Group 3 project materials documentation: Application of cryptDB to a chat application

Kendric Hood[1], Adewole Kazeem Babatunde[2], Yuan Chen[3]

Computer Science, Kent State University
800 E Summit St, Kent, OH 44240

[1]khood5@kent.edu

[2]kadewole@kent.edu

[3]ychen135@kent.edu

***Abstract*— We address the problem of developing a real time chat application that maintains a user's privacy while utilizing the benefits of cloud computing. Chat applications often need to maintain data in a place other than the client system. This eases the transition from one device to another and allows users to join chats without other users needing to have their client systems online. That is the server-client architecture is very popular for chat applications. In recent years 'the cloud' has become a popular way to host and maintain servers for companies. 'The cloud' is a data center hosted by a third party where a company can easily rent computation and storage. While this makes maintenance and upkeep of the server side hardware easier on the company, it puts the users privacy at risk as the third party may only be semi-trusted. Maintaining the user's privacy while taking advantage of 'the cloud' is thus desirable for companies exploring cloud computing as an alternative to their own in-house infrastructure.**

***Keywords*— cloud computing, privacy, chat applications**

I. Introduction

Over the years chat applications have become widely popular. A chat application is software that allows users to send short messages to one another. These messages may include text, images, or videos. Users use these applications both in 1-1 communications and 1-many in 'group chats'. Typically chat applications are developed and maintained by a company. The users are assumed to trust the company that maintains their chat applications. It is assumed by the user that their data (messages) are viewable by the company that makes the application. Oftentimes, the company collects this user data as payment for the application. That is, they forgo payment for the application in extension for access to users data that can be analyzed and sold to third parties. In the process of analysis of the data it may be assumed, whether it is true or not, that the company in question properly anonymizes the data so that the third parties can not gain insight into individuals. Users may disagree on which chat applications companies are truly maintaining their privacy. For example one user may assume that google does indeed anonymize their data but microsoft doesn't, another user may hold the opposite belief. For this reason when a user agrees to use a chat application, they are only consenting to the company that develops and maintains it to have access to their information.

Chat applications commonly follow the server-client architecture. In this architecture the application is hosted on a central machine known as a server. While users access the application via clients. Clients are assumed to be non-permanent and not tied to a specific user. A user may share a client with many other users, or switch between multiple clients regularly. For example a user may share their desktop computer with their whole family and use a cell phone that only they have access to. For the user to have consistent use of the application all data must be stored off the client system in the server. Thus the company that develops the chat application must maintain all user data as well as the application itself.

Many companies that maintain these applications incur high cost in terms of maintenance, monitoring, and upgrading of their servers. This is necessary as the number of users grows and shrinks. In addition the company must maintain other services such as networking. Often many in-house

resources go unused for large portions of time. This is due to the company needing to maintain service reliability during peak usage. In other words, the company is forced to maintain enough resources to support the peak usage at all times, even if the average usage is much lower. As companies seek to lower this cost, cloud computing offers a solution.

Cloud computing allows companies to rent resources on demand. These resources could be computing power, storage, network bandwidth etc. Cloud computing is, in brief terms, a data center for renting. A third party builds, maintains, and upgrades a large pool of resources. Then rents these resources out to other companies on demand. As the cost of renting enough resources for peak usage, only during peak usage, may prove to be much more cost effective for the company's renting. While the cloud computing company can work with multiple other companies to maintain the most usage of their resources at all times. While this seems like a win-win for the companies in question, the user's privacy may be at risk. Users only consent to the chat application company having access to their data, not any cloud provider. Thus chat application providers must take steps to maintain their users privacy if they are to enjoy the use of cloud computing.

We address this issue by developing a chat application that maintains a user's privacy in the cloud. We use cryptDB to make sure that the cloud provider can not see user messages while still leveraging the uses of cloud storage. In section II we outline our applications design and implementation. Section III We discuss how we maintain our users' privacy. Section IV discusses the strengths and weaknesses of our application. Section V we offer a live demo. In section VI we discuss future work. Finally section VII includes our concluding remarks.

II. DESIGN AND IMPLEMENTATION

First we discuss what we consider privacy for this use case. We outline our preliminary assumptions. We present our design consisting of three components: user interface (UI), proxy, and cloud hosted databases (DB). Finally we will briefly overview how each component works together.

Privacy & Preliminary Assumptions

The sender of a message is the user that authors and sends a massage. The users that receive the message are the recipients. We define the problem of privacy in the case of chat applications with two properties.

1. Knowledge: No entity knows *what* the sender sent expect for the recipients
2. Community: No entity knows *who* the recipients are except for the sender and the other recipients.

For the first property, a sender's messages should only be visible by those they want to see and no one else. For the second property, who sees them, the identities of those reading the messages should only be known by the sender and the other recipients. As highlighted in section I there are some exceptions of the users privacy. For example, users of a chat application often, but not always, accept that the developers, sysops engineers, ISP employees, ect. will have access to their messages, or be able to infer who they are sending them too. In particular it is often expected that the developers will use the users messages, communication habits, and other information as data for their own internal analysis. The developers goal is to sell this analysis to other companies, use it for in house product development, marketing, as well as other uses. We make the following assumption

Assumption 1: The developers are exempt from privacy limitations.

This means that the user is both aware and consenting to the developers of the chat application violating the users privacy.

Assumption 2: The intermediate maintenance engineers and developers are exempt from privacy limitations.

This means that, similar to assumption 1, intermediate engineers and maintenance workers can violate the users privacy. The reason for both assumptions 1 and 2 is that the user may assume that these parties either have legitimate reasons for needing to violate their privacy or simply will not out of a lack of interest in the individual user.

We make the following assumptions about the security of systems outside the scope of this project and the cloud.

Assumption 3: All network traffic is secure from attack.

Either no other parties are trying to violate the users privacy by sniffing packets or spying on network traffic, or parties that do want to are unable to. That is, the network traffic is assumed to always maintain the properties of privacy.

Assumption 4: All servers used by the chat application are secure from attack.

Our chat application hosting servers, namely the proxy, are safe from attack and can not leak information.

Assumption 5: All clients used by the chat application are secure from attack, users are not malicious.

Any client used to access our chat application is assumed to be secure from attack and can not leak information. Users are not malicious and use secure passwords, usernames are not shared, and they keep their systems in good condition. That is, they do not have viruses, keyloggers, or other malicious software installed. We also assume that any authentication of the user is secure, and accurate.

The remaining assumptions are at the heart of the problem. They deal with the cloud provider, and their abilities.

Assumption 6: The users only semi-trusted cloud providers.

That is, users do not consent to their privacy being violated by cloud providers or their systems.

Assumption 7: The cloud provider has limited resources to attempt to violate the users privacy. They can only observe users' data in the same form as it is stored by the application in the cloud.

We assume that the cloud provider can not break even rudimentary encryption methods and algorithms. They can not solicit users to send messages or make connections with other users. They can not freely manipulate the data store either. They are limited to only observing the data, in whatever form the applications stores it in.

Assumption 8: The cloud provider has access to any query run in their system.

We assume that the cloud provider can see any query run on a DB in their system.

Components:

We would now like to present the components of our application. We discussed them in the following order UI, proxy, and cloud hosted DB.

A. user interface

Our user interface is implemented in python via a flask app. It consists of a login/logout page, home page, edit group page and a new group page. The login and logout page are self explanatory, they allow the user to login or out of the application. Authentication uses plain text username and password, this is considered secure via assumption 3-5. The home page displays the user's selected group chat messages, and allows them to search the group chat for keywords, see figure 1. The left panel shows each message in the group and the sender of the message. The left panel allows users to search the chat with keywords see figure 2.

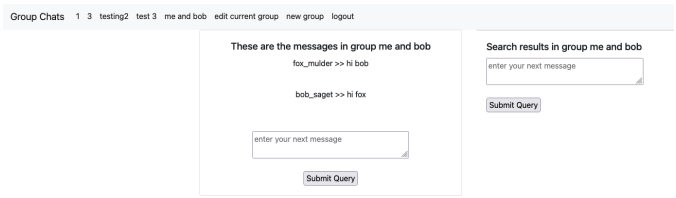

Fig. 1 message board in one group of our chat application

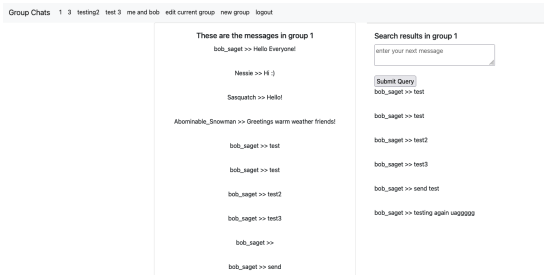

Fig. 2 message board in one group after searching the term test, results shown below the search form.

Users are able to create new chats and add members. First the user must create the chat group. The creator is automatically added as the only member. The creator must then select the group from the list at the top, and then edit the group. The group edit form is shown in figure 3.

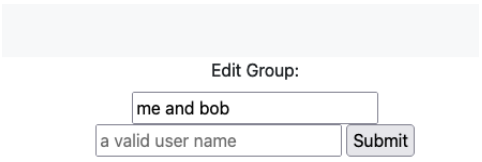

Fig. 3 group edit form for the group 'me and bob'.

Users enter the username of the new group member. Each member must be added one at a time. To maintain the privacy of the other users of the system no user name search function is included. That is the editor of the group must have the usernames off all users they want to add ahead of time. The form does report if the username does not exist. Given assumption 5 this feature is not considered to leak information.

B. proxy

We use the implementation of cryptDB presented in Raluca Ada Popa et al.l's CryptDB: Protecting Confidentiality with Encrypted Query Processing [1]. In this scheme every query is translated from a raw mySQL query into an encrypted version. The proxy encrypts table names and the contents of the tables. Values in each table are stored in layers of encryption or onions. See figure 4 for an overview of the different types of onions used in CryptDB

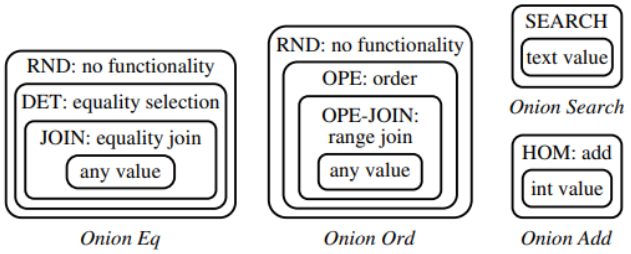

Fig. 4 overview of encryption onions used in CryptDB, retrieved from [1]

The main idea is to always use the most secure encryption for a given query. Each value is thus wrapped in multiple layers of encryption.. There is one encryption key for each column of the DB, however different keys are used for each column and table. All queries by the application are based on comparisons, e.i. searching for a user with username X, group with group id Y ect. We do not sort or order any of the columns, we also never join tables. For this reason we will not discuss CryptDB's functionality in these cases. As our queries are exclusively equality selections all columns stay in the DET encryption, see figure 5 and 6 for an example of an encrypted table.

```
mysql> show tables;
+-----------------+
| Tables_in_chat2  |
+-----------------+
| table_BOPILFVCRR |
| table_LBIKNREQTU |
| table_MPJZDPSPMO |
| table_QLOEBTEYCK |
+-----------------+
4 rows in set (0.00 sec)
```

Fig. 5 A mysql table as stored in the cloud server. Each table name is encrypted.

```
+--------------------------------------------------------------------+
| MDXOATADLQoDET                                                     |
+--------------------------------------------------------------------+
| 0x68076A76B77905FB0F0DCE62555F95A33835E85B0B843842B8F4ED30FF16D2C2 |
| 0xA1E7582BE26FD80DC53DED27FE7ABA2EF7F56E9085FD8327C3E7A99FAD9313B8 |
| 0xA06EC635F8BBB70420DAD3503C5577BB2B5EB21880C0991200E958F6198F475C |
| 0x8A007B856AE1C70F26F2080D86B25D36D77E2E7BD60978841765807508C304EB |
| 0x61F3AC5B93A73AC34B7833BB00359ED90907156363845A395CA2D18C1A11CECC |
| 0x396FD2FE9A75EBF46A24951B10C86404BDBFF2822BABDE7D4E156DC56812E740 |
| 0x101E843BBD3AAD2BBB0E484D388A657F578981A46773C3727CDE966AA3D20CC6 |
+--------------------------------------------------------------------+
```

Fig. 6 A mysql table rows as stored in the cloud server. Each column, and entry is encrypted.

A. *cloud based database*

Our cloud based DB is a normal mysql database. Our chat application exclusively interacts with it via the proxy, since all communication between the cloud and the proxy is encrypted, all communication between the application and the cloud is encrypted. Our database has four tables, *group_id*, *user_id*, *messages*, and *memberships*.

The group_id and user_id tables are the foundations of our databases. The user_id table holds basic information about each user including, time they created an account, username, password, and unique ID. The group_id table holds the time a group was created, its user-friendly name, and a unique ID. The membership table holds pairings of user IDs and group IDs. The pairingings represent the memberships between users and groups. The messages table holds the messages sent and the groups they belong to. Both the memberships, and messages tables reference the group_id table and the user_id table. See figure 7 for a brief overview.

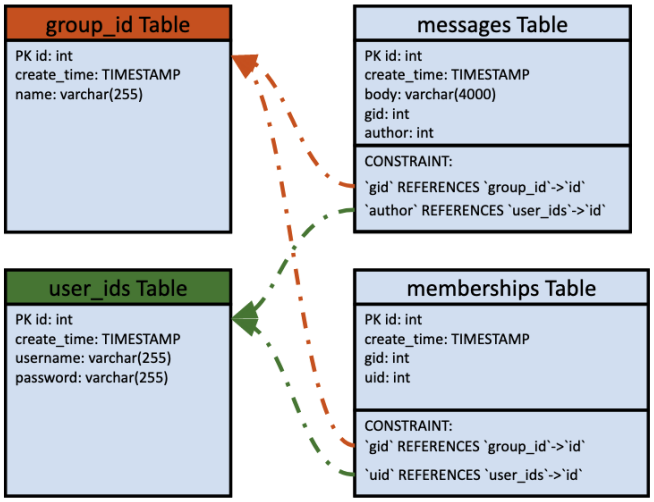

Fig. 7 the database tables and their references. Green arrows show relations between a user ID and another table orange show relations between group ID and other tables.

III. Mechanisms to maintain user privacy

To maintain user privacy, our chat application leverages cryptographic techniques to ensure that only authorized users can access the data. We utilize CryptDB, a system that provides practical and provable confidentiality for applications hosted in the cloud. CryptDB enables the execution of SQL queries over encrypted data, which ensures that the cloud provider cannot read user messages.

To achieve this, we encrypt the user's messages using symmetric encryption schemes with unique keys for each conversation. This ensures that only the sender and the intended recipients can access the content of the messages. Additionally, we employ pseudonyms and other anonymization techniques to mask the identities of the users participating in the conversations, providing an extra layer of privacy.

IV. Analysis of the advantages and disadvantages of Our project

To identify potential weaknesses or shortcomings in the project and to understand the potential risks and challenges associated with the project allows stakeholders to develop contingency plans and strategies to deal with these risks and challenges. In this section we discuss the relative advantages and disadvantages of our group projects.

Advantages:

1. *Enhanced privacy:* Our chat application ensures that user messages and their recipients remain confidential, protecting them from unauthorized access by the cloud provider.

2. *Scalability:* By leveraging cloud computing resources, our chat application can easily scale to accommodate increasing numbers of users and workloads without incurring significant additional costs.

3. *Cost-effectiveness:* Our solution allows companies to minimize infrastructure and maintenance costs while providing a secure and reliable chat service to their users.

4. *Simplicity:* The use of cryptographic techniques and additional privacy-preserving mechanisms via cryptDB offer little changes to the application logic when compared to its unencrypted counterparts.That is using cryptDB is little more difficult then using generic mySQL.

Disadvantages

1. *Trust in encryption:* Users must trust that the encryption techniques used are robust enough to protect their data from potential breaches or attacks.
2. *Key Management:* Ensuring secure key management for the encryption schemes may be challenging and could potentially introduce vulnerabilities if not handled properly. As tables and columns are all encrypted, restoring after a disaster scenario is difficult.
3. *Protection of patterns of behaviour:* We do not hide or conceal when users are using the system, nor do we hide or conceal which users are adding messages. It is possible for the cloud provider to infer an individual's behavior without knowing who the individual is, or who they are exchanging messages with.

V. Demo

We make our project available on github[2], we also offer a live demo available at 10.37.0.32:5000. Please use the following credentials.

Username:bob_saget

Password:password

VI. future work

To further enhance the privacy and security of our chat application, Our future work could explore the following areas:

1. Implementing end-to-end encryption: This would ensure that the messages are encrypted directly on the sender's device and decrypted on the recipient's device, minimizing the risk of unauthorized access.

2. Introducing decentralized or peer-to-peer architectures: This could potentially reduce reliance on central servers and cloud providers, further mitigating privacy concerns.
Investigating more advanced privacy-preserving techniques: Techniques such as zero-knowledge proofs, or secure multi-party computation could be employed to enable even stronger privacy guarantees.

3. Enhancing user privacy controls: Giving users the ability to control their privacy settings and manage data sharing preferences could lead to increased user trust and satisfaction.

4. Evaluating performance and user experience: Assessing the impact of privacy-preserving techniques on application performance and user experience will be essential in determining the success of the chat application.

5. Implementing anonymity networks: Integrating our chat application with anonymity networks such as Tor can help to conceal user activity patterns and make it difficult for cloud providers or third parties to infer individual behaviors. These networks can mask the source, destination, and content of communications, thereby offering an additional layer of privacy protection. This part of the planned future work will address the issues we mentioned in the previous section (5th about the disadvantages of the project).

VII. Conclusions

In our group project, we focused on developing a real-time chat application that protects user privacy while taking advantage of cloud computing. By implementing strong encryption techniques and privacy protection mechanisms, we ensure that user messages remain confidential and cannot be accessed by unauthorized parties, including semi-trusted cloud providers.

Our chat application leverages CryptDB, a system that facilitates SQL queries on encrypted data, enabling secure storage and processing of user messages in the cloud. By employing symmetric encryption schemes, pseudonymization and anonymization techniques, we have successfully achieved our privacy goals of keeping the content of messages and the identity of recipients confidential, in line with user expectations.

The proposed solution offers many benefits to companies seeking to use cloud computing for their chat services, such as enhanced privacy, scalability, and cost-effectiveness. However, the increased complexity of cryptography, trust in cryptography, and challenges associated with key management are significant issues that need to be addressed to ensure a seamless user experience and strong security.

To further improve the privacy and security of the chat application, we also try to explore the implementation of end-to-end encryption, decentralized or peer-to-peer architecture, advanced privacy protection techniques, enhanced user privacy controls, and performance evaluation.

References

[1] Popa, Raluca Ada, et al. "CryptDB: Protecting confidentiality with encrypted query processing." Proceedings of the twenty-third ACM symposium on operating systems principles. 2011.

[2] Hood et al, cloudSecProjectGroup3, (2023), GitHub repository, https://github.com/khood5/cloudSecProjectGroup3