Review of
Improving the Privacy of Tor Onion Services?
by
Edward Eaton, Sajin Sasy, and Ian Goldberg
and
Possible ASP Capstone and Thesis Proposal

What is the problem?
HSDirs assist the client in the task of translating the .onion address of an onion service into its list of introduction points.

An explicit goal of version 3 of the onion services protocol is that it should be difficult for the HSDirs to know i) which onion services they hold descriptors for, and ii) which onion services are being accessed when they are queried. (in version 3 the HSDirs hold descriptors indexed by a blinded public key, and since the identity public key (the onion address) cannot be recovered from the blinded key, the HSDirs cannot link an onion service descriptor with the underlying onion service unless it already knows the identity public key for the onion service. **This process of keyblinding provides the security property of "unlinkability", which states that for an adversary who only observes blinded public keys and signatures under those keys, it is cryptographically impossible to pair two blinded keys as having the same underlying identity key**)

However, previous work has shown that these nodes may be untrustworthy, and can learn or leak the metadata about which onion services are being accessed.

Attacks Targeting Clients:
An adversary hoping to deanonymize a Tor user who uses onion services is placed in a relatively powerful position in the network. When a client resolves an onion service descriptor lookup, they connect to the HSDir via a circuit. Hence if an adversary in addition to controlling the HSDir, controls even the middle node of this circuit to the HSDir, they learn both the client's guard relay and the blinded public key of the service being connected to. For a service that widely distributes their .onion address, this gives the adversary the client's entry point to the network (the guard relay) and their final destination (the onion service).

Attacks Targeting Onion Services:
To track an onion service over time, an HSDir can consider the distribution of queries made to each service over time. Different services are likely to have radically different distributions of queries. By identifying two blinded public keys that received a similar distribution of queries over the course of an epoch, an adversary can ascertain with a reasonable degree of confidence that the two blinded keys correspond to the same identity public key.

(The challenge in this setting is that the database is distributed, and hence the adversary's view is limited to a fraction of the total set of queries made within an epoch. This fraction is defined by the adversarial power a and the total number of nodes *n)*. This notion of building a 'profile' for an onion service based on the query distribution is simply the starting point of our attack, and we will refer to it as the **weak variant** of the attack.

A truly malicious adversary has several other sources of information available to them that strengthens the profiles constructed. For a given onion service, additional sources of metadata that a malicious HSDir could leverage include
(i) the set of guard nodes that make the HSDir lookup requests,
(ii) the frequency distribution of lookup requests from the aforementioned set of guard nodes, or
(iii) the timings of lookup requests within an epoch.

Nonetheless, the common underlying element that makes
the attack feasible is **the ability of an adversarial HSDir to infer which of its onion service descriptors is being looked up in a request, allowing them to link the metadata of the request to that particular onion service.**

These attacks target the unlinkability of onion services, allowing some services to be tracked over time

To prevent the kinds of attacks established, we need to prevent malicious HSDirs from learning which descriptor is being queried by a user.

Summary
To restore unlinkability, we propose a number of concrete designs that use Private Information Retrieval (PIR) to hide information about which service is being queried, even from the HSDirs themselves. We examine the three major classes of PIR schemes, and analyze their performance, security, and how they fit into Tor in this context. We provide and evaluate implementations and end-to-end integrations, and make concrete suggestions to show how these schemes could be used in Tor to minimise the negative impact on performance while providing the most security

Contributions(in detail):
1. We provide a description of the v3 onion service lookup process, which is key to the remainder of the text in Section 2. We then analyze the leakage induced by the descriptor lookup mechanism of v3 onion services, and propose attacks targeting both clients and onion services that leverage this leakage.

2. In Section 3 we discuss the variants of PIR that could solve this problem, and address the challenges of integrating them into a complete end-to-end solution for Tor's onion services,

while also highlighting a network-level optimization that saves an additional network round-trip that PIR would otherwise introduce.

3. We analyze the different PIR schemes proposed to compare the privacy guarantees provided by each in the context of Tor, using enumerative techniques to provide an upper bound on the probability an adversary is able to compromise our multi-server PIR system in Section 4.

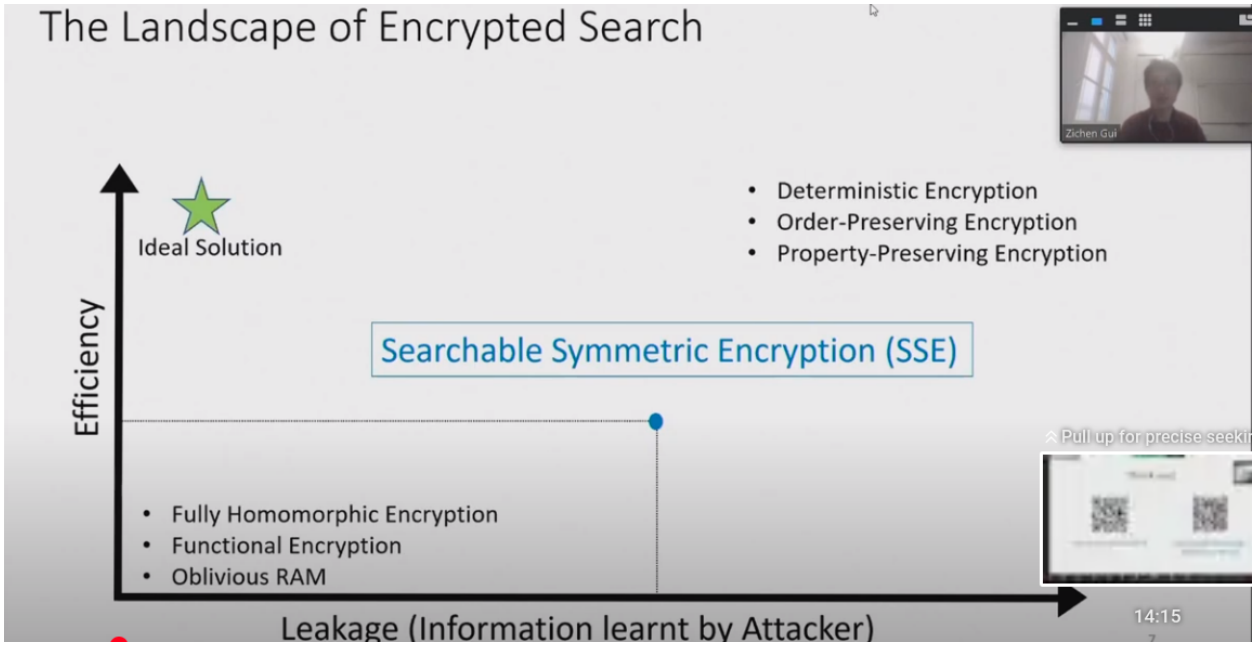
4. Finally, in Section 5 we provide microbenchmarks for all of the defences we propose. Additionally, we also implement and evaluate an end-to-end integration for the best solution from our microbenchmarks on the live Tor network. Our results demonstrate that the performance overhead (or effect on time to load an onion service for clients) of our proposed defence is negligible.

Summary of Key Results

In order to get a more complete picture of how many lookup requests an HSDir handles at a time, as well as the sizes those descriptors, we monitored the activity on an HSDir for around eight months.5 In terms of sizes, about 81% of the v3 descriptors we observed were <16 KiB, 7% were between 16 and 32 KiB, and 12% between 32 and 48 KiB. Hence for all of the PIR schemes we evaluate, we are concerned with large record sizes (approximately 16 KiB) since hidden service descriptors are about that size

Open Questions?

PIR schemes are computationally expensive and hence in order to ensure that integrating PIR guarantees into queries does not impact the performance of the entire application can we use **Searchable Symmetric Encryption**



(https://www.youtube.com/watch?v=HS9X_xrL1Gw)

Single-server PIR does not change how clients decide which HSDir to query from. The structure of the hash ring, how the client decides where to query from, and the logic the onion service uses to decide where to upload can all stay the same

Single-server PIR. In a single-server scheme, a client queries a database by sending an encrypted version of the index that they want to retrieve data for. The server performs some computation over the database, and returns an encrypted response without learning any information about the index. This goal can be accomplished either by using encryption with strong mathematical properties like fully homomorphic encryption, as in XPIR [1] or SealPIR [2], or by using secure hardware, as in ZeroTrace [30].

Can we use SSE here?

—

Proposed Roadmap(please suggest):

1. Present the tor paper: 1.1 descriptor lookup process 1.2 PIR integration
2. Replicate the results
3. **Understand and write a searchable symmetric encryption scheme (asp capstone)**
4. **Build a SSE scheme for the tor network(theoretical)**
5. **Build SSE scheme on the tor network(practical)**
6. **Analysis**

References:

1. On improving the privacy of Onion Services
2. Security analysis of mongo db queryable encryption: https://www.research-collection.ethz.ch/handle/20.500.11850/622165
3. Rethinking searchable symmetric encryption: https://www.research-collection.ethz.ch/handle/20.500.11850/564585?show=full
   A presentation of the same: https://www.youtube.com/watch?v=HS9X_xrL1Gw

September 7, 2023

Capstone: Do something that has already been done, but is challenging but you can complete in 3 months.

1. Implement V simple PIR with homomorphic encryption: https://blintzbase.com/posts/pir-and-fhe-from-scratch/

2. Implement single server PIR: https://web.cs.ucla.edu/~rafail/PUBLIC/34.pdf

3. Implement multi server PIR:

4. Implement one usenix paper from :

1. LWE
2. homomorphism

Explain LWE