**4.0 Research objectives: There are 3 stated objectives with regard to:**

**4.1) secure indexes**

**4.2) relevancy metrics (although the section refers to accuracy, not relevance -- needs to be consistent language)**

**4.3) Information leak mitigation.**

**The Conclusion needs to summarize all three objectives and demonstrate how the research objectives were successfully achieved.**

I’ve attached a “Conclusion” document to this email. Since I’m not sure which document you read, I first want to make sure you are referring to this conclusion section.

2.3 Inefficient solution (is this the same thing as a problem statement?)

No, I don’t think I’d call this the problem statement. Encrypted Search is in general the answer to that problem, but I didn’t event Encrypted Search. I’m only trying to contribute to it in three specific ways.

1. Existing solutions are slow, especially if they use the Bloom filter. This was initially the primary reason I proposed a new type of secure index based on what I call the perfect filter, since it only requires a constant number of hash functions per trapdoor regardless of the size of the document. I show how the Bloom filter’s latency grows with document size (assuming a fixed size block), and show that it very quickly even for modest corpus sizes becomes slower than 1 second.
2. Existing solutions don’t seriously consider an adversary who has access to a history of queries. So, I set out to simulate an adversary that does have access to such a history, and it turns out the adversary was quite successful (100% accuracy can be expected, given enough queries). This motivated the design, implementation, analysis, and experimentation of mitigation techniques like obfuscations and multiple secrets.
3. Once an adversary can query a secure index (e.g., see point 2, above), it can try to reconstruct the documents. I provide an analysis on how these kind of attacks can work, especially if we want to support proximity-sensitive searching (at that point it’s like the game of Jeopardy). I don’t simulate an attacker implementing any of the strategies I outlined, but I was hopeful my theoretical analysis was sufficient justification for me having designed and implemented various mitigation strategies, e.g., secure index poisoning (as other authors refer to it as), and also the desirability of high false positive rates, assuming search accuracy doesn’t degrade to unacceptable levels. (Experiments confirm good search accuracy is maintained for rank-ordered search, not so much for Boolean search.)

So, I could turn that into a problem statement – although I wonder if think it’s proper place is in the research objectives? And some of that is already in the research objectives, but maybe not explicitly enough?

**I don't buy the argument that the problem is energy costs (especially when you're testing this on a PC, rather than a server cluster). The problem is it's slow, right? (But this begs the question, is your solution faster?)**

Yes, the main problem is, it’s slow. I only included energy costs because decryption is computationally expensive, and if they must decrypt an entire corpus, that seems significant, especially on a battery powered devise, like a smartphone. My Encrypted Search scheme doesn’t use reversible decryption—the user just needs to hash the query terms of interest. And the CSP’s energy use isn’t high either, since its not decrypting anything, just using non-cryptographic hashes. But, since I don’t provide any analysis or experimentation to back this up, do you think I should remove it? Maybe it just seems like fluff?

**Abstract concerns I have:**

**• Who is this really a problem for?**

**• Business users? No, because if the data is so confidential that searches need to be encrypted it**

**would never be stored on a CSP due to security compliance controls.**

Ok. I’ll defer to your expertise here.

Perhaps this has a larger motivation for individuals or small groups.

However, as a thought experiment, let’s suppose the following:

With probability p, all the CSP ever sees over the life-time of search activities is indistinguishable from noise; that is, the confidential documents are indistinguishable from noise, the queries are indistinguishable from noise, and the retrieved results are indistinguishable from noise. However, the CSP is able to perform the retrieval operations anyway. At what value of p would this be sufficient for a business? If p=1, that is, confidentiality is guaranteed (which is not possible in any system), I would argue there is no reason why a security compliance control should prevent its use. So the question is, what must be greater than? 0.999999999?  
  
Also, a business may have a large collection of confidential documents, and wish to allow authorized users to search over those documents, but still not adequately trust their own servers – e.g., the administrators who manage the hardware/software aren’t completely trusted. If this is the case, then Encrypted Search (a better name is Oblivious Search, IMO) may be enticing.

**• Military/government users? Are there scenarios where you don't want to leave any trace of what**

**you searched for? It seems to me like the inability to audit with no trail is an even bigger problem, not a solution. Ultimately, aren’t the only people that want to cover their tracks criminals?**

Like any technology, it has both good uses and bad uses. This would undoubtedly be desirable for criminal activity, but it would also be desirable for people who live in a repressive state, like North Korea, so that they can feel safer searching for information that the government may not want them searching for. This is not solved if, at any point in time, the communications is decrypted on an untrusted system, e.g., SSL only encrypts the channel, but the receiving end has access to the plaintext.

**• Is the problem still a problem if the entire network circuit is encrypted such as over a VPN? Or are you claiming a defense-in-depth strategy? Is the "adversary" an internal employee, an external criminal, or a nosy Sysadmin at the CSP? Perhaps the assumption is that the adversary has internal access to data on the network already.**

Yes, Encrypted Search, as I have been describing it, is intended to be used on an untrusted CSP. SSL takes care of eavesdroppers between you and the CSP, but the intention is to prevent even the CSP (e.g., noisy system admin or maybe the CSP has been compromised) from knowing what’s going on. This is why I refer to the CSP as untrusted in the document.

**You might be able to avoid these abstract concerns by re-framing the problem: The real problem isn't the need for secure search itself, nor the inefficiency of encrypting an entire collection of documents.**

**The problem is with the current solution of using Bloom filter-based search indices.**

I’m eager to engage with you more on this point after you read some of my responses above. I certainly think you’ve raised legitimate points.

**Suggestion: Simply cite Google and Bing's conversion to encrypted search in 2014 as evidence of the importance of secure search. I think you have to at least mention this trend somewhere. Then briefly discuss why SSL encryption is not sufficient and easily broken.**

That sounds like a good idea.

**Then discuss the current solution of Bloom filter-based search indices and the disadvantages of the current solutions.**

Maybe I didn’t do this enough in the paper. I think the main disadvantage is the fact that the Bloom filter requires hash functions per block per document, that is, it’s slow. Note: is false positive rate.

Another disadvantage of the Bloom filter is that it doesn’t map members to convenient integers like the Perfect filter does, i.e., use the unique integer from the Perfect filter to map into other data structures that provide location or frequency info as we do in PSIB, PSIF, PSIP, and PSIM. For instance, in PSIM, we map each pair of words in the doc to a minimum pairwise distance for proximity info (without disclosing locations at all).

**Is there a call for more research on non-Bloom filters that you could cite? (Or maybe I missed this in the text.)**

Nope.

**4.1.3.1 Are you merely "proposing" a secure index based on the Perfect filter (which implies it's a pre-**

**existing solution) or did you "design and implement new types of secure indexes" (as you state in the conclusion) as an improvement over Bloom-filter based secure indices? The latter seems like a much stronger position to take.**

I designed and implemented the Perfect filter; it is not based on a pre-existing solution. I looked around for something like it to give it an appropriate name, but I didn’t find anything resembling it so I named it myself—Perfect filter. It’s just a combination of “perfect hash” and “filter” of course. Anyway, since I didn’t consider it to be that novel I’m not sure I want to draw attention to it being a new idea. I wouldn’t be surprised in the least to find prior art here.

**4.1.3.1.4 The disadvantages of the Perfect filter seem serious to me. Mitigation of these stated limitations may need to be discussed more, but I'm not sure where.**

I’m not sure how serious this disadvantage is. I do provide some analysis in the document confidentiality section on compromising the secure index, but they mostly rely upon dictionary-style attacks since a substitution cipher is not practical against it (e.g., if each member maps to 7 bits, i.e., there are only 128 possible patterns of 1’s and 0’s, but there are, assuming strings are bits, up to strings may be represented by it. If you know a secret, you can search through this entire space and find approximate candidates, but even if this could be done quickly (exponential time complexity), you still have a huge number of candidates, many of which are probable. So, I’m not sure how much of a problem it is. Note that N is actually unbounded, but a reasonable bound may be 10 characters, in which case if we store only alphabetic (we actually store alphanumeric) data, that is 7.8125 × 10^23 candidates per member.  
  
The most reasonable strategy I came up with was a dictionary attack, using a language model to pick out the most likely candidates, but this approach works the same for both the Bloom filter and the Perfect filter.

**5.3 All the simulations were run on an AMD PC. Why not cell or laptop or tablet or high-end workstation or server cluster? How would the platform impact the results especially with regard to speed? Is p. 72 with "core count" the only place this issue is addressed?**

First, I should say that I had intended to include multiple platforms in my experiments, but complications – namely, my first round of experimental data being unusable due to flaws in my implementation that were subsequently fixed – prevented that. (I had used the machines at two different labs at SIUE for my first round of experiments.) I was running out of time, so I didn’t make this a priority. I’m not convinced this is much of a problem though. First, most experiments were completely independent of the processor speed/core count, so only the experiments including lag measurements would be effected. I don’t think there’s any question that throwing more cores at it would cause a linear speed up, since the problem is “embarrassingly parallel” in that each secure index is an independent data structure that can be independently queried. So, I didn’t think it was important to show how the lag would respond to core count. And, as for sequential processor performance, while I would be curious to see how each would respond, I think the more interesting result for the latency experiments was whether the secure index’s latency depended upon document size (BSIB scaled linearly, if we make location uncertainty a constant, whereas PSI-based indexes were essentially constant) and query properties, and this was independent of sequential processor performance. In other words, I was more interested in the BigO rather than the coefficients.

Methodology: Is this a true “experiment”? An experiment, to me, has two groups of test subjects where one group receives a treatment and one group does not. I would prefer to consistently call it a simulation” but maybe this language is common in Computer Science.

I called them experiments because I was interested in observing the system’s outputs when its inputs were changed. And, typically, I had some expectation – some hypothesis – that lead me to anticipate a certain behavior. So, this seems to fall in line with the “construct hypothesis, then perform experiments to support or refute the hypothesis”.

But I’m not sure. There many be a more appropriate word to use. Also, I always took “simulation” to mean the simulation of an abstract model of something else, but since this is an actual implementation (the only time it’s not an actual implementation, but rather an abstract model, is when I simulate the adversary employing the attack strategies; but I do call those simulations).

**Results: Might need a summary of the results so far in a tabular format before Section 6.0 on Enhancements.**

**Conclusions: Conclusions section should be the last section after Future Work. Which secure index are you recommending specifically? Which was best and why? Need to summarize all three research objectives and how successfully these were met.**

Okay, I’ll re-arrange it.

Which secure index would depend, I think, on your requirements. They each have a number of trade-offs.

For instance, how much memory does the CSP have? If it is limited, space efficient secure indexes are preferable, e.g., PSIF or PSIB (especially PSIB if documents are small). Or, what sort of searches do you want to support? If proximity sensitivity is required, PSIF is out of the question. If extremely precise sensitivity is required, then PSIP may be preferable – maybe even combined with PSIM for extra proximity sensitivity, but costly in terms of memory. In my experiments, I tried to show the strengths and weaknesses of each. I guess if I was a corporation trying to decide which one to use, I would construct some objective function that I’d want to maximize, e.g., if you have a set of documents typical of your corpus, and a set of queries in which those documents are rank-ordered according to, then see which secure index type optimizes the objective function on that training set.

You do bring up a good question though. The secure index database only requires a secure index to implement the ISecureIndex interface. If it does, then it can be added to the secure index database. This brings up another thing I wanted to test: how is retrieval accuracy when a random mixture of secure indexes are in the secure index database?

**p. 5 Does IR need to be spelled out earlier as “Information Retrieval”?**

Noted.

**p. 32 in 4.2 "They have devised..." who is “they”?**

Good question! Should I just say “the information retrieval community” or something to that effect?

**p. 96 s.t. for “such that” looks weird to me.**

Okay, noted. I will replace all “s.t.” with “such that”.

**p. 97 8.2 says "Due to time constraints, we did perfom" should that be "did not perform"?**

Yes, thank you for the correction!