Introducing PATTY: Peer dATabase securiTY

Tony Young - 20161423
CS 856 - Fall 2004

December 3, 2004

Abstract

Database systems have been in use for decades to store information. No matter where they are used, databases have the power to organize our information and help us access it efficiently. Distributed databases have also been in use for many years. They provide improved query performance, a high level of expandability, improved reliability, easy data sharing, and improved availability of access [18]. Recently, peer-to-peer architecture has been applied to database systems. This architecture allows a system to act as both a client for performing queries and interacting with the user, and as a server to provide results to queries posed by other clients [17].

Peer-to-Peer (peer) databases are fundamentally different from traditional and distributed databases in several key ways: nodes may join and leave a peer network at any time, the schema for a peer database is not global, the data in a peer database might not be complete, and queries in peer databases must be routed to many nodes. Peer database systems can be applied to many fields, for example contact information management, development environment configuration management, genomics, and healthcare.

There are several key issues of importance to peer database systems. Among these issues are user authentication and security. Until now, security services have not been implemented in peer database systems, as the idea of organizing databases in a peer manner is relatively new. This paper presents a framework for Peer dATabase securiTY (PATTY). An implementation and evaluation are proposed. Some analysis of the security offered by PATTY is provided.

Contents

```
SELECT   Min(Price)
FROM     Products
WHERE    ProductName = 'Apple PowerMac G5 Dual 2GHz'
```

Figure 1: An Example Query

1 Introduction to Peer Database Systems

Database systems have been in use for decades to store information. Large organizations store employee, customer and product information. Healthcare professionals store patient information. Researchers keep data on research projects. No matter where they are used, databases have the power to organize our information and help us access it efficiently.

Distributed databases have also been in use for many years. They provide improved query performance, a high level of expandability, improved reliability, easy data sharing, and improved availability of access [18]. Distributed databases provide an organization the flexibility to tune storage and access protocols to suit their infrastructure. For example, data that is frequently used can be replicated to reduce response time. As well, data can be spread across many servers to increase redundancy.

Recently, peer-to-peer architecture has been applied to database systems. This architecture allows a system to act as both a client for performing queries and interacting with the user, and as a server to provide results to queries posed by other clients [17].

1.1 Fundamental Differences

Peer-to-peer (peer) database systems are fundamentally different from distributed and traditional database systems in several ways. One of these fundamental differences is that nodes may join and leave a peer network at any time. In distributed and traditional systems, nodes are added out of necessity (i.e. for redundancy or growth) and are known to the cluster ahead of time [16]. However, a peer may join an information-sharing network at any time, and should only have its resources added to the resource pool when it comes online. Also, peer nodes are not necessarily known ahead of time. This "arbitrary join" paradigm for peer systems poses challenges to finding data with queries, providing complete data through queries, and locating peers on a network.

Take the query in Figure 1 as an example. With this query, the user is looking for the lowest price for a product. If node n_1 contains a cheaper product than peer node n_2, and n_1 is unavailable, the user will receive an incorrect result from their query. Also, how does the node performing the query know to which IP addresses the SQL statement should be sent in order to reach other peers?

A second fundamental difference between peer and distributed or traditional databases is that the schema for a peer database is not global. In traditional and distributed systems, the schema is standardized across each node. In a peer database system, several schemas may be used to represent the same type of data at different nodes [16]. For example, imagine a peer network with two nodes each containing a database of contact information. Node n_3

```
TABLE:  Contacts {
        LName      VARCHAR   25;
        FName      VARCHAR   25;
        Phone      INT;
        Email      VARCHAR   50;
        Address    VARCHAR  100;
        }
```

Figure 2: An Example Schema for Node n_3

```
TABLE:  contact_info {
        Lastname      TEXT  15;
        Firstname     TEXT  15;
        Phonenumber   TEXT  10;
        Emailaddress  TEXT  40;
        Homeaddress   TEXT  50;
        }
```

Figure 3: An Example Schema for Node n_4

may store contact information in a schema such as the one in Figure 2. Also, node n_4 may store contact information in a schema such as the one in Figure 3. This "arbitrary schema" paradigm for peer database systems poses problems for query completeness. For example, n_3 might be searching for contact information and receive a false negative when querying n_4 with a statement such as the one in Figure 4. In this case, n_4 might actually contain a record for John Doe. But, because records are stored in a different schema, the query will not return it.

A third fundamental difference is that the set of available data in a peer network might not be complete. Distributed and traditional systems contain a complete set of data in each server cluster. However, a cluster of peers might not have the complete set of data required to accurately and completely answer a query [16]. This is because a node containing some information required by a query might be offline. Obviously, this query will not be able to return proper and complete results because of missing information. This "missing data" paradigm for peer database systems poses problems for query completeness and correctness. The former completeness, is obvious. If data is offline it will not be found and a complete result set will not be returned. The latter correctness, is somewhat more difficult to understand.

```
SELECT  *
FROM    Contacts
WHERE   FName = 'John' AND LName = 'Doe'
```

Figure 4: An Example Query

```
SELECT   Count(DISTINCT Address)
FROM     Customer
```

Figure 5: An Example Query

An incorrect result would be attributed to incomplete data providing a false result to a query. For example, take the SQL statement in Figure 5. If a peer containing part of the Customer table is offline, the user will not receive the correct number of distinct addresses.

A fourth fundamental difference is that queries in peer database systems must be routed to many nodes. In distributed systems, a query can be routed to a relatively small set of nodes. In a peer database system, the query must be passed to many nodes in order to return an accurate result set [16]. This "query routing" paradigm for peer systems poses problems for locating nodes, routing queries, retrieving results, and doing all this in an efficient manner[1]. For example, a peer system may contain upwards of 10 000 nodes. It is not feasible for a peer performing a query to store the IP address of each other node in the network. Further, contacting each node individually to submit a query and receiving results from all 10 000 nodes does not scale well. Such a system would require mass amounts of space at each node to store IP addresses, consume mass amounts of network bandwidth when submitting queries and retrieving result sets, and experience a long response time.

1.2 Applications

Peer database systems have many different applications. One application of peer database systems is to special interest communities. The internet holds many special interest groups that share information, have meetings, etc. It would be advantageous for such groups to have individual communities that could be used to share files, host chats, etc. [22]. This type of system would distribute information across many nodes and require some sort of efficient search mechanism to find data. A peer database system would be ideally suited for this purpose: information would not have to be centrally located, and users would be able to find information spread across multiple nodes with relative ease.

Finding contact information is another application of peer database systems. Frequently, people do not have a phone number or an email address for a person or business they are trying to contact. In such instances, a peer database system could be queried to retrieve the required information. Each person could store their own vCard, and the vCards of those people they know or have already searched for. Locating contact information would be fast as users would not have to look through old emails or a hierarchy of web pages in order to find the information they need. Also, caching would speed up subsequent queries by oneself or others for the same data.

A third application of peer database systems is development environment configuration management. When engineers work together to develop a piece of software, they must adhere to standards in order to ensure that everyone working on the project can follow the code and documentation. In programming environments, it can be particularly challenging to ensure

[1]efficiency in this context refers to low consumption of network resources.

that everyone uses the same conventions for variable naming, file structure, etc [7]. Due to the scale of some projects, it might also be necessary to distribute templates, project documents, etc. to many different people. Instead of sending this information via email, and in order to ensure that each person working on the project has the most up-to-date information, a query could be performed on a peer database system to allow each employee to download what they require.

Transfer of patient records is another application of peer database systems. Hospitals and doctor's offices must share information about patients: their medical history, contact information, etc. [16]. This sharing usually requires faxing or mailing large amounts of information. This process can take days to accomplish. If, for example, a patient has been taken to an emergency room because of complications resulting from surgery they had at a hospital 1 000 kilometers away, it is imperative that information take minutes rather than days to arrive. In these instances, it would be ideal to search hospital and doctor's office databases remotely to retrieve a patient's information.

Another application of peer database systems is storing banking information. Customers often hold accounts at one single bank branch. Such a branch services the client on a regular basis, but records are often stored elsewhere on a central server. This results in the branch needing to connect to the central server every time a transaction needs to be performed on the client's account. In this situation, a peer database system could speed up access to the client's records by distributing them across individual branches. Queries could then be performed over all branches in order to find account information for a client who goes to a different branch, an ATM, or another bank.

Another application of peer database systems is distributed file storage. Distributing office documents across employee computers is usually done as a consequence of work style rather than as a conscious decision. However, this distribution paradigm is actually quite efficient for storage. Adding a simple searching mechanism would allow employees to access files stored on each other's computers without the need for a large file server, or for file requests to be made by email, phone, etc. [22]. Of course, security protocols would be a required addition to such a system.

Criminology is the final application of peer database systems that will be discussed here. Law enforcement officials around the world keep DNA, fingerprints, criminal records, etc. in nationwide central databases for easy access. This however, creates a bottleneck at the server and the server becomes a single point of failure. As such, it would be advantageous to distribute this data across multiple nodes. For example, the agency that collected the evidence or arrested the individual could be responsible for holding records, updating them as new information is discovered, etc. Using a peer database system, officials from around the world would be able to search this information, and cache it for frequent use and to speed up searching by other agencies.

1.3 Important Characteristics

There are several important characteristics that must be taken into account when designing a peer database system [4, 5, 9, 13]

1.3.1 Scalability

Scalability refers to how well a system performs as the amount of load on it increases. A peer database system must be able to handle an ever-increasing community of users [9]. If a system performs adequately with 1 000 users, it should also perform adequately with 100 000 users. As the size of the community increases, so does the consumption of network bandwidth, the number of messages passed between peers, and the amount of information that a bootstrap node or central server must keep track of. Ideally, a peer database system will be able to scale in a less than linear fashion. Otherwise, an increase in size has the potential to choke the network a user community runs on top of, and overload the bootstrap nodes or central server attempting to hold the community together.

1.3.2 Availability

Nodes should be able to communicate with, and receive data from, each other. As well, data should be replicated at, and retrievable from, several sources [5]. We therefore define availability in the context of this paper to mean the ability to find and retrieve resources using a peer system. The importance of availability should be obvious. If a peer database system does not have a way to find or retrieve resources, it does not serve its purpose. In an ideal system, if user A has a resource and user B caches that resource for later use, user C should be able to obtain that resource from either user A or B.

1.3.3 Performance

The network should return results with the smallest latency possible. As well, communication between nodes should be as efficient as possible [13]. Efficiency in the context of this paper refers to a low consumption of bandwidth and as few messages being passed between nodes as possible. Performance is critical to any peer system because users of such a system expect to have fast access to resources. Also, the usefulness of a system often decreases as performance degrades. As such, for a peer database system to remain useful, it must provide adequate performance.

1.3.4 Data Authenticity

Data authenticity is concerned with determining whether or not responses to a query are factual [5]. As such, a node should be able to tell the difference between a correct and an incorrect query hit. For example, if a user is searching for contact information for John Doe, how does the user know that the information returned is correct? Furthermore, if two different sets of information are returned, both purporting to be for John Doe, how does the user determine which set of information is correct? Ideally, a peer database system would avoid returning incorrect results, or at least provide the user with a mechanism to aid in determining which information is factual.

1.3.5 Security

A peer system must ensure that authorized users are the only ones who make use of privileged data [4]. For example, in a system that shares patient records between doctor's offices and hospitals, only doctors, nurses, and their respective staff members should be able to access private information. As well, information should be encrypted during transport and while in storage to ensure that it cannot be stolen. In the context of this paper, we define security to mean that only authenticated users with proper privileges can access privileged system resources.

Two important aspects of security are:

1. *Authentication of Users*: User authentication is extremely important in order to facilitate sharing of sensitive information across an organization without fear of that information being used by those who are not privileged to see it.

2. *Data Encryption*: To prevent data from being stolen while in storage or transit, it should be encrypted.

This paper will focus on authentication of users and data encryption in peer database systems. Throughout this paper, the term *security services* will be used to mean user authentication and data encryption[2].

The rest of this paper is organized as follows. Section 2 provides an introduction to some peer database systems and provides a critique of the security services offered by them. Section 3 provides an overview of Peer dATabase securiTY (PATTY), as well as an implementation and evaluation proposal. Section 4 evaluates different proposed authentication protocols. Section 5 evaluates different proposed data encryption protocols. Section 6 concludes with some summary remarks.

2 Peer Database Systems

Several different peer database systems have been proposed in recent years [15, 16, 17, 19, 23, 26]. However, very little critical evaluation of such systems has been performed. In this section, four recently proposed peer database systems are presented and the security services they offer are evaluated.

2.1 Introducing APPOINT

APPOINT is an acronym for "Approach for Peer-to-Peer Offloading the INTernet." [23] It is a system that seeks to provide access to large amounts of spatial data in a fast and efficient manner. Users may share or obtain entire spatial data sets over the internet using the Spatial And Non-spatial Data (SAND) browser. SAND is a java-based browser that allows a user to visualize different spatial data sets that are obtained from APPOINT. When users first contact the central APPOINT server, they are asked whether or not they will share their files, extra

[2]The author recognizes these are only two components that make up a traditional set of security services in most distributed environments

disk space and bandwidth in a peer manner. If so, clients can be called upon by the central server to function as servers to other peers. If not, clients operate under the client-server paradigm.

APPOINT is a central server system and functions under a traditional client-server paradigm when system load is low. Users making requests to the system can download files directly from the server. The central server keeps track of what data sets a user has downloaded, as well as whether or not the user is online. When system load starts to increase, APPOINT defers download requests for data sets to serving peers in an attempt to optimize performance. The requesting user then downloads the file from a serving peer who has already cached it and the performance of the central APPOINT server remains acceptable. However, all queries are routed through the central server regardless of whether they will be deferred to serving peers.

Uploading data under APPOINT functions in a manner similar to downloading. When a peer wishes to share a data set and central server load is low, the data set will be uploaded to the central server and made available for download. If central server load is high, data sets are uploaded to other serving peers and eventually propagated to the central server when its load decreases.

Since APPOINT does not require any security mechanism (i.e. it is a freeware system), any user may download any file. For the current free distribution of information that the system is being used for, this does not pose a problem. However, should APPOINT be employed on a network with sensitive or private data, it would require security measures to be built in. As such, in environments that require security, APPOINT would not be able to deliver.

2.2 Introducing DBGlobe

DBGlobe is a knowledge management system that forms dynamic information "communities" of nodes, also called "Primary Mobile Objects" (PMO's), that store information of a common type [19]. Information at a PMO is represented as a service. Thus, PMO's may publish several services of information to the bootstrap servers so that user agents may retrieve and use them. PMO's may also cache services that they make use of in order to publish them for others. Conceptually, each node in the DBGlobe network contains a data store (the PMO) that stores and provides access to information, and an application that registers information with the bootstrap node and performs queries.

DBGlobe makes use of a number of "geographically" distributed Cell Administration Servers (CAS)[3]. These bootstrap servers are responsible for publishing services to PMO's for retrieval and to other CAS's to aid in query routing. A CAS also registers and holds metadata for the PMO's that fall within its assigned geographical area. CAS's receive and route all queries from PMO's to appropriate PMO's and CAS's that should be able to provide meaningful results. Since the CAS's store metadata for each PMO, this query routing is easy.

As mentioned above, DBGlobe tracks information communities. This is done through the use of Community Administration Servers (CoAS's). A CoAS keeps track of each PMO that contains information related to the theme of its assigned community. For example, a "history community" may be assigned to CoAS_2. As well as being tracked by its geographically defined

[3]It is unclear from the literature whether geographical area refers to physical location or network location such as IP address or domain.

CAS, a PMO that stores historical information would also be tracked by CoAS$_2$. When a PMO is looking for information about historical events, its query can then be routed to CoAS$_2$ to receive faster and more accurate results. Conceptually, a CoAS is identical to a CAS except that it tracks PMO's in information communities instead of geographic regions. Communities can be created statically by users, or dynamically by the CAS's analysis of commonalities in stored metadata.

DBGlobe makes use of filters to route queries to the appropriate neighbouring sites (i.e. PMO's, CAS's, and CoAS's) in a network. Each site maintains a "local" filter that summarizes all the services it provides, as well as a set of "merged" filters that summarize the services offered by its neighbours. Once a site receives a query (either from a PMO or from a neighbour), the query can be routed to nodes containing relevant data for further processing.

As with APPOINT, DBGlobe does not provide any authentication or encryption mechanisms to its users. As such, any user may make use of any piece of information in the system. As with APPOINT, security measures would need to be added should DBGlobe be used in a problem domain with sensitive data.

2.3 Introducing Edutella

Edutella makes use of a "super peer" organization strategy for connecting peers to each other [26]. Nodes wishing to join the network are either defined as peers or super peers. Super peers organize themselves into a hypercube topology with edge degrees defining a neighbouring super peer as degree 0, 1, etc. Peers then connect, using a clustering algorithm, to the super peer that will provide them with the best performance. Peers are organized in a star topology around their super peer and route all queries to it.

Super peers maintain two types of indices: "Super Peer / Peer" (SP/P) and "Super Peer / Super Peer" (SP/SP). The function of the two types of indices is the same: to hold metadata about peers connected to the network in order to route queries to only those neighbours who can provide results. The information stored in the two types of indices is what sets them apart. SP/P indices hold metadata for each peer that is connected to a specific super peer. Thus, each super peer has a different SP/P index. Queries arriving at a super peer are then routed to its peers that have relevant data to contribute. Data can then be aggregated and returned to the requesting peer through the connection of super peers. SP/SP indices hold metadata for each peer cluster. So, each super peer would report to its neighbours what information its peers hold. Metadata would propagate around the hypercube in a similar fashion to the way routing information propagates in the distance vector routing algorithm: all SP/SP indices would converge to a steady state over time. Each super peer would then know where to route a query in order to get it to a peer that has appropriate results, using other super peers as intermediaries.

Peers in the Edutella network route queries to their super peer who in turn forwards them through the network according to an edge degree system. When a query request is routed, the edge degree the message was forwarded on is included in the message. Then, when a super peer receives a query request, it only routes it to those relevant super peer neighbours along an edge with degree higher than the degree from which the query originated. This ensures messages are not sent to the same peer or super peer twice.

In the Edutella network, peers may be clustered to super peers in three ways: using “ontology clustering”, “rule-based clustering”, or “query clustering”. With ontology clustering, peers are assigned to clusters according to the data that they are “interested” in. For example, all peers that are interested in holding or searching for DNA data would be clustered into one group. With rule-based clustering, peers are clustered according to a certain set of rules. For example, peers could be clustered according to domain, IP prefix, average response time, average throughput, etc. With query clustering, peers are grouped according to the frequency with which they have performed queries on certain types of data. For example, the number of times they have searched for a person’s name, a product price, or a fingerprint. This requires peers to store statistics about the type of information they search for.

As with DBGlobe and APPOINT, Edutella does not provide any means to secure data on a network. Edutella’s current implementation aims to support the free distribution of information, meaning that it does not require security services. However, employing Edutella on a network requiring security services would mean Edutella would not provide a suitable solution.

2.4 Introducing PeerDB

PeerDB is a robust data management solution that supports data sharing through agent based communication with other peers [15, 16]. In a PeerDB network, each node has a database management system, local and export dictionaries, and an agent. The database management system is responsible for storing local data that the user has entered through the user interface. It also stores cached results of queries that have been performed by the agents. The local dictionary stores metadata and keyword descriptions for all locally stored data. The export dictionary stores metadata and keyword descriptions for all shared data, and is searchable by remote agents. Agents perform both local queries on the local dictionary and remote queries on peers export dictionaries.

PeerDB is built on top of the BestPeer architecture [17]. Using BestPeer, peers are organized into a mesh based upon neighbour assignments from a Location Independent Global Names Lookup (LIGLO) server. A peer initially contacts the LIGLO server and is assigned a unique identifier and a set of neighbour peers to communicate directly with. LIGLO servers store the IP address of the peer, its assigned neighbours, and its unique ID. In this manner, the server can keep track of peer status. When a user logs in again, the LIGLO server knows its new IP address and can assign it new neighbours. As queries are performed, peers make remote nodes neighbours if they are found to be providing many useful query hits. This self-reconfiguration of neighbours allows peers to communicate faster with those nodes that are providing relevant results to its queries. This reduces response time for relevant results, and allows the forming of information “meshes” similar in nature to Edutella and DBGlobe communities.

When a user creates a database schema in PeerDB, they assign keywords to it to describe the data they are storing [15, 16]. This schema is then searchable by keyword in order to avoid the problems associated with the arbitrary schema paradigm discussed previously. In order to perform queries on neighbour export dictionaries, peers make use of agents. Agents are dispatched to each neighbour of a peer when a query is issued. The agents will query the remote export dictionary by keyword and return any results to the user[4]. Each remote query has a time-to-live counter associated with it. If the counter is not zero when it reaches a neighbour, the counter is decremented and the agent is cloned and dispatched to all directly connected peers of the neighbour. Replies to queries are returned directly by the agent to the calling peer. Replies from the agents are then ranked for the user according to agent-determined relevance. The user then selects the data they wish to have returned and the agent fetches the full data set.

PeerDB provides some security for sensitive data by restricting what data may be searched by remote agents. This is done by restricting remote agent queries to the export dictionary. As with Edutella, DBGlobe and APPOINT, there is no mechanism to support searching of sensitive data by authorized users, or data encryption. However, by allowing only certain information to be exportable, PeerDB is one step closer to providing security services. As such, PeerDB still has relatively poor security support, but its mechanisms are slightly better than others.

It appears that much work still needs to be done in the area of security services. None of the four solutions evaluated properly supports these very important functions. However, PeerDB does come closest.

3 Meet PATTY

Peer dATabase securiTY (PATTY) is the main contribution of this paper. PATTY presents a method of authentication, practices for data encryption, and a secure routing protocol. PATTY combines techniques from current literature, and will be shown to provide robust services and protection.

3.1 Design Goals

The design goals of PATTY are simple:

1. *Authentication*: PATTY seeks to provide a secure means of authentication of peers. After authentication, access rights can be granted to data, allowing private data to be shared in a peer manner.

2. *Encryption*: PATTY seeks to provide a secure means of communication and data storage. Encryption will allow PATTY to ensure that private data remains private.

3. *Low Overhead*: PATTY seeks to provide its security services with as little overhead as possible. Thus, performance of user queries should not suffer because of the need to encrypt and decrypt data.

3.2 Influence from Current Literature

PATTY combines several approaches found in the presented literature [10, 11, 20, 26]:

[4]No data is actually returned at this point, only descriptions/keywords.

- *RSA*: The RSA [20] algorithm will be employed to encrypt all communications between peers. RSA was chosen because of its strong encryption and the ease with which a key can be revoked while still allowing the network to function.
- *IDEA*: IDEA [11] will be employed to keep all stored data safe. This method was chosen because of the ease of encryption using current hardware and the additional security provided over the traditional DES algorithm.
- *Remote Authentication*: As with remote authentication [10], PATTY nodes will be able to specify remote users and groups for access control. Authentication will take place using a modified version of the presented algorithm.
- *Edutella*: The peer organization structure of Edutella [26] is used by PATTY in order to ensure that queries are routed securely. Since Edutella peers only route queries to their super peer, the queries can be encrypted and passed securely to all other peers in the network. Also, Edutella appears to provide efficiency using the edge routing system. This reduces overhead, meeting design goal number 3 of PATTY.

3.3 Data Encryption Services

All data that is transferred between peers will be encrypted and decrypted using the RSA algorithm [20]. All data that is stored on a peer's disk is encrypted using the IDEA [11]. These algorithm have been proven secure, and are computationally inexpensive to perform using existing hardware. Using these methods, we argue that our data will be secure, thus meeting design goal number 2 of PATTY and satisfying part of the security requirements for a peer database system.

3.4 Authentication Services

Authentication is performed in much the same way as the remote authentication method [10]. Peers submit authentication information with their query requests, and the receiving node authenticates the requesting node based on its cached user and group records. Each node can specify access rights to tuples, tables, stored procedures, etc. as is possible in any fully functional database management system.

When a query request message is received, the receiving peer decrypts it and extracts the authentication information. The authentication information is decrypted using the included public key. The receiving peer then looks up the record for the user with the provided user name. The password is verified, and the receiver ensures that the public key used to decrypt the authentication information matches the public key stored in the user record. The IP address provided is decrypted with the stored public key. If all of the checks and decryptions are successful, the requesting node has been authenticated, and query processing can begin.

Super peers are outfitted as remote authentication servers. They may contain users and groups, and these remote principals may be added to local groups of peers in the network. In order to register themselves with the network, a peer may contact an administrator and create an account out of hand. Individual peers will still reserve the right to grant or deny access

Query Statement	The text of the query that the sender is attempting to pose to peers in the network
Authentication Information	The user name and password of the requesting peer, encrypted with the requesting peer's private key
Public Key	The public key of the requesting peer
IP Address	The IP address of the requesting peer, encrypted with the requesting peer's private key

Figure 6: A query request message

to their data, meeting PATTY design goal number 1 and satisfying the remaining part of the security requirements for a peer database system.

3.5 Query Routing

Queries are routed securely using a protocol similar to Edutella [26]. This section describes the routing protocol used by PATTY.

- When a peer submits a query to its super peer, it is encrypted with the super peer's public key. The message can then be read only by the super peer when it is decrypted using the super peer's private key. A query message contains the elements in Figure 6. The included authentication information can be decrypted by anyone. However, to authenticate, the information must first be encrypted with the requesting node's private key. Thus, even if a peer knows another's user name and password, they cannot use it to perform impersonation attacks. The requesting peer's IP address is encrypted as well to prevent redirection attacks.
- When the super peer determines where to forward the query next (i.e. to other peers and/or super peers), it is encrypted with each new receiver's public key and transmitted.
- When a peer receives a query, it decrypts the message. Authentication of the user is then performed with the included authentication information. If the requesting node has proper access priviledges, the query is run and a result set is obtained. To return query results to the requesting node, the data is encrypted using the requesting node's public key. Thus, the results will only be viewable by the requesting node. The result is then transmitted directly to the requesting node at the IP address included in the request.

3.6 Evaluation

This subsection outlines PATTY's defense against some common attacks, describes the metrics that will be used to measure the performance of PATTY, as well as the proposed implementation and experiments to be performed.

3.6.1 Attacks

PATTY may have to stand up to several types of attacks. Some of the most common are:

- *Interception*: PATTY is designed to be impervious to interception attacks with the use of RSA encryption for communications. Since messages are encrypted with the receiver's public key, they can only be decrypted and read by the receiver (who holds the private key).

- *Impersonation*: PATTY is impervious to impersonation attacks due to the robustness of the authentication protocol. Any peer may decrypt and read a peer's authentication information using their private key. However, the public key associated with a user name must be included in the query request message, the authentication information must be decrypted using that public key, and the public key must match the one associated with the user in the peer's authentication database. Since it is not possible for a malicious peer to encrypt the authentication information using the requesting peer's private key, authentication will fail if any of the information is tampered with.

- *Wear and Tear*: PATTY may suffer from wear and tear attacks on encryption keys. This is directly due to the number of messages that will be encrypted using these keys. However, the use of public-key cryptography reduces the number of messages that can be collected and analyzed.

- *Replay*: PATTY can fall victim to replay attacks by malicious peers. Query request messages can be resubmitted by a malicious peer. However, this type of attack is possible in traditional and distributed database systems as well. Further, although we are concerned about it, we do not have any way to protect against it at the present time.

- *Redirection*: PATTY is impervious to redirection attacks by malicious peers. Since the IP address of the requesting node is encrypted using their private key, the IP address cannot be modified and still properly decrypt.

- *Data Theft*: Although it is possible to steal base table data from a PATTY node, the data will not be readable as it is encrypted. However, if the encryption key is also stolen, data would be readable by malicious peers. For this reason, PATTY nodes keep the encryption key hidden on the host system.

- *Denial of Service*: It is possible to perform denial of service attacks in a PATTY network. Many requests could be submitted to a super peer at once in order to refuse messages from legitimate peers. No solution to this problem is presently known. However, since there are many super peers in the network, the network will remain usable and connected if a super peer fails.

3.6.2 Metrics

Several things need to be measured in order to evaluate the performance of PATTY:

Time (ms)	*Encryption ON*	*Encryption OFF*	Δ
Minimum			
Mean			
Maximum			

Table 1: Overhead comparison

Time (ms)	*Encryption*	*Decryption*
Minimum		
Mean		
Maximum		

Table 2: Encryption overhead

1. *Encryption Time*: How long does it take to encrypt a query request or reply?
2. *Decryption Time*: How long does it take to decrypt a query request or reply?
3. *Authentication Time*: How long does it take to authenticate a peer once the authentication information has been decrypted?
4. *Overhead*: How much additional overhead is required (on top of "traditional" processing time) to encrypt, decrypt and authenticate?
5. *Data Access*: How long does it take to encrypt and decrypt data in base tables?

3.6.3 Implementation Proposal

PATTY would ideally be implemented in Java in order to make it portable across platforms. In addition, several experiments would be run in order to measure the performance of the system:

1. *Sample Queries*: A batch of sample queries would be run on actual data. The queries would be run with encryption turned on, then run again with encryption turned off. In this manner, the overhead of the encryption services could be measured and recorded in a table such as Table 1. Comparisons can then be made as to the amount of extra overhead the encryption services impose.

Time (ms)	*Encryption*	*Decryption*
Minimum		
Mean		
Maximum		

Table 3: Data access overhead

2. *Encryption Test*: A batch of queries and result sets would be encrypted and decrypted. In this manner, it would be possible to measure the overhead of encrypting and decrypting query results and query request messages. Results would be recorded in a table such as Table 2. The amount of overhead PATTY imposes on communication can then be determined.

3. *Access Test*: A batch of accesses to data tables would be made and recorded. In this manner, it would be possible to measure the overhead associated with decryption of table data, and encryption of inserted data. Results would be recorded in a table such as Table 3. The amount of overhead PATTY imposes on data accesses can then be determined.

In this section, we have described a robust set of security services for peer database systems, and how they can be implemented and tested. The following sections will discuss how PATTY builds on many proposed security services.

4 Authentication Protocols

In this section, we take a critical look at some proposed authentication protocols for distributed and peer systems [1, 2, 6, 10, 12, 14, 24, 25]. Other methods exist but are not presented here as they bear significant similarity to the presented techniques.

4.1 Traditional Authentication

A method of authentication over the wire that involved encryption of user name and password elements with a shared secret key was proposed in [2]. In this manner, a node could authenticate itself against a list of user names and passwords at the authenticating node.

This method is simple and secure. However, it requires that a secret key be distributed. If this secret key is ever cracked or stolen, the entire system must shut down until a new key can be generated and distributed to the member nodes. Also, each node must store user name and password information for all nodes that are to have access to the data. PATTY uses public-key cryptography to avoid the problem of key cracking.

4.2 Group-Based Authentication

Several methods of authenticating a node as a member of a group have been proposed [6, 12, 14, 25]. With these methods, once a node is a member of a group, it is trusted by the group members and can send messages to them.

4.2.1 Troups

As mentioned in [6], a "Troup" is an acronym for a "Trust Group" in which each group member implicitly trusts all other members. A peer is admitted to the group by authenticating itself using a preselected exponent value, y_i. The group controller keeps track of a one-way accumulator value, z, and a peer's auxiliary value, aux_i. The requesting peer provides their exponent and auxiliary values to the controller, who calculates the accumulator and what the actual auxiliary value should be. If the accumulator and auxiliary values provided by the peer are valid, it is admitted to the group. Exponent and auxiliary values are encrypted using a secret key before being sent to the authenticating peer in order to ensure they are not stolen. To add or remove a node from the network, the accumulator and auxiliary values are recalculated to contain the new node's exponent, or without the removed node's exponent. The new auxiliary values for each node are sent as encrypted messages.

Some important details detract from this algorithm's usefulness. The encrypted values can be stolen and used by a malicious party to authenticate themselves to the network. Since these values are presented whenever a node makes a request, the malicious party can then submit requests at will to the system. In addition, the group controller is a single point of failure for the network, and vulnerable to denial of service attacks. Further, significant overhead is required to add or remove a node from the network as values must constantly be recomputed. PATTY does not fall victim to impersonation attacks due to the structure of its messages. Also, no overhead is required to add a node to the set of active nodes.

4.2.2 Secret Shares

A second group authentication protocol was proposed in [14]. With this protocol, a group is formed by a bootstrap node who assigns a secret key to the group, and a "secret share" of data to each member. A node then makes an authentication request to the group using the secret key and their secret share. The active group members vote to allow or deny the requesting peer access to the group's resources. Once a peer is admitted, they are given a group membership certificate that they present each time they make a request. This certificate is signed with the secret key of the group.

This algorithm appears to solve the single point of failure problem that we saw in the Troup approach. However, as with Troups, access after authentication is based on the group membership certificate which can be stolen by a malicious party. In addition, a large number of messages need to be sent when voting to authenticate a member. This introduces an attack vulnerability: a malicious party can continually submit authentication requests, slowing the network significantly. PATTY does not require a large overhead to authenticate users, and authentication requests cannot be used as an attack.

4.2.3 Secure Groups

In [12], Li et al propose a method of authentication to secure groups. This method relies on a bootstrapping node to create a list of user names and passwords for nodes that may join a group. Then, a node submits their user name and password encrypted with the public key of the group controller. The group controller authenticates the node and it may then send messages to group members.

This method avoids the theft of certificate problem we see with Troups and secret shares. This is because operations to be performed are encrypted along with the authentication information. But, this introduces the possibility of replay attacks. Further, as with Troups, we have a single point of failure and denial of service attack problem associated with the group controller. In addition, since user names and passwords are used, the potential of password cracking or theft of the user name/password lists exists. PATTY does not fall victim to cracking due to the encryption required when sending authentication information.

4.2.4 Strong Secure Groups

Weimerskirch et al [25] presented a method (similar to Li et al) that double-hashed user name and password values before storing them to prevent theft of authentication lists. In this manner, a user name and password combination is hashed before being encrypted for transport to the authenticating node. The values are hashed again at the authenticating node before being compared to the values stored in the lists.

This double hashing ensures that password cracking is not possible, and stolen lists are unusable. However, strong secure groups still do not solve the other problems associated with secure groups.

Clearly, the problems associated with group-based authentication preclude it from widespread use in peer database systems.

4.3 Certificate-Based Authentication

Some authentication methods using certificates have recently been proposed [1, 24]. Other methods exist but are not presented here as they bear significant similarity to the presented techniques.

4.3.1 Credential Certificates

In [24], a method of authentication using credential certificates is presented. With this method, a node is given a certificate signed with the issuing site's private key. The certificate contains the user authentication information as well as permissions that the node has at a local site. Then, to perform an operation at a site, a node submits the operation they wish to perform as well as their credential certificate. The site then checks to see if the certificate issued is legitimate, and performs the operation if the certificate contains sufficient privileges.

This method of authentication overcomes many of the problems associated with group-based protocols. Credential certificates cannot be forged as they must be signed by a local site's private key. However, the entire encrypted certificate can still be stolen and presented to perform operations. In addition, each node must store $n - 1$ certificates for a network containing n nodes. Further, these certificates must be generated and distributed in some manner. Thus, the scalability of this method is questionable. PATTY does not require the overhead of certificate generation and distribution. Also, much less data must be stored at the requesting peer for authentication to the receiving peer.

4.3.2 Signed Certificates

Adya et al [1] presented a method of signed certificates for authentication. Using this method, a certificate authority generates a certificate containing a user name and password for a node. The certificate authority then signs the certificate with their private key. A node presents the certificate in order to authenticate. The certificate is presented each time a node performs an operation. The public key of the certificate authority is published in order to allow peers to decrypt the certificate.

Using this method, we see many of the advantages experienced with credential certificates. As well, a peer must only store one certificate for use on the network. But, as with credential certificates, signed certificates are vulnerable to theft. In addition, a certificate authority is a single point of failure and prone to denial of service attacks. Further, if the keys of the certificate authority need to be revoked for any reason, each certificate generated by it becomes invalid and must be regenerated and redistributed. When revoking a key in PATTY, no data needs to be regenerated.

Certificate-based methods come closer to satisfying the authentication requirements of a peer database system. However, some additional work is required to solve the problems with these methods before they can be used in practice.

4.4 Remote Authentication

In [10], Kaminsky et al proposed the use of access control lists and the caching of remote user and group records stored on remote authentication servers. Using this method, each peer adds remote groups and users to local access groups. Then, the local authentication server caches the authentication information stored at the remote site. Periodically, the cached records are compared to the records stored at the remote sites. The changes to each record are propagated to the local servers. To authenticate, a user submits an encrypted authentication request message to the local site. The site then decrypts the request and assigns credentials to the connection that was established. Future requests over the same connection have the same credentials.

This algorithm proposes the specification of remote groups and users to allow them access to data at local sites. It avoids the single point of failure problem as well as denial of service attacks. However, as with many of the other methods presented, it is still vulnerable to theft and replay of the encrypted authentication request message. PATTY makes use of certain aspects of the remote authentication method. Further encryption is used to prevent impersonation attacks if authentication information is stolen.

5 Data Encryption Protocols

In this section, we take a critical look at some data encryption methods that have been proposed [2, 3, 8, 11, 20, 21].

5.1 Data Encryption for Storage

Several methods have been proposed for encrypting data that is to be stored on disk. Three of the most popular are presented here [3, 8, 11, 21].

5.1.1 Chaotic Maps

In [8], a method of encrypting data using 1-D chaotic maps is presented. Chaotic maps use two different encryption functions with different bases and exponents to encrypt data. In this manner, it is virtually impossible to break the cypertext apart to determine the bases and exponents. In addition, the double-encryption prevents a malicious peer from using a wear and tear attack to determine the values used.

This method of encryption requires the storage of several different values on the user's system, and is computationally expensive to perform on data strings of large size (such as those of data tables in a database). Thus, chaotic maps are not ideally suited to encryption for a peer database system as PATTY seeks to minimize the additional overhead imposed by encryption.

5.1.2 Data Encryption Standard

As noted by [3, 21], the Data Encryption Standard (DES) is one of the oldest and most popular encryption methods for storing data. Although DES can be used for data that is to be communicated, it is typically applied to storage of encrypted data on disk. DES is a secret key system with a key length of 56 bits. Sixteen rounds of encryption are performed in which the left and right halves of a document are swapped and alternatively encrypted using a transformation function with XOR operations.

DES is quite useful for encrypting data. However, many researchers believe that the key length is insufficient to prevent current computers (with their increased computing power) from breaking the key. Thus, many incarnations of DES have been proposed that attempt to make it more secure by increasing the bit length. PATTY makes use of a DES variant with a larger key for increased security.

5.1.3 International Data Encryption Algorithm

The International Data Encryption Algorithm (IDEA) [11] was proposed in order to extend the life of DES. IDEA extends the key length of DES from 56 to 128 bits. Then, data is encrypted by performing sixteen encryption rounds on 64-bit chunks of data. Encryption is performed using a subset of 56 bits from the key, and the subset of bits can change during each encryption round. Thus, breaking the cyphertext is nearly impossible.

IDEA provides quite strong encryption using a shared key of longer length than DES. The authors hope that IDEA will remain unbreakable for many years as the pace of computing power increases. IDEA has been designed using functions that are computationally inexpensive using current hardware implementations. Thus, it is the method of data encryption employed by PATTY.

5.2 Data Encryption for Communication

Several methods have been proposed for encrypting data that is to be transmitted to another party. Two of the most popular are presented here [2, 20].

5.2.1 Shared-Key Encryption

In [2], Bird et al review several shared-key encryption techniques. In these techniques, the same key is used for encryption and decryption of data. Thus, the key must be kept secret from all non-members of a group. The difference between the reviewed algorithms is the method in which keys are generated and distributed.

All shared key techniques have the same problems associated with them. First, shared keys are subject to heavy wear and tear attacks. With these attacks, the key can be determined if enough messages are intercepted and analyzed. As well, if the key is ever compromised, a new key must be distributed to each node in the network before the network can continue operation. Due to the high volume of traffic in peer database systems, this method is not appropriate as malicious users will be able to collect a significant number of messages with which to perform wear and tear attacks. PATTY does not need to be shut down if one key is compromized. Update cycles will keep keys up to date as they are revoked and renewed.

5.2.2 Public-Key Encryption

In [20], perhaps the most famous of public-key encryption methods is presented: the Rivest, Shamir, Adleman (RSA) algorithm. The RSA algorithm provides a public and private key pair generation method. In public key systems, a node's public key is published to allow anyone to encrypt or decrypt data that is sent to it. A peer can encrypt data using their private key and send it to other peers who can decrypt it using the sender's public key. Conversely, peers can encrypt data using a receiving node's public key, and the receiving node can then decrypt it using their private key.

Public key encryption systems are very secure. Since the key pair is unique to the node, significantly fewer messages can be collected than with shared-key systems by a malicious peer for use in a wear and tear attack. As well, revoking a public key does not stop the network from functioning. Thus, public key encryption will be used in PATTY.

6 Conclusion

In this paper, we have presented a framework for Peer dATabase securiTY (PATTY). PATTY provides authentication, data encryption, and secure routing of query request messages in a peer database system. PATTY can withstand many common security threats, including impersonation, interception and data theft. An implementation and some performance experiments have been suggested. Unfortunately, due to time constraints, an actual implementation could not be completed.

References

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

[2] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. The kryptoknight family of light-weight protocols for authentication and key distribution. *IEEE/ACM Trans. Netw.*, 3(1):31–41, 1995.

[3] C. Boyd. Modern data encryption. *Electronics & Communication Engineering Journal*, 5:0954–0695, 1993.

[4] M. Ciglaric and T. Vidmar. Management of peer-to-peer systems. *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 1530–2075, 2003.

[5] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *Proceedings of the 9th International Conference on Database Theory*, pages 1–15. Springer-Verlag, 2002.

[6] S. Gokhale and P. Dasgupta. Distributed authentication for peer-to-peer networks. *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on*, 2003.

[7] Andre van der Hoek, Dennis Heimbigner, and Alexander L. Wolf. A generic, peer-to-peer repository for distributed configuration management. In *Proceedings of the 18th international conference on Software engineering*, pages 308–317, Berlin, Germany, 1996. IEEE Computer Society.

[8] M. Jessa. Data encryption algorithms using one-dimensional chaotic maps. *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, 1, 2000.

[9] M. Jovanov. Scalability issues in large peer-to-peer networks. Accessed October 17th, 2004 from http://www.ececs.uc.edu/ mjovanov/Research/paper.html.

[10] Michael Kaminsky, George Savvides, David Mazieres, and M. Frans Kaashoek. Decentralized user authentication in a global file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 60–73. ACM Press, 2003.

[11] M.P. Leong, O.Y.H. Cheung, K.H. Tsoi, and P.H.W. Leong. A bit-serial implementation of the international data encryption. *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000.

[12] Zupeng Li, Yuguo Dong, Lei Zhuang, and Jianhua Huang. Implementation of secure peer group in peer-to-peer network. *Communication Technology Proceedings, 2003. ICCT 2003. International Conference on*, 1, 2003.

[13] J. Liu, Q. Zhang, X. Zhang, and W. Zhu. gmeasure: a group-based network performance measurement service for peer-to-peer applications. In *Proceedings of the 2002 IEEE Global Telecommunications Conference*, volume 3, pages 2528–2532, 2002.

[14] M. Narasimha, G. Tsudik, and Yi Jeong Hyun. On the utility of distributed cryptography in p2p and manets: the case of membership control. *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, pages 1092–1648, 2003.

[15] W. Ng, B. Ooi, and K. Tan. Bestpeer: A self configurable peer-to-peer system. *ICDE*, 2002.

[16] W. S. Ng, B. C. Ooi, K. L. Tan, and A.Y. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *Intl. Conf. on Data Engineering*, 2003.

[17] Beng Chin Ooi, Yanfeng Shu, and Kian-Lee Tan. Relational data sharing in peer-based data management systems. *SIGMOD Rec.*, 32(3):59–64, 2003.

[18] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Englewood Cliffs, NJ, USA, 1991.

[19] Evaggelia Pitoura, Serge Abiteboul, Dieter Pfoser, George Samaras, and Michalis Vazirgiannis. Dbglobe: a service-oriented p2p system for global computing. *SIGMOD Rec.*, 32(3):77–82, 2003.

[20] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1):96–99, 1983.

[21] M.E. Smid and D.K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76:0018–9219, 1988.

[22] H. Sunaga, M. Takemoto, and T. Nakata. Applications of an advanced peer-to-peer platform. *Peer-to-Peer Computing, 2003. (P2P 2003). Proceedings. Third International Conference on*, 2003.

[23] Egemen Tanin, Frantisek Brabec, and Hanan Samet. Remote access to large spatial databases. In *Proceedings of the tenth ACM international symposium on Advances in geographic information systems*, pages 5–10, McLean, Virginia, USA, 2002. ACM Press.

[24] Vijay Varadharajan. Security enhanced mobile agents. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 200–209, Athens, Greece, 2000. ACM Press.

[25] Andre Weimerskirch and Dirk Westhoff. Identity certified authentication for ad-hoc networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 33–40, Fairfax, Virginia, 2003. ACM Press.

[26] Nejdl Wolfgang, Siberski Wolf, and Sintek Michael. Design issues and challenges for rdf- and schema-based peer-to-peer systems. *SIGMOD Rec.*, 32(3):41–46, 2003.