Privacy Preserving Cloud Computing

Aditya Jain
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
adityajain@cse.iitb.ac.in

Bernard Menezes
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
bernard@cse.iitb.ac.in

***Abstract*—To leverage the benefits of Cloud Computing, Data Owners outsource their data management systems from their local sites to the cloud servers. Benefits include quick deployment, cost reduction, on-demand applications and services from a pool of configurable computing resources, etc. To ensure data privacy, Data Owners store their data in encrypted form in the cloud. Unfortunately, conventional encryption schemes don't support the analysis and processing of encrypted data. To make some cash, Data Owner may allow others to query his data. To ensure query privacy, a query user may want to send queries in encrypted form. Now the cloud has to perform computations on encrypted data and encrypted queries. Earlier works have proposed techniques to securely compute k-nearest neighbors (k-NN) on encrypted data (outsourced to the cloud server). These works have used Homomorphic Encryption. We identified vulnerabilities in previous works like Zhu's, Singh's, and Parampalli's work that cannot be handled by homomorphic encryption alone, like code integrity, code confidentiality, etc. We propose a new scheme using secure enclaves to circumvent these vulnerabilities while improving performance. Our proposed design provides data privacy, query privacy, key confidentiality, query controllability, data integrity, code integrity, access pattern hiding, and defense against collusion attacks. Our scheme also improves performance by eliminating Paillier encryption and multiple matrix multiplications, which takes too much time. Our scheme can also be extended to provide code confidentiality.**

***Index Terms*—Secure Cloud Computing, Secure Enclaves, Intel SGX, K Nearest Neighbor, Privacy Preserving**

I. Introduction

A. Motivation

B. Contribution

C. System Model

D. Threat Model

Previous works considered Data Owner, Cloud Server, and Query User as honest-but-curious, which means each party will follow the steps strictly and return correct computation results. At the same time, each party will try to infer as much information about others' data as possible from the data they get from others. But this assumption is not realistic. Data Owner may collude with Cloud Server, or Cloud Server may act as Query User, etc. We have not made such an assumption (like honest-but-curious).

II. Background

A. Paillier Cryptosystem

B. k-Nearest Neighbor (k-NN)

C. Secure Enclaves

1) Intel SGX:

2) SGX Application:

3) How SGX maintains confidentiality:

4) Software Attestation:

III. Prior Work

A. Zhu's

B. Singh's

C. Parampalli's

IV. Vulnerabilities in Prior Work

V. Our Proposed Design

A. Main Idea

Our main idea is to decouple the identity of the query user and the actual query so that it is infeasible for anyone to determine the query submitted by a query user. We have used two secure enclaves in our design:

1) Enclave E1 is responsible for database storage and KNN computation. Enclave E1 belongs to the Data Owner to protect Data Privacy. So Data Owner installs the KNN code in enclave E1.
2) Enclave E2 is responsible for interfacing with the query user and processing user payment. Enclave E2 belongs to the trusted 3rd party to preserve Query Privacy by decoupling Query User's identity from their query.

B. Assumptions

In our design, we assume the following essential things (which is true in previous designs as well):

1) Data Owner has sent the correct database to the secure enclave in the cloud.
2) The cloud server provides its services properly, i.e., it doesn't block the data owner from sending its database into the secure enclave. Similarly, it doesn't block the query user's query.

In addition, we assume that the interfacing code in Enclave E2 is minimal, open-source (known to everyone), and installed by a trusted 3rd party (similar to Certificate Authority). We assume that several parties have verified this code. Once it is loaded in the enclave and verified, it cannot be changed by anyone as enclaves ensure the integrity of the code. In our design, the two enclaves do not collude as Enclave E2 code is minimal and verified to be correct by multiple parties.

C. Stages of Proposed Design

The database point $p = (p_1, p_2, \cdots, p_d)$ is stored as follows in our design:

$$\dot{p} = \left(-2p_1, -2p_2, \cdots, -2p_d, \|p\|^2\right)$$

The query point Q in our design is as follows:

$$Q = (q_1, q_2, \cdots, q_d, 1)$$

1) Sending Database to Enclave E1 (in Cloud Server) in a secured manner: Fig. 1 shows the one-time work done by the data owner.
Steps:

1) Data Owner installs the application (considered untrusted) on the cloud, which creates an enclave E1 (considered trusted) on the cloud. This enclave E1 contains the KNN code. So the KNN code is successfully installed in Enclave E1.
2) Data Owner asks for an attestation report from enclave E1 to prove its integrity, i.e., to prove that it contains the KNN code installed by the Data Owner and not by Cloud Server or any attacker. This process is called Remote Attestation.
3) On receiving the request for the report, it generates a report (called Quote) and an RSA key pair (PK_{E1}, SK_{E1}). It keeps the secret key SK_{E1} within itself and sends the report along with the public key PK_{E1} to Data Owner.
4) Data Owner sends the report (called Quote) to the Intel Attestation Centre for verification.
5) Intel Attestation Centre replies with whether the report is valid or not.
6) Data Owner generates an AES key K_{DO} to encrypt the database. It needs to give the AES key K_{DO} to Enclave E1 for decrypting the database. So it sends the database (encrypted with K_{DO}) and AES key K_{DO} (encrypted with public key PK_{E1} of enclave E1). So Data Owner sends the following to Enclave E1.
   a) $E_{K_{DO}}(Database)$
   b) $E_{PK_{E1}}(K_{DO})$
7) Enclave E1 decrypts the $E_{PK_{E1}}(K_{DO})$ with SK_{E1} (which it had from step 2) and gets the K_{DO}. Now enclave E1 has the AES key K_{DO}. Now it can decrypt the encrypted database using the AES key K_{DO}. This decryption of keys and database happens inside the enclave. So it is not visible to any attacker or Cloud Server.

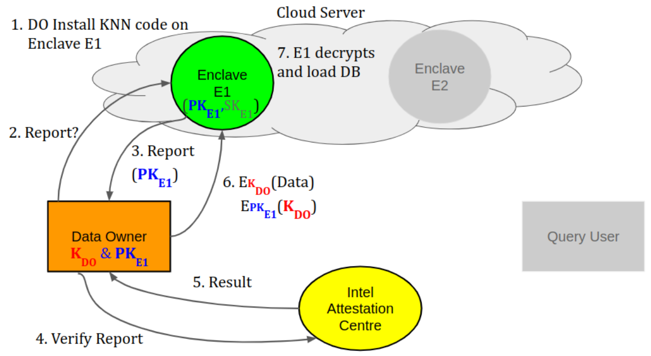

Fig. 1: Sending Database to Enclave E1 (in Cloud Server) in a secured manner

2) Verifying each others' (enclaves') integrity: Fig. 2 shows the one-time work done by both enclaves.
Steps:

1) A trusted 3rd party (similar to Certificate Authority) installs the application (considered untrusted) on the cloud, which creates an enclave E2 (considered trusted) on the cloud. This enclave E2 will have minimal interfacing code (known to everyone and verifiable by everyone). Anyone can verify it through attestation.
2) Enclave E2 asks for an attestation report from Enclave E1 to prove its integrity, i.e., to confirm that Data Owner installs it and not CSP or any other attacker (Local Attestation if they are running on the same processor, otherwise, Remote Attestation).
3) On receiving the report request from E2, Enclave E1 (on the cloud) generates a report. It sends the report along with its Public key PK_{E1} (generated earlier) to enclave E2.
4) On receiving the report, Enclave E2 generates its report and an RSA key pair (PK_{E2}, SK_{E2}). It sends its report along with its Public key PK_{E2} to enclave E1.

If both of them run on the same processor, they can verify the reports locally and get the keys. Otherwise, if they are running on different processors (or on different clouds), they need to go to Intel Attestation Centre to verify the report). In the end, they both have each other's public keys.

3) Query User verifies Enclave E2's integrity: Fig. 3 represents the one time work done by each query user.
Steps:

1) Query User asks for an attestation report from Enclave E2 to prove its integrity, i.e., that it has been installed by a trusted 3rd party and not CSP or any other attacker (Remote Attestation). This step is required only once per query user (as the registered Query User will get enclave E2's public key (PK_{E2}) for later use).
2) On receiving the report request from the Query User, Enclave E2 (on the cloud) generates a report. It sends its report along with the Public key PK_{E1} (of enclave E1) and its public key PK_{E2} to the Query User.

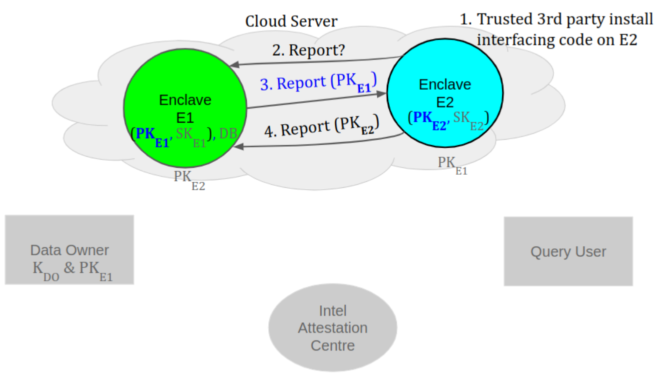

Fig. 2: Verifying each others' (enclaves') integrity

3) QU sends the report to the Intel Attestation Centre for verification.
4) Intel Attestation Centre replies with whether the report is valid.

So now, the QU also has PK_{E1} of enclave E1 and PK_{E2} of enclave E2.

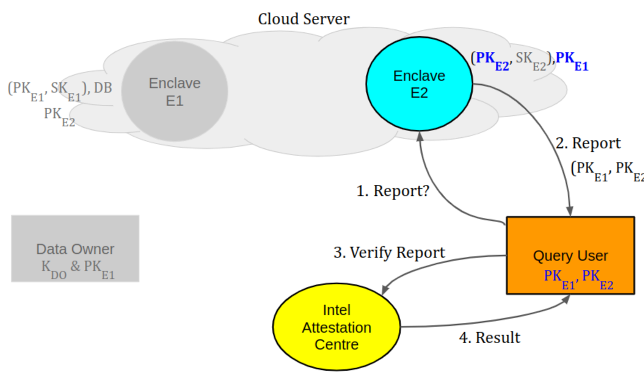

Fig. 3: Query User verifies Enclave E2's integrity

4) Query User sends query to enclave E1 through enclave E2: Fig. 4 represents the work done per query by Query User. Steps:

1) Query User generates its query Q, an RSA key pair (SK_Q, PK_Q), and a unique AES key K_{QU} (different for every query).
2) Query User encrypts the query with its AES key K_{QU}. This encrypted query needs to be sent to enclave E1 through enclave E2. But E1 will need the key K_{QU} to decrypt the query to perform KNN computation. So QU has to send K_{QU} as well. But this K_{QU} should not be visible to enclave E2. Otherwise, it can see the query in clear. Another thing to note is that we need to send the credentials or payment info (Cr) and encrypted query in an encrypted form such that it is not visible to attackers or Cloud Server. So finally, we sent the query in the following way to E2:

$$E_{PK_{E2}}\left(E_{K_{QU}}\left(Q\right)||E_{PK_{E1}}\left(K_{QU}\right)||Cr||PK_Q\right)$$

3) Enclave E2 decrypts the above encrypted query and gets the following:

$$E_{K_{QU}}\left(Q\right)||E_{PK_{E1}}\left(K_{QU}\right)||Cr||PK_Q$$

Based on credentials or payment info, it decides whether to allow query or not [Query Controllability]. Now, the enclave E2 sends the following to enclave E1:

$$E_{K_{QU}}(Q)||E_{PK_{E1}}(K_{QU})$$

Only Enclave E1 can decrypt $E_{PK_{E1}}(K_{QU})$ using SK_{E1} and get K_{QU}. Using K_{QU}, it decrypts the encrypted query. Now, enclave E1 can see the query in the clear, but it doesn't know whose query it is (Query User's identity is decoupled from his query).

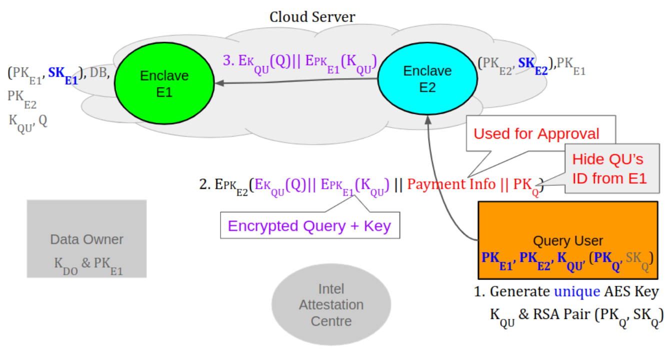

Fig. 4: Query User sends query to enclave E1 through enclave E2

5) KNN computation and sending result (k-nearest neighbors) to Query User: Fig. 5 represents the work done by enclave E1 per query.
Steps:

1) Now, Enclave E1 has both database and query. It computes the KNN and obtains the results.
2) It encrypts the result with K_{QU}, so only QU can decrypt it. Now it returns the encrypted result $E_{K_{QU}}(Result)$ to Enclave E2.
3) Enclave E2 (or anyone else) cannot decrypt the result. E2 sends the $E_{PK_Q}(E_{K_{QU}}(Result))$ to the QU, who can decrypt it to get the result since it has both SK_Q and K_{QU}. $E_{K_{QU}}(Result)$ needs to be encrypted with PK_Q. Otherwise, Enclave E1 can link $E_{K_{QU}}(Result)$ to Query User (As he knows $E_{K_{QU}}(Result)$ and can tap the line between Enclave E2 and Query User to see to whom the result is returned).

Note: Query Users need not use the same K_{QU} again and again. Using a new randomly generated AES key every time will be advisable (as we want per query security).

D. Pros of the Proposed Design

1) ***Code Integrity****:* Data Owner installs the KNN code in Enclave E1. He then performs software attestation to verify the integrity of the code. He gives the database to the enclave only when he is satisfied. SGX will detect any modification in the code later.

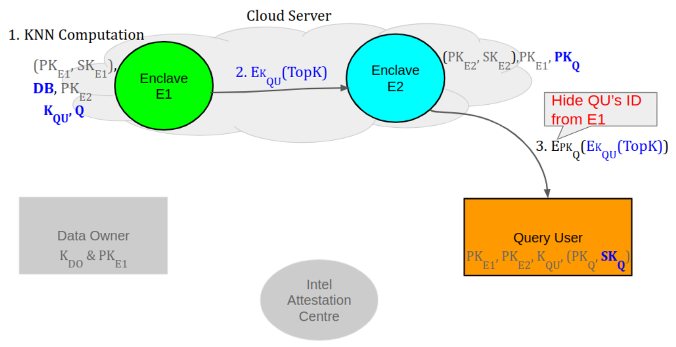
Cloud Server
1. KNN Computation
(PK_{E1}, SK_{E1}), DB, PK_{E2} K_{QU}, Q
Enclave E1
2. $E_{K_{QU}}(TopK)$
Enclave E2
(PK_{E2}, SK_{E2}), PK_{E1}, PK_Q
Hide QU's ID from E1
3. $E_{PK_Q}(E_{K_{QU}}(TopK))$
Data Owner K_{DO} & PK_{E1}
Query User PK_{E1}, PK_{E2}, K_{QU}, (PK_Q, SK_Q)
Intel Attestation Centre

Fig. 5: KNN computation and sending result (k-nearest neighbors) to Query User

2) ***Data Integrity:*** SGX ensures the integrity of the database. Modification by attacker or cloud is possible, but it will be detected. In other words, we are guaranteed that the data is integrity-protected because, otherwise, SGX would stop the system.

3) ***Code Confidentiality:*** In our design, we have not provided code confidentiality as there is no need for code confidentiality in the case of KNN code as it is simple and known to everyone. But code confidentiality can be added to Enclave E1 without worrying about Enclave E1 (an extension of Data Owner) giving queries to Data Owner or others. Even if it provides the query to the Data Owner or others, they won't be able to link Query User's identity to that query. To provide code confidentiality, the KNN code can be sent in encrypted form to Enclave E1 (along with the database). We will need a loader inside the enclave to decrypt and load the code.

4) ***Data Privacy:*** Enclave E1 belongs to the Data Owner. The code is installed in it by Data Owner and has been verified through software attestation. So it is the trusted code. Data Owner can send the database in encrypted form to enclave E1. The enclave E1 will decrypt the code inside itself (as only it has the key). Once the data is decrypted inside the enclave, it will be present inside the enclave. No one from outside can see it. So, data privacy is guaranteed by the secure enclave.

5) ***Key Confidentiality:*** The data is encrypted by an AES key K_{DO} of the data owner. This key is sent to the enclave E1 in encrypted form (encrypted by its public key). So only the enclave E1 can decrypt and get the AES key with which the Database is encrypted. So Data Owner's key is confidential.

6) ***Query Controllability:*** In earlier works like Zhu's and IBM's work, Data Owner decides whether to allow the query or not. But the criteria of decision is what? Data Owner cannot see the query. He has to allow the Query User's query based on his credential (or some payment information). This task can be given to Enclave E2. Enclave E2 can decide whether to allow the query or not based on some payment information.

7) ***Query Privacy:*** Enclave E2 receives the query from the query user in encrypted form. It cannot decrypt the query. It sends the encrypted query to Enclave E1, which decrypts and performs the KNN computation. Enclave E1 gets only the encrypted query and not the credential of the query user from Enclave E2. So E1 cannot link Query User's identity to their query. In other words, Query Privacy is preserved.

8) ***Access Pattern Hiding:*** The KNN computation happens inside the enclave E1. So Cloud Server or the attacker cannot see the result of the KNN computation. In other words, he cannot use the KNN results to figure out corresponding encrypted database tuples. Also, the result is sent in encrypted form to the query user. Only Query User can decrypt the final result. So no one else can see the result as well.

9) ***Per Query Key:*** Each query is encrypted with a different AES key. So even if one key is lost, the privacy of other queries is preserved.

10) ***Data Privacy and Query Privacy against collusion attack:*** Enclave E1 belongs to the data owner. So it won't collude with query users or cloud servers. So data privacy is guaranteed against collusion attacks. Enclave E2 has open-source minimal interfacing code installed and verified by various trusted parties. So E2 won't collude with anyone else. Thus Query Privacy is guaranteed against collusion attacks.

11) ***Data Owner need not be online:*** The task of KNN computation is taken care of by Enclave E1. And the query controllability is handled by Enclave E2. So Data Owner need not be online.

12) ***Paillier Encryption multiple matrix multiplications are not needed:*** The beauty of our scheme is that it uses the AES scheme, which is very fast and secure. We need not worry about the cache-based side-channel attacks because, in recent processors, AES is done on hardware without needing the cache. Also, we don't use fancy matrix multiplications like those used in earlier works.

E. *Cons of the Proposed Design*

1) Couldn't find any security, performance or availability related cons.
2) If we are using Intel SGX then:
   a) Intel SGX support needed.
   b) We have to trust Intel for software attestation (Till SGX version 1). In SGX version 2, we can perform remote attestation without going to Intel Attestation Centre.

VI. Performance Evaluation

We implemented our proposed scheme and two earlier schemes: Zhu's and Singh's, using C language. To implement our scheme, we used the GNU MP library, AES library, and Intel SGX (for secure enclaves). To implement Zhu's and Singh's work, we used the GNU MP library and Paillier library with a 1024 bits key size. We selected the security parameters c = 2, ϵ = 2 (for both Zhu's and Singh's scheme) and l = 5 (for Singh's scheme). We performed all our experiments on a machine having Ubuntu 20.04 with Intel Core i5 2.5 GHz Processor and 8 GB RAM.

A. *Key Generation*

Here, Key Generation Time refers to the time required for generating the key(s) with which the database of the Data Owner is encrypted.

1) Computational Complexity: In our scheme, the computational complexity of Data Owner during Key Generation is O(1) because Data Owner generates an AES Key only for encrypting the database. In Zhu's scheme, the computational complexity of Data Owner during Key Generation is O(η^2) because the costliest step is the generation of $\eta \times \eta$ invertible matrix M. Here, $\eta = d + 1 + c + \epsilon$. In Singh's scheme, the computational complexity of Data Owner during Key Generation is O(n^2) because the costliest step is the generation of $n \times n$ invertible matrix W. Here, $n = \eta + l$.

2) Experiment Results: We evaluated the time for key generation in our design, Zhu's, and Singh's schemes. Fig. 6 shows the average key generation time over ten runs, with the dimension of data points varying from 100 to 1000. Our scheme needs to generate a random AES key, so our design will take constant and negligible time. On the other hand, key generation time in Zhu's and IBM's schemes increases with the dimension of data points. Since Singh's scheme generates two large matrices compared to one in Zhu's scheme, it takes longer.

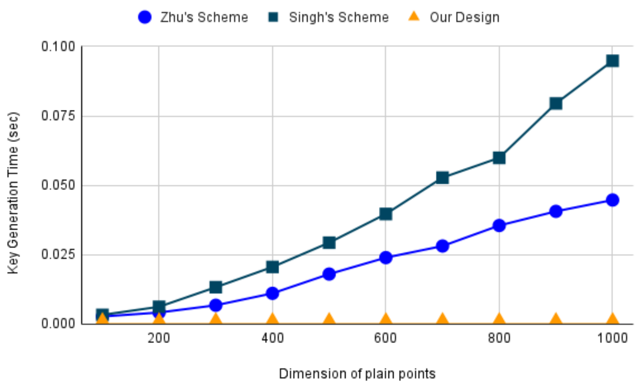

Fig. 6: Average Key Generation Time (in sec)

B. Database Encryption

1) Computational Complexity: In our scheme, the computational complexity of Data Owner during Database Encryption is O(md) because Data Owner encrypts each database point (m points) having d features each, using AES encryption. In Zhu's and Singh's scheme, the computational complexity of the Data Owner during this stage is O($m\eta^2$). The costliest step is the multiplication of m data points with an $\eta * \eta$ invertible matrix M.

2) Experiment Results: We evaluated the time to encrypt a random database consisting of 10000 tuples. We compared our design with both Zhu's and Singh's schemes. Fig. 7 shows the database encryption time, with the dimension of data points varying from 100 to 1000. This stage is the same for both Zhu's and Singh's schemes. So their line graphs match. Since our design uses AES encryption instead of matrix multiplication, ours take far less time than their design. Our design takes just 18 sec compared to 708 sec in their scheme when the dimension of data points is 1000.

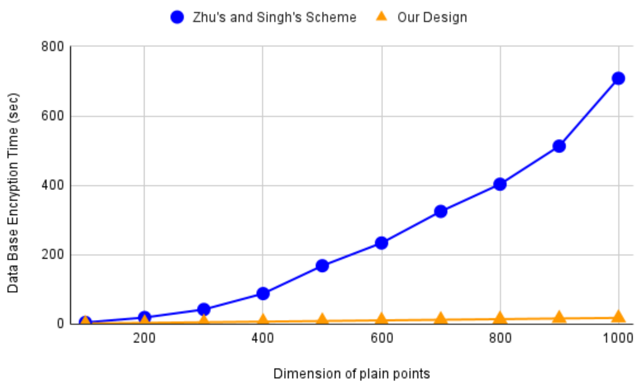

Fig. 7: DataBase Encryption Time (in sec) when number of tuples = 10000

C. Query Encryption at Query User

1) Computational Complexity: In our scheme, the computational complexity of Query User during Query Encryption is O(d) because the costliest step is the encryption (using AES) of query point with d dimensions. In Paillier Cryptosystem, one encryption takes O(log N) computation time (where N = pq). So, in Zhu's and Singh's scheme, the computational complexity of Query Encryption during this stage is O(d log N) because the costliest step is the encryption of the query point with d dimensions.

2) Experiment Results: We evaluated the Query Encryption time (at the Query User side) in our design, Zhu's, and Singh's schemes. Fig. 8 shows the average Query Encryption time over ten runs, with the dimension of query points varying from 100 to 1000. In Zhu's and Singh's scheme, the encryption time is more as Paillier encryption takes more time. Our scheme needs to generate a random AES key and then encrypt the query point with this AES key. Our design takes very little time, as is visible in fig 8.

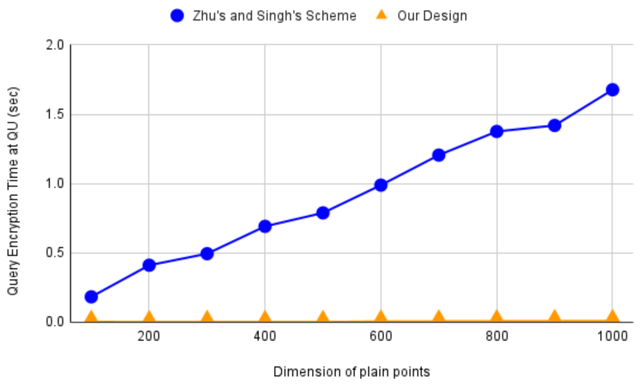

Fig. 8: Average Query Encryption Time at QU's side (in sec)

D. *Query Encryption at Data Owner*

1) Computational Complexity: Our scheme doesn't require Data Owner for Query Encryption. In Zhu's scheme, the computational complexity of Query Encryption during this stage is O($\eta^2 + c\eta log\, N$) because the costliest step is the calculation of $A^{(q)}$, which requires O(η^2) multiplications and O(cη) encryptions (Paillier). In Singh's scheme, the computational complexity of Query Encryption during this stage is O($n^2 + c\eta\, log\, N + ln\, log\, N$) because the costliest step is the calculation of $A^{(q)}$ and then $B^{(q)}$, which requires O(η^2 + n^2) multiplications and O(cη + ln) encryption (Paillier).

2) Experiment Results: We evaluated the Query Encryption time (at the Data Owner side) in Zhu's and Singh's schemes. Fig. 9 shows the average Query Encryption time over ten runs, with the dimension of query points varying from 100 to 1000. In Zhu's scheme, $A^{(q)}$ calculation takes too much time. Since Singh's scheme calculates $A^{(q)}$ and then $B^{(q)}$ from $A^{(q)}$, it requires a lot more time. This puts too much pressure on the Data Owner. Data Owner has to spend 40 sec and 60 sec per query (per query user) in Zhu's and Singh's scheme, respectively. In our design, Data Owner is not involved. So in our design time required on Data Owner's side per query is 0.

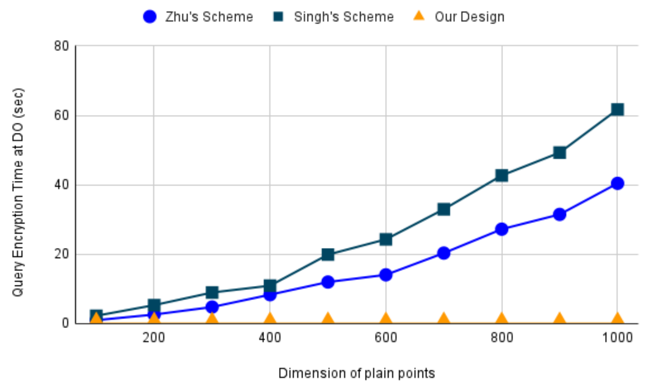

Fig. 9: Average Query Encryption Time at DO's side (in sec)

E. *k-NN Computation*

1) Computational Complexity: In our scheme, the computational complexity for Enclave E1 during k-NN computation is O(dm + m log k). The dot-product distance calculation requires O(d) for the dot product between each data and query point (total m data tuples need O(dm)). Calculating k-NN indexes requires O(m log k) as it can be done using modified heap sort. Similarly, in Zhu's scheme, the computational complexity at Cloud Server for this stage is O(mη + m log k). In Singh's scheme, the computational complexity at Cloud Server for this stage is O(mη + m log k + n^2) because of the extra multiplication of query point with matrix W at the start.

2) Experiment Results: We evaluated the time for k-NN computation in our design, Zhu's, and Singh's schemes. We set k = 10 and m = 10000 (tuples). Fig. 10 shows the average k-NN computation time over ten runs, with the dimension of data points varying from 100 to 1000. Singh's scheme takes a longer computation time than Zhu's scheme because it involves multiplication with extra matrix W at the start. The time complexity of our scheme is lower than theirs. But our design takes a bit more time because of Intel SGX. Intel SGX allocates only 128 MB for secure enclaves. Since we require more, more time is wasted in paging the database in and out of the reserved 128 MB. We don't need to worry about the security as the pages are encrypted while moving out of the reserved memory. Our scheme takes 0.9 sec compared to 0.4 sec and 0.45 sec in the case of Zhu's and Singh's scheme, respectively, when the dimension of data points is 1000. So our design is still practical.

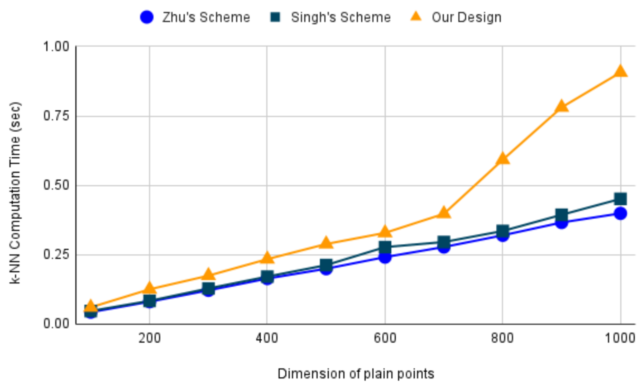

Fig. 10: Average k-NN Computation Time (in sec) when number of tuples = 10000

F. *Other Overheads in our Design*

In our design, we used Intel SGX for secure enclaves. We need to perform remote attestation between Data Owner and Enclave E1, Enclave E1 and Enclave E2, and Enclave E2 and Query Users. During this attestation process, RSA key pairs are generated and shared through the report (or quote) and used later to encrypt the messages between the involved parties for extra security. These attestation steps are one-time tasks only. For each query, an additional cost will be involved for AES and RSA decryption at Enclave E1 and Enclave E2. Similarly, the k-NN result will be encrypted and returned. These overheads will be far less compared to the overhead of Paillier encryption and, at the same time, will provide extra security.

VII. Conclusion

Previous works (using homomorphic encryption alone) hadn't considered collusion between the cloud server and query user (or between the cloud server and data owner). They also failed to provide Code Confidentiality and Integrity. Their scheme needs Paillier encryption to simultaneously provide Query Controllability and Key Confidentiality and requires many matrix multiplications. We identified these vulnerabilities and proposed a new design (using secure enclaves) for performing secure k-NN computation on the cloud. Our design removes previous works' vulnerabilities and provides data privacy, query privacy, key confidentiality, query controllability, data integrity, code integrity, and access pattern hiding. Our scheme also improves performance by eliminating Paillier encryption and multiple matrix multiplications, which takes too much time. Our design uses AES encryption to improve performance without compromising security. Other benefits include preventing collusion attacks and using different keys per query. Also, Data Owner need not be online. Our scheme can also be extended to provide code confidentiality.

References