Searchable Encryption

The University of Texas at Dallas
Juhi Patel - jmp170130@utdallas.edu
Niklas Borcanski - nab160130@utdallas.edu
Dibbya Saha - dxs210061@utdallas.edu
Sarah Burris - seb170005@utdallas.edu

Project Summary

Our goal is to build a client application that can interface with a MySQL database such that the records it inserts into the database have certain columns hidden from the server. The user should be able to search on these columns by specific value. An example would be a shared accounting database system that stores names and social security numbers of customers. The SSNs would be encrypted such that only the accountant inserting records for their own customers would be able to search on and decrypt said records.

Tools

At the onset of implementation, deciding which tools would best fit our needs was imperative. Tool selection included choice of language, cloud service, relational database management system and libraries.

We chose to use the cryptography package in Python. The package provides cryptographic recipes and primitives to Python developers. The package also includes recipes and interfaces for common cryptographic algorithms. The primary function of this library was to provide key generation, encryption and decryption capabilities. The mysql.connector packages were discussed while choosing a language. To achieve the described functionality, the connect() method within mysql.connector was used. This method establishes a connection to a MySQL server. We also used the cursor class to execute operations on the database. Finally the base64 package is also used to encode and decode salts read from a file.

Major Steps of Implementation

Due to choosing a project implementing a method to search an encrypted cloud database, there were a variety of major steps required for completion. The first being the determination of a method to search encrypted data. A significant amount of research was needed to find the best fit for implementation within this semester. The team read white papers for theoretical implementation of Searchable Symmetric Encryption, Secure Keyword Search and a review of a variety of searchable encryption methods. The solution we chose was a hybrid one, based on a number of articles as well as inspired by AcraDB, with a simple client-server architecture instead of a proxy service.

Once a methodology was agreed upon, we began the process of creating the infrastructure needed to complete the project. Our remote database was hosted on AWS and populated with the example world schema published in the MySQL documentation.

After this, we decided on logistics such as which language we wanted to do the project in, what method we wanted to use to encrypt the columns of the database, which column of the database we wanted to encrypt, and how we wanted users to access the data. We decided to use Python due to various built in cryptographic libraries that we were already familiar with within the language. The data is encrypted with AES-GCM with the username of the user provided as additional data and an IV chosen randomly. The blind index value is created using the Scrypt key derivation function with a derived index key provided as the salt. We chose to store all the generated values (encrypted data, IV, and verification tag) in separate columns for the sake of simplicity. In addition, we designed a simple UI functionality to assist users with searching for the data they need.

Once we finished the program that would encrypt selected columns of the database, we decided to create a helper program that would automatically inject these values into said database. After running the helper program, the database would be fully populated with the searchable encrypted data. This data would then be searched upon using the user-interface that we designed.

Division of Labor

Task / Deliverable	Person In Charge
Whitepaper Research	Everyone
Brainstorming Roadmap	Everyone
Create Roadmap Diagrams/Written Plans	Sarah
Database Set-up and management	Juhi
Program to encrypt columns of database	Niklas, Dibbya
Programming GUI	Dibbya
Program to inject encrypted columns into database	Juhi
Report Writing	Juhi, Sarah

Project Specifications

Client Authentication

The client does not provide a direct authentication service. For the sake of simplicity, we assume that the username and password the client is using to encrypt the data is the same as the login information for the database user inserting the records. The client stores saved usernames in a file on the system called users.json along with the encryption key salt index key salt associated with said username. When the program starts, it reads this file into memory as a dictionary and checks to see if the inserted username exists. If it does, it uses those salt values along with a user-entered password to generate the index key and encryption key using Scrypt. If not, it generates new salts for the user and saves the file back to disk.

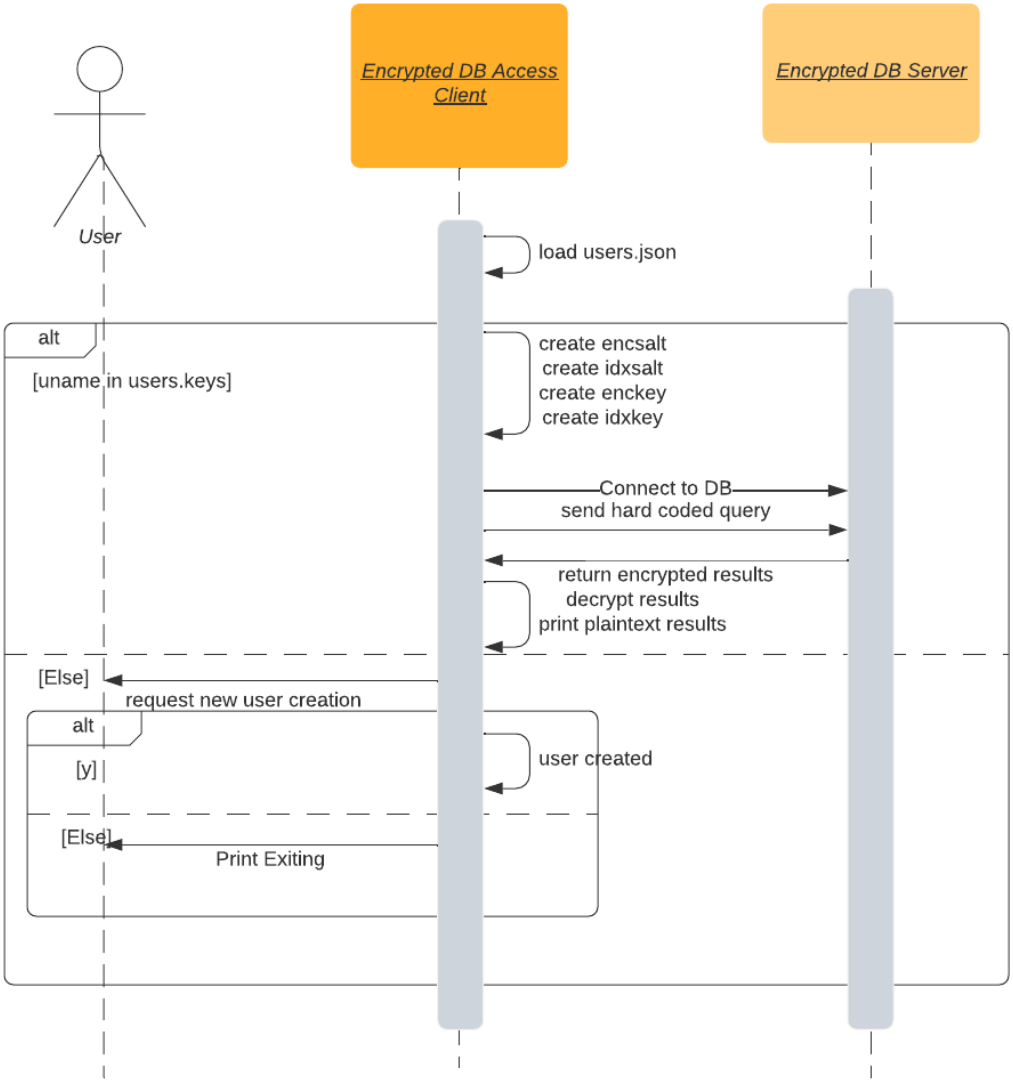
Encryption

To encrypt a record, the client encrypts the data of the encrypted column using AES-GCM, with the username of the user provided as additional data, the encryption key derived earlier as the key, and a random IV. Then, it generates a blind index value by running Scrypt using the plaintext value as the password and the index key derived earlier as the salt. The blind index value, the encrypted data, the verification tag, and the IV together form one record. We chose to store these all in different columns, but really only the blind index values need to be separate.

Searching

To search, the user simply inputs a specific value they wish to search for into the blind index generation function using the index key derived from their index key salt and their password to generate a blind index value. This value can be searched for in the database. They can then decrypt using the encryption key derived from their encryption key salt and their password and verify the tag to check integrity and authenticity. For the sake of simplicity, we chose a column to encrypt that uniquely identifies each record, but if multiple records are returned, all can simply attempt to be decrypted.

UML Sequence Diagram



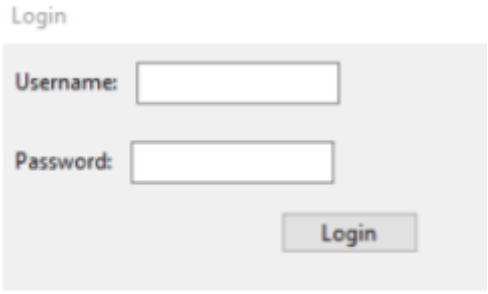
Difficulties Faced

The main difficulty faced by the team was finding a technique that addressed the use case of our application. After researching different methods, we originally considered implementing Searchable Symmetric Encryption. However, this solution seemed more geared toward use in a document set indexed by keywords, such as a MongoDB database or an inverted index for a search engine, rather than a simple MySQL database.

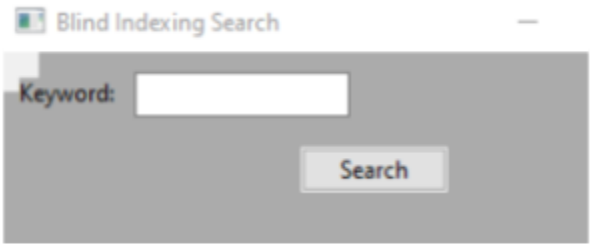
The blind indexing solution, inspired partially by AcraDB, provided the kind of privacy service we were looking for. Our identifying attributes, of which we chose one in our test database randomly, could be simply encrypted using a non-deterministic scheme, and would still be searchable on the derived index. This simple, secure solution gave our application the flexibility it needed to interface with a real-world database solution.

After we had researched the correct solution, the difficulties that followed involved finding the right cryptographic libraries that would allow us to implement the methods described in our research, creating a GUI that would integrate with our program, and creating and integrating a database within AWS. However, most of these problems were due to inexperience, so we were able to solve them with enough time and research.

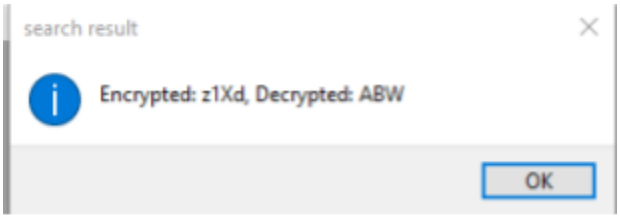
Software Usage and Screenshots



User Log-in Screen



Keyword search interface



Search results window