Abstract-The current trend in the application space towards systems of loosely coupled and dynamically bound components that enables just-in-time integration jeopardizes the security of information that is shared between the broker, the requester, and the provider at runtime. In particular, new advances in data mining and knowledge discovery that allow for the extraction of hidden knowledge in an enormous amount of data impose new threats on the seamless integration of information. We consider the problem of building privacy preserving algorithms for one category of data mining techniques, association rule mining. Suppose Alice owns a k-anonymous database and needs to determine whether her database, when inserted with a tuple owned by Bob, is still k-anonymous. Also, suppose that access to the database is strictly controlled, because for example data are used for certain experiments that need to be maintained confidential. Clearly, allowing Alice to directly read the contents of the tuple breaks the privacy of Bob (e.g., a patient’s medical record); on the other hand, the confidentiality of the database managed by Alice is violated once Bob has access to the contents of the database. Thus, the problem is to check whether the database inserted with the tuple is still k-anonymous, without letting Alice and Bob know the contents of the tuple and the database, respectively. In this paper, we propose two protocols solving this problem on suppression-based and generalization-based k-anonymous and confidential databases. The protocols rely on well-known cryptographic assumptions, and we provide theoretical analyses to prove their soundness and experimental results to illustrate their efficiency. We have presented two secure protocols for privately checking whether a k-anonymous database retains its anonymity once a new tuple is being inserted to it. Since the proposed protocols ensure the updated database remains K-anonymous, the results returned from a user’s (or a medical researcher’s) query are also k-anonymous. Thus, the patient or the data provider’s privacy cannot be violated from any query. As long as the database is updated properly using the proposed protocols, the user queries under our application domain are always privacy-preserving.

1. INTRODUCTION

It is today well understood that databases represent an important asset for many applications and thus their security is crucial. Data confidentiality is particularly relevant because of the value, often not only monetary, that data have. For example, medical data collected by following the history of patients over several years may represent an invaluable asset that needs to be adequately protected. Such a requirement has motivated a large variety of approaches aiming at better protecting data confidentiality and data ownership. Relevant approaches include query processing techniques for encrypted data and data watermarking techniques. Data confidentiality is not, however, the only requirement that needs to be addressed. Today there is an increased concern for privacy. The availability of large numbers of databases recording a large variety of information about individuals makes it possible to discover information about specific individuals by simply correlating all the available databases. Although confidentiality and privacy are often used as synonyms, they are different concepts: data confidentiality is about the difficulty (or impossibility) by an unauthorized user to learn anything about data stored in the database. Usually, confidentiality is achieved by enforcing an access policy, or possibly by using some cryptographic tools. Privacy relates to what data can be safely disclosed without leaking sensitive information regarding the legitimate owner.

Recently, techniques addressing the problem of privacy via data anonymization have been enveloped, thus making it more difficult to link sensitive information to specific individuals. One well-known technique is k-anonymization. Such technique protects privacy by modifying the data so that the probability of linking a given data value, for example a given disease, to a specific individual is very small. So far, the problems of data confidentiality and anonymization have been considered separately. However, a relevant problem arises when data stored in a confidential, anonymity-preserving database need to be updated. The operation of updating such a database, e.g., by inserting a tuple containing information about a given individual, introduces two problems concerning both the anonymity and confidentiality of the data stored in the database and the privacy of the individual to whom the data to be inserted are related:

1) Is the updated database still privacy preserving?
2) Does the database owner need to know the data to be inserted?

Figure-1

---

K.P.Thooyamani, V.khanaa, M.R.Arun Venkatesh
Bharath University, Chennai
Email: thooyamani@hotmail.com, drvkannan62@yahoo.com, mr.arun.venkatesh@gmail.com
II. KEY CHALLENGES

In the existing system data are stored in database directly. Anyone can easily retrieve information like username, password. Etc. The cryptography security is not maintained here. The classification of database is carried out from local system only. Any unauthorized person can easily access the database. Authorized person can view the other user’s data too. Data confidentiality is particularly relevant because of the value, often not only monetary, that data possess. A requirement has motivated a large variety of approaches aiming at better protecting data confidentiality and data ownership. The availability of huge numbers of databases recording a large variety of information about individuals makes it possible to discover information about specific individuals by simply correlating all the available databases. Clearly, the two problems are related in the sense that they can be combined into the following problem: can the database owner decide if the updated database still preserves privacy of individuals without directly knowing the new data to be inserted? The answer we give in this work is affirmative.

III. PROPOSED SYSTEM

We propose two protocols solving this problem on suppression-based and generalization-based k-anonymous and confidential databases. The protocols rely on well-known cryptographic assumptions, and we provide theoretical analyses to prove their soundness and experimental results to illustrate their efficiency. It is today well understood that databases represent an important asset for many applications and thus their security is crucial. Recently, techniques addressing the problem of privacy via data anonymization have been developed, thus making it more difficult to link sensitive information to specific individuals. One well-known technique is k-anonymization. Cryptography technique is using secure data storing in server.

IV. DEVELOPMENT OF THE SYSTEM

After designing the data model, user interface, prototype, documentation plan, functional specifications, detailed design specifications and software quality assurance test plan, the next logical step is to start coding the actual software, writing the documentation and developing the SQA test cases, entrance criteria, and automations. Software coding includes building the data model and programming the user interface and application. If analysis and design efforts are correct, developing the software will not be a hardest task. Several versions include release of certain products. They are

a) Pre-alpha
Pre-alpha basically means that individual modules are ready, but have not yet combined them into a functional unit.

b) Alpha
Alpha is the functional version of software. It is the fundamental structure since it covers about 60 percent of the functionality.

c) Pre-beta
Normally pre-beta testing is used for in-house testing.

d) Beta
The beta version of a software product is its first release to be viewed.

e) GA
The GA or ‘Generally Available’ is a software release ready to install for use. This refers to the finished version of the software.

V. ARCHITECTURE AND EXPERIMENTAL RESULTS

Our prototype of a Private Checker (that is, Alice) is composed by the following modules: a crypto module that is in charge of encrypting all the tuples exchanged between an user (that is, Bob) and the Private Updater, using the techniques exposed in Sections 4 and 5; a checker module that performs all the controls, as prescribed by Protocols 4.1 and 5.1; a loader module that reads chunks of anonymized tuples from the k-anonymous DB. The chunk size is fixed in order to minimize the network overload. In Fig. 3 such modules are represented along with labeled arrows denoting what information are exchanged among them. Note that the functionality provided by the Private Checker prototype regards the check on whether the tuple insertion into the k-anonymous DB is possible. We do not address the issue of actually inserting a properly anonymized version of the tuple.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Objective</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous connection</td>
<td>Protect IP address and sensitive info</td>
<td>Crowds [27], Onion Routing [26]</td>
</tr>
<tr>
<td>Anonymous authentication</td>
<td>Protect sensitive authentication info</td>
<td>Policy-hiding access control [20]</td>
</tr>
<tr>
<td>Anonymous update</td>
<td>Protect non-anonymous data</td>
<td>Proposed in this paper</td>
</tr>
</tbody>
</table>

The information flow across the above mentioned modules is as follows: after an initial setup phase in which the user and the Private Checker prototype exchange public values for correctly performing the subsequent cryptographic operations, the user sends the encryption of her/his tuple to the Private Checker; the loader module reads from the k-anonymous DB the first chunk of tuples to be checked. Such tuples are then encrypted by the crypto module. The checker module performs the abovementioned check one tuple at time in collaboration with the user, according to either Protocol 4.1 (in the case of suppression-based anonymization) or Protocol 5.1 (in the case of generalization-based anonymization). If none of the tuples in the chunk matches the User tuple, then the loader reads another chunk of tuples from the k-anonymous DB. Note the communication between the prototype and User is mediated by an anonymizer (like Crowds, not shown in figure) and that all the tuples are encrypted.

We briefly discuss the complexity of our protocols in terms of the number of messages exchanged and their size. It turns out that the number of messages exchanged during executions of
Protocol 4.1 and Protocol 5.1 is bounded by a linear function of the number of witnesses of the anonymous database. Protocol 4.1 requires that Alice sends Bob the encrypted version of tuple _i_. Bob encrypts it with his own private key and sends it back to Alice. Further, Bob sends Alice the encrypted version of tuple t. Then, Bob sends Alice the encrypted values contained in t, in order to let Alice compute the actual, encrypted version of anonymized tuple t. Finally, Alice and Bob exchange the encrypted version of tuple _i_ for checking whether such tuple and the encrypted, anonymized version of t match.

Assuming the worst-case scenario, this has to be executed w times. Thus, the number of messages is 6 _w_. The complexity of Protocol 5.1 relies on the size of Tw (jTwj) and the complexity of the SSI protocol. The number of calls to the SSI protocol is bounded by Tw, and detailed complexity analyses of SSI can be found in [3], [13]. We implemented both Protocols 4.1 and 5.1 using mySQL 5.1 and C++ using the NTL libraries version 5.5 for the numerical computations. We tested our implementation on the Income database from the UC Irvine Machine Learning Repository [4]. The database has size equal to 50.7 MB and contains about 286k tuples. Such database has been anonymized using both suppression and generalization-based approaches, for values of parameter k equal to 2, 5, 10, 20, and 50. The resulting anonymized databases have been imported into MySQL 5.0. We then tested several times the insertion of a tuple in such anonymized databases. All the experiments were run on an Intel(R) Core(TM)2 1.8 GHz CPU with 1GB of physical memory running Linux Debian. We report the average execution times (expressed in milliseconds) of Protocol 4.1 and Protocol 5.1, respectively, in Figs. 4 and 5. The experiments confirm the fact that the time spent by both protocols in testing whether the tuple can be safely inserted in the anonymized database decreases as the value of k increases. Intuitively, this is due to the fact that the larger the k is, the smaller the witness set. Fewer are the partitions in which table T is divided consequently, fewer protocol runs are needed to check whether the update can be made.

Further, we report that the experiments confirm the fact that the execution times of Protocols 4.1 and 5.1 grow as dataset size P=k. That is, each protocol has to check the anonymized tuple to be inserted against every witness in the worst case, and the larger the parameter k is, the fewer the witnesses are. We report in Figs. 6 and 7 the cpu and network latency times for Protocols 4.1 and 5.1, as the parameter k increases. As it is shown, latency time accounts for a very large portion of the elapsed time for the executions of Protocols 4.1 and 5.1.

VI. RELATED WORK

In this paper, we have presented two secure protocols for privately checking whether a k-anonymous database retains its anonymity once a new tuple is being inserted to it. Since the proposed protocols ensure the updated database remains k-anonymous, the results returned from a user’s (or a medical researcher’s) query are also k-anonymous. Thus, the patient or the data provider’s privacy cannot be violated from any query. As long as the database is updated properly using the proposed protocols, the user queries under our application domain are always privacy-preserving.

Performing the update, once k-anonymity has been verified:
1. The specification of the actions to take in case Protocols 4.1 or 5.1 yield a negative answer;
2. How to initially populate an empty table; and
3. The integration with a privacy-preserving query system.

In the following, we sketch the solutions developed in order to address these questions and which comprise our overall methodology for the private database update. As a general approach, we separate the process of database k-anonymity checking and the actual update into two different phases, managed by two different sub-systems: the Private Checker and the Private Updater. In the first phase, the Private Checker prototype presented in Section 6, following Protocol 4.1 or Protocol 5.1, checks whether the updated database is still k-anonymous, without knowing the content of the user’s tuple. In the second phase, the Private Updater actually updates the database based on the result of the anonymity check; we refer to this step as update execution. At each phase, the database system and the user communicate via an anonymous connection as mentioned in Section 1 by using a protocol like Crowds or Onion routing. Also, legitimate users are authenticated anonymously via the protocol presented. Thus, the system cannot link the user who has entered the update request to the user who actually performs it. Concerning the actual execution of the database update, once the system has verified that the user’s tuple can be safely inserted to the database without compromising k-anonymity, the user is required to send to the Private Updater the non-anonymous attributes’ values to be stored in the k-anonymous database as well. The deployment of an anonymity system ensures that the system cannot associate the sender of the tuple with the subject who made the corresponding insertion’s request.
Suppose that a tuple fails the tests of Protocols 4.1 and 5.1. Then, the system does not insert the tuple to the k-anonymous database, and waits until $k - 1$ other tuples fail the insertion. At this point, the system checks whether such set of tuples, referred to as pending tuple set, are k-anonymous. Such test can be performed on encrypted data by using the methods proposed by Zhong et al. [35]. In the affirmative case, the system proceeds to insert the k-anonymous tuples to the database. In the negative case, the k-anonymization of the set of tuples failing the insertion is periodically checked, again by methods presented. Note that many issues need to be addressed for the approach described above to be effective. For instance, where and who is responsible for keeping the pending tuple set; how to inform and communicate with data users in order to initiate the protocol. We will address these issues in future.

In addition to the problem of falling insertion, there are other interesting and relevant issues that remain to be addressed:

- Devising private update techniques to database systems that supports notions of anonymity different than k-anonymity (see the discussion in [11]).
- Dealing with the case of malicious parties by the introduction of an untrusted, noncolluding third party [12].
- Implementing a real-world anonymous database system.

We believe that all these issues are very important and worthwhile to be pursued in the future.

**VII. ENHANCEMENT**

- Drawback is no original data can be got back.
- Previously using approximate values, data is classified where original data cannot be retrieved.
- In order to take the original data, a reference file is used.
- Reference file will be present in the local system.
- When reference file data is encrypted, the path will be taken.

**REFERENCES**