Comparing Security, Latency, and Throughput of PIR and the Tor Network

Alexander Gaidis
Brown University

Andrew Park
Brown University

Abstract

We attempt to show that the slightly less secure privacy system of the Tor network can be considered to be sufficient in certain PIR use cases, especially in a real-world environment in which millions of users are on Tor. By considering the ε-private model presented by [13], we investigate mathematically the security differences between PIR and Tor. In addition, we empirically show that although Tor is less secure, the latency and throughput of Tor vastly outperforms PIR, especially as the size of the database grows.

1 Introduction

Consider dark pools, which allow investors and traders to request a large order of stocks from a bank or other financial entity without publicly revealing their intentions. This allows an investment to avoid substantial price devaluation and speculation from competing financial entities, both of which are existing issues in standard exchanges. For example, Goldman Sachs can hide their business interests from competitors by requesting a trade through a dark pool. However, this does not protect Goldman Sachs' investments from the owner of the dark pool, thus leaving opportunities for front running. This motivates the need for a system in which users can hide their queries from a potentially compromised database server. Private information retrieval (PIR) is a set of cryptographic protocols designed to enable users to do just this [5].

PIR guarantees a strong definition of privacy whereby a database server learns nothing of a user's query. Yet, PIR has been shown to be infeasible in physical deployments, because current PIR servers must access every element in a database to protect a user's query (and the desired record by extension). Therefore, both the computation and the communication costs of the server increase superlinearly with respect to the database size [4].

Rather than fix the PIR complexity problem, we examine the case where the security guarantees of PIR are excessive and the more computationally efficient, albeit less secure [12], anonymous communication channels are sufficient. In particular, we consider the The Onion Routing (Tor) network as an alternative for many use cases of PIR. The Tor network sends onion-encrypted messages over a pre-established circuit made up of a number of relay nodes that sit between a user and some end-point service. We believe that Tor can be comparable to PIR in many cases such as the dark pool example given above. While PIR does not hide the user query from the database, Tor hides the user but not what is queried. For example, the PIR server would not learn what record Goldman Sachs is interested in, while the Tor server does not learn that Goldman Sachs is making a query in the first place.

Although Tor is not able to offer the complete security guarantees as PIR, previous research has shown that privacy can be increased with more users and more nodes [13]. Specifically, leakage is minimal in a real-world environment with millions of users while still being computationally feasible. In this paper, we attempt to mathematically define the leakage and security of Tor while also empirically comparing latency and throughput of queries to a database server to substantiate our computational complexity saving claims. The paper is organized as follows:

- Section 2 describes relevant usage cases of PIR and TOR in other privacy-preserving systems, while also describing the CPIR scheme used in our model.
- Section 3 discusses the theoretical results of [13], and explains how Tor could behave like a less secure CPIR system.
- Section 4 and 5 describes the design and implementation of our client/server model, as well as the assumptions made in regards to the query distribution of a single client.
- Section 6 presents the experimental data of the latency and throughput of the communication networks.

2 Background

2.1 PIR

First described by Chor et al [7] in 1995, Private Information Retrieval (PIR) is a protocol that allows private access of public databases. Specifically, PIR schemes allow a client to access an element of a potentially compromised public database without exposing to the database which specific element was accessed. Although this is clearly a very powerful security guarantee, it is computationally very expensive, because existing PIR schemes must operate on every element of a database [4].

Chor et al provides two different levels of privacy protections: Information-Theoretic PIR (PIR-IT), and Computational PIR (CPIR). In PIR-IT schemes, the query is sent to several identical, non-colluding servers. The client collects the responses from each of the servers and combines the results locally. PIR-IT schemes provide a computation-unbounded guarantee on privacy. More specifically, database servers would *never* be able to learn the query result. On the other hand, PIR-IT is difficult to employ practically, because the non-collusion assumption among servers is infeasible in practice.

In the computational variant of PIR, hardness assumptions are used to reduce the overall computation cost of the server. Specifically, we reduce security guarantees only to adversaries which are computationally bounded. Unfortunately, the overall cost of PIR-systems are still so high that CPIR-systems are still impractical for databases with over 100k entries [4]. However, recent research by [4] introduces SealPIR, a CPIR system which utilizes query compression techniques with existing efficient CPIR protocols in order to further reduce overall network costs.

2.2 SealPIR

2.2.1 Query Compression

Here, we provide a simple explanation of the cryptosystem utilizes by SealPIR to compress and send queries. When querying from the database, the client must in actuality send n ciphertexts to the server. Specifically, the ith ciphertext is an encrypted 1 for the desired ith element, and an encrypted 0 otherwise. In this fashion, the query touches all elements of the database, while FHE allows the client to recover the desired record. However, sending n ciphertexts for one desired result is clearly inefficient. Therefore, SealPIR provides a method of sending one ciphertext to the server, while the server expands the ciphertext. Although this is relatively more expensive for the server, it significantly saves communication cost between the client and the server.

SealPIR utilizes a Fan-Vercauteren FHE cryptosystem (FV). In FV, plaintexts are represented as polynomials of degree at most N with integer coefficients modulo t, where N and t are parameters chosen by the user [4]. Specifically, the polynomials are elements in the quotient ring $R_t = \frac{Z_t[x]}{x^N+1}$. In addition, ciphertexts are polynomials in the quotient ring $R_q = \frac{Z_q[x]}{x^N+1}$. For a security parameter q.

Therefore, for a client whose desired record is the ith element of the server, SealPIR encrypts the plaintext $x^i \in R_t$ as $query = Enc(x^i) \in R_q$. Thus, the server receives $Enc(x^i)$ as the query from the client, which is then expanded and handler server-side.

2.2.2 Query Expansion

At a high level, the server expands $query = Enc(x^i)$ into n ciphertexts, where the ith ciphertext is $Enc(1)$ and all other ciphertexts are $Enc(0)$. [4] describes both the method and the proof of correctness of their algorithm, which is not described fully here. However, for the sake of our implementation, it is important to note that the cost of the n ciphertexts, therefore, has been shifted from the client to the server. More specifically, the client is now only in charge of sending 1 ciphertext, while the server must expand that query obliviously. We discuss the empirical results of this cost in a later section.

2.3 Tor

The Tor network as employed today was born out of a paper written by the Free Haven Project and the U.S. Naval Research Labs in 2004 [9]. The number of total daily international users of the network range from roughly 1.5 million to 2 million [1]. It is an overlay network which means it works atop common internet protocols such as TCP which has made its adoption quick. It works as follows.

Say Alice's client wants to communicate with Bob's server through Tor. Alice has her Tor client query a directory server for nodes in the network and their public keys in order to establish a circuit of nodes or "hops", the default being three. Once Alice's client has chosen the nodes that will participate in her circuit, they do a Diffie-Hellman handshake to exchange shared symmetric keys. It is important to note that while second and third routers know that someone has a shared key with them, they don't know who has the other key.

When Alice sends a packet to Bob's server, her client first encrypts the packet with the shared key of the last router in her established circuit, followed by the shared key of the middle router, and then the shared key of the entry node or first router. We say Alice has now created an "onion of encryption" which she will send to Bob. As the onion travels through the network, a layer of encryption is peeled back at each router that has the necessary key. Eventually the last hop, or exit node, decrypts the final layer of encryption and can send the packet onto Bob's server that is unaware Alice is the one communicating with it (barring authentication of some form).

While adversaries can see that users are on Tor and could theoretically see what elements of a database are accessed,

users are basically anonymous. In particular, unless a majority of the onion routers are compromised, TOR has a certain level of privacy guarantees [3].

3 Private Retrieval

In this section, we discuss a theoretical result proved by [13] and discuss their formulation of ε-privacy, a differential-privacy based definition of privacy that extends from not only IT-PIR and CPIR, but also to other less secure privacy systems. More specifically, they provide a flexible metric to describe all privacy protocols. The definition is provided below.

Definition 3.1 (ε,δ)*-Privacy. A protocol is* (ε,δ)*-private if there are non-negative constants* ε *and* δ *such that for any possible adversary-provided queries* Q_i *and* Q_j *and for all possible adversarial observations O, we have:*

$$P(O|Q_i) \leq e^{\varepsilon} P(O|Q_j) + \delta$$

Intuitively, this means that an adversary could provide a user with two queries and the adversary would not be able to distinguish the resulting networks between the two. (Note that under this model, the adversary is assumed to be passive and all users are assumed to be honest.)

Under this definition, it is clear that IT-PIR is (ε,δ)- private for $\varepsilon = 0$ and $\delta = 0$ (a perfectly secure system with no leakage). In addition, [13] shows that CPIR systems are ε-private where δ is negligible. Clearly, this is a very strong definition of privacy. The adversary has a lot of control over the users' actions and can limit the space of potential observations and queries substantially. Particularly, it was shown that Anonymous Request Networks such as Tor are actually not ε-private [13], and for anonymous communication to be achieved, a certain number of users and requests must be completed on the network [3]. However, in the case where we are not attempting to achieve anonymous communication, but rather avoid the adversary learning the request for a specific user, this PIR-privacy definition is excessive. Therefore, it is not a suitable definition for the usage case we are considering, where we are more concerned about the adversary learning the request of a certain user.

Although Tor is not secure when there is only one user making a request (since an adversary can theoretically observe a route of a request through the network), consider the case where there are u users simultaneously using the network. In this case, the adversary could not link a request with a user unless all users made the exact same request (Note other information could be leaked, but we are only interested in linking a user with a request). Therefore, for u users and a database of size k we could bound the probability of all u users requesting the same output. This happens with probability:

$$P(\text{all users select same index}) =$$

$$\sum_{i=1}^{k} P(\text{all users select index i}) =$$

$$\frac{k}{k^u} = \frac{1}{k^{u-1}}$$

More specifically, this probability decreases with both the database size and the number of users. Thus, when considering the fact that PIR is often usable in large database settings, and that the number of users of Tor are large, the probability of this event becomes negligible, and the privacy guarantees of PIR is unnecessary for these usage cases.

4 Design

At a high-level, we use a client/server model where the server hosts a database and the client wants to query an element in it. This section is broken down into three subsections to describe this model. The first discusses the server and client design, and the second and third describe the two methods we use to query the database that resides on the server.

4.1 Client / Server Design

The client and server we created communicate over the internet through the use of sockets. To simplify the programming and communication of the server and client, they both must agree on certain parameters out-of-band before beginning a round of communication such as the the number of elements in the database, the size of each element, and the database abstraction to use. The following sections describe the specifics of their implementation.

4.1.1 Client

The client, as found in `client_main.cpp`, consists of one large `main` function that handles all interaction with the server. The interesting piece of the client's code is the timing functionality. It is within this code that we measure round-trip latency which includes (random) query generation, writing and reading the query and response from sockets respectively, and doing any necessary parsing of the response to make it human readable/printable. The results of these latency tests are printed out to a text file which we then uploaded to a spreadsheet application for analysis.

4.1.2 Server

We created an asynchronous server whose control flow starts in `server_main.cpp` where it creates either a `PlainServer` instance or a `PIRServer` instance and generates a database corresponding to command-line options. Both of these classes will be described in more detail in the two querying sections. For now, it suffices to know that both `PlainServer`

and `PIRServer` inherit from base class `TCPServer` which accepts incoming connections and passes them to its inner-class `TCPConnection` for handling.

`TCPConnection` contains a handful of handler functions in it that correspond to the type of server that was created (this is specified at run-time) such as `handle_read_galkeys()` that reads, processes, and stores the Galois keys generated by the client which are used for the PIR protocol. In general, `handle_read_<protocol>` handles the reading of information from the client, processing it, and generating a response. `handle_write_<protocol>` simply writes the response to the client back out to the socket. The read and write handlers call each other and thus create a loop that is only exited when the client sends an `EOF`.

4.2 PIR Querying

PIR querying follows the process described above in §2.2 using the SealPIR library. A few changes were made to integrate the library such as extending the `PIRServer` class by `TCPServer` and adding a generate database method. But the overall control flow of structuring a query client-side and generating a response server-side remains the same. Each time a client connects to a fresh instance of a `PirServer`, they must do a Galois key exchange "handshake" to sync with the server. Once this is complete, they can query as many times as they wish without re-exchanging keys.

4.3 Tor Querying

Tor querying from a client/server perspective is nothing more than querying a database abstraction that is an n-byte array where an element of size l at position i is contained at indices $[i, i+l)$. Tor's main functionality comes from its large user-base and the fact that it is an overlay network so TCP applications need not change their implementation to work with Tor. To get our application to route traffic through the Tor network, we use Tor version 0.4.1.6 along with `torify(1)` which acts as a wrapper for Tor.

Since Tor revolves around its network, we chose a database representation that was similar to a bare-minimum PIR database so as to keep our tests similar but ensure that what Tor lost in communication overhead, it gained back in computation overhead.

5 Implementation

Our client/server implementation consists of around 800 lines of C++ code and around 2,000 lines of C++ code if the code from SealPIR that was merged into our code-base is included. To change the number of hops ($[2, 9)$) in our Tor service, we edited the source code of version 0.4.1.6 and recompiled the binary. Our server was hosted via an Amazon t3a.medium EC2 instance with a 2.5 GHz AMD EPYC 7000 series processor, 4096MiB of memory, and up to 5 Gigabit network speeds. Our client was a MacBook Pro with a 2.9GHz Dual-Core Intel Core i5 processor, 16GB of memory, with up to 1 Gigabit network speeds.

The most challenging implementation-related parts of this project were three-fold. First, getting the hang of CMake and the idiosyncrasies of C++ took quite a bit of time. Different compiler versions, library versions, and more made getting everything set-up a multi-day affair. Second, reading through and making sense of dense C++ cryptographic library code without comments slowed down development and often made debugging tricky. Finally, the concept of this project contains a lot of subtleties that took a while for us to come to terms with through reading papers and discussion.

6 Evaluation

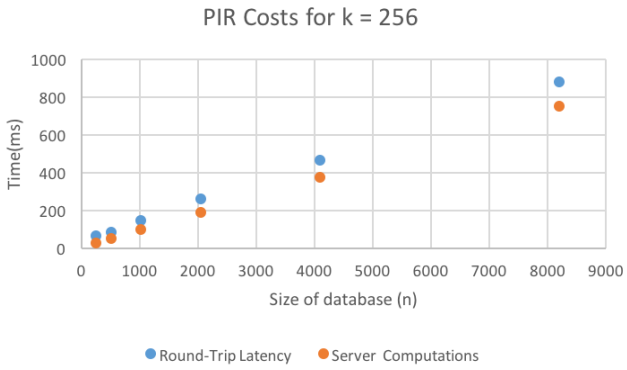
In this section, we present some preliminary empirical results from testing query send and retrieval through our client and server implementations.

In Figure 1, we plotted the latency of our PIR design for varying database sizes and element sizes with 10 trials for each point in order to reduce variance. This metric of round-trip latency includes the sending of the query, query expansion server-side, and then the receiving of the result back at the client.

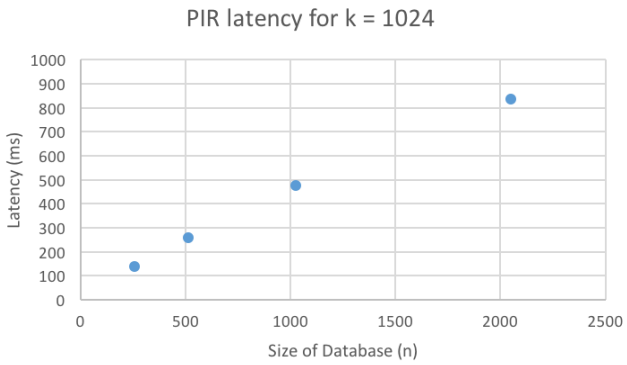
The above figure clearly shows that increasing the database size and the element size results in a higher latency. Particularly, larger database size implies that the server must compute more in order to expand the query in our FHE scheme, and the larger element size implies more information that must be communicated. In the largest case, the system could about 800 milliseconds to process approximately 2MB of data. Our implementation of SealPIR was not able to handle larger databases, which shows how PIR is very limited to smaller DBs. In addition, when comparing the latencies to Tor, we note that the latencies are much higher comopared to the latencies that we see in the Tor network, even for a high number of nodes.

When looking at Figure 2, we note that more hops in the circuit leads to a higher latency. This intuitively makes sense; if there are more hops in the circuit, it will naturally take a longer time to reach the end of the path. In addition, we see that for longer paths in the circuit, there is much more variance in the resulting latency. In particular, the box plot is much more spread out, which implies that the network would not reliably perform faster/slower solely based on the number of onion nodes. This could be based on the circuit construction step that occurs for every request. Because the sites of the nodes are chosen randomly all over the world, there is a lot of variance for where the onions could be located.

When examining throughput of the Tor network for increasing circuit size, we find that the throughput decreases. When



(a) PIR costs for element size of 256 bits



(b) PIR latencies for element size of 1024 bits

Figure 1: Latency of PIR with respect to different database and element size.

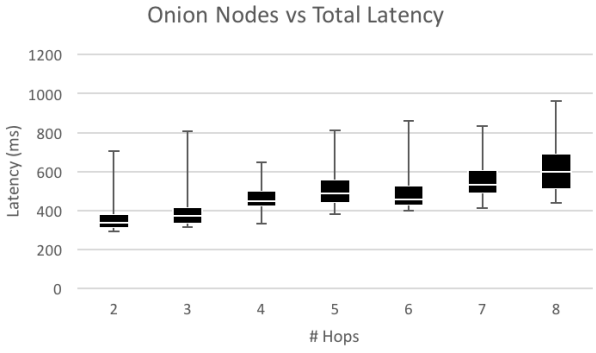


Figure 2: Latency of Tor with respect to the number of hops in the network. The database was calibrated to be of size 65,536 with an element size of 512.

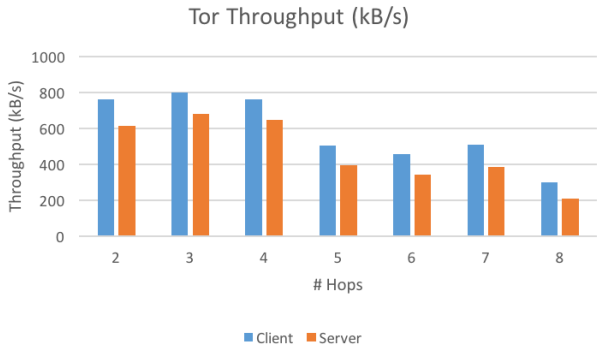


Figure 3: Throughput of Tor both client and server side with respect to the size of the Tor network. There exists a negative, albeit noisy, correlation between the two.

testing throughput against a plain query request, the throughput was found to be approximately 5mB/s, which demonstrates the contrast in information both sent and received in a cryptographic network.

7 Future Work

Going into the winter break and next semester we are planning a few avenues of future work. Namely, improved testing, improved proofs, quantification of other PIR protocols, and application of knowledge.

7.1 Testing

Automating our test framework more to gather additional data could prove to interesting as both of us agree that we have just started to scratch the surface with our experimental results. Also, more work could be put into standardizing and streamlining our test framework. One easy improvement would be to host the client as an Amazon EC2 instance to allow for stable internet connection and configurable hardware settings. We did not implement this currently out of convenience and cost.

We also did not test our design and implememtation again PIR databases in which the large size severly affected the communication cost. More specifically, we not able to test against databases that are hundreds of megabytes in size, because we were limited by our SealPIR framework. By further generalizing the SealPIR framework to allow testing for larger databases, we can hopefully find further significant evidence in latency differences between PIR and Tor.

7.2 Proofs

Our proofs assume a passive or network adversary. Particularly, this means that adversaries can observe but not manipulate traffic or nodes. Building upon this to prove things for active adversaries is an open line of research and one that we think would be insightful for our research question. In addition, we currently have a very simplistic definition of security for Tor. One potential question is whether we can strengthen this privacy definition to be closer to the ε-private definition stated above.

7.3 Additional PIR Protocols

SealPIR came off the back of XPIR [2], and it claims to be the fastest CPIR implementation. However, we have not seen a paper that compares the real-world cost of more than two PIR protocols. Other schemes such as Percy++ [10] and ITPIR protocols such upPIR [6] and RAID-PIR [8] would be interesting to compare.

7.4 Application

This project was born out of interest in developing a PIR protocol for Yodel [11], meta-data anonymous voice calls. After delving into the literature more and doing our own quantification of PIR, we believe an interesting path for future work to be developing a middle-ground between Tor and PIR that both scales well and offers strong privacy guarantees.

8 Conclusion

Our preliminary results seem to show that the Tor network is a potential alternative to PIR systems especially in cases where the PIR security guarantees are excessive, while significantly saving computation costs. Although Tor has more variance in the query result by the nature of the circuit construction, it still outperforms PIR queries on larger databases. Extending our research to handle larger PIR databases and strengthening our mathematical argument will be done to eventually attempt to show that Tor is comparable to CPIR in certain environments.

References

[1] Tor metrics. https://tinyurl.com/pfk3yty.

[2] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir : Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016, 08 2015.

[3] Megumi Ando, Anna Lysyanskaya, and Eli Upfal. Practical and provably secure onion routing, 2017.

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. pages 962–979, 2018.

[5] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, Savannah, GA, November 2016. USENIX Association.

[6] John Cappos. uppir, 2011. https://uppir.poly.edu/projects/project.

[7] B. Chor, Oded Goldreich, and E. Kushilevitz. Private information retrieval. volume 45, pages 41–50, 11 1995.

[8] Daniel Demmler, Amir Herzberg, and Thomas Schneider. Raid-pir: Practical multi-server pir. volume 2014, 11 2014.

[9] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. *Paul Syverson*, 13, 06 2004.

[10] Ian Goldberg. Improving the robustness of private information retrieval. pages 131–148, 06 2007.

[11] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Yodel: strong metadata security for voice calls. pages 211–224, 10 2019.

[12] S.J. Murdoch and George Danezis. Low-cost traffic analysis of tor. pages 183– 195, 06 2005.

[13] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost epsilon-private information retrieval, 2016.