Enhancing the Security of Yioop Discussion Board

A Project Report

Presented to

Dr Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Prajna Gururaj Puranik

December 2022

ABSTRACT

Yioop is an open-source web portal that works as a search engine and a discussion board. To function as a wiki system and as a discussion board, Yioop provides groups feature that allows users to create, join and share content with each other. Such a system needs to store and access data, so data security is an important concern. Apart from database security, Yioop generates statistical data that needs to be protected. The security mechanism also covers access restriction to the groups as well as the data shared among group members.

While Yioop has a security mechanism in place, the goal for this semester was to extend it by implementing several security measures like differential privacy, secret sharing, and homomorphic encryption. The main objective is to protect user data and make the portal more secure. To that end, the project goals for this semester were to understand the existing security concepts of Yioop as well as explore new encryption techniques that could make data sharing more secure. These techniques will be incorporated in Yioop as part of the project for next semester.

Keywords –Yioop, Security, Groups, Thread, Encryption, Differential Privacy, Secret Sharing, Homomorphic Encryption

TABLE OF CONTENTS

I. INTRODUCTION

Data security is of critical importance in the current age of information. It is important to protect data against threats like identity theft, data tampering etc. Web security is crucial for preventing hackers from gaining access to confidential data. Without a proactive security policy, web services run the risk of attacks and possible data leakage. This project focuses on enhancing security measures of one such system – Yioop by implementing security systems like differential privacy, secret sharing and homomorphic encryption.

Yioop is open-source software system that functions as a search engine and a social platform with groups, discussion board and wiki. Users can create and join groups, start discussions with other users in their groups, configure the rules of their groups to restrict access to group members and so on. The groups can be made secure by turning on encryption for the group and the group data can be hidden by turning on the differential privacy option. This option uses the concept of differential privacy: a technique to protect a statistical database such that sensitive data in the database can be protected while allowing useful information to be obtained from the database. The goal of this project for this semester is to extend differential privacy mechanism and implement two new encryption schemes that can potentially make Yioop more secure.

The project for this semester was executed in two phases: the first phase deals with understanding and implementing differential privacy. The second phase involves implementing two encryption schemes. The two phases are split into four deliverables, each of which is detailed in a separate section in this report. Deliverable 1 focuses on understanding the existing differential privacy mechanism in Yioop while Deliverable 2 is an implementation of differential privacy to mask the number of users in a group. Deliverable 3 involves implementing secret sharing while deliverable 4 is homomorphic encryption implementation. The conclusion section summarizes the work done this semester and discusses the future goals of the project

II. DELIVERABLE 1

The goal of the first deliverable is to understand the existing privacy mechanism in Yioop. To achieve this, an encrypted group was created in Yioop, and the database was checked to see whether group creation and addition of users, threads were reflected in the database tables. This deliverable was instrumental in clarifying the security concepts already implemented in Yioop.

The first finding of this deliverable was the concept of database encryption used to protect the data in the Yioop database. Here, data is converted into ciphertext and stored it in the database. This transformed data needs to be decrypted first before users can access it.

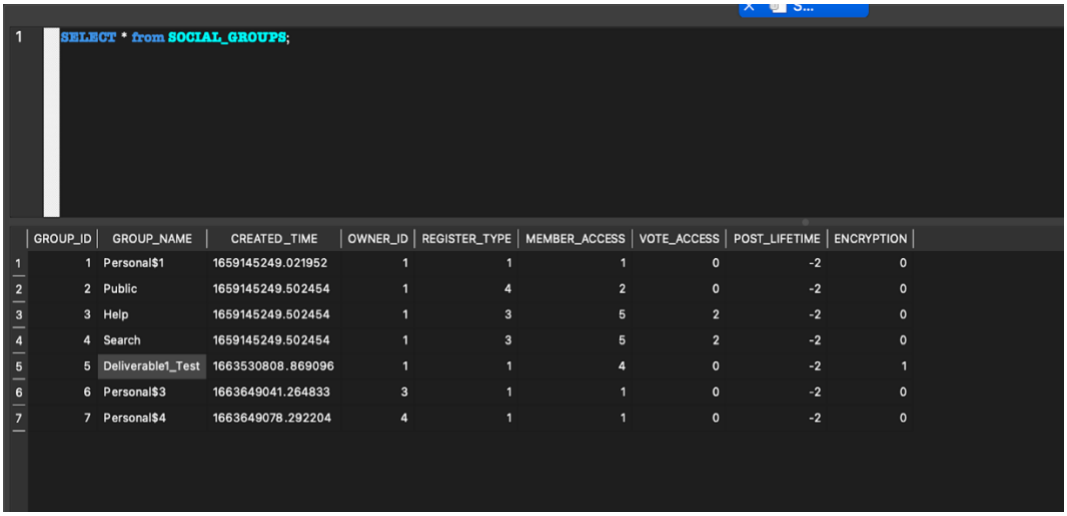
There are different techniques available for database encryption like external database encryption, column level encryption, field level encryption, application-level encryption and so on. Yioop uses application-level encryption and the key for data encryption and decryption is used in an external database. The key is symmetric, generated by AES-256 encryption scheme. This external database provides an additional layer of security. If an intruder gets access to the main database, they will not be able to decrypt the data in the database without having access to the external database. To make it more secure, we inject random values to the actual data every time any database operation like insert, update etc. are performed.

Not all data inside a group needs to be encrypted since not all data is sensitive. We are interested in encrypting the threads in the group and this includes encrypting the title and description. Since it is selective encryption, column level encryption is used so that only the title and description columns are encrypted/decrypted.

To verify these concepts, an encrypted group called ‘Deliverable1_Test’ was created and users were added to this group as well as several threads. The objective was to find the affected databases, query them to check if the group creation was reflected and if the appropriate tables

contain the added users and posts. Additionally, database had to be queried to check if thread title and description were present in an encrypted form.

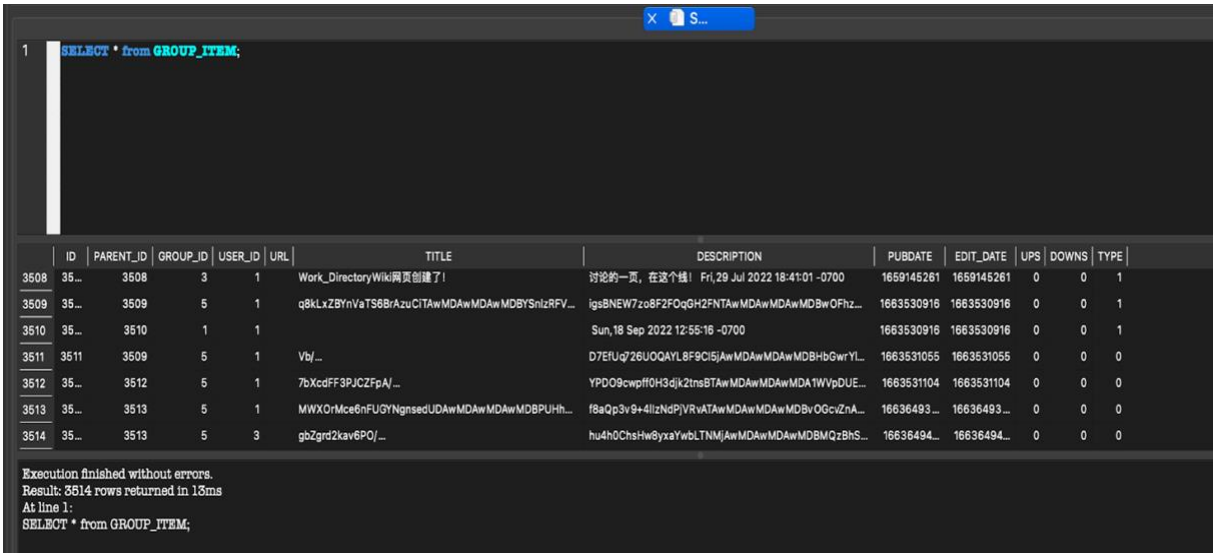
Figure 1 shows the presence of created group ‘Deliverable1_Test’ with group id=5 while Figure 2 shows existence of posts added to the created group. Additionally, figure 3 shows that title and description of threads are encrypted.



```
SELECT * from SOCIAL_GROUPS;
```

	GROUP_ID	GROUP_NAME	CREATED_TIME	OWNER_ID	REGISTER_TYPE	MEMBER_ACCESS	VOTE_ACCESS	POST_LIFETIME	ENCRYPTION
1	1	Personal$1	1659145249.021952	1	1	1	0	-2	0
2	2	Public	1659145249.502454	1	4	2	0	-2	0
3	3	Help	1659145249.502454	1	3	5	2	-2	0
4	4	Search	1659145249.502454	1	3	5	2	-2	0
5	5	Deliverable1_Test	1663530808.869096	1	1	4	0	-2	1
6	6	Personal$3	1663649041.264833	3	1	1	0	-2	0
7	7	Personal$4	1663649078.292204	4	1	1	0	-2	0

Figure 1: Query table to check if created group is present

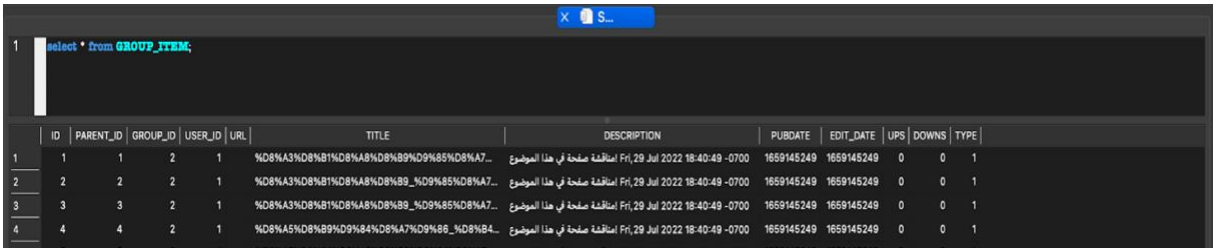


```
SELECT * from GROUP_ITEM;
```

	ID	PARENT_ID	GROUP_ID	USER_ID	URL	TITLE	DESCRIPTION	PUBDATE	EDIT_DATE	UPS	DOWNS	TYPE
3508	35...	3508	3	1		Work_DirectoryWiki网页创建了！	讨论的一页，在这个线！ Fri,29 Jul 2022 18:41:01 -0700	1659145261	1659145261	0	0	1
3509	35...	3509	5	1		q8kLxZBYnVaTS6BrAzuCiTAwMDAwMDAwMDBYSnlzRFV...	igsBNEW7zo8F2FOqGH2FNTAwMDAwMDAwMDBwOFhz...	1663530916	1663530916	0	0	1
3510	35...	3510	1	1			Sun,18 Sep 2022 12:55:16 -0700	1663530916	1663530916	0	0	1
3511	3511	3509	5	1		Vb/...	D7EfUq726UOQAYL8F9CI5jAwMDAwMDAwMDBHbGwrYl...	1663531055	1663531055	0	0	0
3512	35...	3512	5	1		7bXcdFF3PJCZFpA/...	YPDO9cwpff0H3djk2tnsBTAwMDAwMDAwMDA1WVpDUE...	1663531104	1663531104	0	0	0
3513	35...	3513	5	1		MWXOrMce6nFUGYNgnsedUDAwMDAwMDAwMDBPUHh...	f8aQp3v9+4IlzNdPjVRvATAwMDAwMDAwMDBvOGcvZnA...	16636493...	16636493...	0	0	0
3514	35...	3513	5	3		gbZgrd2kav6PO/...	hu4h0ChsHw8yxaYwbLTNMjAwMDAwMDAwMDBMQzBhS...	16636494...	16636494...	0	0	0

```
Execution finished without errors.
Result: 3514 rows returned in 13ms
At line 1:
SELECT * from GROUP_ITEM;
```

Figure 2: Query table to check if posts created in the group are reflected



```
select * from GROUP_ITEM;
```

	ID	PARENT_ID	GROUP_ID	USER_ID	URL	TITLE	DESCRIPTION	PUBDATE	EDIT_DATE	UPS	DOWNS	TYPE
1	1	1	2	1		%D8%A3%D8%B1%D8%A8%D8%B9%D9%85%D8%A7...	امتافئة صفحة في هذا الموضوع Fri,29 Jul 2022 18:40:49 -0700	1659145249	1659145249	0	0	1
2	2	2	2	1		%D8%A3%D8%B1%D8%A8%D8%B9_%D9%85%D8%A7...	امتافئة صفحة في هذا الموضوع Fri,29 Jul 2022 18:40:49 -0700	1659145249	1659145249	0	0	1
3	3	3	2	1		%D8%A3%D8%B1%D8%A8%D8%B9_%D9%85%D8%A7...	امتافئة صفحة في هذا الموضوع Fri,29 Jul 2022 18:40:49 -0700	1659145249	1659145249	0	0	1
4	4	4	2	1		%D8%A5%D8%B9%D9%84%D8%A7%D9%86_%D8%B4...	امتافئة صفحة في هذا الموضوع Fri,29 Jul 2022 18:40:49 -0700	1659145249	1659145249	0	0	1

Figure 3: Check if title and description of threads are encrypted

III. DELIVERABLE 2

Deliverable 2 involves implementing differential privacy using the concepts learnt in the previous deliverable. Differential privacy is used here to mask user statistics in Yioop i.e., number of users in groups. This deliverable ensures that Yioop is protected against statistical attacks.

To understand differential privacy, we first define statistic as quantity computed from a sample. A statistical database is said to protect privacy when it enables the user to learn properties of the population, while protecting the privacy of the individuals in the sample. Differential privacy is a mechanism that ensures that the risk to one's privacy does not substantially increase because of participating in a statistical database.

A randomized function K gives ε-differential privacy if for all data sets D1 and D2 differing on at most one element, and all $S \subseteq \text{Range}(K)$,

$\Pr[K(D1) \in S] \leq \exp(\varepsilon) \times \Pr[K(D2) \in S]$

where ε is a positive real number and S is a subset of image K is a set of all output values that it might produce

A mechanism K that satisfies this definition addresses concerns that any participant might have about the leakage of her personal information x: even if the participant removed her data from the data set, no outputs (and thus consequences of outputs) would become significantly more or less likely

The mechanism works by adding appropriately chosen random noise to the answer a = f(X), where f is the query function and X is the database. Magnitude of the random noise is chosen as a function of the largest change a single participant could have on the output to the query function => sensitivity of the function

For $f: D \rightarrow R^d$, the L1-sensitivity of f is: $\Delta f = \max \| f(D1) - f(D2) \|$ for all D1, D2 differing in at most one element and D is the collection of datasets

The privacy mechanism (Kf for a query function f) computes f(X) and adds noise with a scaled symmetric exponential distribution with variance σ2 in each component, described by the density function:

$$\Pr[Kf(X) = a] \propto \exp(-\|f(X) - a\|/\sigma)$$

For f: D → Rd, the mechanism Kf gives (Δf/σ)- differential privacy. For query strategy F = {fρ: D → Rd}, the mechanism Kf gives (ΔF/σ)-differential privacy.

This deliverable was executed by first identifying the existing code that calculates the number of users in each group. Understanding how this user count is stored and fetched from the database helps identify the point in the codebase where differential privacy function can be called.

This functionality is part of the SocialComponent's function getGroupUsersData, which calls GroupModel's countGroupUsers function to calculate the number of users per group in the database. This is where noise needs to be added to prevent the actual user count from being displayed. The next step is to add addDifferentialPrivacy function to the getGroupUsersData function so that the number of users per count is fuzzified. This masked count of users is then displayed.

There are three instances in the UI where the group user count is displayed. The root user can see group count in the Edit Group section while a general user can see the group user count when accessing information about the group. So, to test this deliverable, these three UI instances need to be checked to ensure that the group user count is fuzzified. These three instances are shown in Figure 4.

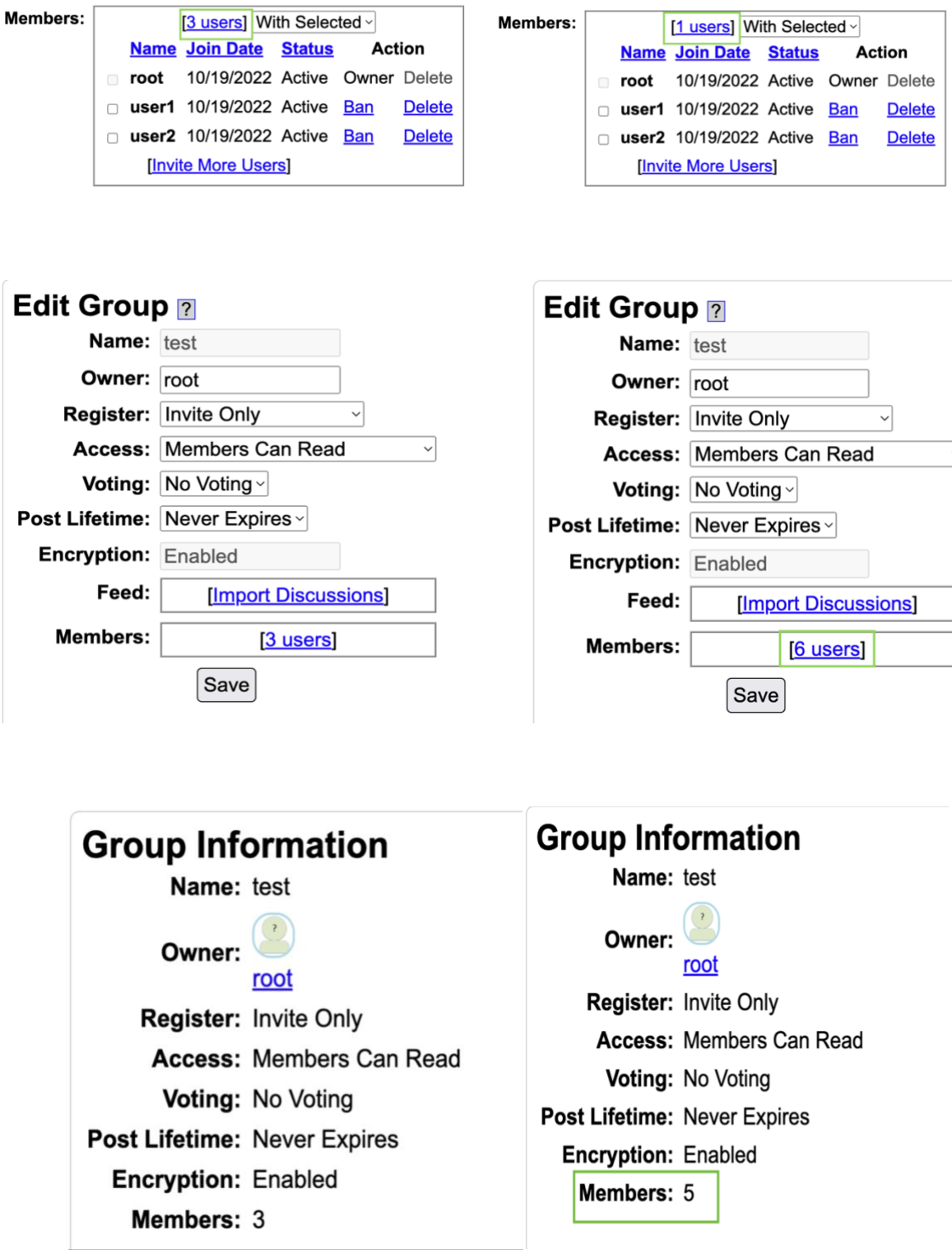


Figure 4: Differential Privacy implementation result

IV. DELIVERABLE 3

This deliverable consists of secret sharing encryption scheme implementation. Secret sharing involves distributing a secret among a group such that no individual holds any intelligible information about the secret. When a sufficient number of individuals combine their shares, the secret can be reconstructed. An adversary without enough shares cannot reconstruct the secret even with infinite time and computing capacity. We use (k, n) threshold scheme which divides data D into n pieces D_1, $D_{2,}$,…, D_n such that:

- Knowledge of any k or more D_i pieces makes D easily computable
- Knowledge of any k - 1 or fewer D_i pieces leaves D completely undetermined, in the sense that all its possible values are equally likely

This is especially useful in the management of cryptographic keys. Keeping the key in a single, well-guarded location is unreliable and storing multiple copies of the key at different locations could be dangerous. So, the best solution is (n, k) threshold scheme to keep the encryption keys a secret. Threshold schemes are ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate.

Polynomial interpolation is used to implement secret sharing. Assume that the data to be encrypted is D, n is number of shares and k is threshold or the minimum number of shares required to retrieve the secret. We create D_i shares of the data by choosing a polynomial with degree k-1 and encoding the secret as a coefficient of P. The polynomial is then evaluated to get Di points as shares. The steps can be formally defined as follows:

- Select a random k-1-degree polynomial $q(x) = a_0 + a_1(x) + a_2(x^2) +.... +a_{k-1}(x^{k-1})$ in which $a_0 = D$.
- Evaluate: $D_1 = q(1)$ .... $D_i = q(i)$ ..... $D_n = q(n)$

Given any subset of k of these Di values, we can find the coefficients of q(x) by interpolation, and then evaluate D=q (0). Knowledge of just k - 1 of these values does not suffice to calculate D.

Secret sharing can be implemented using integer arithmetic or finite field arithmetic. Using integer arithmetic provides weak security wherein the attacker could possibly guess the secret by solving algebraic equations. This problem can be solved by using finite field arithmetic. The steps to be followed are:

Choose a prime number such that prime number is greater than number of participants and prime number is greater than every coefficient, including the secret. The points on the polynomial must also be calculated as (x, f(x) mod prime) instead of (x, f(x)). This prime number is known but even with this information, attacker will not be able to stop guessing, unlike integer arithmetic problem. (-prime.m_x) is added to each equation. Subtraction of previous equations with give prime (m_1 - m2) because of mod and it becomes impossible to guess.

Encryption function is created using the approach outlined above. Special care must be taken while decrypting the secret. If threshold number of shares are given, use Lagrange basis polynomial equations to decrypt the secret.

If 1 point is given, find $l_0(x) = (x - x_1)/ (x_0 - x_1) * (x - x_2) / (x_0 - x_2)$

If 2 points are given, find $l_1(x)$ in addition to $l_0(x)$ and so on

In general, the formula for interpolation is:

$$f(x) = \sum y_j * l_j(x)$$

Reconstruction of the secret involves finding the multiplicative modulo inverse of a number. Extended GCD can be used for this purpose.

There are many advantages of using secret sharing scheme. The size of each piece does not exceed the size of the original data. When k is kept fixed, Di pieces can be dynamically added or deleted without affecting the other Di pieces. It is easy to change the Di pieces without

changing the original data. All we need is a new polynomial q(x) with the same free term. A frequent change of this type can greatly enhance security. Secret sharing also wllows a hierarchical scheme where number of pieces needed to determine the secret depends on their importance.

The result of secret sharing implementation is shown below in Figure 5

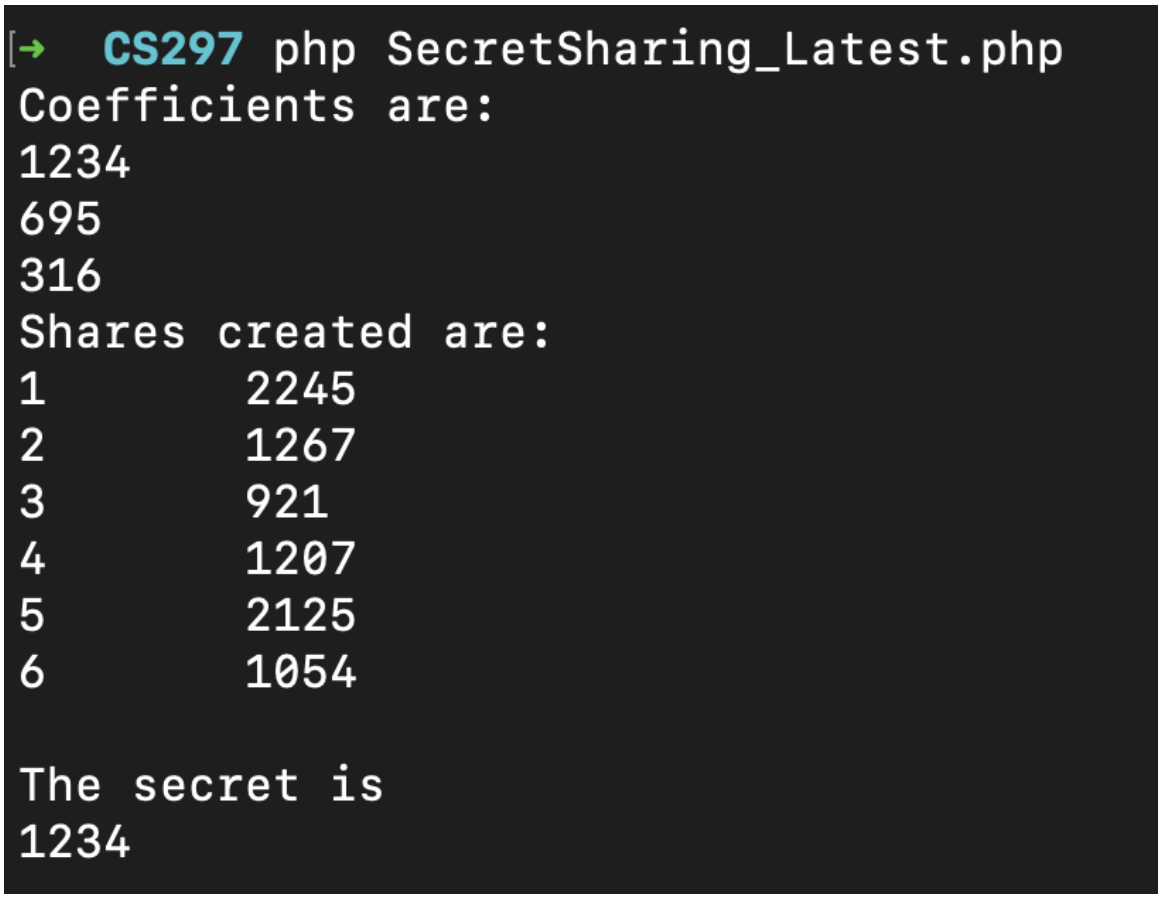


Figure 5: Secret Sharing implementation result

V. DELIVERABLE 4

Deliverable 4 involves implementation of a homomorphic encryption scheme. The methodology chosen for implementation is Paillier encryption, a partial homomorphic encryption scheme.

Homomorphic encryption permits users to perform computations on encrypted data without first decrypting it. This result is in an encrypted form When it is decrypted, result is identical to that produced if the operations been performed on the unencrypted data. Homomorphic encryption makes it possible to analyze or manipulate encrypted data without revealing the data to anyone.

Just like other forms of encryption, homomorphic encryption uses a public key to encrypt the data. Unlike other forms of encryption, it allows functions to be performed on the data while it's still encrypted. Then, the individual with the matching private key can access the unencrypted data after the functions and manipulation are complete. This allows the data to be and remain secure and private even when someone is using it.

Homomorphic encryption has huge potential in areas with sensitive personal data such as in financial services or healthcare when the privacy of a person is paramount. Another bonus of homomorphic encryption is that unlike other encryption models in use today, it is safe from getting broken by quantum computers.

There are three main types of homomorphic encryption:

- Fully-HE (FHE): keeps information secure and accessible while allowing any number of addition and multiplication operations.
- Somewhat-HE (SHE): It allows both addition and multiplication, but we are limited in term of the number of operations we can perform.

- Partially-HE (PHE): Keeps sensitive data secure by only allowing select mathematical functions to be performed on encrypted data. This type of scheme either allows addition or multiplication, but in an unlimited fashion.

  Paillier encryption scheme is partially homomorphic over addition. That is,

$$E(m_1) * E(m_2) = (g^{m1}\, r_1^{n} \pmod{n^2}) * (g^{m2}\, r_2^{n} \pmod{n^2})$$

$$= g^{m1+m2}\, (r_1 * r_2)^{n} \bmod n^2$$

$$= E\,(m_1 + m_2)$$

As the encryption function is additively homomorphic, we can define the following properties:

1) Multiplying encrypted messages results in the addition of the original plaintexts mod n:

$$D\,(E\,(m_1, r_1) * E\,(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n$$

2) Homomorphic multiplication of ciphertext to power of plaintext: A ciphertext raised to the power of a plaintext will decrypt to the product of the two plaintexts. More generally, a ciphertext raised to a constant k will decrypt to the product of the plaintext and the constant:

$$D\,(E\,(m_1, r_1)^{k} \bmod n^2) = k\, m_1 \bmod n$$

Implementation steps involve generating a public-private key pair and using it to encrypt and decrypt the plaintext. The key generation function starts with picking two large prime numbers p and q randomly and independently. We need to confirm that GCD (pq, (1-p) (1-q)) is 1. This property is assured if both primes are of equal length. We then compute n as the product of p and q and lambda as the lcm of p-1 and q-1. Next, pick a random integer g in the set of integers from 1 to n^2 and define function L which is division of x-1 by n. Last step is to calculate multiplicative modulo inverse, mu. If it doesn't exist, choose new p and q and repeat. The public key is (n, g) and the private key is (λ, μ)

For encryption of plaintext m in the range 0 to n, select random number r such that r value is between 0 and n and compute ciphertext C as:

$$C = g^m r^n \bmod n^2$$

To decrypt the ciphertext, plaintext = L ($c^{\lambda} \bmod n^2$). $\mu \bmod n$

The result of the implementation is given by Figure 6

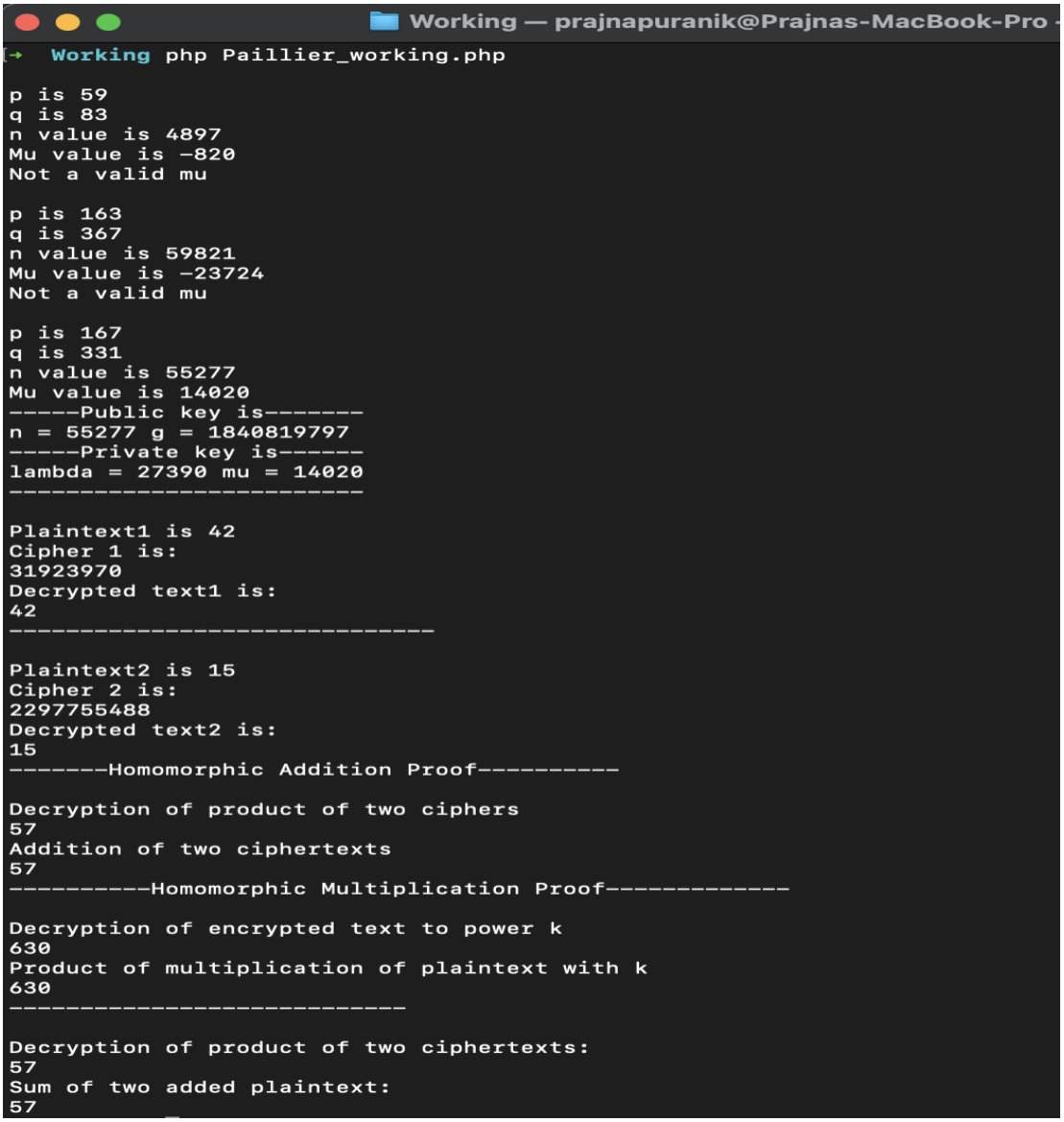


Figure 6: Paillier encryption implementation result

VI. CONCLUSION

Ensuring the security of Yioop and safety of user data is the objective of this project. This project has taken several steps to achieve that objective. Deliverable 1 was the groundwork that helped understand Yioop, from the coding guidelines to the execution flow and database tables involved. Deliverable 1 and 2 focused on the concept of differential privacy and the scope at which it was already part of Yioop. It was also instrumental in helping identify use cases to extend differential privacy and one such use case was implemented as part of deliverable 2 to mask the number of users in groups. Deliverables 3 and 4 were an exploration into encryption concepts that could potentially be used to make Yioop more secure. These two deliverables drive home the importance of information security.

The encryption schemes executed in Deliverables 3 and 4 will be integrated into Yioop in the next semester. More such schemes and use cases will be explored in the next project. Possible areas for extending differential privacy will be investigated so that sensitive information is hidden while not hindering the functionality of the system. This project, as well as the project next semester, will be focused on the overall objective of making Yioop more secure.

REFERENCES

[1] C. Dwork, "Differential Privacy, 33rd International Colloquium on Automata, Languages and Programming, part II", 2006

[2] C. Dwork *et al.*, "Differential Privacy - A Primer for the Perplexed", *Conference of European Statisticians*, 201

[3] A. Shamir, "How to share a secret", *Communications of the ACM*, 612,613

[4] https://blog.boot.dev/cryptography/shamirs-secret-sharing/

[5] D. R. Stinson, M.R.Patterson, “Cryptograph Theory and Practice”, 4th edition, page 467

[6] https://www.freecodecamp.org/news/introduction-to-homomorphic-encryption/

[7] M. O’Keeffe, “The Paillier Cryptosystem: A Look into The Cryptosystem and Its Potential Application”, College of New Jersey, 2008

[8] A. Acar et al., “A Survey on Homomorphic Encryption Schemes: Theory and Implementation”, Florida International University