Space-Efficient Private Search with Applications to Rateless Codes

George Danezis and Claudia Diaz

K.U. Leuven, ESAT/COSIC,
Kasteelpark Arenberg 10,
B-3001 Leuven-Heverlee, Belgium
{george.danezis, claudia.diaz}@esat.kuleuven.be

Abstract. Private keyword search is a technique that allows for searching and retrieving documents matching certain keywords without revealing the search criteria. We improve the space efficiency of the Ostrovsky *et al.* Private Search [9] scheme, by describing methods that require considerably shorter buffers for returning the results of the search. Our basic decoding scheme *recursive extraction*, requires buffers of length less than twice the number of returned results and is still simple and highly efficient. Our extended decoding schemes rely on solving systems of simultaneous equations, and in special cases can uncover documents in buffers that are close to 95% full. Finally we note the similarity between our decoding techniques and the ones used to decode rateless codes, and show how such codes can be extracted from encrypted documents.

1 Introduction

Private search allows for keyword searching on a stream of documents (typical of online environments) without revealing the search criteria. Its applications include intelligence gathering, medical privacy, private information retrieval and financial applications. Financial applications that can benefit from this technique are, for example, corporate searches on a patents database, searches for financial transactions meeting specific but private criteria and periodic updates of filtered financial news or stock values.

Rafail Ostrovsky *et al.* presented in [9] a scheme that allows a server to filter a stream of documents, based on matching keywords, and only return the relevant documents without gaining any information about the query string. This allows searching to be outsourced, and only relevant results to be returned, economising on communications costs. The authors of [9] show that the communication cost is linear in the number of results expected. We extend their scheme to improve the space-efficiency of the returned results considerably by using more efficient coding and decoding techniques.

Our first key contribution is a method called *recursive extraction* for efficiently decoding encrypted buffers resulting from the Ostrovsky *et al.* scheme. The second method, based on solving systems of linear equations, is applied after recursive extraction and allows for the recovery of extra matching documents

S. Dietrich and R. Dhamija (Eds.): FC 2007 and USEC 2007, LNCS 4886, pp. 148–162, 2007.

from the encrypted buffers. Recursive extraction results in the full decoding of buffers of length twice the size of the expected number of matches, and has a linear time-complexity. Shorter buffers can also be decrypted with high probability. Solving the remaining equations at colliding buffer positions allows for even more documents to be retrieved from short buffers, and in the special case of documents only matching one keyword, we can decode buffers that are only 10% longer than the expected matches, with high probability. We present simulations to assess the decoding performance of our techniques, and estimate optimal parameters for our schemes.

In this work we also present some observations that may be of general interest beyond the context of private search. We show how arrays of small integers can be represented in a space efficient manner using Pailler ciphertexts, while maintaining the homomorphic properties of the scheme. These techniques can be used to make private search more space-efficient, but also implement other data structures like Bloom filters, or vectors in a compact way. Finally we show how rateless codes, block based erasure resistant multi-source codes, can be extracted from encrypted documents, while maintaining all their desirable properties.

This paper is structured as follows: We introduce the related work in Section 2; present more in detail in Section 3 the original Ostrovsky scheme whose efficiency we are trying to improve; and explain in Section 4 the required modifications. Sections 5 and 6 present the proposed efficient decoding techniques, which are evaluated in Section 7. In Section 8 we explain how our techniques can be applied to rateless codes; and we present our conclusions in Section 9.

2 Related Work

Our results can be applied to improve the decoding efficiency of the Private Search scheme proposed by Rafail Ostrovsky *et al.* in [9]. This scheme is described in detail in Section 3. Danezis and Diaz proposed in [5] some preliminary ideas on how to improve the decoding efficiency of the Ostrovsky Private Search scheme, which are elaborated in this paper.

Bethencourt *et al.* [1,2] have independently proposed several modifications to the Ostrovsky private search scheme which include solving a system of linear equations to recover the documents. As such, the time complexity of their approach is $\mathcal{O}(n^3)$, while our base technique, recursive extraction, is $\mathcal{O}(n)$. Their technique also requires some changes to the original scheme [9], such as the addition of an encrypted buffer that acts as a Bloom filter [3]. This buffer by itself increases by 50% the data returned. Some of our techniques presented in section 6.2, that allow for efficient space representation of concatenated data, are complementary to their work, and would greatly benefit the efficiency of their techniques.

The rateless codes for big downloads proposed by Maymounkov and Mazières in [8] use a technique similar to ours for efficient decoding, indicating that our ideas can be applied beyond private search applications. We explore further this relation in Section 8, where we show how homomorphic encryption can be used to create rateless codes for encrypted data.

Pfitzmann and Wainer [13] also notice that collisions in DC networks [4] do not destroy all information transmitted. They use this observation to allow n messages to be transmitted in n steps despite collisions.

3 Private Search

The Private Search scheme proposed by Ostrovsky *et al.* [9] is based on the properties of the homomorphic Paillier public key cryptosystem [10], in which the multiplication of two ciphertexts leads to the encryption of the sum of the corresponding plaintexts ($E(x) \cdot E(y) = E(x + y)$). Constructions with El-Gamal [6] are also possible but do not allow for full recovery of documents.

The searching party provides a dictionary of terms and a corresponding Paillier ciphertext, that is the encryption of one ($t_i = E(1)$), if the term is to be matched, or the encryption of zero ($t'_i = E(0)$) if the term is of no interest. Because of the semantic security properties of the Paillier cryptosystem this leaks no information about the matching criteria.

The dictionary ciphertexts corresponding to the terms in the document d_j are multiplied together to form $g_j = \prod_k t_k = E(m_j)$, where m_j is the number of matching words in document d_j. A tuple $(g_j, g_j^{E(d_j)})$ is then computed. The second term will be an encryption of zero ($E(0)$) if there has been no match, and the encryption $E(m_j d_j)$ otherwise. Note that repeated words in the document are not taken into account, meaning that each matching word is counted only once, and m_j represents the number of different matching words found in a document.

Each document tuple is then multiplied into a set of l random positions in a buffer of size b (smaller than the total number of searched documents, but bigger than the number of matching documents). All buffer positions are initialized with tuples $(E(0), E(0))$. The documents that do not match any of the keywords, do not contribute to changing the contents of these positions in the buffer (since zero is being added to the plaintexts), but the matched documents do.

Collisions will occur when two matching documents are inserted at the same position in the buffer. These collisions can be detected by adding some redundancy to the documents. The color survival theorem [9] can be used to show that the probability that all copies of a single document are overwritten becomes negligibly small as the number of l copies and the size of the buffer b increase (the suggested buffer length is $b = 2 \cdot l \cdot M$, where M is the expected number of matching documents). The searcher can decode all positions, ignoring the collisions, and dividing the second term of the tuples by the first term to retrieve the documents.

4 Modifications to the Original Scheme

A prerequisite for more efficient decoding schemes is to reduce the uncertainty of the party that performs the decoding. At the same time, the party performing the search should gain no additional information with respect to the original scheme. In order to make sure of this, we note that the modifications to the original scheme involve only information flows from the searching (encoding) party back to the matching (decoding) party, and therefore cannot introduce any additional vulnerabilities in this respect.

Our basic decoding algorithm (presented in Section 5) only requires that the document copies are stored in buffer positions known to the decoder. In practice, the mapping of documents to buffer positions can be done using a good hash function $H(\cdot)$ that can be agreed by both parties or fixed by the protocol. We give an example of how this function can be constructed.

Notation:

- l is the total of copies stored per document;
- d_{ij} is the j-th copy of document d_i ($j = 1 \ldots l$) – note that all copies of d_i are equal;
- b is the size of the buffer;
- q is the number of bits needed to represent b ($2^{q-1} < b \leq 2^q$);
- p_{ij} is the position of document copy d_{ij} in the buffer ($0 \leq p_{ij} < b$).

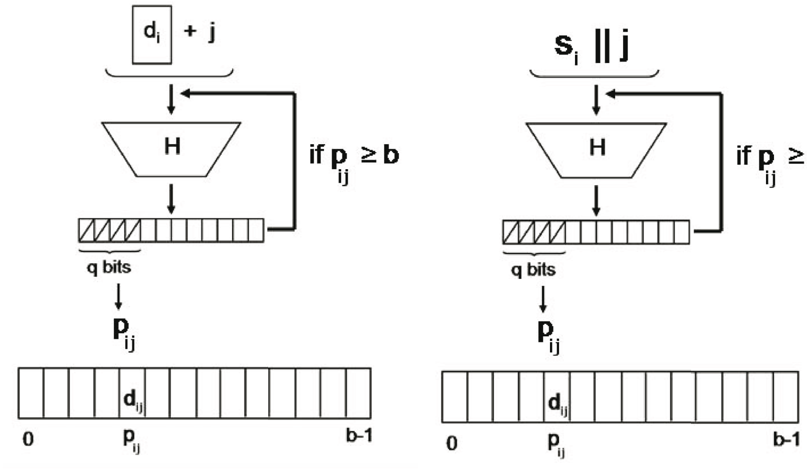
The hash function is applied to the sum of the the document d_i and the copy number j, $H(d_i + j)$. The position p_{ij} is then represented by the q most significant bits of the result of the hash. If there is index overflow (i.e., $b \leq p_{ij}$), then we apply the hash function again ($H(H(d_i + j))$) and repeat the process, until we obtain a result $p_{ij} < b$. This is illustrated in Figure 1(a).

With this method, once the decoding party sees a copy of a matched document, d_i, it can compute the positions of the buffer where all l copies of d_i have been stored (and thus extract them from those positions) by applying the function to $d_i + j$, with $j = 1 \ldots l$.

We present in Section 6 an extension to our decoding algorithm that further improves its decoding efficiency. The extension requires that the total number N of searched documents is known to the decoder, and that the positions of *all* (not just matched) searched documents are known by the decoder. This can be achieved by adding a serial number s_i to the documents, and then deriving the position p_{ij} of the document copies as a function of the document serial number and the number of the copy $H(s_i||j)$, as shown in Figure 1(b). We then take the q most significant bits of the result and proceed as in the previous case.

With respect to the original Ostrovsky scheme, our basic algorithm only requires the substitution of the random function $U[0, b-1]$ used to select the buffer positions for the document copies by a pseudorandom function dependent on the document and the copy number, that can be computed by the decoder.

The extension requires that the encoder transmits to the decoder the total number N of documents searched. The encoder should also append a serial number to the documents (before encrypting them). We assume that the serial numbers take values between 1 and N (i.e., $s_i = i$).



(a) in the basic algorithm (b) in the extended algorithm

Fig. 1. Function to determine the position of a document copy d_{ij}

5 Basic Decoding Algorithm: Recursive Extraction

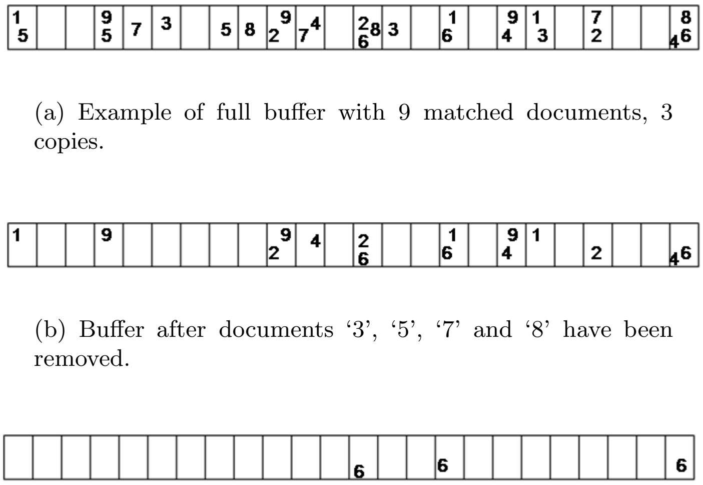
Given the minor modifications above, we note that much more efficient decoding algorithms can be used, that would allow the use of significantly smaller buffers for the same recovery probability.

While collisions are ignored in the original Ostrovsky scheme, our key intuition is that collisions are in fact not destroying all information, but merely adding together the encrypted plaintexts. This property can be used to recover a plaintext if the values of the other plaintexts with which it collides are known.

The decoder decrypts the buffer, and thanks to the redundancy included in the documents, it can discern three states of a particular buffer position: whether it is empty, contains a single document, or contains a collision.

In this basic scheme, the empty buffer positions are of no interest to the decoder (they do provide useful information in the extended algorithm, as we shall see in the next section). In the case of it containing a single document d_i, then the document can be recovered. By applying the hash function as described in Section 4 to $d_i + j$ with $j = 1 \dots l$, the decoder can locate all the other copies of d_i and extract them from the buffer. This hopefully uncovers some new buffer positions containing only one document. This simple algorithm is repeated multiple times until all documents are recovered or no more progress can be made.

In the example shown in Figure 2(a), we match 9 documents and store 3 copies of each in a buffer of size 24. Documents '3', '5', '7' and '8' can be trivially recovered (note that these four documents would be the only ones recovered in the original scheme). All copies of these documents are located and extracted



(a) Example of full buffer with 9 matched documents, 3 copies.

(b) Buffer after documents ‘3’, ‘5’, ‘7’ and ‘8’ have been removed.

(c) Buffer after documents ‘1’, ‘2’, ‘4’ and ‘9’ have been removed.

Fig. 2. Example of recursive extraction

from the buffer. At this point, documents ‘1’, ‘2’, ‘4’ and ‘9’ appear alone in at least one position, and can therefore be extracted. Once they are removed from the buffer, document ‘6’ can be also retrieved.

This very simple algorithm already provides an improvement of a factor l (number of document copies) over the original Ostrovsky scheme, as we show in our evaluation in Section 7.

6 Extended Decoding Algorithm: Solving Equations

Our basic decoding algorithm may terminate without recovering all matching documents if we run into a situation where a group of documents is copied to the same set of buffer positions. Figure 3 gives an example of such a case: 3 matching documents (2 copies each) have been stored in 3 colliding buffer positions. Our key observation is that by expressing these buffer positions as linear equations, we can still retrieve the 3 colliding documents.

Our basic decoding based on recursive extraction algorithm takes advantage of the fact that, once one document copy is retrieved, all other copies can be extracted from the buffer. This does not require the buffer positions of the document

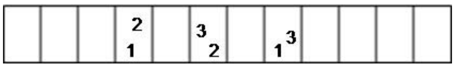


Fig. 3. Example of buffer with 2 copies of 3 documents colliding in 3 buffer positions

copies to be predictable for the decoder, and he needs to see at least one copy of the document before being able to compute the positions of the other copies.

Making predictable the positions of the document copies in the buffer, further reduces uncertainty and allows us to further improve the decoding efficiency. In order to achieve this we include, as pointed out in Section 4, a serial number in the document and tell the decoder the total number N of documents searched. Then we make the document copy position dependent on its serial and copy numbers (as shown in Figure 1(b)), so that the position of each searched document copy is known *a priori* to the decoder.

The end resulting buffer can be modeled as a system of simultaneous equations. Each equation represents a buffer position, which leads to at most b equations. Each document that has a copy in this buffer position is a variable in the equation, and the sum of the actual matched documents equals the value of the bucket. Note that non-matching documents are set to zero, and therefore they do not contribute to the sum.

The example shown in Figure 3 represents a mapping of three documents d_1, d_2 and d_3 into three buckets b_3, b_5 and b_7. The corresponding set of equations would be:

$$d_1 + d_2 = b_3 \tag{1}$$

$$d_2 + d_3 = b_5 \tag{2}$$

$$d_1 + d_3 = b_7 \tag{3}$$

We can solve the system of linear equations as long as the number of unknowns (i.e., documents) is lower or equal than the number of equations (i.e., buffer positions).

Note that, if a non-matching document d_4 is also allocated in two of the buffer positions (say, b_3 and b_7), we cannot retrieve any document because there are more unknowns that equations. Therefore the key to the success of this decoding technique is applying first the recursive extraction decoding to identify as many matching and non-matching documents possible; empty buffer positions give key information about non-matching documents. This aims to reduce the number of unknown documents to be less than the available equations, allowing us to solve the linear system. As such, solving equations is complementary to the first decoding technique, and it is always applied after recursive extraction.

6.1 Special Case: Searching for One Keyword

In some applications, the decoder may be interested in searching only one keyword in the documents. For example, when retrieving pseudonymous email [11], the decoder would provide his email address as the only keyword for searching in the documents.

In these cases, we can further improve the decoding efficiency to the extent that the buffer length needs to be less than 10% larger than the expected number of matches. As we show in Section 7, with this technique we can with high probability retrieve more than 900 matched documents from a buffer with 1000 positions.

The technique works as follows: the serial number is appended to the lower end of the document, leaving enough space to accommodate the sum of serial numbers of documents present in the bucket. The modified documents d'_i are computed as: $d'_i = d_i \cdot 2^S + i$, where S is the number of bits needed to make sure that the serial numbers add up without overflowing the appended bit space. Given that the total number of searched documents is N, the worst case would be that all searched documents match and that there is a bucket that contains a copy of each document. In this case, the sum of all serial numbers is $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$. Therefore, it suffices to append $S = 2\log_2(N)$ bits to the document, where the serial number i is included. Note that a lower number of bits between $\log_2(N)$ and $2\log_2(N)$ may be sufficient, since the average number of matched documents in a buffer position is generally much lower than the worst case.

As we are considering the special case in which only one keyword is being searched, matching documents would be multiplied by a one, while non-matching documents are multiplied by a zero (as opposed to the general case of searching K keywords, where the document may be multiplied by up to K if all searched keywords are contained in it). The serial numbers, as part of the document, are also multiplied by either a one or a zero.

This scheme has the following property: given that 2 matching documents d_i and d_j are colliding in a buffer position b_k, the serial number bits of the collision will contain $i + j$. More generally, the sum of serial numbers in a given buffer position is given by: $R = \sum_{i=1}^{N}(i \cdot x_i \cdot y_i)$, where x_i is one if a document copy has been stored in that buffer position and zero otherwise (the values of x_i are known *a priori* to both the encoder and the decoder); and the variable y_i takes the value one if the document has been matched and zero otherwise (the value of y_i is unknown both to the encoder and the decoder). Note that in the general case of searching K keywords y_i takes values between zero and K.

In order to perform the decoding, we use the following information:

- M: number of matching documents in a buffer position (given by the decryption of the multiplication of first parts of the tuple $(g_i, g_i^{d_i})$ for each document d_i in the buffer position, $M = D(\prod_i(g_i)) = \sum_i(x_i \cdot y_i)$)
- R: result of summing all serial numbers of matching documents in the position, $R = \sum_{i=1}^{N}(i \cdot x_i \cdot y_i)$.

We then seek the subset of M documents stored in the position whose serial numbers sum R. We illustrate the method with the example shown in Figure 4. In this example, we can express the equations for the buffer positions as:

$$d_1 + d_2 + d_3 = b_3 \tag{4}$$

$$d_2 + d_4 + d_5 = b_5 \tag{5}$$

$$d_1 + d_3 + d_4 + d_5 = b_7 \tag{6}$$

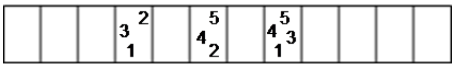


Fig. 4. Example of buffer with 2 copies of 5 documents colliding in 3 buffer positions

If there are two matches in b_3, b_5 and three in b_7 (i.e., $M = 2$ for b_3 and b_5; and $M = 3$ for b_7); and the accumulative serial numbers R of b_3, b_5 and b_7 are 4, 7 and 9, respectively, then we know that d_2 and d_4 are zeros and we can solve the system of 3 linear equations with 3 unknowns.

This technique may still be applicable in cases where 2 or 3 keywords are searched, but given that the complexity grows exponentially with the number K of searched keywords, it quickly becomes infeasible to use it, even with a moderate K.

6.2 Tight Packing of Encrypted Lists and Bit Fields

In the previous section we described how we can use a single Paillier ciphertext to encode, space permitting, both the serial number of the document and the document itself. This encoding still allows for the homomorphic property; i.e., for the different plaintext parts of the message to be added together field-wise, when two ciphertexts are multiplied. This is a special case of a generic mechanism we can use to further improve the space-efficiency of the original Ostrovsky *et al.* scheme [9], the encoding of the Bloom filter buffer in Bethencourt *et al.* [1,2], and our decoding schemes.

A Paillier ciphertext can fully represent a plaintext of up to $\lceil \log(n) - 1 \rceil$ bits of length. In many cases (e.g., the first element of each buffer position in the Ostrovsky scheme, the representation of the serial number, or the Bloom filter entry) only a small plaintext is to be represented. We can do this by using only one Paillier ciphertext and packing as many elements as possible into it. Consider that we have two ciphertexts $E(i)$ and $E(j)$, representing two fields. We assume that the cryptographic protocols we shall perform will never result in a sum of those fields being greater than b bits long. We can then encode these two ciphertexts as:

$$E(i) \cdot E(j)^{2^b} = E(j \cdot 2^b + i) \tag{7}$$

It is possible to add a value to any element of the encrypted list. First, the ciphertext is shifted to the appropriate position and then multiplied to the tightly packed buffer. For example:

$$E(j \cdot 2^b + i) \cdot E(i') = E(j \cdot 2^b + (i + i')) \tag{8}$$

$$E(j \cdot 2^b + i) \cdot E(j')^{2^b} = E((j + j') \cdot 2^b + i) \tag{9}$$

As long as the sum of each element can be represented using only b bits (which is much smaller than the bits needed to represent the modulus n) we prevent the carry interfering with other fields in the ciphertext and achieve a significant gain in space efficiency.

7 Experimental Results

We present in Figures 5 and 6 simulation results[1] illustrating the performance of recursive extraction and solving equations for different sets of parameters.

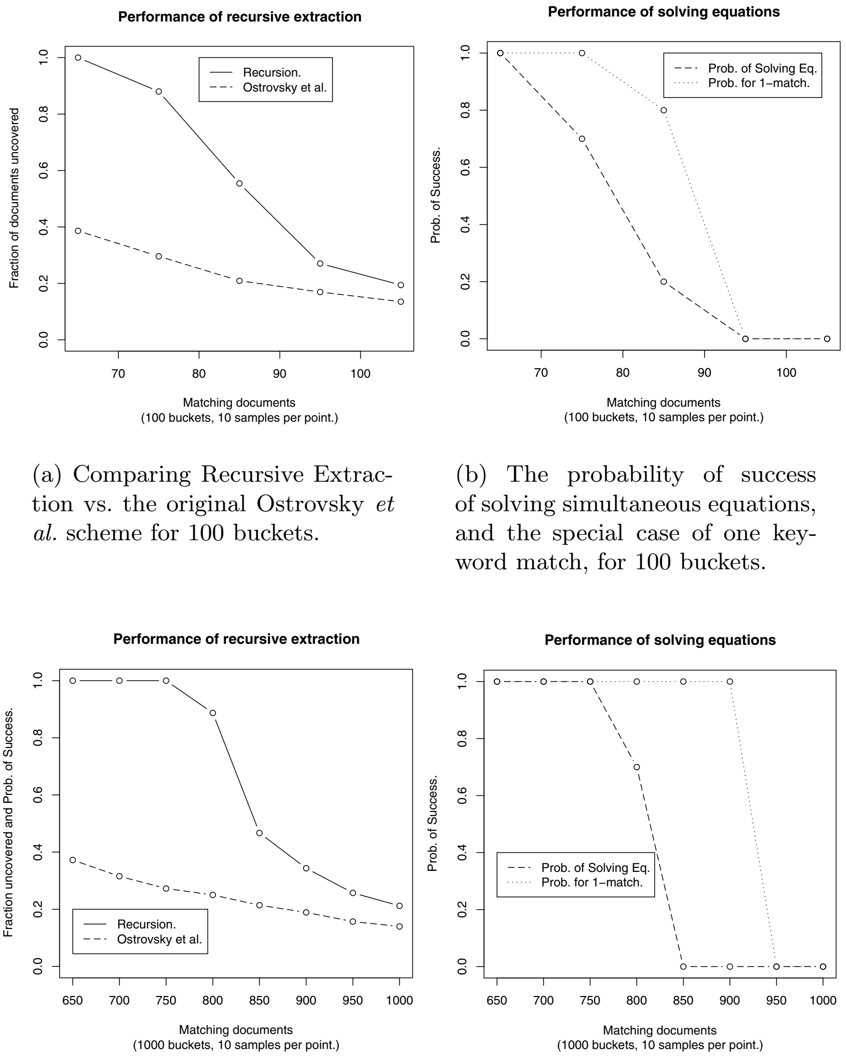
Figure 5 illustrates how the recovery rate for recursive extraction changes, as the number of matched documents increases in a buffer of 100 places (figure 5(a)), and 1000 places (figure 5(c)). The recovery rate is defined as the fraction of documents retrieved from the buffer. When the number of matches is lower than half the size of the buffer, the recovery of all matching documents is virtually guaranteed. We plot the recovery rate for a number of documents that is greater than half the size of the buffer, in order to better illustrate the limits of our technique. We also plot the recovery rate for the original decoding scheme proposed by Ostrovsky *et al.*, in order to show the improvement offered by our techniques.

Figures 5(b) and 5(d) show the probability of success of our techniques based on solving equations, for buffers of length 100 and 1000, respectively. We show the probability of success for the general case (K searched keywords), and for the special case (one searched keyword). These "probabilities of success" have to be interpreted differently from the recovery rates of recursive extraction: whether a set of equations in a buffer can be solved is an all-or-nothing event – meaning that either all remaining unknown variables are extracted or none. Since solving equations can only be done *after* recursive extraction, these probabilities of success have to be seen as providing the possibility to get *all* documents (*in addition* to the documents already retrieved as shown in figures 5(a) and 5(c)).

Figure 6 illustrates the effect that different number of copies have on the recovery rate of the documents, for all techniques. We graph "measures of success" computed over numbers of matches that range from half the size of the buffer to slightly more than the buffer size (i.e., edge cases). For this reason, the graphs cannot be used to compare techniques, but rather the relative performance of the number of copies in each technique. As we expect, recursive extraction provides no advantage over the Ostrovsky scheme when only one copy of the document is used. Using three copies seems to be optimal for most cases (and hence we used it for our experiments shown in figure 5).

We observe that too few or too many copies, reduce the recovery rate of documents. In the first case not enough copies of the document are present to guarantee retrieval by ensuring [9] that at least one copy is alone in the buffer. In the latter case too many documents are inserted, which lowers the probability that any document is found on its own in the buffer. The original claim in [9], that recovery becomes better as the number of copies increases may also mislead implementers. This is only the case if one assumes a buffer of infinite size – and therefore an optimal parameter for the number of copies has to be calculated for each set of practical values of buffer size and expected matches.

[1] The full source code to produce all experiments and graphs is publicly available at `http://homes.esat.kuleuven.be/~gdanezis/`



(a) Comparing Recursive Extraction vs. the original Ostrovsky *et al.* scheme for 100 buckets.

(b) The probability of success of solving simultaneous equations, and the special case of one keyword match, for 100 buckets.

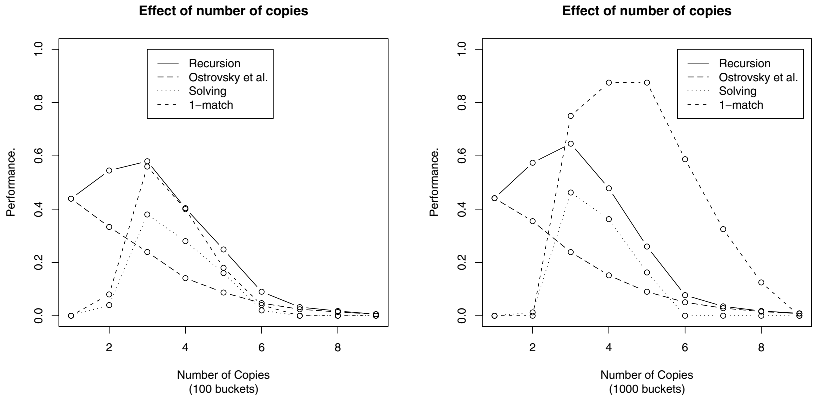
(c) Comparing Recursive Extraction vs. the original Ostrovsky *et al.* scheme for 1000 buckets.

(d) The probability of success of solving simultaneous equations, and the special case of one keyword match, for 1000 buckets.

Fig. 5. Performance evaluation of our techniques and comparison with the original scheme

8 Applications to Rateless Codes

Maymounkov and Mazires in [8] introduce "rateless codes", a method for erasure resistant, multi-source coding. These codes have been designed to be used in a



(a) Comparison for 100 buckets.

(b) Comparison for 1000 buckets.

Fig. 6. The effect of the number of copies used on the performance of all techniques

peer-to-peer context, where Alice maybe downloading the same file from multiple sources, with no coordination between sources. The network is assumed to be unreliable and dropping packets transporting message blocks with some probability. We have already pointed out that the recursive decoding technique we use to decode the buffer resulting from private search is similar to the decoding strategy for such codes.

Many proposals for censorship resistant peer-to-peer systems [12] rely on peers storing encrypted files. We show that we can produce rateless codes from Paillier encrypted files, or using other homomorphic schemes like El-Gamal. Furthermore we can combine blocks received from multiple sources in order to retrieve the original file. Since Paillier encryption is probabilistic, the encrypted files on the different peers are unlinkable to each other for anyone not knowing the private decryption key.

We consider that Bob stores the file blocks F_i of a file, as Paillier encrypted ciphertexts, under the public key of Alice, i.e. $E_A(F_0), E_A(F_1), \ldots E_A(F_n)$. As in the original proposal, Bob generates the rateless code in two phases. First, the message is expanded into a composite message by appending auxiliary blocks. Each auxiliary block is the multiplication encrypted message blocks selected uniformly at random from all message blocks using a pseudo-random number generator (for more details see [7]). In the second phase, an “infinite” stream of check blocks can be generated. Those are the blocks to be transmitted to Alice, along with their serial numbers and all the seeds of the pseudo-random number generators used to generate them. Each check block is generated as follows: a random degree d is chosen using a pseudo-random number generator from a distribution ρ_d defined in [8]. A number d of blocks chosen uniformly at random using a pseudo-random number generator are then multiplied together to form the check block. In both cases, due to the Paillier homomorphic properties, the product of the ciphertexts becomes the ciphertext corresponding to the sum of the plaintexts.

Bob sends check blocks to Alice, that decodes them. First of all, Alice decrypts the check block and recovers the sum of the auxiliary blocks from which it is composed. A similar algorithm as for the recursive decoding is then applied: check blocks that contain only one unknown auxiliary block are simplified by subtracting all the known blocks. As shown in [8], after a linear number of applications the message is recovered.

Allowing the computation of codes on encrypted data provides two key advantages:

- Alice can accept check blocks from two different sources, that are not in any way coordinating with each other, and use both streams to recover the original file. Yet, because of the randomized nature of Paillier, the different sources cannot know that the encrypted files correspond to the same source file.
- Alice knows when she has received sufficient blocks to recover the original message (from any number of sources), even if she is not able to decrypt the blocks and recover it. The mapping between auxiliary blocks and check blocks is determined only by the pseudo-random number sequences for which the seeds are known, and therefore she is able to tell when a decryption would be successful.

El-Gamal [6] encryption can also be used instead of Paillier, with certain advantages. In the El-Gamal variant ciphertexts are multiplied, which leads to the encryption of the product of the corresponding plaintexts. The decoder decrypts all blocks, as before, and divides (instead of subtracting) known blocks to simplify others.

A key observation is that the division necessary to reconstruct the original message can actually be performed on the encrypted check blocks, even without the knowledge of the secret key. As a result of this property of El-Gamal ciphertexts, not only messages encrypted blockwise using this cipher can be expanded into rateless codes served from multiple sources, but also the receiver of these blocks can perform the decoding and reconstruct a valid El-Gamal encrypted representation of the original message. The new representation of the message can in turn be rendered unlinkable to the two (or more) source representations by re-encrypting (also called self-blinding) the recovered ciphertexts.

9 Conclusions

We have presented in this paper efficient decoding mechanisms for private search. The very simple basic mechanism consists of recursively extracting documents from the returned buffer. The extended mechanism additionally solves equations in order to retrieve the remaining colliding documents in the buffer.

The size of the returned buffer with matched documents is the key to the success of private search schemes. If the size of this buffer is too long, any scheme can simply be reduced to transmitting back all the documents, which would save in complexity and cryptographic costs.

Our proposed decoding methods reduce by a significant constant factor the buffer sizes required by the Ostrovsky *et al.* Private Search [9] scheme. They require buffers less than twice the size of the matched documents, and a linear decoding complexity, meaning that the buffers at at least three times shorter than in the original scheme. Recursive extraction and solving equations are complementary and can be applied sequentially to extract a maximum number of documents from short buffers. Compression can be achieved by packing lists of values more efficiently, as presented in Section 6.2, further halving the base cost of the original scheme. In the important special case of matching documents that only contain one keyword, we achieve an overhead of less than 10%, making private search very practical.

We have presented simulation results to assess the decoding performance and estimate optimal parameters for our schemes.

Finally, we have shown how rateless codes can be constructed from encrypted data, while preserving the unlinkability of files across multiple sources.

Acknowledgements

This work has benefited from discussions with Paul Syverson at the Dagstuhl Seminar on Anonymous Communications (October 9-14, 2005). Claudia Diaz is supported by the IWT SBO ADAPID project (Advanced Applications for e-ID cards in Flanders) and George Danezis is sponsored by the F.W.O (Fund for Scientific Research) Flanders (Belgium).

References

1. Bethencourt, J., Song, D., Waters, B.: New constructions and practical applications for private stream searching (extended abstract). In: SP 2006. Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, USA, pp. 132–139. IEEE Computer Society Press, Los Alamitos (2006)
2. Bethencourt, J., Song, D., Waters, B.: New techniques for private stream searching. Technical report, Carnegie Mellon University (2006)
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
4. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
5. Danezis, G., Diaz, C.: Improving the decoding efficiency of private search. Dagstuhl Seminar on Anonymity and its Applications (October 2005)
6. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
7. Maymounkov, P.: Online codes. Technical report, New York University (2003)

8. Maymounkov, P., Mazieres, D.: Rateless codes and big downloads. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 247–255. Springer, Heidelberg (2003)
9. Ostrovsky, R., Skeith III, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005)
10. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
11. Sassaman, L., Cohen, B., Mathewson, N.: The pynchon gate: A secure method of pseudonymous mail retrieval. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2005), Arlington, VA, USA (November 2005)
12. Serjantov, A.: Anonymizing censorship resistant systems. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, Springer, Heidelberg (2002)
13. Waidner, M., Pfitzmann, B.: The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure servicability. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, Springer, Heidelberg (1990)