Information Security For Multiple Storage Systems

Baby.D.Dayana, V.M.Hemeshwar, S.Aravind, Sachin S.Srivathsa, Krishna Purushoat

***Abstract:** In this paper we investigate a private information retrieval (PIR) scheme for secure distributed storage systems to provide additional security for systems in the presence of external security threats such as eavesdroppers. In the proposed system we use a query distribution system for splitting query requests across multiple databases to provide additional security.*

I. INTRODUCTION

In recent years there has been a large increase in the worldwide consumption of data. With internet being more accessible and more devices being connected to the internet the amount of data sent through the internet has increased exponentially. As the usage of storage systems dramatically, the amount of security vulnerabilities increases. Data is a valuable commodity in today's world. With the power of artificial intelligence and machine learning able to gather targeted information from large data dump, the privacy of data becomes a huge concern to an individual. With many major corporations collecting and selling private data worldwide, data security is important now more than ever. To address this issue there needs to be advancements made in the field of data security and privacy. The proposed system addresses this issue and implements a new system of query distribution which provides an incremental layer of security.

The majority of current services make use of a single centralized database to store their data along with backup databases. However, the storage of data completely in a database poses a lot of security vulnerabilities. The security of these databases varies from organization to organization. While bigger technology giants who spend huge amounts of money on security implement the best security measures possible, smaller startups cannot afford to so. If one were to gain access to the database through a backdoor or an internal eavesdropper, all the data in the database is exposed to the malicious party. The privacy of an individual data depends on the organization. The proposed system takes into consideration a private information retrieval (PIR) scheme that will help in safeguarding data privacy as well as data security. PIR has long been considered to be an important cryptographic problem in computer science. A PIR scheme is a protocol that enables a client to secretly recover the required messages from the databases, while hiding the index of the required message from every individual database. The goal of the PIR problem is to look for the most efficient solution for secretly recovering the ideal message with a little measure of downloaded data from the databases [1]. In a data theoretic definition, the limit of a PIR problem is defined as the most maximum proportion of the size of the required message that can be secretly recovered to the size of the total downloaded data from the databases. Two different scenarios are considered with respect to security breaches. The first is a scenario where the databases do not have the knowledge on the location of the fragments i.e. which database they are stored in or their indices. The second is a scenario where the databases are aware of the locations of the other data fragments. The latter scenario is not preferable as it may interfere with the PIR procedure as it is known that the capacity of the PIR is affected when the locations of the fragments are known. Regardless the PIR scheme used by the proposed system ensures that data privacy and security is never compromised in any scenario.The system proposed makes use of multiple databases located in various countries. Data to be stored is split into n subsets of data that do not make any sense when viewed individually or together with n-1 fragments of the rest of the data. To view the message, one must have all the n total fragments. These fragments are stored randomly across the databases located around the world. A Who-is database is also used to process the DNS requests that are generated. When a query is submitted by the user, the query is split into n fragments and sent to the different servers. The request is processed by the individual servers and a partial response is generated for each corresponding request. These responses are sent back to the user. The user then puts together the responses to view the data requested. The proposed system uses Python and Flask framework for its implementation. The databases are created and managed using Microsoft SQL Server Express. Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries[]. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. Microsoft SQL Server Express is a version of Microsoft's SQL Server relational database management system that is free to download, distribute and use.

Manuscript published on 30 April 2019.
* Correspondence Author (s)
Baby D Dayana, Assistant Professor, SRM Institute of Science and Technology, Chennai, India.
V.M.Hemeshwar, B.Tech CSE, SRM Institute of Science and Technology, Chennai, India.
Mr.S.Aravind, B.Tech CSE, SRM Institute of Science and Technology, Chennai, India..
Mr.Sachin.S.Srivathsa, B.Tech CSE, SRM Institute of Science and Technology, Chennai, India
Mr.Krishna Purushoat, B.Tech CSE, SRM Institute of Science and Technology, Chennai, India

It comprises a database specifically targeted for embedded and smaller-scale applications. The product traces its roots to the Microsoft Database Engine (MSDE) product, which was shipped with SQL Server 2000. The "Express" branding has been used since the release of SQL Server 2005.

II. SYSTEM ARCHITECTURE

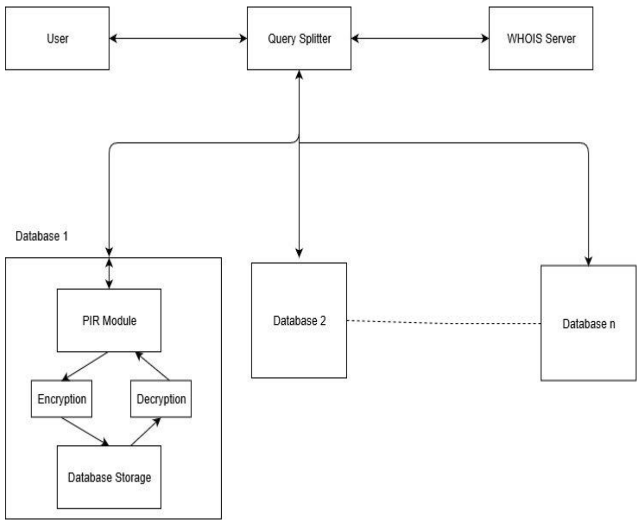


Fig. 1. Architecture of the proposed system.

Diagram 2.1 Architecture of Proposed System

The current system uses a single database situated in a centralized location. The proposed system utilizes multiple databases located in various locations all around the world. The system also utilizes a public Who-is server for DNS resolution. The databases utilize dual channel indexing. This indexing method involves the indexes of the data being stored being hidden from the server itself. This method provides anonymity for the data stored. External entities will not know the indices of the data. The data is split into fragments of equal sizes and stored across the databases. These fragments of data have no meaning when viewed individually or coupled with other fragments. If the data is split across n databases, then all the n fragments are needed to make sense of the data. Even with n-1 fragments, the data will not be usable. The data is encrypted using AES encryption algorithms for security. The user device first creates a request for data from the server. This request is split into n fragments where n is the number of databases used. These fragments are sent to the databases in a random order. The fragments are assigned random databases for every request to make sure that the data does not get compromised. The requests are resolved by the public Who-is server which resolves the DNS addresses of the databases to direct the requests to the respective databases. The requests sent by the devices are received by the database servers. The requests are arbitrary integer numbers. These requests are processed by the server using pre specified algorithms. The algorithm uses the request and performs computations with the index of the data stored. The result is generated and sent back to the user. The user upon receiving the response uses a reverse algorithm of the same nature to convert the response into the data needed. The n responses from the server are decrypted using the above method. Upon decryption of all the fragments, the result is compiled together to get the data required.

III. MODULES

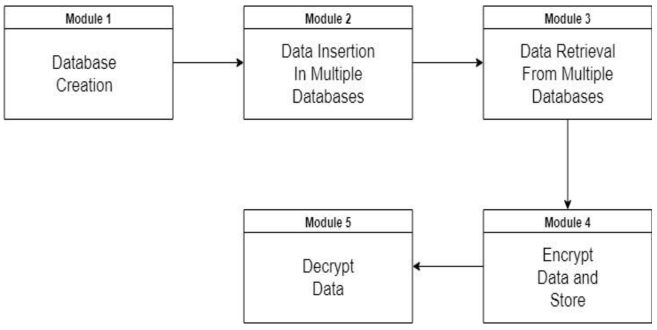


Fig. 2. Flowchart of the modules present in the proposed system

Diagram 3.1 Module Flowchart

The proposed system consists of five modules which are shown in the diagram above.

A. *Database Creation*

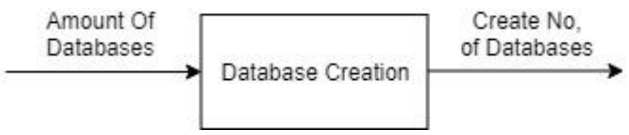


Fig. 3. Architecture of the Database Creation Module.

The first step towards creating the proposed system is to create multiple databases to store the data. In the real world application of the project, database servers would need to be rented from service providers across the world. Some examples of server providers are Microsoft Azure, Amazon Web Services (AWS), Oracle. For the project's simulation multiple virtual database servers will be created. The user inputs the number of servers needed to be created. The module creates the required number of servers. Theoretically, the more the no. of databases used, the better the security of the stored data. This increases the complexity of the task however and will require more bandwidth and consume more time.

B. *Data Insertion Into Multiple Databases*

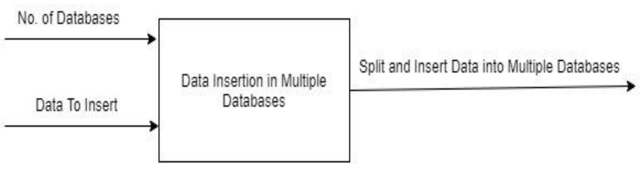

No. of Databases
Data To Insert
Data Insertion in Multiple Databases
Split and Insert Data into Multiple Databases

Fig. 4. Architecture of the Data Insertion Module

This module is responsible for inserting the data into the databases. The module first checks the number of databases in operation. After this it requests data to be input by the user. Upon receiving the data, the module splits the data into n fragments of equal size, where n is the no. of databases in operation. The fragments are encrypted using standard encryption algorithms and stored in the database.

Algorithm 1:

```
    Input: Data K, No. of Databases N

    for all K do

    Generate (N^(k-1) +(N-1)N^(K-1),N^K) fragments

    End for
    Randomize the indices of the fragments
    for n belonging to N do

    Store N^(k-1) fragments of message at Database N
    End for
  Data Retrieval From Multiple Databases
```

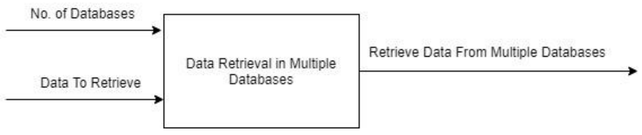

No. of Databases
Data To Retrieve
Data Retrieval in Multiple Databases
Retrieve Data From Multiple Databases

Fig. 5. Architecture of the Data Retrieval Module

The user submits a request for data. The module splits the query into n parts where n is the number of databases in operation. The subqueries are randomly assigned to a database. The public WHOIS server is used for DNS resolution. Each database upon receiving the request uses the PIR scheme to create a response. This response is the fragment of data present in that database. The response is sent to the user client. The responses from the n databases are collected and compiled.

Algorithm 2:

```
        Randomize the indexes of the fragments.

  Initialize N,K,D

        for i=1 to K do

          Download fragment $F_n$ from $DB_n$

        End for

        Arrange fragments in order of the indices
```

C. *Encryption of Data*

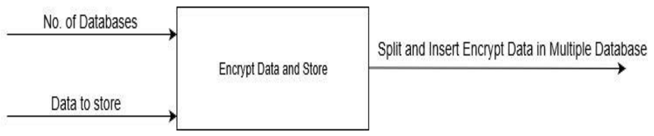


Fig. 6. Architecture of the Data Encryption Module

This module encrypts the data and stores it in the database. AES encryption is used.

D. *Decryption of Data*

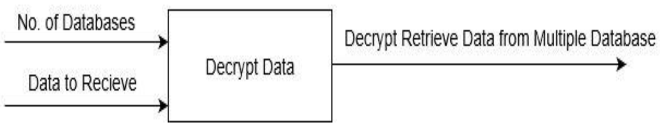


Fig. 7. Architecture of the Data Decryption Module

IV. CODE

A Database Creation

```
def ProcessCreateDatabase():
    global noofdb
    temp = request.form['dbcnt']
    dbcnt = int(temp)
    noofdb = dbcnt
    print(type(dbcnt))
    if type(dbcnt) != int :
        return  render_template('CreateDatabase.html',
processResult='No of Databases must be Integer')
  if dbcnt > 10 :
        return  render_template('CreateDatabase.html',
processResult='No of Databases must be less than
10')
    for index in range(dbcnt):
        connection = pypyodbc.connect('Driver={SQL
Server};Server=THARUNARJUN\SQLEXPRESS
3;Integrated_Security=true;', autocommit=True)
          cursor = connection.cursor()
        sqlcmd  =  "CREATE  DATABASE  db"  +
str(index+1)
        cursor.execute(sqlcmd)
        cursor.commit()
        connection.close()
        for index in range(dbcnt):
        conn1    =    pypyodbc.connect('Driver={SQL
Server};Server=THARUNARJUN\SQLEXPRESS
3;Integrated_Security=true;Database=db'+
str(index+1), autocommit=True)
        cur1 = conn1.cursor()
        if        not        cur1.tables(table='MyTable',
tableType='TABLE').fetchone():
            sqlcmd1  =  "CREATE  Table  MyTable
(SecretData nvarchar(MAX))"
            cur1.execute(sqlcmd1)
            cur1.commit()
            conn1.close()
```

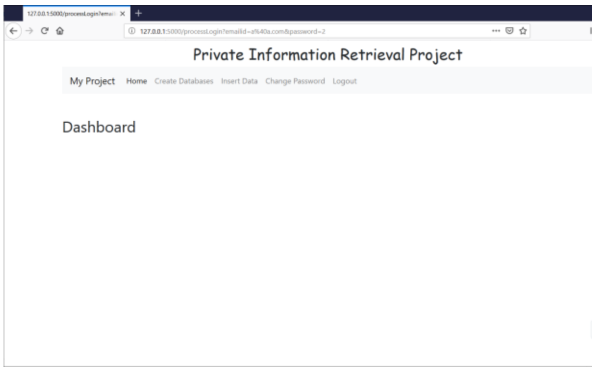
B. Data Insertion and Encryption

```
def ProcessInsertData():

    global noofdb
    sdata = request.form['sdata']
    if len(sdata.strip()) < noofdb :
        return  render_template('CreateDatabase.html',
processResult='Invalid Input Data')
    connection  =  pypyodbc.connect('Driver={SQL
Server};Server=THARUNARJUN\SQLEXPRESS
```

```
            3;Integrated_Security=true;', autocommit=True)
                cursor = connection.cursor()
                domain = sdata
                data = textwrap.wrap(domain,
            math.ceil(len(domain) / noofdb))
                for index in range(noofdb):
                    conn1 = pypyodbc.connect('Driver={SQL
            Server};Server=THARUNARJUN\SQLEXPRESS
            3;Integrated_Security=true;Database=db'+
            str(index+1), autocommit=True)
                    cur1 = conn1.cursor()
          if cur1.tables(table='MyTable',
    tableType='TABLE').fetchone():
              hash_object =
    hashlib.sha384(data[index].encode('utf-8'))
              hashdata = hash_object.hexdigest()
              sqlcmd1 = "INSERT INTO MyTable (SecretData)
    VALUES('"+hashdata+"')";
                        cur1.execute(sqlcmd1)
                        cur1.commit()
                        pass
                    conn1.close()
                    connection.close()
```

V. EXPERIMENTAL RESULTS

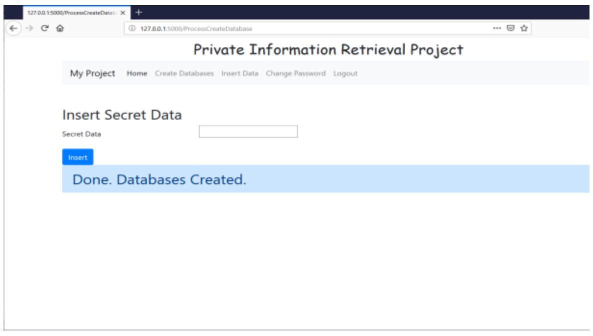

Private Information Retrieval Project

My Project Home Create Databases Insert Data Change Password Logout

Dashboard

Fig. 8. Homepage of the proposed system

The above image shows the Front End Homepage of the proposed system,where the frontend is created using HTML and JavaScript.


Private Information Retrieval Project

My Project Home Create Databases Insert Data Change Password Logout

Insert Secret Data

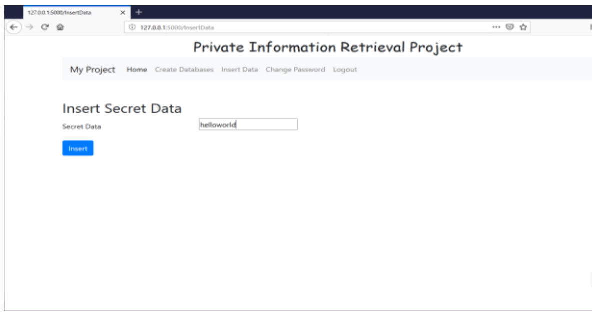
Secret Data

Insert

Done. Databases Created.

Fig. 9. Database Creation Module.

The number of databases is retrieved from the user as input and the databases are created.This module uses JQuery to submit the data in the JSON format.


Private Information Retrieval Project

My Project Home Create Databases Insert Data Change Password Logout

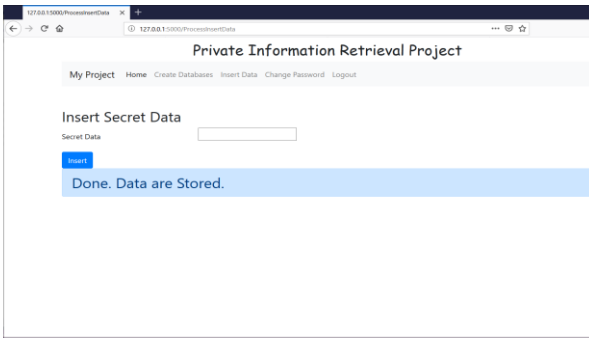
Insert Secret Data

Secret Data helloworld

Insert

Fig. 10. Data Insertion Module.

The data to be encrypted and fragmented is retrieved from the user, using Jquery in the JSON format.


Private Information Retrieval Project

My Project Home Create Databases Insert Data Change Password Logout

Insert Secret Data

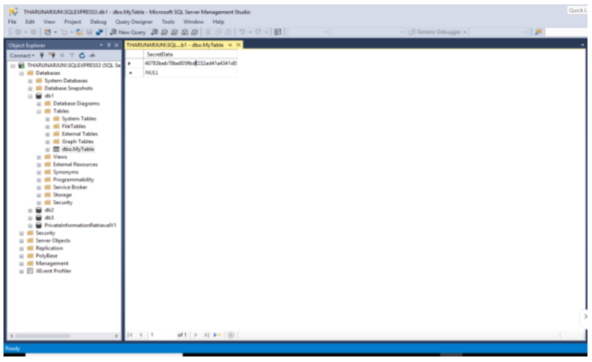
Secret Data

Insert

Done. Data are Stored.

Fig. 11. Acknowledgement of Data Insertion

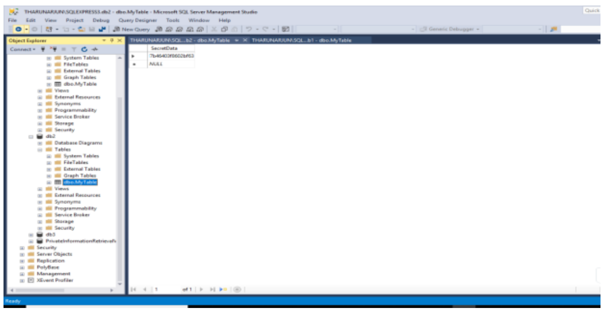
Once the data is successfully inserted, an acknowledgement is displayed.


THARUNARJUN\SQLEXPRESS3.db1 - dbo.MyTable - Microsoft SQL Server Management Studio

	SecretData
▶	40783beb78be805fbd232ad41a4341d0
*	NULL

Fig. 12. Database 1 with Encrypted Fragement

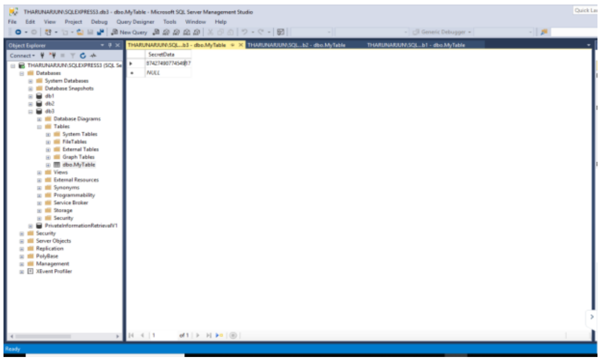
The given data is encrypted and fragmented and stored in the database in the form of an Hash.


THARUNARJUN\SQLEXPRESS3.db2 - dbo.MyTable - Microsoft SQL Server Management Studio

	SecretData
▶	7b46403f0602bf63
*	NULL

Fig. 13. Database 2 with Encrypted Fragement

The next part of the fragmented and encrypted data is stored in the next database.



```
SecretData
8742748077454817
NULL
```

Fig. 14. Database 3 with Encrypted Fragement

The final part of the data is stored in the last database. The given data is encrypted using AES Encryption Algorithm..

VI. Conclusion

In this paper, we considered a PIR problem for securing distributed databases in the case of an eavesdropper.
We have showed an application developed which applies the concepts considered to deliver a more secure system that protects information from both external and internal eavesdroppers. The proposed system, though currently small in scale, can be implemented on a larger scale for commercial purposes in the near future

REFERENCES

1. Heechol Yang, Wojae Shin and Jungwoo Lee "Private information retrieval for secure distributed storage systems,"
2. JS. Pawar, S. Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," IEEE Transactions on Infomation Theory, vol. 57, no. 10, pp. 6734-6753, Oct. 2011.
3. K. Banawan and S. Ulukus, "The capacity of private information retrieval from Byzantine and colluding databases," arXiv preprint arXiv:1706.01442, Jun. 2017.
4. W. Huang and J. Bruck, "Generic secure repair for distributed storage," arxiv preprint arXiv:1706.00500, Jun. 2017.
5. H. Sun and S. A. Jafar, "Multiround private information retrieval: capacity and storage overhead," arXiv preprint arXiv:1611.02257, Nov. 2016.
6. O. O. Koyluoglu, A. S. Rawat, and S. Vishwanath, "Secure cooperative regenerating codes for distributed storage systems," IEEE Transactions on Infomation Theory, vol. 60, no. 9, pp. 5228-5244, Sep. 2014.
7. A. Beimel, Y. Ishai, E. Kushilevitz, and J. -F. Raymond, "Breaking the O(n1/(2k-1)) barrier for information-theoretic private information retrieval," in Proc. the 43rd Annual Symposium on Foundations of Computer Science, Vancouver, Canada, pp. 261-270, Nov. 2002..

Authors Profile

Mrs.Baby D Dayana M.E, Assistant Professor (Department of Computer Science)
Publication : Spontaneous Message Detection of an annoying situation in community networks using mining algorithm.
Achievements: 36th Rank in Anna University.

V.M.Hemeshwar B.Tech CSE (Student)
Publication: First Person Shooter Game

Aravind Sriram B.Tech CSE (Student)
Publication: First Person Shooter Game

Sachin S Srivathsa B.Tech CSE (Student)
Publication: First Person Shooter Game

Krishna Purushoat B.Tech CSE (Student)
Publication: First Person Shooter Game

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)
© Copyright: All rights reserved.