Outsourced Bits

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

How to Search on Encrypted Data: Searchable Symmetric Encryption (Part 5)

This entry was posted on August 21, 2014, in Crypto Design, Encrypted Search, Privacy and tagged cloud storage, encrypted search, privacy, search on encrypted data, searchable encryption. Bookmark the permalink. 24 Comments

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

This is the fifth part of a series on searching on encrypted data. See parts 1 (http://outsourcedbits.org/2013/10/06/how-to-search-on-encrypted-data-part-1/), 2 (http://outsourcedbits.org/2013/10/14/how-to-search-on-encrypted-data-part-2/) and 3 (http://outsourcedbits.org/2013/10/30/how-to-search-on-encrypted-data-part-3/) and 4 (http://outsourcedbits.org/2013/10/30/how-to-search-on-encrypted-data-part-4/).

In the previous post we covered the most secure way to search on encrypted data: oblivious RAMs (ORAM). I always recommend ORAM-based solutions for encrypted search whenever possible; namely, for small- to moderate-size data [1]. Of course, the main limitation of ORAM is efficiency so this motivates us to keep looking for additional approaches.

The solution I discuss in this post is *searchable symmetric encryption* (SSE). For readers who are not familiar with this area, let me stress that this has *nothing* to do with CipherCloud's searchable strong encryption (http://www.ciphercloud.com/company/about-ciphercloud/press-releases/ciphercloud-delivers-breakthrough-searchable-strong-encryption/). I don't know why CipherCloud chose to call its "breakthrough" product SSE. No one knows exactly what CipherCloud does at the crypto level but everything points to them using some form of tokenization which, as far as I know, is an industry term for deterministic encryption. This is neither a breakthrough nor really secure for that matter but that's the last thing I'll say about CipherCloud here so every reference to SSE that follows is about searchable symmetric encryption.

SSE was first introduced by Song, Wagner and Perrig [SWP00 (http://www.cs.berkeley.edu/~dawnsong/papers/se.pdf)] in 2001. SSE tries to achieve the best of all worlds. It is as efficient as the most efficient encrypted search solutions (e.g., deterministic encryption) but provides a lot more security.

The Security of Encrypted Search

One of the most interesting aspects of encrypted search from a research point of view has to do with security definitions; that is, what does it mean for an encrypted search solution to be secure? This is not an obvious question and I talked about this a bit in the previous post on ORAM (http://outsourcedbits.org/2013/12/20/how-to-search-on-encrypted-data-part-4-oblivious-rams/).

The first paper to explicitly address this question was an important paper by Eu-Jin Goh [Goh03] [2] who was a graduate student at Stanford at the time. This paper had many contributions but one of the most important ones was simply to point out that SSE schemes were not normal encryption schemes and, therefore, the standard notion of CPA-security was not meaningful/relevant for SSE. The problem is essentially that when an adversary interacts with an SSE scheme he has access to more than an encryption oracle; he also has access to a search oracle. Goh's point was that this had to be captured in the security definition otherwise it was meaningless.

To address this, he proposed the first security definition for SSE. Roughly speaking, the definition guaranteed that given an EDB and the encrypted documents, the adversary would learn nothing about the underlying documents beyond the search results *even if it had access to a search oracle*. Let me highlight a few things about Goh's definition: (1) it was a game-based definition; and (2) it did not provide query privacy (i.e., no privacy guarantees for user queries) [3]. A follow up paper by Chang and Mitzenmacher [CM05] proposed a new definition that was simulation-based and that guaranteed query privacy in addition to data privacy.

I won't go into details, but simulation-based definitions have some advantages over game-based definitions and, generally speaking, are preferable and can be easier to work with—especially when composing various primitives to build larger protocols. So we're done right? Not exactly.

During this time, Reza Curtmola, Juan Garay, Rafail Ostrovsky and myself were also thinking about SSE and one of the things we noticed while thinking about the security of SSE schemes was that the previous security definitions didn't seem to really capture what was going on. There were primarily two issues: (1) the definitions were (implicitly) restricting the adversary's power; and (2) they didn't explicitly capture the fact that the constructions were leaking information.

Adaptivity. The first problem was that in these definitions, the adversary was never given the search tokens, the EDB or the results of its searches. The implication of this was that—in the definition—the adversary could not choose its search oracle queries as a function of the EDB, the tokens or previous search results. In other words, it's behavior was being implicitly restricted to making *non-adaptive* queries to its search oracle. This was clearly an issue because in the real-world the adversary we are trying to protect against is a server that stores the EDB, that receives tokens from the client and that sees the results of the of the search. So if we allow this adversary to query a search oracle, then we also have to allow him to query the oracle as a function of the EDB, the tokens and previous search results. More concretely, this captures a form of attack where the server crafts some clever oracle queries based on the EDB, the tokens or previous search results.

Now let's take a step back. At this point—unless you are a cryptographer—you are likely thinking something to the effect of: "this sounds contrived and honestly I can't see how one could craft queries of this form that would lead to an actual attack of this form. This is all academic!". I know this because, unfortunately, I've heard this many times over the years.

But this is roughly the reaction people have every time cryptographers point out that an adversarial model needs to be strenghtened. Usually, what happens is the following: (1) non-cryptographers ignore this and build their systems using primitives that satisfy the weaker model because they don't believe the stronger attacks are realistic; (2) someone comes along and carries out some form of the stronger attack; and (3) the systems need to be re-designed and patched. This has happened in the cases of encryption (CPA- vs. CCA2-security) and key exchange.

In any case, having observed this, we wrote about it in the following paper [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] and proposed a new and stronger definition where the adversary was allowed to generate its queries as a function of the EDB, the tokens and previous search results. We called this *adaptive* security and gave two formulations of this definition: one game-based and one simulation-based. This turned out to be quite interesting from a theoretical point of view because the simulation-based formulations were slightly stronger than the game-based formulations; which is not the case for the standard notion of CPA-security [4].

Now, to be honest, I do not know of an explicit attack on a concrete SSE construction that takes advantage of adaptivity. But that shouldn't matter anymore because we now know how to construct adaptively-secure SSE schemes that are as efficient as non-adaptively-secure ones. So there is no excuse for not using an adaptively-secure scheme. Another important reason to consider adaptive security is for situations where SSE schemes are used as building blocks in larger protocols. In these kinds of situations, the primitive can be used in unorthodox ways which open up subtle new oracles that one may not have considered when designing the primitive for its more standard uses.

This exact issue comes up in a paper I wrote recently [K14] (http://research.microsoft.com/en-us/um/people/senyk/pubs/metacrypt.pdf) that combines structured encryption (which is a form of SSE) with secure multi-party computation to design a private alternative to the NSA metadata program. In this case, it turns out that the adversary for the larger protocol (i.e., the NSA analyst) can easily influence the inputs to the underlying SSE scheme and implicitly carry out adaptive attacks on it. So in this case, it is crucial that whatever structured encryption scheme is used be adaptively-secure.

Leakage. Another important issue that was overlooked in previous work was leakage. As I've discussed in previous posts, non-ORAM solutions leak some information. Everyone was basically aware that SSE revealed the search results (i.e., the identifiers of the documents that contained the keyword). This was the whole point of SSE and most people believed that this was why it was more efficient than ORAM [5]. But this was not treated appropriately. In addition, we also pointed in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] that all the known SSE constructions leaked more that the search results. In particular, they also revealed whether a search query was being repeated. This was very easy to see by just looking at the constructions: the search tokens were usually the output of a PRF applied to the keyword being searched for.

The main problem was that the definitions did not capture any of this [6]. To address it we decided to treat leakage in SSE more formally and to capture it very explicitly in our security definitions. Our thinking was that leakage was an integral part of SSE (since it seemed to be one of the reasons why SSE was so efficient) and that it deserved to be properly studied and understood. At this stage we only really considered two types of leakage: the access pattern and the search pattern. The access pattern is basically the search results (the identifiers of the documents that contain the keyword) and the search pattern is whether a search query is repeated. At the time these were the only leakages that had appeared in the literature. In a later paper with Melissa Chase [CK10 (http://eprint.iacr.org/2011/010.pdf)], we generalized the definitional approach of [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] so that the definition could include *any* kind of leakage.

Leakage is of course undesirable from a security point of view, but it is fascinating from a research point of view. I hope to discuss this further in later posts. For the purposes of this discussion, I'll just point out that there are (mostly) two kinds of leakages: setup leakage, which is revealed just by the EDB; and query leakage, which is revealed by a combination of the EDB and a token. One of the main issues with any solution based on deterministic encryption or, more generally, on property-preserving encryption is that they have a high degree of setup leakage: their EDB's have non-trivial leakage. In that sense, SSE-based solutions are better because their setup leakage is usually minimal/trivial and the non-trivial leakage is only query leakage which is controlled by the client since queries can only be executed with knowledge of the secret key.

Summing up. So in the end, what we tried to argue in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] was that what we should be asking for from an SSE security definition is a guarantee that:

> The adversary cannot learn anything about the data and the queries beyond the explicitly allowed leakage; even if the adversary can make adaptive queries to a search oracle.

But once we settled on this definition and formalized it, the following natural problems came up: (1) how do we distinguish between reasonable and unreasonable leakage?; and (2) is it even possible to design SSE schemes that are adaptively-secure? [5]

Initially, the answers to these questions weren't obvious to us. We thought about them for a while and eventually answered the second question by finding an SSE construction that was adaptively-secure. Unfortunately, while the scheme had optimal asymptotic search complexity, it was not really practical. But at least we knew adaptive security was achievable—though we did not know whether it was achievable efficiently.

We didn't really have any answer for the first question. In fact, we still don't. We don't really have a good way to understand and analyze the leakage of SSE schemes. For now, the best we can do is to try and describe it precisely.

Searchable Symmetric Encryption

There are many variants of SSE (see this paper [CK10 (http://eprint.iacr.org/2011/010.pdf)] for a discussion) including interactive schemes, where the search operation is interactive (i.e., a two-party protocol); and response-hiding schemes, where search results are not revealed to the server but only to the client. I'll focus on non-interactive and response-revealing schemes here because they were the first kind of SSE schemes considered and also because they are very useful as building blocks for more complex constructions and protocols. It also happens that they are the most difficult to construct.

In our formulation we will ignore the document collection itself and just assume that the individual documents are encrypted using some symmetric encryption scheme and that the documents each have a unique identifier that is independent of their content (so that knowing the identifier reveals nothing about a file's contents).

We assume that the client processes the data collection $\mathbf{D} = (D_1, \ldots, D_n)$ and sets up a "database" DB that maps every keyword w in the collection to the identifiers of the documents that contain it. Recall that in our context, we use the term database loosely to refer to a data structure optimized for keyword search (i.e., a search structure). For a keyword w, we'll write $\mathrm{DB}[w]$ to refer to the list of identifiers of documents that contain w.

A non-interactive and response-revealing SSE scheme $(\mathrm{Setup}, \mathrm{Token}, \mathrm{Search})$ consists of

- a Setup algorithm run by the client that takes as input a security parameter 1^k and a database DB; it returns a secret key K and an encrypted database EDB;
- a Token algorithm also run by the client that takes as input a secret key K and a keyword w; it returns a token tk;
- a Search algorithm run by the server that takes as input an encrypted database EDB and a token tk; it returns a set of identifiers $\mathrm{DB}[w]$.In addition to security, of course, the most important thing we want from an SSE solution is low search complexity. Fast, for our purposes will mean *sub-linear* in the number of documents and, ideally, linear in the number of documents that contain the search term. Note that the latter is optimal since at a minimum the server needs to fetch the relevant documents just to return them.Requiring sub-linear search complexity is *crucial* for practical purposes. Unless you are working with a very small dataset, linear search is just not realistic—try to imagine if your desktop search application or email search function did sequential search over your hard drive or email collection *every time you searched*. Or if your favorite search engine sequentially scanned the entire Web every time you performed a web search [7].The sub-linear requirement has consequences, however. In particular it means that we must be willing to work in a offline/online setting where we run a one-time (linear) pre-processing phase to setup a search structure so that we can then execute search queries on the data structure in sub-linear time. And this is exactly the approach we'll take.
  The Inverted Index Solution

  The particular solution I describe here is referred to as the *inverted index solution* and was proposed in the same [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] paper in which we studied the security of encrypted search.

  This is a good construction to understand for several reasons: (1) it is the basis of almost all subsequent SSE constructions; and (2) many of the tricks and techniques that are used in recent SSE schemes (and the more general setting of structured encryption) originated in this construction.

Setup. The scheme makes use of a symmetric encryption scheme $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$, of a pseudo-random function (PRF) $F : \{0,1\}^k \times W \rightarrow \{0,1\}^k$ and of a pseudo-random permutation (PRP) $P : \{0,1\}^k \times W \rightarrow \{1, \ldots, |W|\}$. To setup the EDB, the client first samples two k-bit keys K_{T} and K_{R} for F and P, respectively. It then creates two arrays T and RAM_1. For all keywords $w \in W$, the client builds a list for $\mathrm{DB}[w]$ and stores the nodes in RAM_1. More precisely, for every keyword $w \in W$ and every $1 \leq i \leq |\mathrm{DB}[w]|$, it stores

$$\mathrm{N}_{w,i} = \left\langle \mathrm{id}_{w,i}, \mathrm{ptr}_1(w, i+1) \right\rangle$$

in RAM_1, where $\mathrm{id}_{w,i}$ is the i th identifier in $\mathrm{DB}[w]$ and $\mathrm{ptr}_1(w, i+1)$ is the address (in RAM_1) of the $(i+1)$ th identifier in $\mathrm{DB}[w]$. Of course, $\mathrm{ptr}_1(w, |\mathrm{DB}[w]|+1) = \perp$.

It then randomly permutes the locations of the nodes; that is, it creates a new array RAM_2 stores all the nodes in RAM_1 but at locations chosen uniformly at random and with appropriately updated pointers.

After this shuffling step, the client encrypts each node in RAM_2; that is, it creates a new array RAM_3 such that for all $w \in W$ and all $1 \leq i \leq |\mathrm{DB}[w]|$,

$$\mathrm{RAM}_3\left[\mathrm{addr}_2(\mathrm{N}_{w,i})\right] = \mathrm{Enc}_{K_w}\left(\mathrm{RAM}_2\left[\mathrm{addr}_2(\mathrm{N}_{w,i})\right]\right)$$

where $K_w = F_{K_{\mathrm{R}}}(w)$ and addr_2 is just a function that maps nodes to their location in RAM_2 (this just makes notation easier). Now, for all keywords $w \in W$, the client sets

$$\mathrm{T}\big[P_{K_{\mathrm{T}}}(w)\big] = \mathrm{Enc}_{K_w}\big(\mathrm{addr}_3(\mathrm{N}_{w,1})\big),$$

where addr_3 is a function that maps nodes to their locations in RAM_3. Finally, the client sets

$\mathrm{EDB} = (\mathrm{T}, \mathrm{RAM}_3)$.

Now the version I just described is simpler than the one presented in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)]. There are two main differences. The first has to do with the domain of the pseudo-random permutation P. In practice, PRPs have a fixed domain size. For example, if we view AES as a PRP then it is a PRP that maps 128-bit strings to 128-bit strings. But in our case we need a PRP that maps keywords in W to the numbers 1 through $|W|$. The problem here is that in practice the size of W will be *much* smaller than 2^{128}. So the question becomes how we can use a PRP built for a large domain to build a PRP for a small domain? There are ways of doing this but at the time the known solutions had several important limitations. So we solved the problem using the following approach.

Suppose we used a large-domain PRP. The problem would be that the table T would be large as well, i.e., it would have to hold 2^{128} elements if we were using a PRP over 128-bit strings (e.g., AES). Obviously this is too large to be practical. So the idea was to “shrink” T by using something called a Fredman-Komlos-Szemeredi (FKS) table. I won’t go into the details, but the point is that by using FKS tables, we could use a large-domain PRP and still have a compact table T.

The other difference has to do with the symmetric encryption scheme $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ that we use. In the version described here, it is important for security that the encryption scheme be *anonymous* which means that, given two ciphertexts, one cannot tell whether they were encrypted under the same key or not. Why is this important? Because each list of nodes $\{\mathrm{N}_{w,i}\}_{i\leq|\mathrm{DB}[w]|}$ is encrypted under the same key K_w. And if, given RAM_3, the adversary can tell which ciphertexts are encrypted under the same key, then it can learn the frequency $|\mathrm{DB}[w]|$ of each keyword. Note that this would be revealed by the EDB; without the client ever having made any queries.

The problem with anonymity is that it is not implied by the standard notion of CPA-security. In practice, it seems that most block ciphers (including AES) would be anonymous but again maybe not. In [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] we didn't assume that the underlying symmetric encryption scheme was anonymous so we had to use a different approach. At a high-level, what we did is to encrypt each node under a different key and store that key in its predecessor in the list. The fact that every node is encrypted under a different key solves our problem.

Token and search. If the client wants to search for keyword w , he simply generates a token

$$\mathrm{tk} = (\mathrm{tk}_1, \mathrm{tk}_2) = (P_{K_{\mathrm{T}}}(w), F_{K_{\mathrm{R}}}(w)),$$

which he sends to the server. To query $\mathrm{EDB} = (\mathrm{T}, \mathrm{RAM}_3)$, the server first recovers the ciphertext $c = \mathrm{T}[\mathrm{tk}_1]$ which it decrypts to recover address $a_1 = \mathrm{Dec}_{\mathrm{tk}_2}(c)$. Then, for all i until $a_i = \perp$, it decrypts the nodes $(\mathrm{N}_{w,1}, \ldots, \mathrm{N}_{w,|\mathrm{DB}[w]|})$ by computing

$$(\mathrm{id}_i, a_{i+1}) \leftarrow \mathrm{Dec}_{K_{\mathrm{R}}}(\mathrm{RAM}_3[a_i]).$$

It then finds and returns the encrypted documents with identifiers $(\mathrm{id}_1, \ldots, \mathrm{id}_{|\mathrm{DB}[w]|})$.

Efficiency and security. To search, the server needs to do one lookup in T , which is $O(1)$ and then one decryption for each node $(\mathrm{N}_{w,1}, \ldots, \mathrm{N}_{w,|\mathrm{DB}[w]|})$, which is $O(|\mathrm{DB}[w]|)$. So the search complexity of this approach is $O(|\mathrm{DB}[w]|)$, which is optimal since it would take at least that much time just for the server to send back the relevant documents.

The construction is clearly efficient (asymptotically speaking, as efficient as possible) but is it secure? Yes and no. The security of the solution (at least the more complex version) is proved secure in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] but it is only shown to be *non-adaptively-secure* with trivial setup leakage and query leakage that includes the access pattern (the search results) and the search pattern (whether a query is repeated).

Intuitively, given $\mathrm{EDB} = (\mathrm{T}, \mathrm{RAM}_3)$ the adversary learns at most the number of keywords (by the size of T) and $\sum_{w \in W} |\mathrm{DB}[w]|$ by the size of RAM_3. So that is the setup leakage. Notice that unlike solutions based on deterministic encryption, the EDB by itself does not leak any non-trivial information like the frequency of a keyword. At query time, the server obviously learns the search results $\mathrm{DB}[w]$ but it also learns whether the client is repeating a keyword search since in that case the tokens $\mathrm{tk} = (P_{K_\mathrm{T}}(w), F_{K_\mathbb{R}}(w))$ will be the same.

Improvements. The inverted index solution has been improved over several works. Its main limitations were that: (1) it was only non-adaptively secure; (2) the use of FKS dictionaries made the solution hard to understand and implement; and (3) it was a static scheme, in the sense that one could not modify the EDB to add or remove keywords and/or document identifiers [8].

The first problem was addressed in a joint paper with my MSR colleague Melissa Chase [CK10 (http://eprint.iacr.org/2011/010.pdf)]. One of the observations in that work was that the inverted index solution could be made adaptively-secure by replacing the symmetric encryption scheme by a non-committing encryption scheme. Non-committing encryption schemes are usually either very expensive or require very strong assumptions (i.e., random oracles). Fortunately, in our setting we only need a *symmetric* non-committing encryption scheme and such a scheme can be instantiated very efficiently. In fact, it turns out that the simplest possible symmetric encryption scheme is non-committing! In retrospect this is a very simple observation, but it’s been a very useful one since it allows us to design adaptively-secure schemes very efficiently (and under standard assumptions). In fact, this has been used in most subsequent SSE constructions.

The second issue was also addressed in [CK10 (http://eprint.iacr.org/2011/010.pdf)]. Obviously one could just replace the PRP with a small-domain PRP but the approach taken in [CK10 (http://eprint.iacr.org/2011/010.pdf)] was different. The idea is to replace the array T with a dictionary DX. A dictionary is a data structure that stores label/value pairs and that supports lookup operations that map labels to their values. Dictionaries can be instantiated as hash tables, binary search trees etc. So instead of populating T with

$$\mathrm{T}\left[P_{K_\mathrm{T}}(w)\right] = \mathrm{Enc}_{K_w}\left(\mathrm{addr}_3(\mathrm{N}_{w,1})\right)$$

for all $w \in W$, we instead use a PRF G and store the pair

$$\left(G_{K_T}(w), \mathrm{Enc}_{K_w}(\mathrm{addr}_3(\mathrm{N}_{w,1}))\right)$$

in DX for all $w \in W$. With this approach we remove the need for a PRP altogether and, in turn, the need for either small-domain PRPs or FKS dictionaries.

The third issue was addressed in a joint paper with Charalampos (Babis) Papamanthou who was an MSR intern at the time and Tom Roeder who was an MSR colleague at the time. In this paper [KPR12 (http://eprint.iacr.org/2012/530.pdf)], we show how to make the inverted index solution dynamic while maintaining its efficiency. The solution is complex so I won't discuss it here.

In another paper with Babis [KP13 (https://research.microsoft.com/en-us/um/people/senyk/pubs/psse.pdf)] we propose a much simpler dynamic solution. Our approach here is tree-based and not based on the inverted index solution at all. It's search complexity, however, is not optimal but sub-linear; in particular, it incurs a logarithmic (in the number of documents) factor over optimal. It has other good properties, however, like parallizable search and good I/O complexity.

In a more recent paper [CJJJKSR14 (http://www.internetsociety.org/sites/default/files/07_4_1.pdf)], Cash, Jarecki, Jaeger, Jutla, Krawczyk, Steiner and Rosu describe a dynamic solution that is very simple, has optimal and parallelizable search and has good I/O complexity.

In another recent paper [NPG14 (http://web.engr.illinois.edu/~naveed2/pub/Oakland2014BlindStorage.pdf)], Naveed, Prabakharan and Gunther propose a very interesting dynamic solution based on the notion of blind storage. In a way, their notion of blind storage can be viewed as an abstraction of the RAM_3 structure in the inverted index solution. What [NPG14 (http://web.engr.illinois.edu/~naveed2/pub/Oakland2014BlindStorage.pdf)] shows, however, is that there is an alternative—and much better—way of achieving the properties needed from RAM_3 than how it is done in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)]. I won't say much else because this really gets into the weeds of SSE techniques but I recommend the paper if you're interested in this area (see this talk (http://research.microsoft.com/apps/video/default.aspx?id=212229) by Naveed for more details)

Finally, the last paper I'll mention is a work by Cash, Jarecki, Jutla, Krawczyk, Rosu and Steiner [CJJKRS13 (http://eprint.iacr.org/2013/169)] that shows how to extend the inverted index solution to handle *boolean* queries while keeping its optimal search complexity. Prior to this work we knew how to handle conjunctive search queries (i.e., $w_1 \wedge w_2$) in linear time. This paper showed not only how to do it in optimal time but also showed how to handle disjunctive queries (i.e., $w_1 \vee w_2$) and combinations of conjunctions and disjunctions!

Update: Two very nice resources on SSE I should have mentioned in the original post are a great blog post (http://bristolcrypto.blogspot.com/2013/11/how-to-search-on-encrypted-data-in.html) by Peter Scholl from University of Bristol and an incredibly comprehensive survey (http://eprints.eemcs.utwente.nl/24788/01/a18-bosch.pdf) by Bosch, Hartel, Jonker and Peter from University of Twente.

Notes

[1] I discuss how to use ORAM for encrypted search towards the end of the previous post of this series.

[2] Amazingly, this paper was never accepted for publication; which tells you something about the current state of our publication process. [3] This wasn't an omission on Goh's part; he defined it this way on purpose. His reasoning was that SSE schemes could have a variety of applications where token privacy was not needed. This made sense but it still left open the question of how one should define security with token privacy.

[4] A similar situation was later observed by Boneh, Sahai and Waters and O' Neill in the setting of functional encryption.

[5] Technically, this is *not* true! The reason SSE schemes tend to be more efficient than ORAM is not because they reveal the search results (access pattern) but because they reveal whether searches were repeated (search pattern).

[6] At this point you might be wondering how the proofs went through. In the definition of [Goh03], the tokens did not appear at all since he was not considering query privacy. In the case of [CM05], the adversary in the proof is restricted to never repeating queries.

[7] A criticism I often hear from colleagues and reviewers is that SSE constructions are not really *searching* over data. The underlying issue is that no computation is being performed. In my opinion, this reflects a very uninformed understanding of the real world. Given the amounts of data we currently produce and have to search over, search has become analogous to *sub-linear-time search* and therefore to some form of indexed-based search. In other words, the kind of scale we now have to deal with has fundamentally changed what we mean by the term search.

[8] Actually, in [CGKO06 (http://eprint.iacr.org/2006/210.pdf)] we describe a way to make our constructions (and any other) dynamic. There are limitations to this approach, however, including the tokens growing in length with the number of updates and interaction. So when we ask for a dynamic SSE scheme we typically want the update process not to affect the token size and, preferably, the update mechanism to be non-interactive —though the latter doesn't matter much from a practical point of view.

24 thoughts on "How to Search on Encrypted Data: Searchable Symmetric Encryption (Part 5)"

1. Pingback: Interesting technology posts this week Oct 2, 2014 | What Would Dan Do?

2. **Pravin Kothari says:**
   October 6, 2014 at 12:22 pm
   CipherCloud actually does 'Strong Searchable Encryption' by encrypting all data with a randomized encryption but also indexing a plaintext->ciphertext mapping using standard key-value data stores like Elastic Search. They can then store the encrypted data in a cloud service (e.g. Salesforce). When a user wants to 'search' the encrypted data in the cloud service, CipherCloud's reverse proxy instead issues a search to the plaintext-

>ciphertext mapping. Typically, this is stored on a CipherCloud customer's premise. It finds all (randomized) ciphertexts that match the query, then it constructs a big long search query (SOQL in the case of Salesforce) to fetch every value which matches the query. It's clever software engineering but bears little resemblance to real searchable encryption techniques.

Reply

senykam says:
October 7, 2014 at 9:14 am
Thanks for the details Pravin! It's hard to figure out what's going on with many of the startups in this space. I may be missing something but what exactly is the value proposition for clients if they have to run and maintain a CipherCloud proxy on premise? With this approach the client is trading off having to manage its data on premise to having to manage the "mapping/proxy" on premise. What do they gain exactly?

Reply

Pravin Kothari says:
October 7, 2014 at 1:42 pm
Basically nothing; it's really a question of degree. Ciphercloud would probably argue that it requires fewer resources to manage the on-prem proxy (with or without local search index) than an entire CRM application. Though, their deployments are still quite complex; they typically require hundreds of hours of (billable) professional services work. And that's just the standard proxy – if you want tokenization (e.g. for PCI compliance) that's another cluster that needs to be managed.

3. **senykam says:**
October 8, 2014 at 9:06 am
Interesting. For this to make any kind of sense, running a CRM on premise (including the required consulting services) should be more expensive than running a CRM in the cloud + a CipherCloud proxy (including the required consulting services).

Reply

4. **biswatosh says:**
October 10, 2014 at 2:07 am
Hi Pravin, Are you the same Pravin Kothari of Cipher Cloud–the founder?

Reply

Pravin Kothari says:
October 13, 2014 at 11:57 am
I can neither confirm nor deny this.

Reply

5. **blf says:**
November 7, 2014 at 10:27 am
All SSE schemes talked about here seem to, when given a query, return a full list of all documents containing the keyword(s) in the query. However, in modern search engines results come usually ordered by some relevance ranking. Aren't there any provably-secure SSE works giving this functionality? If not, why hasn't the security research community tried to tackle this problem, in your opinion?

Reply

senykam says:
November 7, 2014 at 10:21 pm

There are a few papers out there that try to do this and there are more coming. Most of these papers focus on text documents so the ranking is of the TF/IDF style.

With respect to web search engine rankings, this paper: http://eprint.iacr.org/2011/010.pdf shows how to encrypt web graphs in such a way to support focused subgraph queries. Based on this, one can run Jon Kleinberg's HITS algorithm to get a ranking.

Reply

6. Pingback: How to securely store SSN details in a database while maintaining searches? - DL-UAT

7. **sairalouise says:**
January 19, 2015 at 4:30 am
I have noticed that in general PEKS schemes use secure channels between the user and the server to transport the search tokens to prevent an adversary who cannot derive search tokens themselves from intercepting them and learning the search results. Does SSE prevent against this? Is a secure channel assumed? Or does the security model for SSE allow for any adversary to intercept a search token and learn the corresponding search results?

Reply

senykam says:
January 23, 2015 at 11:45 am
This is a great question. As you point out, in the PEKS setting we need a secure channel because the tokens are not hiding and do not protect the queries. So by using a secure channel we can at least protect the queries from a simple eavesdropper even if we can't achieve that against the server itself.

In the symmetric case the tokens offer more protection so in some sense there is less of a need for a secure channel. That being said, it is important to realize that the tokens do leak some information: namely whether two queries are for the same keyword or not; so there is some benefit to using a secure channel. The question boils down to whether you care about revealing the equality of keywords to an eavesdropper.

In practice I would always recommend to just use a secure channel since you never know and the cost of using one isn't that great.

Reply

8. Pingback: How to securely store SSN details in a database while maintaining searches? | Question and Answer

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

10. **Rik says:**
November 3, 2015 at 12:54 pm
Sorry for being late to the party, but how would this work with changing databases? SSE allows for searching in the EBD, but how would one add documents to the EDB when it is already deployed. Would it involve decrypting the entire EDB?

Reply

senykam says:
November 5, 2015 at 9:46 pm
Hi Rik. Great question. There are SSE schemes that can handle updates (without decrypting the entire EDB). They are typically referred to as "Dynamic SSE schemes". A particularly nice one is by Cash et al.: http://eprint.iacr.org/2014/853

Reply

11. **fastportal says:**
November 6, 2015 at 1:45 am
Thanks! Are there any implementations or libraries that we can have a look at?

Reply

senykam says:
November 6, 2015 at 11:41 am
As far as I know, there aren't any publicly available implementations available (all the ones I know of are proprietary and there are no plans on releasing them) but I hope that will change soon.

Reply

12. **blf says:**
November 6, 2015 at 2:31 pm
I have a C++ implementation of that scheme as I'm building on it to propose something new. I want to open source it in the future, but it's still a scientific prototype, not exactly production level code 😟

Reply

13. **maxpmagee says:**
December 16, 2015 at 12:20 pm
I think you have a typo in the following paragraph:
"We didn't really have any answer for the second question. In fact, we still don't. We don't really have a good way to understand and analyze the leakage of SSE schemes. For now, the best we can do is to try and describe it precisely."

You had just discussed said you found an SSE implementation that was adaptively-secure (the second question), and I believe you are referring to the first question (leakage) in this paragraph.

Reply

senykam says:
December 18, 2015 at 7:46 am
Fixed, thanks!

Reply

14. **Jidu says:**
December 17, 2015 at 2:53 am
Hi Kamara…thanks for this nice blog on your work on SSE……I have a question regarding the following quote

"Notice that unlike solutions based on deterministic encryption, the {\mathrm{EDB}} by itself does not leak any non-trivial information like the frequency of a keyword. At query time, the server obviously learns the search results {\mathrm{DB}[w]} but it also learns whether the client is repeating a keyword search since in that case the tokens {\mathrm{tk} = (P_{K_\mathrm{T}}(w), F_{K_{\mathbb R}}(w))} will be the same."

Does that mean after queries the above described SSE will reveal the frequency?
I guess the assumption that a database will be queried once is not practical at all. Adding dummy documents will hide the frequency after queries at the cost of large storage at the server. Right?

Reply

senykam says:

December 18, 2015 at 7:51 am

No one assumes it will queried only once. Every time it is queried it will reveal more information and this is captured in the definitions and in the analysis. The point, however, is that this is still considerably less than what is leaked by other approaches; especially some approaches based on PPE (i.e., deterministic and order-preserving encryption) which leak (more) information even if no queries are ever made.

Reply

15. **waleedalgomyli says:**

January 18, 2016 at 9:39 am

Hi, can you explain in example what does “query predicate P() ” mean in searchable encryption.

Reply

Blog at WordPress.com. WPExplorer.