7] which allows to compute linear combinations of plaintexts by multiplying the associated ciphertexts. More precisely, if we denote by $\mathcal{E}$ and $\mathcal{D}$ the encryption and decryption functions, the Paillier cryptosystem has the following properties:

- $\mathcal{D}(\mathcal{E}(M_1) \times \mathcal{E}(M_2)) = M_1 + M_2$,
- $\mathcal{D}(\mathcal{E}(M_1)^c) = c \times M_1$.

This scheme is a public key encryption scheme (anyone can encrypt data, but only the legitimate key owner can decrypt it) and offers semantic security (it is computationally infeasible to distinguish between $\mathcal{E}(0)$ and $\mathcal{E}(1)$). Semantic security is a necessary property for PSS, but it also implies that encryption should be randomized and, as such, induce a message expansion. In the case of Paillier's cryptosystem, documents are elements of $\mathbb{Z}_N$ (integers modulo a large composite number N) and ciphertexts belong in $\mathbb{Z}_{N^2}$, inducing an expansion factor of 2. For larger documents, the Damgård-Jurik extension [8] allows documents in $\mathbb{Z}_{N^s}$ with ciphertexts in $\mathbb{Z}_{N^{s+1}}$ (for any integer s), meaning the document is expanded by a constant number of bits $\log N$, depending only on the security parameter N (typically of 1024 bits).

B. The Original Ostrovsky-Skeith Scheme

A PSS scheme works in three steps: first the user builds a query and sends it to the server, then the server executes the

This research was funded by the NSF grant CCF-0964018.

978-1-4673-2579-0/12/$31.00 ©2012 IEEE

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 Ostrovsky-Skeith scheme.

1) Query Construction: Let $\Omega = \{w_1, \ldots, w_{|\Omega|}\}$ be the dictionary of possible keywords and $K \subseteq \Omega$ the set of keywords the user wants to query. The query is $Q = \{q_1, \ldots, q_{|\Omega|}\}$, where $q_j = \mathcal{E}(\mathbf{1}_{w_j \in K})$ and $\mathcal{E}$ denotes a Paillier encryption 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 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, but increases the communication cost).

2) Query Execution: When 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 $\mathcal{E}(0)$.

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 $F_i = \big(\prod_{w_j \in W_i} q_j\big)^{f_i} = \prod_{w_j \in W_i} \mathcal{E}(\mathbf{1}_{w_j \in K})^{f_i}$. Thanks to the homomorphic property of $\mathcal{E}$, we also have: $F_i = \mathcal{E}\big(f_i \sum_{w_j \in W_i} \mathbf{1}_{w_j \in K}\big)$.

Let us define $c_i = \sum_{w_j \in W_i} \mathbf{1}_{w_j \in K}$, the number of keywords of K that match f_i. We thus 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 $B_j = \prod_i F_i^{\mathbf{1}_{j \in b_i}} = \mathcal{E}\big(\sum_i \mathbf{1}_{j \in b_i} c_i f_i\big)$, 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 $B = \mathcal{E}(H \times (c_i f_i)_{i \in [1,t]})$.

3) 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.

Of course, this operation is only possible if it is possible to detect singletons and if extracting the value of f_i from a singleton is easy. This is possible by embedding a small checksum in each document.

4) Asymptotic Cost: The encrypted buffer that is sent back by the server to the user has size γm. In order for the user to be able 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, the answer is thus of order $O(2mS \log m)$ using Paillier's cryptosystem, or $O(m(S + 1024) \log m)$ using the Damgård-Jurik extension (with a 1024-bits modulus N). For a large number of matching documents, this construction is thus not very practical and some improvements are needed to keep the buffer size *linear* in the number of matching documents.

III. Two New PSS Constructions

Looking at the Ostrovsky-Skeith construction from an information theory point of view, it appears that the server starts by computing an encrypted sparse[1] vector $\big(\mathcal{E}(c_i f_i)\big)_{i \in [1,t]}$ and optimizing communications consists in "compressing" this vector before sending it to the user. As this vector is encrypted, standard compression techniques cannot be applied. However, the homomorphic property of $\mathcal{E}$ makes it possible to homomorphically multiply it by a matrix: if we consider this vector as an error pattern, we can compute its syndrome with respect to any parity check matrix. This is what we propose to do (with a few additional tweaks) in our new constructions.

A. A Zero-Error Construction Using Reed-Solomon Codes

Our first construction uses Reed-Solomon codes [4] and exploits their MDS 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.

A direct application of this would consist in multiplying the $\mathcal{E}(c_i f_i)$ vector 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. This gives a PSS scheme with communications linear in the number of matching documents: $4mS$ bits have to be sent using the standard Paillier cryptosystem. However, it is possible to do better than a factor 4 expansion.

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 (remember that for security reasons N will be at least of order 2^{1024}), and using a Reed-Solomon code over $\mathbb{Z}_N$ 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 our algorithm works.

1) Query Construction: This step is identical to the original Ostrovsky-Skeith algorithm, but without a parameter γ.

[1] The corresponding decrypted vector is of Hamming weight m if there are m matching documents.

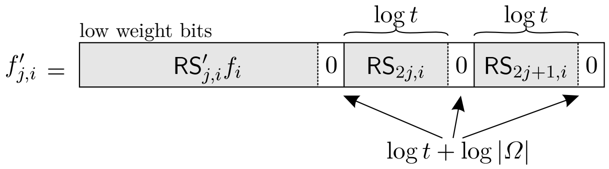

Fig. 1. Embedding of two Reed-Solomon codes and f_i inside a element of $\mathbb{Z}_N$.$|\Omega|$ is the dictionary size and t the database size: the zero-paddings allow to avoid overflows when computing linear combinations of $f'_{j,i}$.

2) Query Execution: As in the previous construction, for each document f_i, the server computes the encryption $\mathcal{E}(c_i)$ of the number of 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 Paillier plaintext. 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 $\mathcal{E}(c_i)^{f'_{j,i}}$.

3) Document Extraction: 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 $\sum_i c_i \mathsf{RS}_{2j+1,i}$. Reducing the last two elements modulo p_t for each $j \in [1, m]$, the user 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.

4) 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 every coefficient in the matrix, that is mt exponentiations instead of t as in the other algorithm.

However, on the user side, the computational cost remains identical 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.

5) Asymptotic 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 to avoid overflows. 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[2] 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 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.

B. An Asymptotically Optimal Construction Using Irregular LDPC Codes

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:

- instead of using a fix LDPC matrix, pseudo-randomly generate its columns from the documents f_i,
- use a decoding algorithm similar to the erasure correction algorithm proposed by Luby and Mitzenmacher for verification codes [9].

The construction we propose works as follows.

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

2) 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:

- it computes $\mathcal{E}(c_i f_i)$ exactly as in the original scheme,
- using a pseudo-random number generator seeded by f_i, it generates a column weight d following a given distribution (the best choice for this distribution is discussed in Section III-B4) and generates a random binary vector H_i of length ℓ and Hamming weight d,
- for every non-zero position in H_i, it multiplies the corresponding positions in buffer B by $\mathcal{E}(c_i f_i)$.

In the end, B contains the encrypted syndrome of the $(c_i f_i)$ vector with respect to an irregular LDPC code (with a chosen column weight distribution), which the server sends to the user.

3) Document Extraction: As in the original scheme, when receiving buffer B, the user starts by decrypting it and looks for singletons. The difference here is that, for every singleton, the user gets the value of one document f_i and can regenerate (using the same PRNG as the server) the corresponding column H_i. Knowing H_i it is possible to remove document f_i from the other syndrome positions where it has been added,

[2] In practice, this can be reduced to $3\log m + 3\log|K|$ bits, but this requires to modify the scheme depending on m and K which might not be convenient.

TABLE I
MINIMAL EXPANSION RATIOS WHEN USING COLUMNS OF CONSTANT WEIGHT d

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

thus uncovering new singletons, which in turn can reveal new documents f_i. This gives an iterative algorithm which can be analyzed in the same way as in [6]: knowing the column and row weight distributions of the parity check matrix it is possible to analyze the asymptotic behavior of the algorithm and compute the probability of recovering all documents.

4) Choosing an Optimal Column Weight Distribution: In order to analyze the decoding algorithm, the matrix H must be transformed into a bipartite graph. On the left of the graph are m information nodes and on the right are ℓ parity nodes. These nodes are connected by edges: each 1 in H is an edge in the graph, linking an information node to 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.

Studying the probability of success of this algorithm for given parameters m and ℓ is difficult, however, as proven in [10], 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 quite easily. 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 $b_{j+1} = \lambda(1-\rho(1-b_j))$. Asymptotically, the decoding algorithm is successful if $b_j \overset{j\to\infty}{\longrightarrow} 0$, which will be the case if:

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

a) Constant Column Weight: Danezis and Díaz [3] improved the original Ostrovsky-Skeith scheme by using a similar iterative decoding algorithm with constant column weight d. They do not provide any asymptotic analysis, but their construction is equivalent to choosing $\lambda(x) = x^{d-1}$ and $\rho(x) = \exp\left(-\frac{md}{\ell}(1-x)\right)$. The best expansion ratios $\frac{\ell}{m}$ one can achieve with constant column weight d (and an asymptotic probability of recovering all documents of 1) are reported in Table I. The best choice is $d = 3$ leading to an expansion of at least 22% and an asymptotic communication amount of at least $1.22m(S+1024)$ bits for m documents of S bits.

b) Harmonic Distribution: The harmonic distribution of order D consists in a normalized truncated (at order D) Taylor series expansion of $-\ln(1-x)$. 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 (1) 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.

c) Enhanced Harmonic Distribution: A problem of the plain harmonic distribution is that it contains columns of weight 2 which have a high probability of producing collisions: two identical columns H_i leading to a deadlock in the iterative decoding algorithm. To avoid this, the best solution is to combine the harmonic and constant weight distributions into what we call the enhanced harmonic distribution. Each column of length ℓ is split into a weight 3 column of length ℓ_3 and a harmonic column of length $\ell - \ell_3$. Choosing $\ell_3 = O(\sqrt{\ell})$ is enough to ensure that (asymptotically) no collisions take place and the probability of recovering all documents tends to 1, while not modifying the expansion factor $\frac{\ell}{m}$. Fig. 2 shows how the 3 distributions compare for various values of ℓ.

Using the enhanced harmonic distribution, the asymptotic communication cost of PSS with m matching documents of S bits becomes $2mS$ using the standard Paillier cryptosystem or $m(S+1024)$ using the Damgård-Jurik extension.

IV. 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 compared to a non-private query. 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. 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. Our offline-online 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,

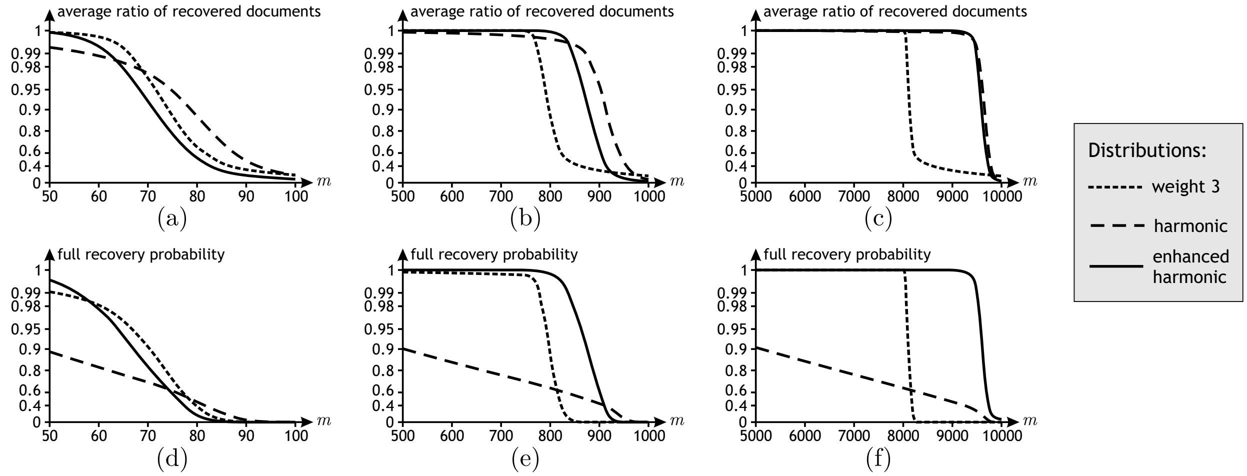

Fig. 2. Simulation results for the LDPC decoding algorithm for 3 different distributions. The curves represent the average ratio of recovered documents and the probability of recovering all m documents as a function of the number of matching documents m for different buffer sizes ℓ. In (a) and (d) $\ell = 100$ and $\ell_3 = 10$, in (b) and (e) $\ell = 1000$ and $\ell_3 = 40$, and in (c) and (f) $\ell = 10000$ and $\ell_3 = 60$.

- 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.

B. Dealing with Multiple Keywords.

One constraint with the previous scheme is that 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.

V. Conclusion

We presented two new constructions for private stream search that allow a communication complexity very close to that of a non-private search. As in all known PSS schemes, the workload can be quite heavy on the server side but it remains very small on the user side. In that sense, these constructions could be practical for many applications where privacy matters. We also present an offline-online variant of our construction that can make these schemes practical even if a delay between the query and the reply cannot be accepted.

References

[1] R. Ostrovsky and W. E. Skeith, "Private searching on streaming data," in *Advances in Cryptology - CRYPTO 2005*, ser. LNCS, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 223–240.

[2] J. Bethencourt, D. X. Song, and B. Waters, "New techniques for private stream searching," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, 2009.

[3] G. Danezis and C. Díaz, "Space-efficient private search with applications to rateless codes," in *Financial Cryptography 2007*, ser. LNCS, S. Dietrich and R. Dhamija, Eds., vol. 4886. Springer, 2007, pp. 148–162.

[4] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the SIAM*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[5] R. G. Gallager, "Low-density parity-check codes," Cambridge, Massachussets: M.I.T. Press, 1963.

[6] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. 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*, ser. STOC '98. ACM, 1998, pp. 249–258.

[7] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99*, ser. LNCS, J. Stern, Ed., vol. 1592. Springer, 1999, pp. 223–238.

[8] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *PKC 2001*, ser. LNCS, K. Kim, Ed., vol. 1992. Springer, 2001, pp. 119–136.

[9] M. G. Luby and M. Mitzenmacher, "Verification codes," in *Proc. Allerton Conf. on Communication, Control, and Computing*, 2002.

[10] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *SODA*, 1998, pp. 364–373.