Outsourced Bits

A research blog on cloud computing, cryptography, security, privacy, …

How to Search on Encrypted Data: Deterministic Encryption (Part 2)

This entry was posted on October 14, 2013, in Crypto Design, Encrypted Search, Privacy and tagged cloud storage, deterministic encryption, encrypted search, privacy, property-preserving encryption, search on encrypted data, searchable encryption. Bookmark the permalink. 15 Comments

(https://cloudcrypto.files.wordpress.com/2013/10/sse.jpg)

This is the second part of a series on searching on encrypted data. See part 1 (http://outsourcedbits.org/2013/10/06/how-to-search-on-encrypted-data-part-1/), 3 (http://outsourcedbits.org/2013/10/30/how-to-search-on-encrypted-data-part-3/) and 4 (http://outsourcedbits.org/2013/12/20/how-to-search-on-encrypted-data-part-4-oblivious-rams/).

In this post we'll cover the simplest way to search on encrypted data. This is usually the solution people come up with when they first think of the problem of encrypted search and, as we'll see, this approach has some nice properties but also some limitations.

To make this work we' ll need a special type of encryption scheme called a *property-preserving* encryption (PPE) scheme. PPE schemes encrypt messages in a way that leaks certain properties of the underlying message.

There are different types of PPE schemes that each leak different properties. The simplest form is *deterministic* encryption which always encrypts the same message to the same ciphertext [1]. The property preserved by deterministic encryption is *equality* since, given two encryptions

$$c_1 = E_K(m_1) \quad \text{and} \quad c_2 = E_K(m_2),$$

one can test if the underlying messages are equal by just checking if $c_1 = c_2$ [2]. More complex PPE schemes include order preserving encryption (http://www.cc.gatech.edu/~aboldyre/papers/bclo.pdf) (OPE) and orthogonality-preserving encryption (http://www.cs.utexas.edu/~jrous/documents/191.pdf).

The approach we'll describe here works with any PPE scheme but is easier to explain with deterministic encryption so that's what we'll use.

PPE-based encrypted search was first proposed in the Database community and later studied more formally in the Cryptography community. The first paper to provide a cryptographic treatment of this approach was a paper (http://eprint.iacr.org/2006/186.pdf) by Bellare, Boldyreva and O' Neill from 2006.

The authors formally study the notion of deterministic encryption and show how to apply it to the problem of encrypted search. While Bellare et al. were motivated in part by searching on encrypted data, the notion of deterministic encryption is interesting to cryptographers in its own right and the formal study of deterministic encryption that was initiated by this paper has led to other applications and deepened our understanding of encryption.

As mentioned above, another important type of PPE is order preserving encryption which was introduced by Agrawal, Kiernan, Srikant and Xu in this paper (http://rsrikant.com/papers/sigmod04.pdf) and studied more formally by Boldyreva, Chenette, Lee and O' Neill here (http://www.cc.gatech.edu/~aboldyre/papers/bclo.pdf) and here (http://www.cc.gatech.edu/~aboldyre/papers/operev.pdf).

The High-Level Idea

Suppose we have both a deterministic encryption scheme E^D and a standard CPA-secure (http://en.wikipedia.org/wiki/Ciphertext_indistinguishability) (i.e., randomized) encryption scheme E^R.

We can then create an encrypted database EDB as follows. For each document D_i in the collection $(D_1, \ldots, D_n)$, the client computes deterministic encryptions of each keyword of D_i. Assuming each document D_i has m keywords $(w_{i,1}, \ldots, w_{i,m})$, the EDB then simply consists of n tuples

$$r_i = (d_{i,1}, \ldots, d_{i,m}, ptr(c_i)),$$

where $d_{i,j} = E^D_{K_2}(w_{i,j})$, $c_i = E^R_{K_1}(D_i)$ and $ptr(c_i)$ is a pointer to ciphertext c_i.

Recall that in our setting, the client sends the encrypted database

$$EDB = (r_1, \ldots, r_n)$$

to the server along with the randomized encryptions of the documents $(c_1, \ldots, c_n)$.

To search for keyword w, the client just sends a deterministic encryption of w to the server. This encryption of the keyword, $d_w = E^D_{K_2}(w)$, will serve as the token. Now all the server has to do is compare d_w to all the deterministic encryptions in EDB. If d_w is equal to any of them, the server follows the corresponding pointer and returns the encrypted document. In other words, for all $1 \leq i \leq n$ and $1 \leq j \leq m$, the server tests if $d_w = d_{i,j}$ and if they are equal it follows $ptr(c_i)$ and returns c_i.

This clearly works but there is a limitation in the way the scheme is described: the search time for the server is $O(nm)$, i.e., linear in the number of documents (let's just assume m is very small here). Obviously, linear-time search is too slow for practice but in reality this is not a problem because we can just store the deterministic encryptions of EDB in data structures that support fast search (e.g., a binary search tree) so that search can be performed very quickly (e.g., in time $O(\log(n))$.

Is This Secure?

As we've seen, PPE-based solutions can achieve fast (i.e., sub-linear) server-side encrypted search. They do, however, have some limitations with respect to security as discussed and formalized by Bellare et al.

The first problem is that the encrypted database EDB leaks quite a bit of information to the server about the data collection—even before the client has performed any searches. In particular, recall that the keywords in EDB are encrypted using a deterministic encryption scheme so the same keyword w will always encrypt to the same ciphertext.

This means that if the server sees two or more equal ciphertexts in EDB it knows that the corresponding encrypted documents contain a keyword in common. In addition, the server learns the frequency with which keywords appear which makes the encrypted database vulnerable to frequency analysis (http://en.wikipedia.org/wiki/Frequency_analysis).

Another issue is that since tokens are deterministic encryptions of the search terms, the server will always know whether the client is repeating a search a not.

A third issue occurs when the deterministic encryption scheme (or any form of PPE scheme) is public-key. In this case, all the deterministic encryptions (both in EDB and in the tokens) are encrypted under the client's public key which is, obviously, public and available to the server. The server can then mount a dictionary attack on the encrypted database by encrypting a list of possible keywords and comparing them to the ones found in EDB and in the tokens. If it gets a match then it knows the keyword.

This attack clearly shows that public-key PPE-based solutions should probably not be used for "normal" data (e.g., text, emails, PII etc.). But does it mean we can't use it all? Not quite. As Bellare et al. observe in their paper, such a solution can be used when the data has high min-entropy, which is a way of saying that the data looks random to the server.

The problem of course is that it's not clear when this applies in practice. Also, note that even if the keywords do have high min-entropy, the other two issues still remain.

Conclusions

So we've seen one approach to searching on encrypted data based on property-preserving encryption. It results in a solution that supports fast search on encrypted data but, unfortunately, leaks quite a bit of information to the server. In the next post we'll go over another solution that provides a different tradeoff: it leaks less information to the server but achieves slower search time.

Notes

[1] In general it is a terrible idea to encrypt with deterministic encryption since we've known since this (http://theory.lcs.mit.edu/~cis/pubs/shafi/1984-jcss.pdf) 1984 paper by Goldwasser and Micali that any secure (http://en.wikipedia.org/wiki/Semantic_security) encryption scheme has to be randomized. The point here, however, is that we may be willing to use weaker primitives in order to design more functional encryption schemes.

[2] Technically, we do not need a deterministic *encryption* scheme since we'll never need to decrypt so (in the symmetric setting) one could use a pseudo-random function (http://en.wikipedia.org/wiki/Pseudorandom_function_family).

15 thoughts on "How to Search on Encrypted Data: Deterministic Encryption (Part 2)"

1. Pingback: How to Search on Encrypted Data (Part 1) | Outsourced Bits
2. Pingback: How to Search on Encrypted Data: Functional Encryption (Part 3) | Outsourced Bits
3. **Megha says:**
   November 16, 2013 at 9:52 pm
   During construction of EDB we are using key K2 to encrypt keyword but while searching for a keyword, we are encrypting it using K1 and then check. Is this typo or I understand it wrong?
   And what is K1 and K2 here ?

Reply

senykam says:
November 17, 2013 at 4:58 am
Fixed; thanks for pointing that out. Also, K1 and K2 are just keys chosen for the randomized and deterministic encryption schemes, respectively.

Reply

Megha says:
November 17, 2013 at 5:03 am
Ok.. Thanks for your reply.

4. Pingback: How to Search on Encrypted Data (Part 4): Oblivious RAMs | Outsourced Bits

5. **Marvin says:**
May 13, 2014 at 12:29 pm
Hi Seny, this article series is really great.

Is it possible to use deterministic encryption search for data of multiple users? I just want to check which users know the same secret, the content doesn't matter to me. But as I understand it, then I have to use the same salt, which makes it probably insecure. (The users don't know each other, so they cannot exchanges keys.)

Reply

senykam says:
May 19, 2014 at 3:15 pm
Hi Marvin,

Yes, you could do it with deterministic encryption. Usually I would advise not to use it but depending on how large the secret space is and how random the secrets are, it may be OK.

Reply

Marvin says:
May 26, 2014 at 11:14 am
Thanks. I changed my plan to use a deterministic salt, that should make it a little bit more secure. In my use case there is no prose text, only data like email addresses or names, therefore frequency analysis wouldn't leak much information.

senykam says:
May 26, 2014 at 4:29 pm
Actually it probably would. Email addresses and names don't have much entropy. I would strongly recommend to not roll out an encrypted search solution yet and would recommend waiting that a strong open source team team rolls out a good implementation. The technology is maturing in terms of research but unfortunately there aren't any open real-world implementations available yet.

Marvin says:
June 10, 2014 at 8:42 am
"would recommend waiting that a strong open source team team rolls out a good implementation. "

Oh okay, but as I understood there isn't any theoretical approach known yet for unlimited multiuser search, or is it? Therefore I could only implement a better-than-nothing solution which is tailored to my use case.

senykam says:
June 19, 2014 at 9:36 am
There are a few papers that consider multi-user search: http://eprint.iacr.org/2006/210.pdf (Section 6) and http://eprint.iacr.org/2013/720.pdf

6. Pingback: How to Search on Encrypted Data: Searchable Symmetric Encryption (Part 5) | Outsourced Bits

7. **Prasanna says:**
   December 10, 2014 at 10:01 am
   Sir,
   is it possible to use RSA algorithm to encrypt B tree index , and search for the keyword. if Yes plz let me know how we can search on RSA encrypted data.
   -Prasanna

   Reply
8. Pingback: Applied Crypto Highlights: Searchable Encryption with Ranked Results | Outsourced Bits