Tackling Prying Eyes in The Public Cloud: A Privacy-Preserving Search Tool for Sensitive Information

An updated problem description, solution approach, and work plan submitted in partial fulfilment of the requirements for the degree of:

BACHELOR OF SCIENCE in Computer Science

in:

The Queen’s University of Belfast

by:

Nathan James Totten

22/04/2024

GitLab Repository [1]

# 0.1 Acknowledgements

I would like to begin by expressing my gratitude towards Dr Indranil Ghosh-Ray for his ongoing support with this project. Without his expertise on this subject the work would surely not be possible.

I would also like to give my thanks to Moira Watson for the coordination of the final year project opportunity, as well as the guidance offered to help students like me hit the ground running on a piece of academic work such as this.

Finally, I extend my thanks and appreciation to the Student Wellbeing Service and staff within EEECS who have offered their support and understanding at every stage of this degree, without the help on offer I might not have managed to balance the demands of life and university enough to reach this stage.

# 0.2 Abstract

The proliferation of cloud computing has led to a record number of individuals and opting to outsource their data storage to the public cloud. This comes with the security concerns that surround any cloud service provider. Users want privacy, and to do so they must encrypt the data they store in the public cloud. This paper will develop on the work of my supervisor [2] – Dr Indranil Ghosh-Ray – by implementing an efficient, search-friendly tool for symmetric-key encryption of cloud data. The aim of this project will be to preserve privacy of the client’s data at all stages, whilst maximising the utility of the encrypted data by enabling lightweight keyword searching.

Table of Contents

[0.1 Acknowledgements 2](#_Toc163398482)

[0.2 Abstract 2](#_Toc163398483)

[1 Introduction and Problem Area 5](#_Toc163398484)

[1.1 The Cloud – An Overview 5](#_Toc163398485)

[1.2 The Problem Area 5](#_Toc163398486)

[1.3 Security Questions and Considerations 6](#_Toc163398487)

[1.3.1 What is Symmetric Encryption? 6](#_Toc163398488)

[1.3.2 What is Searchable Encryption? 6](#_Toc163398489)

[1.3.3 How does Searchable Encryption interest users of public cloud services? 6](#_Toc163398490)

[1.4 Building on An Existing Approach 7](#_Toc163398491)

[1.4.1 Users of Proposed Solution 7](#_Toc163398492)

[1.4.2 Performance Considerations 7](#_Toc163398493)

[2 System Requirements and Specification 9](#_Toc163398494)

[2.1 Functional Requirements 9](#_Toc163398495)

[2.2 Non-Functional Requirements 10](#_Toc163398496)

[2.3 Use Cases 10](#_Toc163398497)

[3 Design 13](#_Toc163398498)

[3.1 Technology Stack 13](#_Toc163398499)

[3.2 System Architecture 13](#_Toc163398500)

[3.3 Data Design 13](#_Toc163398501)

[3.4 Security Design 13](#_Toc163398502)

[3.5 Future Considerations 13](#_Toc163398503)

[4 Implementation 13](#_Toc163398504)

[5 Testing 13](#_Toc163398505)

[6 System Evaluation & Experimental Results 13](#_Toc163398506)

[References 15](#_Toc163398507)

# Introduction and Problem Area

## The Cloud – An Overview

We can draw a concrete definition of The Cloud from [3], “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

The Cloud affects us all in more ways than we might think, from storing photos and documents in the cloud with services such as Google Photos and Google Drive, to social media ( e.g. Facebook ) and e-commerce (e.g. Amazon ), we all make use of cloud services day-to-day, often without much thought to how it works under the hood.

At a high level, the cloud can be imagined as a vast network of computers, servers, and storage that is connected to the public internet. Instead of storing your data and performing computations and logic on your own device(s), you can have the cloud handle most of the heavy lifting. It allows us to conveniently access our data anytime, anywhere, through a complex network of data centres across what are commonly known as availability zones, all around the world.

The cloud enables low-cost mass storage of encrypted data to resource constrained individuals and businesses, allowing for high availability, scalability, and resilience that previously would have been prohibitively expensive for many. This mass storage of encrypted data is difficult to search through and process, which – as discussed in [2] gives rise to an emerging field of research called *searchable encryption* (SE).

## The Problem Area

The problem faced by this project is one of solving cloud data privacy concerns. This project aims to provide an accessible solution to securing your data in the public cloud by going beyond simple encryption of the data that clients store.

To fully realise the benefits offered by the cloud, a level of utility must be preserved with client data that does not come at the cost of security.

The scheme proposed in this project – A new lightweight Symmetric Searchable Encryption (SSE) Scheme for public and commercial use – aims to solve the problem of securely accessing and processing clients’ data in the cloud.

## Security Questions and Considerations

As the solution will focus on the public cloud, this raises some concerns around data security. Users must be assured that their sensitive data is secure at every stage of processing and storage on the public cloud, so it is critical that the solution uses a vetted and tested encryption scheme. For this project, AES (Advanced Encryption Standard)[4] will be used as it has proven itself as a dependable and robust encryption algorithm with its adoption as a U.S. Federal Government Standard.

### What is Symmetric Encryption?

Symmetric Encryption is the practice of using the same cryptographic encryption key(s) for both the encryption of plaintext and the decryption of the resultant ciphertext. It can be defined as “the result of the algorithms Gen, Enc, Dec” as outlined in *Definition 1* of [2]

### What is Searchable Encryption?

Searchable encryption concerns the inner workings of how we securely access and process our data in the cloud. The scheme operates across the client side, and the server side – with only the client having visibility to any of the plaintext data. We store the encrypted data on the server using the data owner’s secret key. A secure index table is then generated to create index values/tokens for each of the words in each encrypted file – the index values are encrypted and hashed in such a way that no information about the underlying plaintext is revealed to the server. A series of trapdoors can then be used to search through the encrypted server-side data and retrieve an encrypted file identifier or identifiers for any files matching the search key word. The user may then use this to retrieve and decrypt a file on their end.

“Searchable (symmetric) encryption allows encryption while still enabling search for keywords. Its immediate application is cloud storage where a client outsources its files while the (cloud) service provider should search and selectively retrieve those.” [5]

### How does Searchable Encryption interest users of public cloud services?

As discussed above, in an evolving technologically connected landscape that affects us all, the average user of online services is becoming increasingly concerned with the whereabouts and security of their data. The news of cyber-attacks, be it carefully organised large scale operations or simple social engineering schemes and scams has instilled a new factor for us all to consider in how we conduct our lives – we are grappling with the reality that cyber security and mindfulness over how we manage our data is more important than ever before. It is unlikely that this will ever change.

In the face of this, adopters of the cloud have found themselves increasingly interested in how they can protect what is theirs when it is being routed, stored, and processed on the near-unavoidable public cloud. This is where a lightweight searchable encryption scheme comes in. By proposing a solution to cloud data security that maintains the convenience and utility that we come to expect from the cloud, but bolsters security, users can place more faith in the cloud and the services it enables. This may allow for further adoption of the tools that allow us to continually connect and grow together – across the world.

## Building on An Existing Approach

My solution will aim to understand and build on the work carried out by my supervisor and their colleagues [2]. An area of proposed further work is to improve the efficiency / performance of the SSE scheme. This will be the goal of my project.

Many existing pieces of literature focus on the theoretical applications and experimentation on SSE (Searchable Symmetric Encryption), my paper will differ with its focus on implementing a modified variant of the SSE Scheme from [2] as an efficient, portable, and user-friendly Java CLI application.

### Users of Proposed Solution

The implementation of the scheme will be carried out in such a manner that little to no technical knowledge is required to utilise the security offered. As such, it is the hope of the project to make this symmetric searchable encryption scheme easy to understand and use for both private/personal users, as well as commercial clients. As the focus of this project is on providing a performant implementation of the scheme in [2] I will broadly consider any user to simply be the Data Owner.

An area of proposed future extension would be to implement suitable authentication to enable encrypted searching through the document collection by those who are not the Data Owner but have been given permissions to search over a subset of documents.

### Performance Considerations

My aim will be to consider the compute performance and efficiency of the proposed solution. I will need to strike a balance between security and speed and adjust the tools and approaches I use accordingly. For example, the overhead of securely generating secret keys, building the secure index table, and crucially, the processing of large plaintext files will likely be the major pain points when developing an efficient and lightweight key-word search approach.

Network overheads for large volumes of traffic would be another consideration in a real-world deployment – as discussed in [2] however the proof-of-concept tool that is being developed in this project will not handle any LAN or WAN traffic, and so this will not be a considerable factor due to the considerably higher speed and bandwidth of any on-machine data processing.

# System Requirements and Specification

## Functional Requirements

This encryption scheme does not make distinction between the types of users it interacts with – as such the fundamental requirements can be neatly summarised as follows:

1. As a user, I must be able to generate a set of secret encryption keys in line with the AES-GCM standards. [6], [7]
   1. A Data-Owner Secret Key – to encrypt the file data in a manner that hides it from the server.
   2. A Mask Secret Key – for the secure search functionality, generating trapdoors etc.
   3. A Server/Session Secret Key – for the generation of the secure index and searching functionality.
2. As a user, I must be able to upload a specified file or files by providing a valid directory.
   1. The system must encrypt any files uploaded using .
   2. The system must encrypt any files on the client-side before they are stored on the server. This ensures the server never obtains details on the contents of the files.
3. As a user, I must be able to retrieve and decrypt a file of my choosing without exposing plaintext data to the server.
   1. The system must use an encrypted, secure file ID to store and retrieve the file so that the server cannot gain info on the potential contents of the encrypted file (e.g. no plaintext filenames such as joe\_bloggs\_health\_data.txt).
   2. Decryption will be done on the client-side using to avoid unveiling plaintext data to the server.
4. The system must generate a secure index table that enables privacy-preserving searching through the encrypted document collection on the server-side.
   1. This secure index table must not reveal information to the server that compromises the secrecy of the encrypted documents.
5. As a user, I must be able to search for a keyword and see file(s) that contain a match for the provided search term.
   1. The file ID cannot be decrypted until it is on the client-side – this prevents the server from gaining information on the possible file contents.
   2. The system must automatically generate the necessary trapdoors to enable searching for the keyword in the encrypted document collection.

## Non-Functional Requirements

As for non-functional requirements, I will focus on Usability, Performance, Maintainability, Security, Portability, and Reliability.

* The system must provide clear instructions or help information for the user that will be available at any stage of program execution.
* The system must use clear and intuitive commands for each of its use cases.
* The system must provide clear feedback to the user in the form of warnings, error messages, and success messages throughout the user’s interaction with the system – It should handle errors gracefully.
* The system should abstract unnecessary complexity from the user to lower the barrier to entry and automate features that do not need to be manually executed by the user.
* The system should take on average less than 500ms to respond to any user action.
* The system should sanitise and validate inputs as necessary to prevent injection attacks and handle general user error gracefully.
* The application codebase should be clearly structured and well-documented to improve ease of maintenance and future extension.
* The system should be able to handle large datasets without significant degradation in performance or user experience.

## A diagram of a user Description automatically generatedUse Cases

At a high-level, the requirements can be seen in Figure 1. The user must be able to:

* Generate Private Encryption Keys

Upload documents to be encrypted and securely stored.

* Generate the secure index table to enable searching of the encrypted documents on the server-side.
* Searching for a keyword on the server-side using the secure index table.
* Decrypting and exporting a document on demand based on results of the searching operation.

Figure 1 - High-level use case overview.

A diagram of a person's encrypted key

Description automatically generated**Generating Encryption Keys:**

KeyGen encompasses the generation of each required secret key and the large probabilistic prime number security parameter.

Figure 2 – UC1 Key Generation and Security Param.

The Data Owner key and Mask Key must always be kept secure in the hands of the User – the Server Key may be given to the server without compromising the plaintext data. This is required for secure keyword search.  
**Upload Document(s):**

A diagram of a document utility

Description automatically generated

Figure 3 – UC2 Document Upload and Encryption

Document Upload will enable the user to upload a specific file, or all files in a provided directory. This must automatically encrypt any uploaded documents using the Data Owner Key before any data is passed on to the server.

**Build Secure Index:**

A diagram of a secure index

Description automatically generatedThe Secure Index table will be used in conjunction with trapdoors to enable matching of encrypted documents with a provided keyword search term.  
**Search for Keyword:**

Figure 4 - UC3 Build Secure Index Table

A diagram of a search engine

Description automatically generated

Figure 5 - UC4 Search for Keyword

Searching will require a valid keyword before being passed off to the final implementation of the searching algorithm. The output will consist of encrypted file identifiers of any documents containing the keyword. These IDs will only be decrypted on the client-side to avoid leakage of encrypted information to the server.

**Decrypt Document:**

A diagram of a document

Description automatically generatedFile decryption will only occur on the client-side. File IDs will be encrypted before passing to server.

Figure 6 - UC5 Retrieve and Decrypt Document

# Design

## Technology Stack

## System Architecture

## Data Design

## Security Design

## Future Considerations

# Implementation

# Testing

# System Evaluation & Experimental Results

[Figure 1 - High-level use case overview. 10](#_Toc163398472)

[Figure 2 – UC1 Key Generation and Security Param. 11](#_Toc163398473)

[Figure 3 – UC2 Document Upload and Encryption 11](#_Toc163398474)

[Figure 4 - UC3 Build Secure Index Table 12](#_Toc163398475)

[Figure 5 - UC4 Search for Keyword 12](#_Toc163398476)

[Figure 6 - UC5 Retrieve and Decrypt Document 12](#_Toc163398477)

# References

[1] Nathan James Totten, ‘GitLab Repository’. [Online]. Available: https://gitlab.eeecs.qub.ac.uk/40290303/cs\_final\_year\_project\_40290303

[2] I. G. Ray, Y. Rahulamathavan, and M. Rajarajan, ‘A New Lightweight Symmetric Searchable Encryption Scheme for String Identification’, *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 672–684, Jul. 2020, doi: 10.1109/TCC.2018.2820014.

[3] P. M. Mell and T. Grance, ‘The NIST definition of cloud computing’, Gaithersburg, MD, 2011. doi: 10.6028/NIST.SP.800-145.

[4] M. J. Dworkin, ‘Advanced Encryption Standard (AES)’, May 2023. doi: 10.6028/NIST.FIPS.197-upd1.

[5] F. Hahn and F. Kerschbaum, ‘Searchable encryption with secure and efficient updates’, in *Proceedings of the ACM Conference on Computer and Communications Security*, Association for Computing Machinery, Nov. 2014, pp. 310–320. doi: 10.1145/2660267.2660297.

[6] M. Dworkin, ‘Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality’, Jul. 2007, doi: 10.6028/NIST.SP.800-38C.

[7] D. L. Evans and K. H. Brown, ‘Advanced Encryption Standard (AES)’, May 2023, doi: 10.6028/NIST.FIPS.197-UPD1.

# 