Implementing O-DORY: Reproducing Secure Keyword Search in Encrypted File-storing Systems

Emmie He, *Brown University*

Abstract

O-DORY is a DORY-style [1] end-to-end encrypted file-storing application. It allows secure and efficient server-side keyword search by storing search indices as encrypted bloom filters, distributing trust among n servers (n>=2), and verifying the authenticity of search results with aggregated message authentication codes. These system and cryptographic techniques have been discussed in detail in the DORY research [1]. This report will focus on implementing and integrating these techniques into a functional application. Currently, on a 2017 MacBook Pro with a Intel(R) Core(TM) i7-7567U CPU @ 3.50GHz, uploading 100 documents that contain 1000 keywords to the servers takes 13 seconds. A single keyword search with a false positive rate of 1% over these documents takes less than 3 seconds. I consider the performance to be acceptable since the program employs little to no parallelism or computation optimizations, which leaves plenty of room for performance improvement in the future.

1. Introduction

With the popularity of cloud storage services, users have increasingly higher requirements for data and system security. For data privacy and security considerations, more and more people resort to end-to-end encrypted storage systems. These systems protect data safety by storing only encrypted data. Due to the fact that the servers have no knowledge of the original data, it is often challenging to provide secure and efficient server-side keyword search and document retrieval functionality without exposing search access patterns. DORY (Decentralized Oblivious Retrieval sYstem) is a search system that addresses this problem by encrypting searches and distributing trust between multiple servers [1].

This report describes the implementation of a DORY-style file-storing application O-DORY that utilizes related cryptographic and system techniques to allow server-side encrypted search. The implementation focuses on keyword search functionality rather than document encryption. One difference between O-DORY and DORY is that O-DORY does not support data sharing, meaning each document will be uploaded and queried by the same user.

2. Background & Related Techniques

DORY proposes to use (1) encrypted bloom filters to space-efficiently store the keyword membership information for documents, (2) Distributed Point Functions (DPFs) to distribute the keyword search on n servers and assemble the search results on the client-side so that the search access patterns are kept secret even if (n-1) servers are compromised, and (3) Message Authentication Codes to verify the authenticity of the received search results.

O-DORY implementation involves the following concepts and techniques from DORY.

2.1. Document Bitmap

A document bitmap is a mapping from a range of words to bits. The bitmap is uploaded or updated every time a client inserts or updates a document. Each server holds a bitmap table. In this table, each column represents a keyword while each row represents a stored document [1].

2.2. Bloom Filter

DORY leverages bloom filters [5] to further compress document bitmaps. Each bit should be masked to avoid search index leaking [1]. This design effectively reduces space for storage and performance overhead for queries.

2.3. Private Information Retrieval (PIR)

The PIR protocol enables a user to retrieve information from a server database without it knowing which item is retrieved [2].

2.4. Distributed Point Functions (DPF)

DPF is a cryptographic primitive that allows distributing a secret between two processes without revealing the secret to any of the processes [3, 4]. DORY uses this algorithm to distribute trust for search access patterns [1].

2.5. Aggregate Message Authentication Codes (Aggregate MACs)

MACs allow the DORY client to check the integrity of the search results [1]. Aggregate MACs produce a shorter tag for verifying multiple MAC tags [6].

3. Design & Implementation

O-DORY runs multiple servers that store encrypted data and the metadata necessary to perform keyword searches. End users can run a client locally to upload/update/remove files and to search for documents that contain certain keywords.

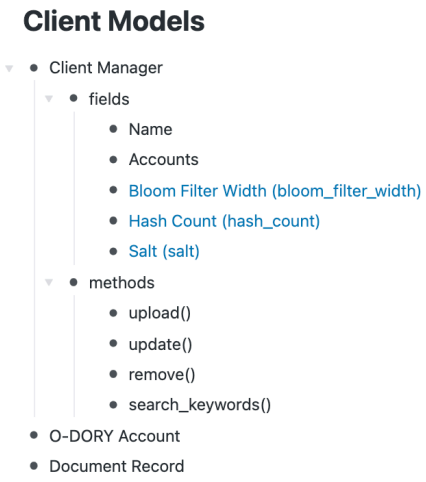
Each server executes search requests without knowing the searched terms or the search access patterns.

O-DORY is implemented with an open-source web application framework Odoo [10]. The majority of the implementation is in Python 3. Following the Model-View-Controller architecture, Odoo provides the ORM layer and view templates. The O-DORY server and client also add Odoo's open-source modules "base" and "web" as dependencies for user authentication and interface setup [11, 12, 13]. The O-DORY server and client communicate with each other over XML-RPC.

O-DORY uses the Sycret Python library [14] to generate secret sharing functions.

3.1. Client

3.1.1. Models



** The blue fields are set once during creation and hidden from the users*

The O-DORY client has three data models: the client manager, the O-DORY account, and the document record.

The client manager holds information about the user's server accounts and main functions that enable the user to communicate to the server. The client manager has fields like bloom filter width and hash count, which default to 9585 and 7 respectively and are hidden from the user. These technical fields should be only set once when a user creates an account. The default values allow a 1% false-positive rate for 1000 keywords. The client manager also generates a unique random string (salt) upon creation. This salt corresponds to the per-folder key mentioned in the DORY research paper.

The document record model is for users to take notes about the documents they uploaded. This model's data is only for the client's convenience and is never sent to the server.

The client also has a set of transient models (called "wizards" in Odoo) [15]. These wizards help to store user input temporarily for a better user experience.

3.1.2. Upload

When a user uploads a file to the servers, the client extracts all the words from the file, creates a bloom filter according to the client manager's bloom filter width and hash count value, computes each word's bloom filter indices, and sets the bits of those indices to 1. It also generates a random string as the version number for this document and uses this document version and the client-side salt to mask the bloom filter.

Next, the client acquires the existing Message Authentication Codes (MACs) for this user from the server, generates a new list of MACs for all the bits in the current bloom filter with the document version, and XORs the new list with the existing MACs to get the updated MACs.

Finally, the client sends the encrypted data, the masked bloom filter, and the updated MACs to the servers.

3.1.3. Update

The update process is similar to upload, except that updating MACs requires both XORing out the previous version's MACs and XORing in the new MACs.

3.1.4. Remove

Removing a document from the client sends requests to the servers to remove the related bloom filter entries from their bitmap tables.

The client also sends the new aggregated MACS (by XORing out the removed document's MACs) to the servers.

3.1.5. Search

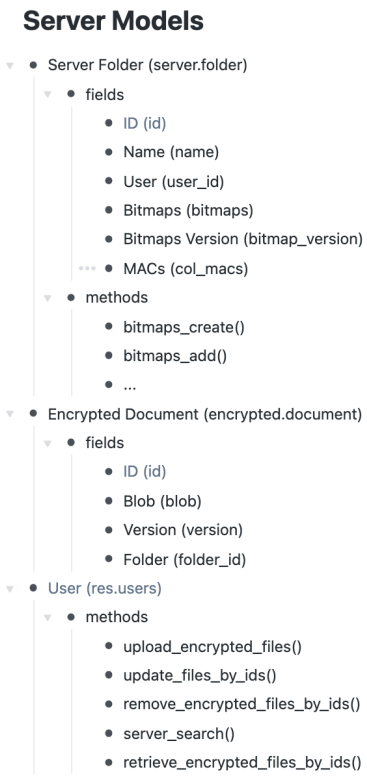
When a user initiates a keyword search (from the keyword search wizard), the client first verifies the bitmaps consistency between servers and retrieves the aggregated MACs. It then computes the supplied keyword's bloom filter indices and prepares the corresponding distributed point functions (in the format of an input x - an encrypted integer list - and two keys key_a, key_b). The client sends input x and key_a to server one and input x and key_b to server two.

After receiving the responses from both servers, the client assembles the responses and verifies if the MACs for the targeted indices match the previously retrieved aggregated MACs. This is to examine whether the search bitmaps are tampered with on the servers or not.

Once the authenticity of the search result is verified, the client unmasks the targeted bits and finds the documents (rows) that match the keyword (columns).

3.2. Server

3.2.1. Models



** The grey items are Odoo default models/fields*

The O-DORY server contains two main data models: the server folder and the encrypted document. Each server folder holds the encrypted bloom filters (bitmaps) and aggregated MACs (col_macs) for each user. The bitmaps are implemented with a Python dictionary that maps each document ID to its bloom filter (Python list) instead of using actual bit arrays. All bitmaps-related operations (create, add, remove, flip, etc.) are attributed to the server folder model. Therefore, it should be fairly convenient for any future work to replace or optimize the bitmaps implementation.

The O-DORY server handles document uploading/updating/removing/searching requests under an inherited user model (res.users) [16].

3.2.2. Upload, Update, & Remove

The server updates the corresponding bitmaps with the supplied bloom filter entries or document IDs accordingly.

3.2.3. Search

The server performs the keyword search after evaluating the input x (an encrypted, bloom filter column-sized integer list) and a key from the client. The evaluated output represents an encoded binary vector that indicates which columns we want to search on (recall that each server will only get "half" of the original binary vector). The server then goes over each document's bloom filter and computes, for each column, the AND result of the bloom filter entry and the output bit. Finally, the server sends the result matrix back to the client.

In O-DORY, all servers are replicas of each other. In order for the user to perform a search, there must be at least two available servers that hold the same bitmaps for that user. O-DORY verifies consistency when performing operations - when the servers are not consistent, it raises a validation error to the client. Resolving server inconsistency is out of the scope of this project.

3.3. Implementation

The server module has less than 400 lines of code, the client module has less than 1000 lines of code, and the testing script has less than 500 lines of code (calculated with cloc [17]).

For installation and other instructions, please see the GitHub repository: https://github.com/emmiehe/o-dory.

4. Evaluation

4.1. Script

To test uploading, searching, and removing functionalities, I have included a script "upload_and_search.py". The script expects a series of inputs: username, number of keywords, false-positive rate, number of documents, a “needle” word to search for, auto remove flag, and async flag.

Running "python3 upload_and_search.py Bob 100 0.1 10 needle 1 0" will create a user/folder for Bob and create Bob's client manager with 100 keywords and a 10% false positive rate (the keyword number and false-positive rate inputs help determine the bloom filter width and hash count values in the O-DORY client). The script will then upload 10 documents that contain 100 random strings. A random subset of these documents will contain the word "needle". The script will then perform an O-DORY keyword search and compare the search result with the expected result. After the comparison, when the auto remove flag is set to 1, the script will remove all the objects it has created; otherwise, the script will keep all the objects and the user can go to the

O-DORY client/server in a web browser to examine the data and objects.

The async flag allows the keyword searching functionality to be multi-processed. Currently, the multiprocessing implementation is still in progress and may not speed up as expected.

4.2. Experiments

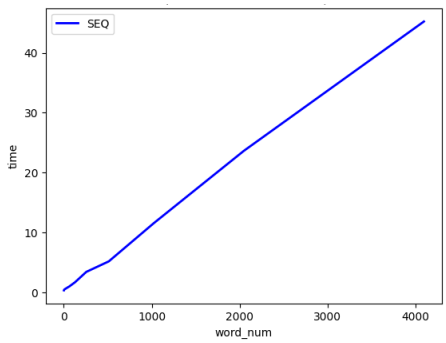


Figure 4.2.1.

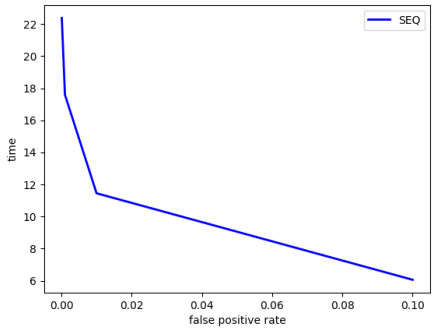


Figure 4.2.2.

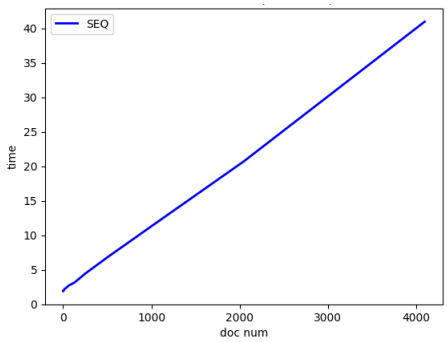


Figure 4.2.3.

Figure 4.2.1. shows the relationship between search time and the number of keywords, with the false positive rate being 1% and the number of documents 1024.

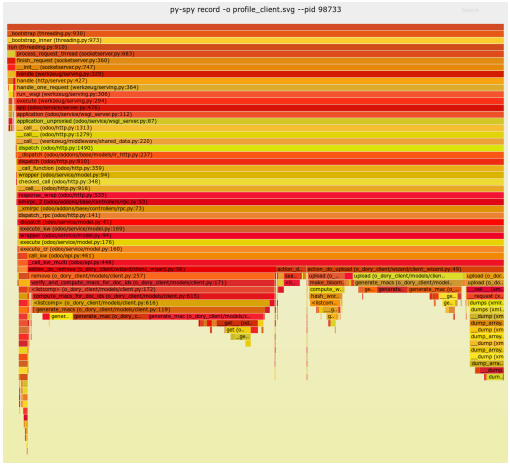
Figure 4.2.2. shows the relationship between search time and the false positive rate, with the number of keywords being 1024 and the number of documents 1024.

Figure 4.2.3. shows the relationship between search time and the number of documents, with the false positive rate being 1% and the number of keywords 1024.

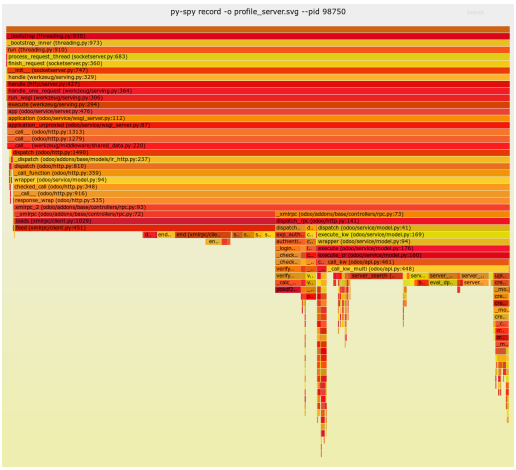
Not surprisingly, the amount of time needed for O-DORY to perform a keyword search grows linearly with higher numbers of keywords and higher numbers of documents. The amount of time needed for O-DORY to perform a keyword search grows exponentially with lower false-positive rates.

5. Future Work

The current implementation of O-DORY heavily focuses on the keyword search functionality and does not employ any optimizations or parallelism for uploading and removing documents. As shown in the client flame graph [18], when running the script specifying 1000 keywords, 100 documents, and a false-positive rate of 1%, the majority of time is spent on document uploading and removing (more specifically, the MACs generation and computation). The server flame graph also indicates that data deserialization (loads) takes a significant amount of time. Working on advancing these operations will improve O-DORY's performance in the future.



Client Flame Graph



Server Flame Graph

References

[1] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119. USENIX Association, November 2020. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/dauterman-dory

[2] Chor, Benny & Kushilevitz, Eyal & Goldreich, Oded. (1998). Private Information Retrieval. J. ACM. 45. 965-981. 10.1145/293347.293350.

[3] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *EUROCRYPT*, pages 640–658. Springer, 2014.

[4] Distributed point function. https://en.wikipedia.org/wiki/Distributed_point_function. (Accessed: September 2021)

[5] Bloom filter. https://en.wikipedia.org/wiki/Bloom_filter. (Accessed: September 2021)

[6] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Cryptographers' Track at the RSA Conference*, pages 155–169, 2008.

[7] Advanced Encryption Standard. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. (Accessed: September 2021)

[8] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. Journal of the ACM (JACM), 43(3):431–473, 1996.

[9] F.31.1. Trigram (or Trigraph) Concepts. https://www.postgresql.org/docs/9.6/pgtrgm.html. (Accessed: September 2021)

[10] Odoo. Open Source Apps To Grow Your Business. https://github.com/odoo/odoo. (Accessed: October 2021)

[11] Odoo. Architecture Overview. https://www.odoo.com/documentation/15.0/developer/howtos/rdtraining/01_architecture.html. (Accessed: December 2021)

[12] Odoo. Base Module. https://github.com/odoo/odoo/tree/15.0/odoo/addons/base. (Accessed: December 2021)

[13] Odoo. Web Module. https://github.com/odoo/odoo/tree/15.0/addons/web. (Accessed: December 2021)

[14] Sycret. Function Secret Sharing library for Python and Rust with hardware acceleration. https://github.com/OpenMined/sycret. (Accessed: December 2021)

[15] Odoo. Wizard. https://www.odoo.com/documentation/15.0/developer/howtos/backend.html. (Accessed: December 2021)

[16] Odoo. Inheritance. https://www.odoo.com/documentation/15.0/developer/howtos/rdtraining/13_inheritance.html. (Accessed: December 2021)

[17] cloc. Count Lines of Code. https://github.com/AlDanial/cloc/. (Accessed: December 2021)

[18] Brendan Gregg. Flame Graphs. https://www.brendangregg.com/flamegraphs.html. (Accessed: December 2021)