Improving the Decoding Efficiency of Private Search

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. We show two ways of recovering all matching documents, in the Ostrovsky *et al.* Private Search [3], while requiring considerably shorter buffers. Both schemes rely on the fact that documents colliding in a buffer position provide the sum of their plaintexts. Efficient decoding algorithms can make use of this property to recover documents never present alone in a buffer position.

1 Introduction

Rafail Ostrovsky *et al.* [3] have presented a scheme that allows a server to filter a stream of data, based on keywords, and only return the resulting documents without learning the query string. In this way searching can be outsourced, and only relevant results have to be returned, economizing on communications costs. The authors show that the communication cost is linear on the number of results expected. We extend their scheme to improve its efficinecy considerably by using more efficient coding and decoding techniques.

In this paper we show how to modify the basic scheme to support such encodings and present two techniques that require shorter buffers for the same probability of recovering all returned matches. The first, simplest and very practical technique allows us to use buffers that are twice the size of the expected number of matching documents. The second technique allows for documents to be recovered from even shorter buffers, but it is sensitive to the number of non-matching documents searched. This makes it less practical. It is of historical relevance that similar schemes have been used before by Michael Wainer and Birgit Pfitzmann to addapt DC-nets [1] and make them resistant to collisions [5].

2 Private Search

We shall start by briefly describing the private search [3] scheme. The scheme is based on the properties of the homomorphic Pallier public key cryptosystem [4], in which the multiplication of two ciphertext leads to the encryption of the sum of the correspondence plaintexts ($E(x) \cdot E(y) = E(x + y)$). Constructions with El-Gamal [2] are also possible but do not allow for full recovery of documents.

The searching party provides a dictionary of terms and a corresponding Pallier 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 Pallier 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 $g_j = \prod_k t_k = E(\mathcal{E}[\text{Number of matching terms}])$. A tuple $(g_j, g_j^{d_j})$ is then computed. The second term will be an encryption of zero ($E(0)$) if there has been no match, and the encryption $E(g_k d_j)$ otherwise.

Each document tuple is then multiplied into a set of l random positions in a buffer (of smaller size b than the number of documents) that has all positions initialized with tuples $(E(0), E(0))$. The documents that are not matched do not contribute to changing the contents of these positions in the buffer (since zero is being added to the plaintexts), but the matches 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 [3] can be used to show that the probability that all copies of a single document are overwritten becomes negligibly smaller as the number of l copies and the size of the buffer b increases (the suggested buffer length is $b = 2 \cdot l \cdot m$). 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.

3 Reducing Uncertainty

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 than in the original scheme. To make sure of this we note that all modifications to the original scheme involve only information flows from the searching party back to the matching party, and therefore cannot introduce any additional vulnerabilities in this respect.

The party performing the matching provides two additional pieces of information to the party that has requested the searching. First it needs to provide the total number of documents that were searched (not matched since this information is hidden), and a mapping of all searched documents to the buffer positions they were inserted.

In practice a good hash function can be agreed by both parties or fixed by the protocol, that given the size of the buffer b and the number of searched documents is used to produce the mapping. For example, a hash function $H : [0, (s \cdot l) - 1] \rightarrow [0, b - 1]$, where b is the buffer size, s is the number of documents and l the number of copies, can be used. To determine the position in the buffer of the ith copy of document j, one can just calculate $H(i \cdot j - 1)$.

A more generic solution would be for the searching party to provide the decoding party a fresh nonce, used to feed the pseudo-random number generator used to make any 'random' choices. Care must be taken in this case that the

decoding party can reconstruct all the random choices and map them reliably to their corresponding encoding actions.

Finally we require the plaintext of each document to contain the serial number of the document, a number from 0 to $s - 1$, where s is the total number of searched documents.

4 Making the Best out of Collisions

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

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 some plaintext if the values of the other plaintexts that it collides with are known.

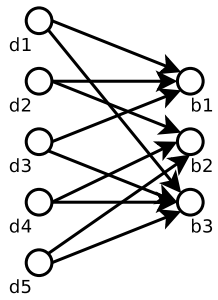

Fig. 1. An example mapping of five documents into 3 buckets (with 2 copies each.)

Two decoding algorithms are possible. Both model the returned buffer as a bipartite graph (as illustrated in figure 1): One set of edges represents documents and the other set of edges represents positions in the buffer. We draw vertexes connecting the document edges with the positions in the buffer in which they are held. The first very simple algorithm is based on removing known edges from a graph, and the second, sightly more complex, is based on solving systems of simultaneous equations.

4.1 Decoding by Removing Known Edges

The decoder can discern three states of a particular buffer position: whether it is empty, contains a single document, or contains a collision. In the case of it

containing a single document its serial number can also be recovered, since we require it to be included in the plaintext.

Our algorithm proceeds as follows: One removes from the graph the edges corresponding to documents in buffer positions that are empty. Then one recovers documents that are contained in buffer positions with only one document. The remaining documents in these buffer positions can also be removed from the graph since they contain empty documents. When removing an edge, all vertexes associated with it are also removed. This means in practice, that the plaintext of the recovered document is substracted from all other buffer positions containing a copy of this document. This hopefully uncovers some 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.

Simulation results are presented in figures 2 and 3. Figure 2 illustrates how the recovery rate changes with the size of the buffer. The recovery rate is defined as the fraction of messages that can be recovered using our techniques out of the 100 matching documents. For the purposes of this experiment we have set the number of copies to 5 (a good choice as we shall see). Note that the probability of recovery is very high (nearly one) for a buffer size that is double from the number of matches (200 in this example). This is a considerable improvement, of a factor l, over the original proposal.

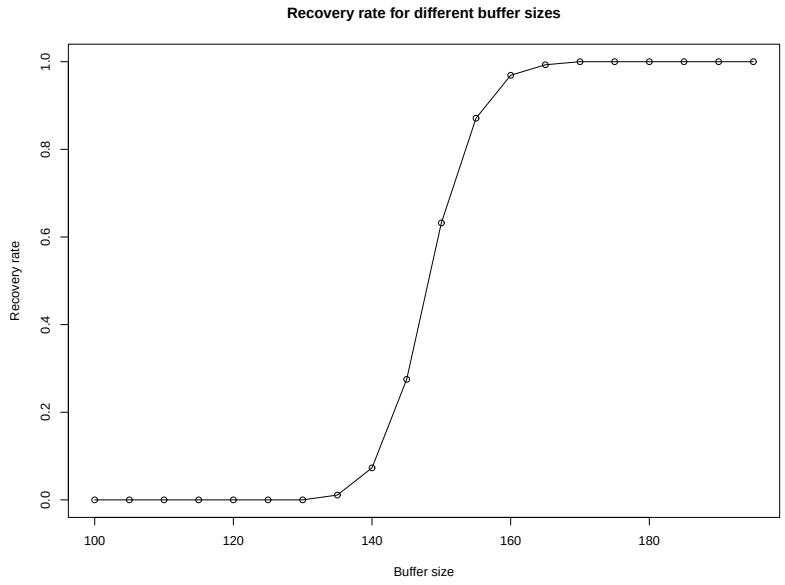

Fig. 2. Recovery rate for different buffer sizes. (1000 samples, 100 matches, 5 copies.)

Figure 3 illustrates the effect of different number of copies on the retrieval rate of the documents. 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 insuring [3] 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. We note that for our parameters 5 copies seems to provide optimal recovery (and hence the parameter of our previous experiment). The original claim in [3], 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.

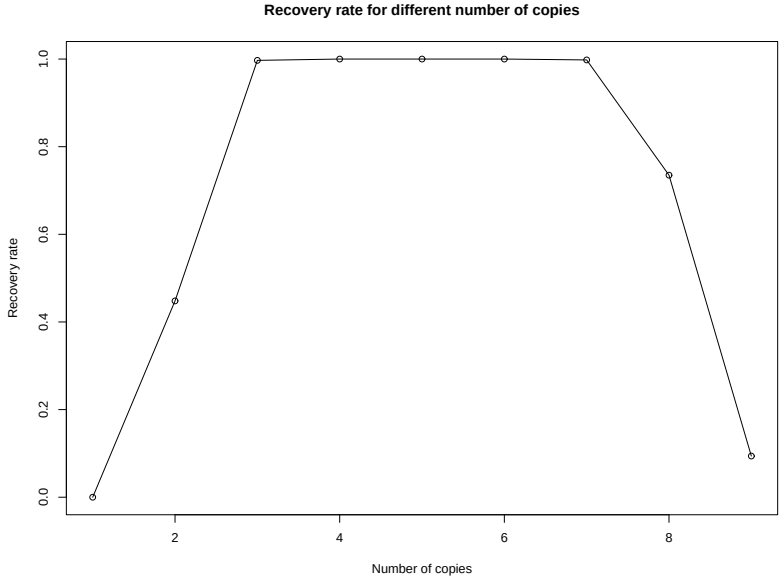

Fig. 3. Recovery rate for different number of copies. (1000 samples, 100 matches, 200 buffer places.)

Further efficiency improvements are possible in this scheme. For instance we can do without the need for any additional communication, of the allocation graph or the inclusion of the serial number in the documents themselves. The party performing the matching generates the positions of each document in the buffer by hashing, using a good hash function, each document's contents and using the result as a seed for a random number generator. In case the document is in a buffer position on its own, the searching party can recover its contents, hash

it and determine the positions of the other copies. Our protocol then continues as before, but with no need to know the full graph in advance – it is instead generated as the decoding makes progress. There is no need to have access to a serial number or the total number of documents in advance.

Yet additional information – allowing us to extract the number of documents as well as the serial number of each retrieved document – allows for better recovery, as we see in the next section.

4.2 Decoding by Solving Simultaneous Equations

The graph representing the assignments of document copies into buckets, as well as the end resulting buffer, can also be modeled as a system of simultaneous equations. Each equation represents a buffer position, and therefore there are at most b equations. Each document that has a copy in this buffer position is a variable in the equation, and their sum should equal the value of the bucket.

As an example, Figure 1 represents a mapping of five documents $d_1, \ldots, d_5$ into three buckets $b_1, \ldots, b_3$. The corresponding set of equations would be:

$$d_1 + d_2 + d_3 = b_1 \tag{1}$$

$$d_2 + d_4 + d_5 = b_2 \tag{2}$$

$$d_1 + d_3 + d_4 + d_5 = b_3 \tag{3}$$

It is clear that this set of equations cannot be solved in the general case – there are more variables than equations. It is only because some documents are actually represented by zero (since they do not match) that the number of variables can be reduced. This reduction is performed by applying the heuristics presented in the previous section. If after applying these heuristics we are left with a number of equations (i.e. buckets that contain more than one document) that is equal or higher to the number of unknown variables (representing documents), we can solve the system and retrieve the documents.

For a certain number of matching documents m and a certain number of non-matching documents d we can calculate the most probable number of decoded messages after we apply the heuristic decoding. This number is also dependent on the replication factor l and the number of buckets b. The analysis is simple:

- A bucket that contains no document or one document can be recognized as such, and all other documents that have copies in this buffer position can be tagged as non-matches (and assigned a value of zero). We first calculate the probability $p_{(0 \text{ or } 1)}$ a buffer position contains one or zero matching documents:

$$p_{(0 \text{ or } 1)} = b \cdot \left[\left(\frac{b-1}{b} \right)^{lm} + \frac{lm}{b} \left(\frac{b-1}{b} \right)^{lm-1} \right] \tag{4}$$

- For a non matching document to be excluded it should only have copies in buckets that contain more than one matching document. The probability

p_{unknown} of this happening to an individual non-matching document, given l copies, is:

$$p_{\text{unknown}} = (1 - p_{(0 \text{ or } 1)})^l \tag{5}$$

– We define an indicator random variable for each non matching document I_i, with probability of taking a value of 1 equal to p_{unknown}. By linearity of expectation we know that:

$$\mu = \mathcal{E}[\sum^{d} I_i] = \sum^{d} \mathcal{E}[I_i] = d \cdot p_{\text{unknown}} \tag{6}$$

The expectation μ is the expected number of undetected non-matching documents after the first stage of the heuristic decoding.

The above calculation is a conservative estimate of the decoding efficiency for two reasons. Firstly it does not take into account that the distribution into buffer positions is done without replacement. This makes it more likely for a non-matching document to be in a bucket that would allow it to be detected as such. Second, only the very first stage of heuristic decoding is taken into account, when actually the process is repeated until the set of equations cannot be simplified further. This, as we have seen in the previous section, will often lead to complete recovery, but would also reduce further the number of non-matching documents that are left unknown.

It is important to note that if only matching documents were to be returned all of them would be recoverable (since there are as many unknown documents as buffer positions we could simply solve the equations.) It is only because of the non-matching documents, that have to be identified as such, that the difficulty occurs. Figure 4 illustrates the number of non-matching documents that can be tolerated, given different buffer sizes, to still be able to retrieve all 100 matching documents (the replication factor is 5). When there are exactly as many buffer positions as matching documents, we expect this number to be zero (adding any further documents would lead to s system of equations with more unknowns than equations). As the number of buckets increases we can use the 'spare' equations to tolerate and detect more and more non-matching documents. After a buffer size of about 160 we expect our heuristic technique, presented above, to decode more messages, so that the number of tolerated non-matching documents becomes infinite.

The experimental procedure used to produce figure 4 is as follows: for each set of parameters of buffer size, we ran 1000 experiments. In each experiment we performed first the heuristic decoding to establish the remaining number of unknown documents, and the number of equations that were available. We then used equation 6 to estimate the number of non-matches that this balance of non-decoded documents versus number of equations could still decode. The average number of such non-matches is plotted for each value of the buffer size.

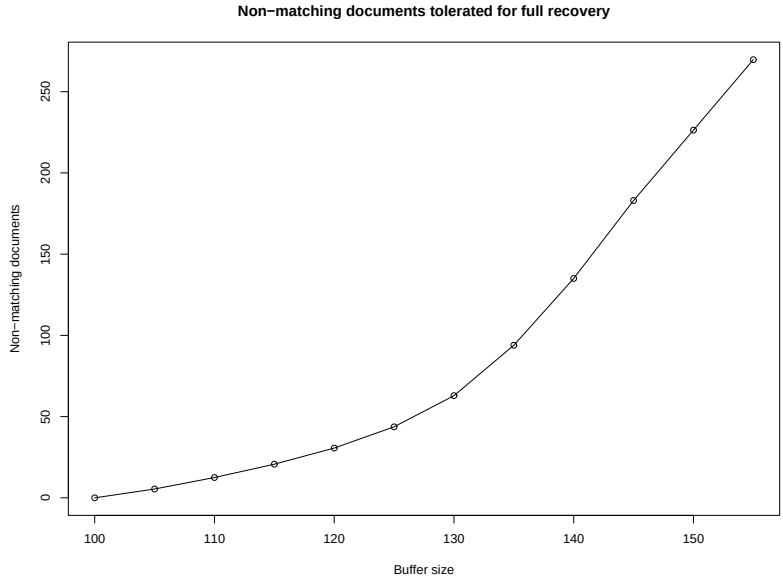

Fig. 4. Number of tolerated non-matching documents to achieve decoding (1000 samples, 100 matches, 5 copies.)

5 Open Issues and Conclusions

Following the original proposal we have assigned document copies pseudo-randomly into the buffer. Yet we also see that the distribution of the document copies in the buffer affects the decoding efficiency. It might be the case that a different graph is more appropriate to maximize the number of positions that are empty or populated by only one matching document, and minimizing the probability that the set of resulting equations cannot be solved.

Our simulations suggest that our schemes achieve a significant efficiency improvement over the original proposal. The size of the buffer that needs to be transferred has been reduced from $2lm$ to $2m$ to allow for full recovery using our heuristic scheme (m is the number of matching documents, l is the number of copies). As an example, to retrieve 100 documents we need a buffer of 200 positions instead of 1000! We have also uncovered some sensitivity to the number of copies l used, and suggest that implementers need to pay special attention: too few or too many copies will decrease the decoding efficiency of the scheme.

Our technique based on solving simultaneous equations is more expensive (inverting a matrix is about $\mathcal{O}(n^3)$), but is very effective – it will decode documents even if no singleton copy is available. On the downside this technique is too sensitive to the presence of non-matching documents, and is therefore less practical.

Deriving analytic results about the recovery probabilities and the non-matching document tolerance of both schemes remains an open problem.

Acknowledgments. We are indebted to our colleagues Andrei Serjantov and Paul Syverson for insightful discussions about the decoding schemes and Anna Lysyanskaya for giving us early positive feedback on the feasibility of our schemes. We are also grateful to the organizers of the Dagstuhl seminar on "Anonymous Communication and its Applications", Shlomi Dolev, Rafail Ostrovsky and Andreas Pfitzmann as well as all the staff of the school for providing such a productive as well as pleasant environment.

References

1. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.
2. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1984.
3. Rafail Ostrovsky and William E. Skeith III. Private searching on streaming data. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.
4. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.
5. Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco - underconditional sender and recipient untraceability with computationally secure serviceability (abstract). In *EUROCRYPT*, page 690, 1989.

A Some simulation code

```
import random
import sets
from Numeric import *
from LinearAlgebra import *

def exp(numbin,matches,leng,eqsolve=False):
   # First fill the bins with numbers
   bins = {}
   bn = {}
   for i in range(matches):
      bn[i] = random.sample(xrange(numbin),leng)
      for j in bn[i]:
         try:
            bins[j] += [i]
         except:
            bins[j] = [i]
   # Now start removing!
   ret = [] # List to return
   Recovered = sets.Set()
```

```
while True:
   lone = [bins[i][0] for i in bins.keys() if len(bins[i])==1]
   s = sets.Set(lone)
   Recovered = Recovered.union(s)
   ret += [len(s)]
   # Stop if there is nothing to remove
   if len(s) == 0:
      break
   # Otherwise remove the lone ones
   for r in s:
       for rl in bn[r]:
          bins[rl] = [x for x in bins[rl] if x is not r]
# Calculate the number of equations...
u = matches - len(Recovered)
eq = len([bins[i] for i in bins.keys() if len(bins[i])>1])
if not eqsolve:
   return ret, sum(ret), u, eq
# Lets try to solve it by solving the equations...
if u > 0 and u <= eq: # A necessary condition to solve the problem
    matrix = []
    col = []
    fullbins = [bins[i] for i in bins.keys() if len(bins[i])>1]
    for b in fullbins:
       row = []
       for i in range(matches):
           if not i in Recovered:
              if i in b:
                 row += [1]
              else:
                 row += [0]
       assert len(row) == (matches - len(Recovered))
       # Now we are going to reduce the matrix in size
       if len(col) < u:
          col += [[sum(b)]]
          # So far we have fewer equations than unknowns so keep going.
          matrix += [row]
       else:
          # now we need to 'fold' the new equation in an older one!
          s = random.sample(xrange(u),1)[0] # choose an equation
          matrix[s] = [r+n for (r,n) in zip(matrix[s],row)] # Add the new row to an older one
          col[s] = [col[s][0] + sum(b)]
    try:
        m = array(matrix) # the equations
        mInv = inverse(m)
        c = array(col) # the vector of values
        z = matrixmultiply(mInv,c)
        return ret, sum(ret), u, eq
    except LinAlgError, e:
        print 'Singular matrix'
        print e
```

```
   return ret, sum(ret), u, eq

def Medexp(numbin,matches,leng, t):
   s = 0
   sq = 0
   sqnum = 0
   for i in range(t):
      (ret, sret, u, eq) = exp(numbin,matches,leng)
      s +=  sret / matches
      # here we can also estimate the recovery rates if we solved the equations
      # In particular we can tolerate a certain number of additonal documents
      tolerance = (float(eq) / numbin)**leng
      if not (tolerance == 1 or eq < u or tolerance == 0):
         d = float(eq - u) / tolerance
         sq += d
         sqnum += 1
   if sqnum == 0:
      sq = 0
   else:
      sq = float(sq)/ sqnum
   return float(s) / t, sq
```