Private Queries on Public Certificate Transparency Data

Vy An Phan

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-27
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-27.html

May 9, 2019

Acknowledgement

I would like to thank Jean-Luc Watson and Edward Oakes, for their great contributions to the system-level improvements discussed in this paper.

I would also like to thank my advisors, Doug Tygar and Raluca Ada Popa, for their support and advice throughout this project.

Private Queries on Public Certificate Transparency Data

by Vy-An Phan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee

J. Doug Tygar
Research Advisor

9 May 2019
(Date)

* * * * * * *

RPopa

Raluca Ada Popa
Second Reader

May 6, 2019
(Date)

Abstract

Despite increasing advancements in today's information exchange infrastructure, the preservation of user data and privacy still remains a problem. Both insecure baselines and secure solutions leak user data. For example, Certificate Transparency (CT) promises significant security improvements to existing Public Key Infrastructure solutions that up-to-now have solely relied on the Certificate Authority hierarchy. CT provides a robust auditing layer and transparency solution to quickly detect such compromises, but introduces the requirement that client browsers interact with third-party servers when validating a site certificate.

In the existing CT system, these requests leak information about each user's browsing habits to the hosting server. It is not a stretch to think that this valuable data could be collected and exploited, as corporations and governments have plenty of financial and political incentive to do so. In this project, we seek to address this problem by using an oblivious file sharing system with strong anonymity properties, to provide a more scalable, performant solution to privacy-preserving queries.

Contents

1 Introduction

Modern secure communication between users and websites requires on the exchange of public keys. The Public Key Infrastructure (PKI) is a core underpinning technology of the modern internet, enabling secure traffic (e.g. via TLS) between clients and servers.

Traditionally, users depended on central certificate authorities (CAs) for issuing, moderating, and verifying certificates. This model of putting all trust into a single point of failure has proven to be far from perfect. There are documented cases of CAs incorrectly issuing certificates [1, 11], being attacked by malicious third parties [5], and even engaging in malicious behavior themselves [20]. Even worse, failure in the CA system is an "or" condition, not an "and" condition. There are dozens of CAs in existence, and it only takes one of them to be compromised in order for all the sites dependent on it to be compromised, even if said websites had their public keys stored with other CAs. These incidents can expose large numbers of users to active man-in-the-middle (MITM) attacks.

In response to these growing concerns, Certificate Transparency (CT) is a new protocol that addresses these issues by enabling users, CAs, and domain owners to audit certificate changes and identify and monitor potential modifications directly relevant to them [4]. Recently, CT has seen uptake by both web browsers and CAs, with Google Chrome requiring all trusted CAs to partake in the protocol since 2017 [2]. All changes are logged via hashed chains of Merkle tree roots, preventing editing of prior history and allowing easy auditing of future changes.

However, CT is not without its flaws. In particular, it represents an additional communication channel — as part of normal operation, users query servers hosting CT log for certificates matching domains with which they wish to communicate as shown in Figure 1.1. This querying is a central part of the CT protocol, but has the side effect that a malicious log server can glean private information about the user from their queries, namely, correlating the IP address of the request with the domains visited over time.

Such tracking could easily be performed without consent. Particularly worrisome is that these log servers have financial incentive to do exactly that: transparently collect user browsing data and sell it to industry players in advertisement and tracking. This example is just one of many instances that illustrate the privacy concerns that arise from requiring users to contact a third

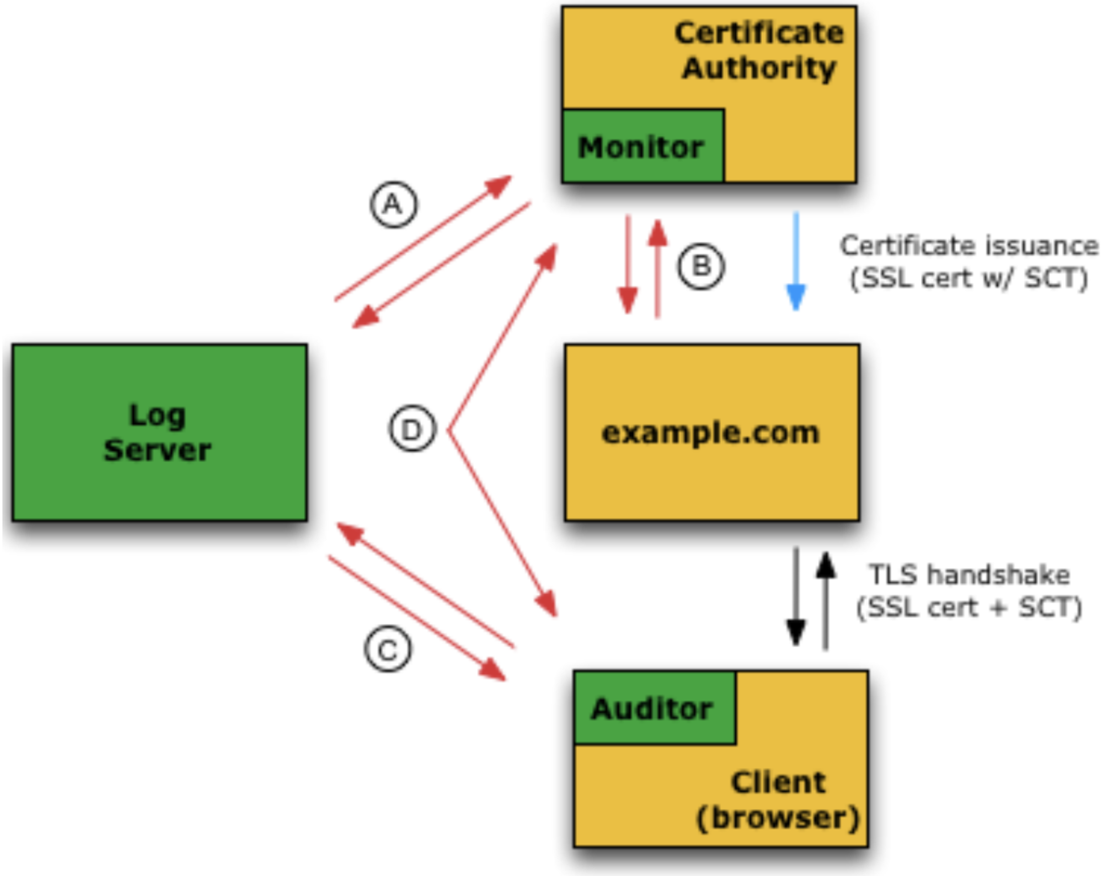

Figure 1.1: A simplified architecture diagram for Certificate Transparency. Assuming an honest-but-curious log server, request C is problematic as it reveals to the log server what sites the client is visiting.

party (CT logs) to communicate with any desired domain. In general, users should be able to benefit from CT without sacrificing their privacy.

Previous efforts to maintain privacy in Certificate Transparency and other data retrieval systems are discussed in Section 7. In this project, we seek to extend the CT protocol to preserve users' privacy from the log servers that they query.

We outline our definition of privacy with subsequent design goals in this setting in Section 3. Our solution should be usable and scalable, in order to efficiently service hundreds of millions of users, without sacrificing user privacy by revealing their queries to the server.

In Section 4 we describe our solution. We use an ORAM file sharing system [**oram-filesharing**] to achieve the base security properties, and add system-level performance improvements to make it more usable. We do this by precomputing encryption exponentiations and garbled circuits, pipelining phases of the transfer protocol, and batching requests.

Despite implementing all of our improvements, we discovered that we did not meet our scalability goals. Even so, we saw better performance that points to a possibility of meeting those goals in the future. This will be further discussed in Section 5.

2 Related Work

2.1 Privacy in Certificate Transparency

Since Certificate Transparency's inception, there have been significant extensions to increase its functionality and security. For example, CT has been extended to support certificate revocation and subsequent applications including end-to-end encrypted mail [6].

Additionally, the security guarantees of CT have been demonstrated via cryptographic properties [7] and measurement studies have been conducted to evaluate how it is used in practice and the differences between public CT logs [10].

However, evaluation and extension of the privacy properties of CT have not been as thoroughly scrutinized. Eskandarian et al. identified and proposed a solution to privacy issues related to browsers auditing a CT log and extended CT to support non-public subdomains [8]. To the best of our knowledge, however, there does not exist an extension to CT that preserves users' privacy when interacting with CT logs.

2.2 Oblivious RAM

An adversary with access to a machine's data access patterns can gain nontrivial information about a particular process, even if the data itself is encrypted. Oblivious RAM (ORAM) [9] schemes prevent this by randomizing data access patterns in such a way that observing them reveals no information about the data being accessed. This is especially useful in the case where a client or user wishes to access data on a server or machine without revealing what that data is.

Primitive approaches to ORAM are not very efficient. Path ORAM [17] is an improvement on traditional ORAM by storing data blocks in a tree structure. It only requires that the client store a location mapping of data blocks. As such, its bandwidth efficiency can be reduced to logarithmic or poly-logarithmic if the client wishes to recursively store the location mappings as another oblivious tree. This construction has the best bandwidth efficiency among the known ORAM protocols and requires little storage on the client's part. While it is very good in the secure processor setting, however, it still suffers from a complex memory block eviction strategy ($O(D \log^2 N)$) that, while more optimal for certain problems, does not work well in the multi-party computation setting.

Circuit ORAM [19] is a related ORAM variant which achieves a near-optimal circuit size for realistic memory block sizes. Also tree-based, it takes advantage of preemptive metadata scans to complete the eviction algorithm in one pass, substantially improving the eviction efficiency. This makes it more useful for multi-party computation between servers than Path ORAM.

2.3 Two-Party Computation

Two-Party Computation (2PC) protocols provide a way for two parties A band B to compute a mutual result $f(a, b)$ without revealing their respective secret inputs a and b to each other [12].

Both parties start by agreeing on the evaluating function f and obfuscating their secret inputs, most commonly by XOR-ing with a random value. A generates garbled circuits [21] that compute f (as well as the required commitments for consistency checks), while B evaluates and decodes the result based on the information sent by A.

Privacy is preserved because each party is involved in a different share of the computation. A knows the random values used in generating the garbled circuit but never comes in contact with input b. B receives A's inputs, but does not know what it is because A is in charge of the scrambling and encryption.

2.4 Oblix

Oblix [14], or OBLivious IndeX, is a space- and time- efficient search index that neither leaks access patterns nor result size. It works by maintaining an obliviously sorted multimap, which essentially links keys to values while ensuring that any given client requests are computationally indistinguishable from random requests. To do so it requires many of the same cryptographic techniques that we will use in our proposed solution, including garbling, ORAM, Merkle hash trees, and oblivious data structures (including the obliviously sorted multimap).

Oblix has been demonstrated to be useful in private retrieval of public keys in the case of certificate transparency, and in fact solves the basic problem statement presented at the beginning of this paper. However, it does rely on trusting secure hardware enclaves, which our solution will avoid.

3 Design

3.1 Security

3.1.1 Threat Model

We assume that we have access to two *non-colluding* servers. These servers can be passive eavesdroppers (honest-but-curious) — they can observe any local state not secret-shared between the servers and have full access to client communications with itself.

While requiring two non-colluding servers is perhaps unrealistic in many domains, it fits the certificate transparency system well, where many different companies are providing CT services. These different entities could very well be financial competitors (e.g. Google versus Symantec), and sharing collected datasets would eliminate their commercial advantages. Under our threat model, a CT provider running in this way does not have to be trusted by any clients making requests for certificate proofs.

We assume that CT security properties are still valid, namely, that unauthorized and/or malicious modifications to certificates will be detected with extremely high probability by a sufficient number of auditing nodes. Thus, we focus specifically on the privacy implications of CT queries.

3.1.2 Security Definition

We define security as preserving the user's privacy. When a user queries the CT server, they should be able to receive the correct proof for the desired domain without revealing to the server what they asked for.

Even if given two potential domains a and b, and observing many client requests for one of the domains, a malicious CT log provider should not be able to infer the identity of the queried site any better than a random guesser. Similarly, patterns of requests for the same domain should not reveal any information to the server (a new movie trailer drives a sudden traffic spike to streaming website `a.com` so any new repeated requests to a given domain is more likely to be a request for that website).

Our security definition only concerns itself with user privacy, and not that of the server. Since certificate information is meant to be public anyway, it is not a major security threat for a client to know the public keys of domains that they did not ask for. However, we still wish to avoid this scenario to minimize overhead and unnecessary transfers.

3.2 Design Proposal

3.2.1 Goals

We focus on two primary design goals:

- Client queries for a domain's certificate do not leak the requested domain to the CT server, as discussed above, without relying on a trusted third party or hardware enclave.

- Preserving client privacy imposes reasonable overheads on CT log administration and user requests.

3.2.2 Oblivious Merkle Trees

Merkle trees are a special type of binary tree in which a node's value is equal to a hash of its children. Thus, if the value of a node changes even a tiny bit, that change will propagate up to the value of the Merkle tree root [13].

In the context of certificate transparency, Merkle trees provide a mechanism for users to check the validity of a certificate from a log server with logarithmic efficiency [4]. An attacker trying to falsify a certificate cannot realistically find hash collisions well enough to give a valid proof. Merkle trees make it infeasible for an attacker to edit or append certificates without being quickly caught.

However, traditional CT allows the server storing the Merkle tree can easily determine which node a client is requesting the validity proof for (and thus which website a client is trying to visit). This is because the proof consists of the siblings of the nodes on the path from the root to the node the client wishes to validate.

To make certificate transparency work privately, we need to store Merkle trees obliviously. While Oblix [14] is successful at this, our goal is to do the same without relying on secure hardware enclaves. Specifically, we wish to access specific nodes of the Merkle tree, just as in the non-oblivious certificate transparency public key lookup and proof, without revealing which node we accessed.

We do this by using two non-colluding servers that are capable of 2PC. Similar to the Oblix Merkle tree implementation, nodes are identified using an oblivious index [14]. Instead of relying on secure enclave, however, we take advantage of the "separation of powers" provided by two-party computation.

We store the Merkle tree nodes and certificate data using ORAM [19] on one server, and the position mappings on the other. One server knows the values of the nodes, and the other server knows the positions of the nodes, but neither knows both. Only the user gets both the position and value shares, and can assemble this information privately to generate the Merkle tree proof. This setup still uses the original tree structure of the data nodes and operates at $O(\log n)$ per request.

3.2.3 Scalability

The crux of our solution to providing private queries on public CT data relies on a scalable private storage mechanism. Many existing private storage solutions (as discussed in Section 7 are inefficient, do not scale to many users, rely on a trusted proxy or specialized hardware, or some combination of these.

Using ORAM and 2PC alone would fulfill the first requirement. However, while oblivious file sharing performs significantly better than existing private file storage systems, its initial version is does not scale to serve the entire world's CT requests. The basic system can only process a single request at a time in serial due to the ORAM construction, so multiple users making many simultaneous requests results in fast-growing latencies. Currently, reading a 4KB file from this basic system takes about 2.5s when there are many files in the system under our testing setup, which we will discuss further in Section 5.

One key design consideration for CT is that because certificate validation is performed online and in batches, providing latencies that are good enough for real-time browsing is unnecessary in this context. However, to realistically serve many clients, our system must be able to more gracefully handle multiple requests in parallel.

A primary focus of our project is to modify this existing system to make it more efficiently serve multiple concurrent requests. We do so by precomputing encryption exponentiations and garbled circuits, pipelining relevant stages, and batching similar requests.

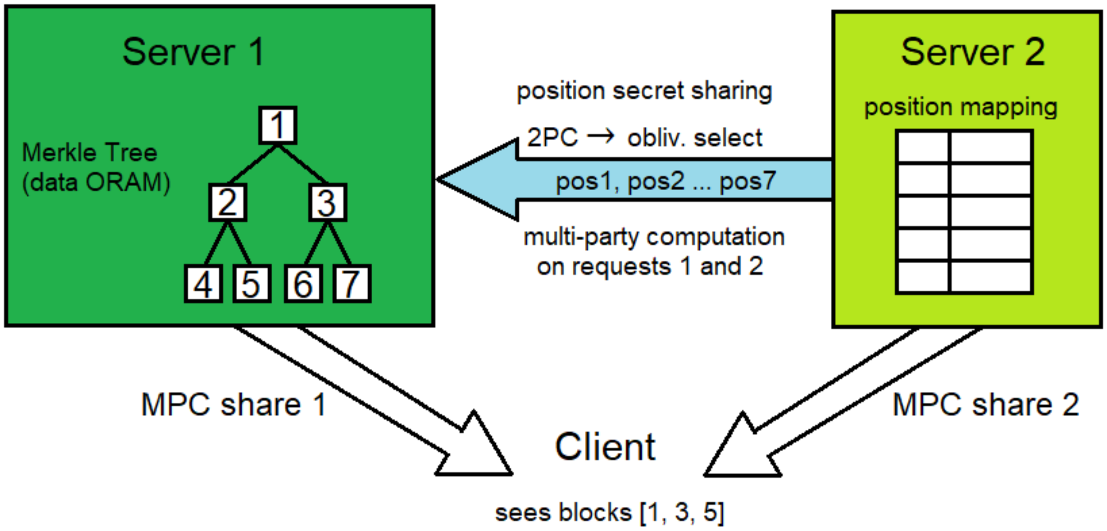

Figure 3.1: The two servers use two-party computation to create the client's request, without knowing the other's share.

4 Implementation

4.1 Implementing Oblivious Merkle Trees

We make the same assumptions as was mentioned in the threat model in Section 3: the client has access to two mutually distrustful servers. These servers can both independently be passively malicious, but they do not collude. In implementing the oblivious Merkle tree, we use C and its associated libraries like Obliv-C [22] for the baseline system implementation.

One server is responsible for storing the actual data, that is, the certificate transparency Merkle tree nodes. The other server knows the random position mapping for the nodes (analogous to the oblivious index of Oblix). By using oblivious transfer, the two servers can secret share the position mapping and use two-party computation to generate the two shares of the final value. As the client is the only one that possesses both shares simultaneously, they are they only one who receives the final result. A basic version of this protocol is shown in Algorithm 1.

Once a node has been accessed, it must be evicted and and randomly placed back in the server as in Circuit ORAM to prevent the server from linking a previous request to a later request. Otherwise, a server will be able to tell that the user has accessed the same node multiple times. The eviction protocol is shown in Algorithm 2.

Algorithm 1: The basic strategy for requesting the Merkle tree proof in our oblivious CT scheme, as seen in 2PC [12] and oblivious transfer [3]. For simplicity we exclude the standard permissions and correctness checks.

```
Input: server1 (Merkle nodes), server2 (position mappings)

circuit = server1 and server 2 agree on a circuit to compute the Merkle proof
garbledCircuit, labels = server2.garble(circuit)
encryptedPosmap = server2.encrypt(positionMap, labels)
encryptedMerkle = server1.obliviousRequest(ctMerkle, labels)

server2.send(server1, garbledCircuit)
server2.send(server1, encryptedPosmap)
encryptedProof = server1.eval(garbledCircuit, encryptedMerkle, encryptedPosmap)
outputLabel = server1.decrypt(garbledCircuit, encryptedProof)

share1 = server1.format2PC(outputLabel)
server1.send(client, share1)
share2 = server2.format2PC(garbledCircuit)
server2.send(client, share2)
merkleProof = client.recompute(share1, share2)

Result: Obliviously fetches and returns the required blocks for a Merkle tree proof.
```

Algorithm 2: `evict()`: The basic eviction strategy when obliviously requesting the Merkle tree proof in our private CT scheme, as seen in 2PC [12] and Circuit ORAM [19].

```
Input: CertificateNode

merkTree = Server1.initMerkleTree();
posMap = Server2.initPositionMapping();
List proofNodes = [required nodes for Merkle proof of CertificateNode]

for node in proofNodes do
    nodePosition = posMap.getPosition(node)
    path = path to nodePosition
    deepest = Prepare deepest legal node for eviction
    target = Prepare target block for writing
    holdBlock = $\perp$
    destination = $\perp$
    for i in [0 ... length of path] do
        writeBlock = $\perp$
        if holdBlock != $\perp$ and i == destination then
            writeBlock = holdBlock
            holdBlock = $\perp$
            destination = $\perp$
        end
        if target != $\perp$ then
            holdBlock = read and remove deepest legal block in path[i]
            destination = target[i]
        end
        if writeBlock != $\perp$ then
            path[i].write(writeBlock)
        end
    end
end

Result: After fetching the required blocks for the Merkle tree proof, reshuffles the blocks in
        ORAM.
```

4.2 Implementing Scaling

In this section, we describe our changes to the baseline oblivious file sharing system implementation that allow it to scale more efficiently when servicing multiple requests concurrently.

4.2.1 Precomputation

Oblivious transfer protocols use asymmetric encryption and garbled circuits. Both of these processes are computationally expensive and used many times in the sharing protocol, but are also independent of the inputs and CT itself. Thus, they can be precomputed offline, separate from the actual requests. This should improve both individual request latency as well as the total throughput of the system.

Modern asymmetric encryption schemes involve computing exponents of very large numbers. This can be done in a separate thread from the one that services requests. Afterwards, the values can be retrieved as needed. We only need to be careful to protect the precomputed exponentiations from cache timing attacks.

Garbled circuits [21] are also very time-consuming because they cannot be reused, and a new one must be computed for each transfer. Having one server generate a garbled circuit and then transfer it to the other only when needed introduces significant latency and could be offloaded to another set of servers that computes them offline or in the background. Again, this can be computed simultaneously with and separately from the rest of the transfer protocol. While the garbled circuit itself is important for the transfer, the composition of the circuit itself is independent of the request so long as it takes the proper number of inputs, is properly randomized, and changed for every request. We generate the garbled circuits in a separate thread on demand.

4.2.2 Pipelining

The system's server-side protocol to access files can be seen as a series of sequential stages. Some of these stages must be performed sequentially for a single request, while others can happen in parallel.

The nine main stages of a single request are listed below:

1. Receiving client requests and validating commitments.
2. Performing permissions checks on the user and files.
3. Obliviously fetching the CT Merkle tree node positions.
4. Obliviously fetching the CT Merkle tree proof nodes.

5. Generating a permutation to shuffle the paths in ORAM.
6. Applying the generated permutation to ORAM.
7. Computing the 2PC result shares for each server.
8. Secret sharing the result to the client.
9. Recomputing the result based on the given secret shares.

The system cannot concurrently fetch a new path from ORAM while the previous permutation has not been applied to ORAM. This results either in a loss of obliviousness if the permutation is not performed at all, or an incorrect lookup if the two operations happen concurrently.

On the other hand, stages 1-2 are completely independent of the following stages, as long as we are careful not to finish the revealing process before the validations and permissions checks are finished. This suggests that the system could benefit from pipelining these stages, or overlapping the execution stages across requests to provide partial parallelism and improve performance when servicing multiple requests.

Our design is limited by the fact that Obliv-C [22] currently only supports a single thread running two-party computations. This means, for example, that while stages 2 and 3 could be pipelined, we are not able to do so as we must perform all 2PC in a single thread. Adding parallelism would require major modifications to Obliv-C that are outside the scope of this project.

Our implementation consists of three logical groups of stages that run in separate threads: stage 1, stages 2-6, and stages 7-9. While performing all 2PC in a single stage is not ideal, in fact the group consisting of stages 2-6 is the bottleneck, so splitting stages 2-6 would not result in a significant speedup.

4.2.3 Batching

In scenarios such as Certificate Transparency, client latency is a small concern compared to the impact of raw throughput on scalability. We explore how to widen bottlenecks in the pipeline discussed above by batching and processing multiple independent requests in parallel. While this approach results in longer per-request processing times, we can increase the ability of server pairs to handle higher load.

We first enable the system to successfully process interleaved requests – two or more simultaneously transmitted query shares may arrive at a server in any order and still be processed correctly. We synchronize and update the pending, incomplete queries by their commitment values and expose a queue-based interface to the main file access handler, similar to the queues used to implement pipelining. The handler withdraws requests from the queue, synchronizes with the second server, and processes the query as normal. This capability allows for processing sets of batched requests, simply by withdrawing more than 1 query at a time from the incoming queue.

Interacting with the ORAM requires a set of individual path queries, so it is difficult to fully merge a batch of queries. However, we observe that the Circuit ORAM's tree-based structure results in notable overlap between different operations. In addition, while computing over the small file-index ORAM is quite quick in a 2-party computation setting, operations on the larger filesystem tree is more costly. Rather than synchronizing the data ORAM serially for every request, we compose the changes caused by the batch as a whole to require at most one update per block in the tree.

As a simple example, imagine a query that results in a block eviction into the root, followed by a second query whose result moves the same block from the tree root to a bucket at a leaf. Operating sequentially, we would follow these exact steps and relocate the same data block twice. On the other hand, applying multiple operations to the index ORAM tree at once, while performing the appropriate flushes, can yield more efficient updates, in this case directly from the stash into a leaf bucket.

Implementing this requires a fundamental change to the type of permutation generated by the 2PC before it is applied by each server to the data ORAM. Whereas previous permutations consist solely of two paths intersecting only at the stash and root, composing the result of many individual evictions requires combining a subtree of varying shape, depending on the particular slice of the eviction schedule. Maintaining a full-tree permutation representation is not memory efficient or necessary; instead, we continue to track changes to the ORAM tree in a global context (rather than relative to the current eviction path), but in a lightweight C macro-based map [18] that scales the batch size.

We found that removing path-relative tracking reduces the complexity of index management required to track ORAM changes, as we can reset the index values at any point and derive any changes made to the entire tree after some amount of processing. This structure can be collapsed into an array with explicit index labels in order to create permutation shares for the individual machines and apply the tracked changes to the filesystem itself.

5 Evaluation

Our primary evaluation mechanism was observing the behavior of the modified oblivious file sharing system under load from multiple concurrent clients, requesting as many randomly-selected files as possible over a 60 second window, extracting latency and throughput behavior as the number of competing clients (e.g. potential CT-enabled browsers) increases.

For our testing we used an Ubuntu m5d.xlarge NVMe SSD type AWS instance running on one core with 150GB memory. Without any improvements at all, a single serial request may take 2.5s or more on our experimental setup depending on the request size and database size.

While we did observe positive trends associated with our systems-level optimizations, they did not significantly change the overall complexity of processing each request. Due to general resource constraints and the aforementioned request complexity, we had to limit our evaluation to small numbers of parallel clients instead of the larger scale that would be more realistically seen in a potential real-world datacenter.

5.1 Precomputation

As discussed previously, encryption caching by itself provides a little bit of speedup but does not change the overall complexity. Similarly, precomputing the garbled circuits technically does not change the big-O complexity, but in practice shaves off more time than caching of exponentiation results. Together, they make the latency more bearable.

5.2 Pipelining

Figure 5.1 evaluates the impact of encryption and garbled circuit precomputation/caching and the pipelining mentioned in Section 4 on client latency. The filled in measurements represent clients running a two-stage pipeline, and indicate that such a technique can slightly increase the effectiveness of other optimization. Specifically, as the number of concurrent clients increases, simply pre-computing exponentiations does not benefit client latency, but pipelining lowers average latency by a constant factor and benefits exponentiation. This is likely due to the fact

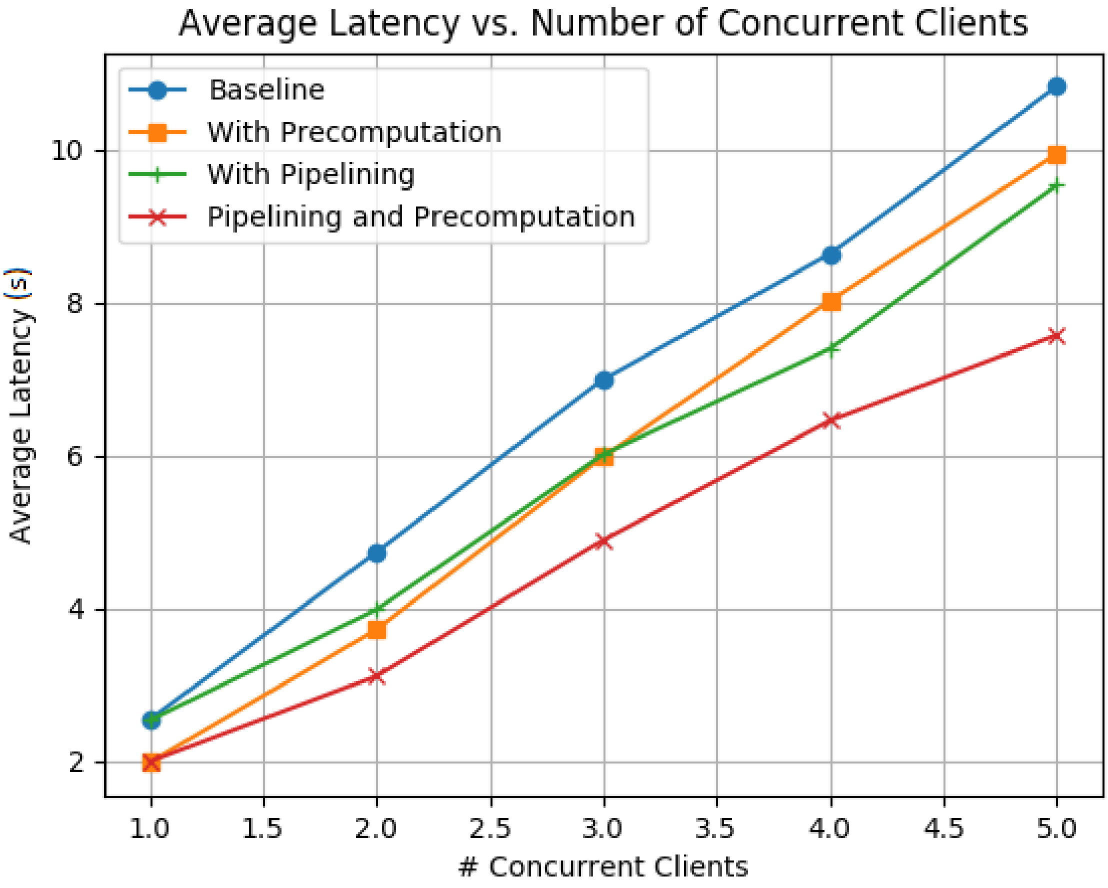

Figure 5.1: Average client latency under contention, with pipelining enabled and disabled.

that the caching in the baseline serial implementation only occurs when there are no requests to be handled, whereas the inclusion of multiple pipeline stage threads allows exponentiation to persist in the background. Thus, we believe the ability to concurrently schedule different tasks can directly impact overall system effectiveness.

Figure 5.2 tracks server-side throughput with multiple outstanding requests (in these experiments, the clients only send one query at a time). As expected, pipelining performs at the same level as the baseline when only one request is processed at a time since each of the other stages are empty. Throughput rises drastically with two parallel requests (filling up both current pipeline stages) and flattens out with increased load as the pipeline is completely full. As mentioned earlier, a more flexible MPC implementation with multithreading capabilities would enable much finer granularity (more pipeline stages) and therefore more significant performance gains. The best-performing combination in terms of throughput was again both precomputation and pipelining with an approximate 22% performance gain, although a pure pipelining-based approach was eventually able to exceed the initial raw performance boost offered by pre-computation. Again, as with latency, which these improvements do make the system faster, they do not perform significantly better as more and more requests are provided.

5.3 Batching

We successfully modified the oblivious Circuit ORAM implementation used by our file sharing system to track and collapse subtree-based modifications for arbitrary batch sizes as described in Section 4. However, we encountered an late implementation issue using Obliv-C to expose the per-server permutations to each party in the 2PC. Thus, while we can demonstrate that such a method will work in practice, we were not able to perform the same evaluation as above while batching was enabled.

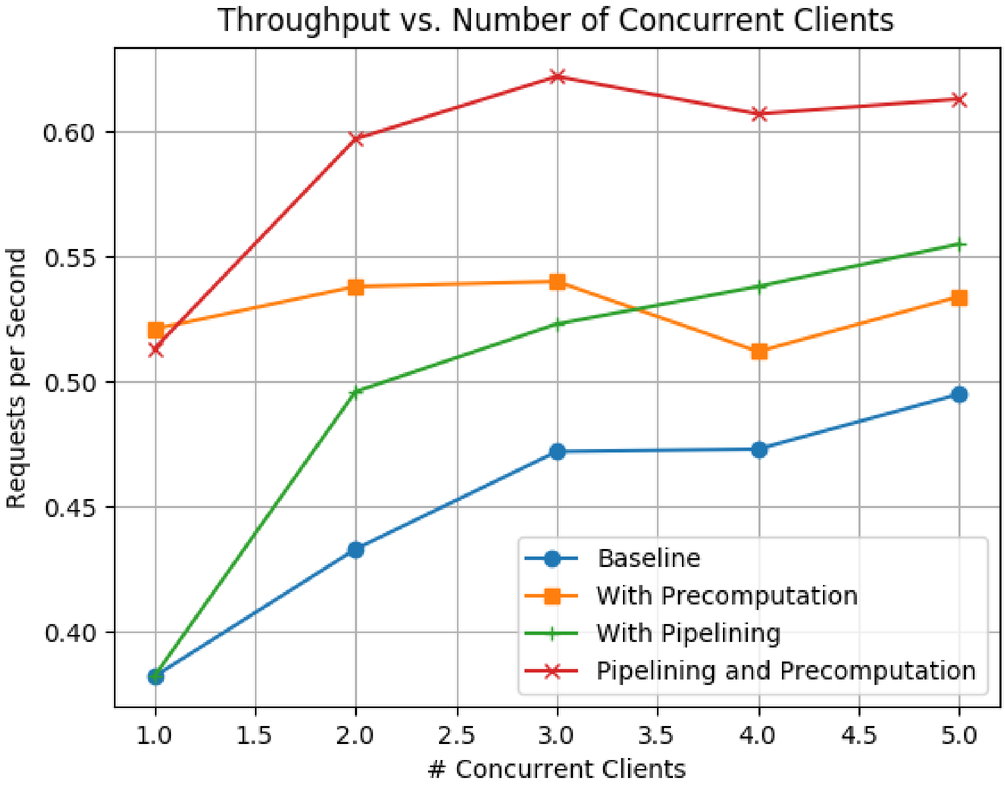

Figure 5.2: Server throughput as the number of parallel requests grows, with pipelining enabled and disabled.

6 Conclusion and Open Questions

We have shown that while conducting private queries on public databases is possible without the use of hardware enclaves, the opportunity for systems-based approaches to make scalability gains is significant. Neither our upgraded oblivious file sharing system nor similar privacy-preserving systems are ready to serve hundreds of millions of users, but they provide an important first step. This demonstrates that improvement of performance while preserving privacy is possible, and more work will make our goal a reality.

Currently, it is possible that specific individuals with the right personal incentives, such as human rights workers or asylum seekers, would be willing to deal with the extra latency for the sake of privacy and security.

To serve users at scale, our system needs to be able to scale to multiple machines. There has been work on distributed ORAM in the past such as such as ObliviStore [16] and TaoStore [15], but designing a similar construction that works with 2PC and in the larger oblivious file sharing system presents a significant challenge.

One possible area for even future improvement is making the requests for the Merkle tree proof more efficient. Currently, we individually request each node in the Merkle tree proof. However, being able to do the requests and evictions in one pass would reduce the overall request time. We must be careful, however, to ensure that none of the requests reveal anything about the others, and thus hide from the server the path of the Merkle tree proof and the ultimate value of the originally requested certificate at the leaf.

As a side note, it would be helpful to see how our implementation holds up against a real-world CT server. The publicly available source code on Github by Google has two versions, a deprecated C version and a newer Golang version. Because the existing oblivious file sharing system and most of our codebase has already been implemented in C, we have been working with the deprecated C version of CT as well for simplicity's sake.

While the protocols often used in such projects are highly studied and constructed for efficiency, real-world cryptosystems are only broadly adopted if they are practical to deploy and operate at reasonable cost. We hope to continue our work on both privacy-sensitive, secure internet infrastructure, including Certificate Transparency, and scalable private systems in general.

7 Acknowledgements

I would like to thank Jean-Luc Watson and Edward Oakes, for their great contributions to the system-level improvements discussed in this paper.

I would also like to thank my advisors, Doug Tygar and Raluca Ada Popa, for their support and advice throughout this project.

Bibliography

1. H. Adkins. *An update on attempted man-in-the-middle attacks.* https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html.
2. *Announcement: Requiring Certificate Transparency in 2017.* https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/78N3SMcqUGw.
3. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. "More Efficient Oblivious Transfer and Extensions for Faster Secure Computation". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security.* CCS '13. ACM, Berlin, Germany, 2013, pp. 535–548. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516738. URL: http://doi.acm.org/10.1145/2508859.2516738.
4. A. L. B. Laurie and E. Kasper. *RFC 6962: Certificate Transparency.* https://www.rfc-editor.org/rfc/rfc6962.txt.
5. *Comodo Report of Incident - Comodo detected and thwarted an intrusion on 26-MAR-2011.* https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html.
6. M. D. Ryan. "Enhanced Certificate Transparency and End-to-End Encrypted Mail". In: *Network and Distributed System Security Symposium.* 2014. ISBN: 1-891562-35-5.
7. B. Dowling, F. Günther, U. Herath, and D. Stebila. "Secure Logging Schemes and Certificate Transparency". In: *21st European Symposium on Research in Computer Security.* Vol. 9879. 2016, pp. 140–158. ISBN: 978-3-319-45740-6.
8. S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh. "Certificate Transparency with Privacy". In: *Proceedings on Privacy Enhancing Technologies.* Vol. 4. 2017, pp. 232–247. DOI: 10.1109/SP.2018.00015. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00015.
9. O. Goldreich. "Towards a Theory of Software Protection and Simulation by Oblivious RAMs". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing.* STOC '87. ACM, New York, New York, USA, 1987, pp. 182–194. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28416. URL: http://doi.acm.org/10.1145/28395.28416.
10. G. J., O. G., A. M., and C. N. "A First Look at the CT Landscape: Certificate Transparency Logs in Practice". In: *Passive and Active Measurement Conference.* 2017.

11. D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey. "Tracking Certificate Misissuance in the Wild". In: *2018 IEEE Symposium on Security and Privacy (SP)*. Vol. 00. Pp. 288–301. DOI: 10.1109/SP.2018.00015. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00015.

12. Y. Lindell and B. Pinkas. "An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries". In: *Advances in Cryptology - EUROCRYPT 2007*. Ed. by M. Naor. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 52–78. ISBN: 978-3-540-72540-4.

13. R. C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology — CRYPTO '87*. Ed. by C. Pomerance. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 369–378. ISBN: 978-3-540-48184-3.

14. P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa. "Oblix: An Efficient Oblivious Search Index". In: *2018 IEEE Symposium on Security and Privacy (SP)*. Vol. 00. Pp. 775–792. DOI: 10.1109/SP.2018.00045. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00045.

15. C. Sahin, V. Zakhary, A. El Abbadi, H. Lin, and S. Tessaro. "Taostore: Overcoming asynchronicity in oblivious data storage". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 198–217.

16. E. Stefanov and E. Shi. "ObliviStore: High Performance Oblivious Cloud Storage". In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 253–267. DOI: 10.1109/SP.2013.25.

17. E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas. "Path ORAM: An Extremely Simple Oblivious RAM Protocol". In: vol. 65. 4. ACM, New York, NY, USA, 2018, 18:1–18:26. DOI: 10.1145/3177872. URL: http://doi.acm.org/10.1145/3177872.

18. *uthash: a hash table for C structures*. troydhanson.github.io/uthash/.

19. X. Wang, H. Chan, and E. Shi. "Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. ACM, Denver, Colorado, USA, 2015, pp. 850–861. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813634. URL: http://doi.acm.org/10.1145/2810103.2813634.

20. A. Whalley. *Distrusting WoSign and StartCom Certificates*. https://security.googleblog.com/2016/10/distrusting-wosign-and-startcom.html.

21. A. C. Yao. "How to generate and exchange secrets". In: *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25.

22. S. Zahur and D. Evans. "Obliv-C: A Language for Extensible Data-Oblivious Computation". In: *IACR Cryptology ePrint Archive* 2015, 2015, p. 1153.