View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Sabanci University Research Database

A Practical Privacy-Preserving Public Key Repository

By

RAMIN ARMANFAR

Submitted to the Graduate School of Engineering and Natural Sciences

In partial fulfillment of

the requirements for the degree of

Master of Science

Sabancı University

March, 2017

A Practical Privacy-Preserving Public Key Repository

Approved By:

Prof. Dr. Albert Levi .....
(Thesis Supervisor)

Assist. Prof. Dr. Cemal Yılmaz .....

Assist. Prof. Dr. Cengiz Toğay .....
(Uludağ University)

Date of Approval: 27.03.2017

© 2017 Ramin Armanfar

All Rights Reserved

Dedication

To my beloved family...

Acknowledgments

First of all, I would like to express my sincere gratitude to Prof. Albert Levi, for his continuous support and worthwhile guidance throughout my academic life. Without his support and mentoring, this thesis would not have been completed. Prof. Levi was always accessible and willing to help anytime I needed; I feel proud as he placed confidence in me more than I do. Also I am very thankful to my thesis defense committee members: Prof. Albert Levi, Assist. Prof. Cemal Yılmaz, Assist. Prof. Cengiz Toğay for their support and presence.

I appreciate Merve Can Kuş Khalilov for her help during the evaluation process. I would also like to thank to my friends Mahmoud Al-Ewiwi and Marco Chiappetta for their help in the curriculum courses. Prof. Kemal Kılıç deserves special thanks for his precious supports. Last, but not the least, I am immensely thankful to my family, for being there when I needed them to be. The project under which this thesis has been produced is supported by Kuveyt Türk R&D center and Kobil Security Solutions, part of TÜBİTAK TEYDEB project 3141000.

Abstract

Internet and mobile users have been using financial institutions' alternative channels for their financial transactions with an increasing rate. In order to avoid frauds, the financial institutions make use of second factor authentication tokens such as one-time passwords sent to mobile phones as text. Another trend of such transaction verification is utilizing fully cryptographic protocols, in which the transactions are signed by the users. In the implementation of such an approach, in order to provide end-to-end security between the financial institution and its client, each client must have a public-private key pair. In some cases, especially for small-scale institutions, such a transaction verification system is fully outsourced as a Cloud service including clients' public keys. However, even in this outsourced model, the institutions need to access their clients' public keys for end-to-end security. In such a case, in order to provide privacy of the clients against the outsourced database, we need a privacy-preserving public key repository. In this thesis, we developed such a privacy-preserving public key repository based on Path ORAM mechanism. We have developed adaptation layers for Path ORAM so that the queries are performed via regular SQL queries and the data is stored in

a regular relational database, rather than Path ORAM's non-standard data structure. In this way, the non-standard features are hidden from both the financial institutions and the Cloud provider. We analyzed the performance of our system under different database sizes, network connection models and query types. We conclude that such a Path ORAM based system is feasible to be used in a practical system since even with a regular computer used as a server, the computational overhead is at marginal level.

Özet

Internet ve mobil kullanıcıların finansal kurumların alternatif kanallarını kullanımı gittikçe artmaktadır. Dolandırıcılıktan korunmak için finansal kurumlar telefonlara kısa mesaj olarak gönderilen tek kullanımlık şifreler gibi iki faktörlü kimlik doğrulama yöntemlerini kullanmaktadırlar. İşlem doğrulama için kullanılan diğer bir yönelim ise işlemlerin kullanıcılar tarafından imzalandığı kriptografik protokollerdir. Bu tip yaklaşımların gerçeklenmesinde müşteriler ile finansal kurumlar arasındaki uçtan uca güvenliği sağlamak adına her müşterinin bir açık-gizli anahtar ikilisine sahip olması gerekir. Özellikle küçük ölçekli kurumlarda, bu tip bir işlem doğrulama sistemi ve açık anahtarların saklanması Bulut servisi olarak taşeron hizmeti olarak alınabilmektedir. Ancak bu tip bir taşeron modelinde bile kurumlar uçtan uca güvenliği sağlamak için kullanıcılarının açık anahtarlarına erişmek isteyebilirler. Bu nedenle, kullanıcıların mahremiyetlerini taşeron veritabanı hizmet sağlayıcısına karşı koruyabilmek için mahremiyeti koruyan açık anahtar deposuna ihtiyaç vardır. Bu tezde Path ORAM mekanizmasını kullanan bu tip bir mahremiyet korumalı açık anahtar deposu geliştirilmiştir. Bu kapsamda sorguların normal SQL sorguları olduğu ve verilerin de Path ORAM'in stan-

dart dışı ağaç veri yapısı yerine ilişkisel veritabanlarında saklandığı bir durum için Path ORAM bağdaştırma katmanları geliştirdik. Böylece standart dışı öğeler hem finansal kuruluştan hem de Bulut sağlayıcıdan saklanmış oldu. Sistemimizin başarımını değişik veritabanı boyları, bağlantı modelleri ve sorgu tipleri için analiz ettik. Bunun sonucunda da Path ORAM tabanlı sistemimizin normal bir bilgisayarın sunucu olarak kullanıldığı durumda bile marjinal seviyede ek işlemsel maliyet getirdiğine ve pratik olarak kullanılabileceğine kanaat getirdik.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Privacy of the Internet users has became one of the most important concerns as the Internet and mobile usage increase. Privacy in the Internet world is directly related to data privacy, which is defined as keeping the individuals' identity, habits, personal information and other critical data immune from unauthorized disclosure.

User's shared data such as photos, comments, etc could be considered as his/her private information. However, there are also some other types of data that are being used by the users to perform users request which can reveal some sensitive facts about user's habits, customs, behavior, or any other statistical information. Those kind of user data can be abused by malicious users or even servers. Over the last few years, by developing modern types of communications as well as faster ever communication equipment, public database services become more and more widespread. From hotel booking

services to stock exchange market, public databases are being used in a wide variety of areas. Hence, users need to send their requests in the form of SQL queries to those public database servers in order to fetch their requested data. In some cases, however, users' queries could be considered as very sensitive and private user information since it is possible to infer very crucial and private information about the user and his/her habits. Thus, it is extremely important to provide privacy of the users' queries against malicious database server administrators or any other authorities, which are suspicious to take advantages of inferred data for their personal benefits.

Keeping users' queries private against database administrators is considered as a complex problem, because without accessing user's query it is not straightforward for the database server to perform operations to fetch user's requested data from database. Furthermore, cryptographic primitives such as encryption/decryption operations are not proper to be used as a "privacy-provider" against malicious database servers, although it can be used to keep information secure form any third parties. Hence, in order to provide user's query privacy, we need to take advantage of cryptographic protocols. In the literature, there are several such protocols, such as PIR (Private Information Retrieval) [3–5, 11, 17, 20, 27, 30], OT (Oblivious Transfer) [1, 2, 7, 8, 18, 19, 22, 29], and ORAM (Oblivious RAM) [6, 10, 12, 13, 16, 26, 28, 31, 34, 35].

Among these cryptographic protocols, Path ORAM [9, 36] is the most appropriate and suitable to be utilized in practical and real-world applications thanks to its properties which makes it faster and provides acceptable level of privacy.

In this thesis, we utilized Path ORAM protocol to adapt it as a "privacy-preserving" tools in an outsourced database systems to keep users' data private from untrusted database managers in a cloud settings. More specifically, we utilized Path ORAM in order for a banking system to outsource its customer database to the cloud. We developed a model for a middleware to implement Path ORAM in SQL environment so that query operations are performed in a transparent manner. Eventually, we evaluate efficiency and effectiveness of our system in such a banking application setting by developing a practical software systems.

The rest of the thesis is organized as follows. Chapter 2 gives the motivation and the problem definition. Chapter 3 reviews some existing cryptographic privacy protocols in the literature. Chapter 4 presents our proposed adaptation for Path ORAM in a practical setting; mainly for outsourcing the customer database of a bank to be queried using SQL in a privacy-preserving manner. Chapter 5 gives details of our implementation and discovers experimental results. Finally, Chapter 6 concludes the thesis.

Chapter 2

Motivation and Problem Definition

In recent years, by developing technology, modern types of communications are being used by most people and enterprises around the world. Nowadays, by utilizing smart phones and the Internet, we are able to do anything anywhere and faster than ever before. However, those benefits are not without cost and they bring along several challenges. One of major concerns in any kind of modern communication methods is privacy and security of users and their information which are being transferred in the network. These concerns become more significant in case they are used for more sensitive or critical purposes such as financial, commercials, military, etc.

Financial and commercial data transactions methods have also being changed rapidly by appearing cutting-edge communication technologies. De-

pends on the application, there are wide variety of security and privacy methods and protocols such as authentication, authorization, confidentiality, non-repudiation necessary to be used to ensure that the transactions are being performed in a safer and reliable way.

For any financial transaction operation in banking system, there are some security and privacy operations that have to be applied in order to guarantee users' security as well as authenticity of that transaction per se. Utilizing cryptographic protocols can help us to provide security and privacy which are required to perform those transactions. One of most important part of any financial transaction is users' authentication. In other words, banks need to confirm users' identity before starting to perform transaction operations.

In today's world, financial institutions generally use classical username/password authentication technique with OTP (One-Time-Password) based second factors. However, this strategy has some disadvantages as will be discussed in the next subsections. Thus, there is a need for a more securer and privacy-providing user and transaction authentication and verification methods, which is our main focus in this thesis.

2.1 Two Factor Authentication Protocol

Two Factor Authentication (TFA) is an authentication protocol, which is being used in many banking systems as well as other corporations. The

protocol works based on separate secure communication channel to confirm sender's identity. In such protocols, users send their primary authentication information, such as password, PIN code, along with his/her requested transaction information, to the bank. Immediately after receiving user's request, bank checks for user's identification information in its database. In case those information is entered correctly, bank sends a randomly generated One-Time-Password (OTP) back to the user but this time using separate communication channel than the first one, such as SMS, email, etc. Subsequently, user has a limited amount of time (usually between 1 to 3 minutes) to resend received OTP back to the bank server using primary communication channel. If user can successfully send OTP back to the server in the given period of time, bank will confirm the identity of the user. Finally, in the next stage bank will perform user's requested transaction operation. Figure 2.1 indicates two factor authentication operation steps. Note that OTP is a randomly generated code and it is allowed to be used just once for any kind of transaction. Therefore, for each distinct transaction of any user there should be a unique randomly generated code in order to prevents any abuse or attack.

Most significant advantage of this OTP based mechanism is that it prevents possible attacks in the primary communication channel. However, it still has some vulnerabilities. One of known attacks to this protocol is message forwarding attack that exists in some platforms and operating systems such as Android. In message forwarding attack, any received message to the user is sent to other third party applications (malwares) without user's awareness. Therefore, an attacker can use it to get OTP code. Phishing is

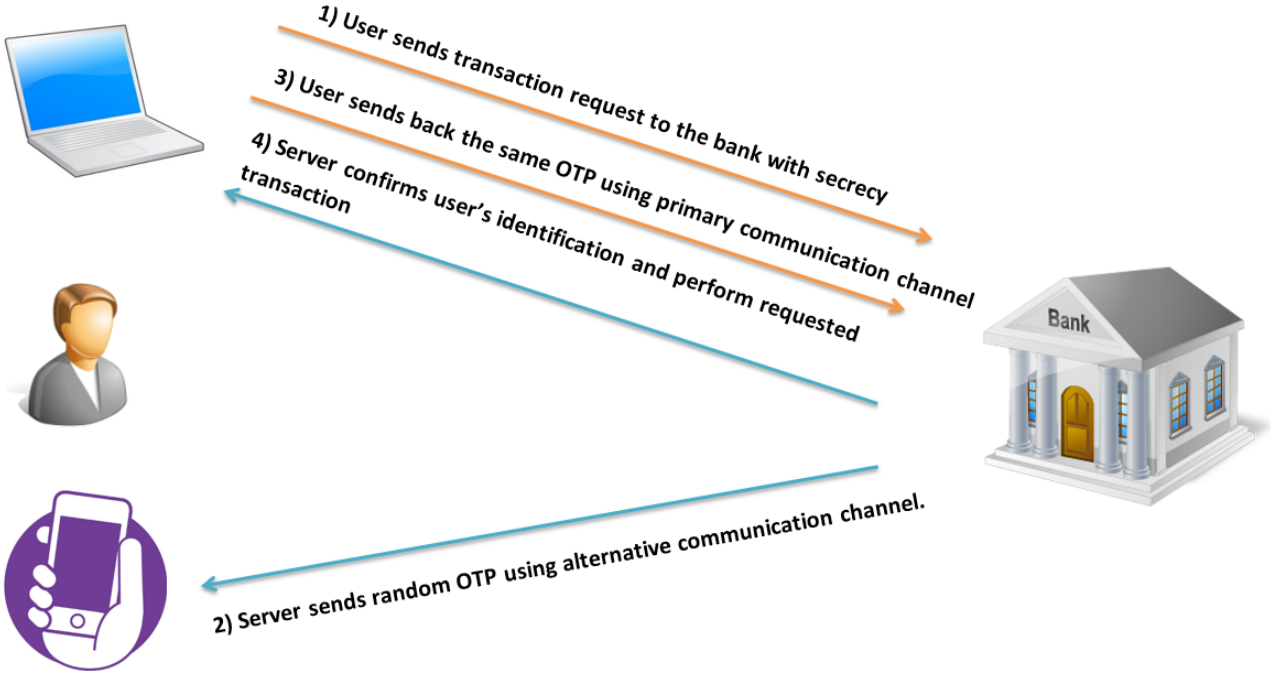

Figure 2.1: A Generic Two Factor Authentication Protocol.

another possible attack. Here, attacker tries to opens its own designed application in users device, which is very similar to the bank application in terms of GUI, to get user's OTP. Then, attacker asks user to enter his/her received OTP. As soon as user enters the code, attacker can obtain the OTP then uses it instead of user. To prevent phishing attack, users must make sure that he/she is connected to the genuine bank site by verifying the SSL/TLS certificates before entering their OTP. However, most of the ordinary users have no technical security knowledge; therefore, it is considered as a big challenge in this protocol.

Main transactions of all banks are financial operations. Security operations, specially in small banks could be outsourced. However, outsourced

operations can cause vulnerabilities in the financial transaction system. For instance, users' information and keys may needs to be stored in the outsourced database. Because of this reason, is very important, security and privacy of this method to be addressed.

2.2 Problem Definition and the Main Working Model

Instead of using OTP based mechanism, employing cryptographic mechanisms (such as digital signature) for transactions verification would improve the security level of the system in a more positive way. On the other hand, small banks and companies may prefer not to employ separate servers for this purpose and prefer to outsource this as a cloud service. Outsourcing of services such as identification and signature based transaction confirmation operations means to delegate cryptographic operations to cloud servers. In such a setting, cloud servers will also keep necessary keys of the customers so that cryptographic operation can be outsourced as well.

However, using such a cloud-based outsourcing model has its own challenges; cloud servers may not be completely trusted; therefore, it is essential to establish end-to-end encrypted communication between banks and customers through the cloud. Figures 2.2 and 2.3 show the steps of operations in an outsourced cloud-based model.

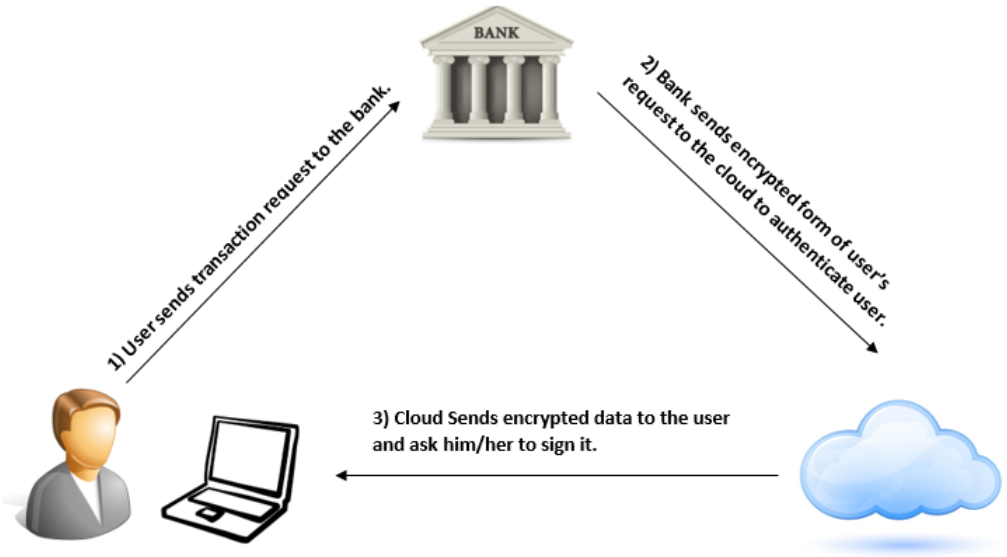

Figure 2.2: Transaction operations in an outsourced cloud-based authentication and transaction verification model (Phase 1).

In this model of authentication and verification, there are two pairs of key. First key pair (public/private keys) is utilized for end-to-end encryption and decryption between the bank and customers. Second pair is used to authenticate customers; each customer of the bank has a private key which is stored in his/her device, while the public keys of all customers are stored in the outsourced server database. The reason is that, outsourced server is responsible to perform verification and authentication operations of the customers during transaction verification process. As indicated in Figure 2.2, in the first step, customer sends transaction request to the bank server. Then, bank encrypts customer's request in such a way that only corresponding customer will be able to decrypt it. Then, the bank server sends encrypted request of the customer to the cloud server to get the customer's

authorization on the transaction. Afterwards, cloud server passes encrypted information to the customer and asks him/her to sign it. At this point, the customer decrypts received message and checks whether the transaction request belongs to him/her. After that, he/she signs the received message using his/her private key and sends it back to the cloud server. As shown in Figure 2.3, immediately after receiving the signature of the transaction request from customer, cloud server validates customer's signature using his/her corresponding customer public key. However, the idea here is that the cloud server should not be able to learn customer's identity because of his/her privacy against outsourced server, which is considered as an untrustworthy server. Owing to the fact that, outsourced servers are not trusted, we need a privacy-preserving mechanism to let cloud server confirm user's identity but without leakage any information about his/her identity. Finally, cloud server sends the authentication result back to the bank server and bank server can either perform customer's transaction request or reject it based on the verification result.

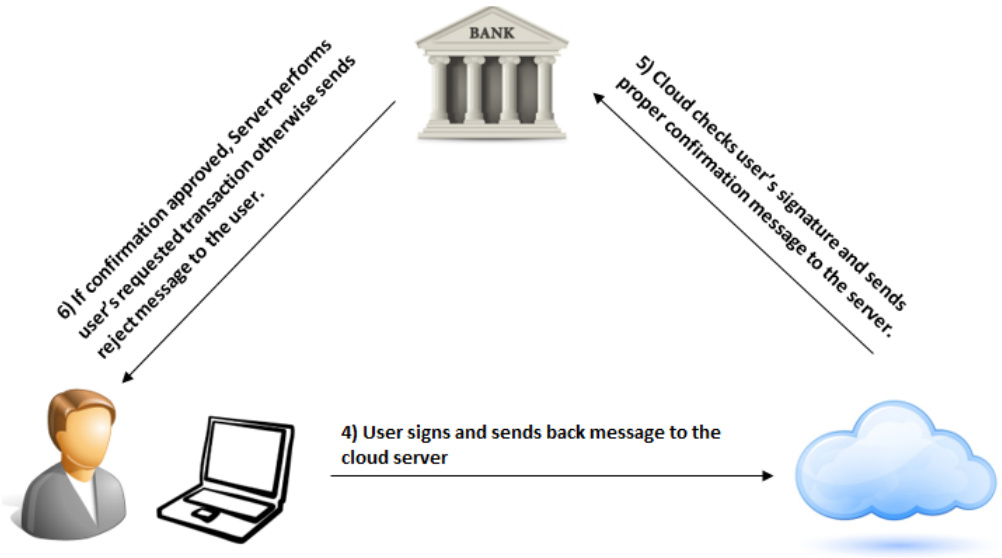

Figure 2.3: Transaction operations in an outsourced cloud-based model (Phase 2).

As it is described above, and shown in Figures 2.2 and 2.3, the bank is considered as a client for outsourced cloud server. Therefore, the bank sends its request to the cloud server in order to perform the authentication and verification process. However, since outsourced server is not trusted, we need to keep customer information private. This is considered as a challenging problem and a hot research topic.

One of the subtopics in this challenging privacy problem is that cloud server may need to keep the customer database of the bank including the customer's public key for end-to-end (bank-to-customer) encryption. The bank needs to query this database to get the customer public key in order to encrypt the transaction details. Thus, here it is needed to have a system in

which both (i) customer data will be kept in a secret way, (ii) bank queries will not be seen and processed by the cloud in a clear way.

Furthermore, nowadays, almost all database systems are rational database systems such as MS-SQL Server [21], SQLite [24], mySql [25]. The problem here is that, however, there is no any built-in mechanism to provide customers' privacy against database server administrator in rational database systems. Thus, we have to either design an intermediate application to be placed between rational database server and its users to hide their access pattern track or devise completely different database system to be deployed instead of rational database systems.

2.3 The Contribution of this Thesis

In this thesis, we focus on the problem of customer database outsourcing for a bank to be used in transaction verification model. In this model, the bank needs to query the customers' public keys in a secure and privacy preserving manner. Moreover, due to existing practices of the bank, a rational database system must be used and the queries must be performed using regular SQL queries. We adopt on existing privacy preserving data access model, namely Path ORAM [35, 36], for these purposes.

Chapter 3

Background and Literature Overview

Nowadays, users' privacy has become a major concern in any aspect of communication networks. In the last few years, by appearing new communication methods such as cellphones and smart cards, almost all users' information are available on the network and opens doors for many types of electronic crimes. Therefore, it is vital to provide not only security mechanisms but also privacy. On the other hand, each application or communication model requires diverse models of privacy-enhancing protocols in order to make sure that data transactions meet acceptable level of privacy and security. However, all of those privacy models have a similar goal which is hiding users' data from untrustworthy access of any third-party except authorized parties.

3.1 Privacy in Database Systems

There are several studies and researches in academia to provide users' privacy in many different communication and networking systems. Thus, hiding users' access pattern as well as their requested data from database server is a specific aspect of users' privacy which is still a hot topic in academia. Accordingly, in most of applications, a database system is demanded in server side to keep users' or products information in order to perform users' requested transactions. Since database server is considered as an owner of the stored data in the database, it can access all data and users access patterns. However, as we discussed in the previous chapter, in some applications such as outsourced servers, database administrators are not trusted (considered as a third-party application). Nevertheless, we need to provide privacy of users' against untrustworthy database server which is a tough problem in database systems, as database server needs to access the queries of the users in order to process them. Furthermore, privacy-providing protocols in database servers can be categorized into two main categories: (*i*) utilizing an intermediate application to provide cryptographic protocol along with existing database systems, (*ii*) creation of a new database system with the required cryptographic protocols from scratch and replace it with existing database system. These two models are shown in Figures 3.1 and 3.2.

Figure 3.1: Providing user' privacy with rational database system using intermediate application.

Figure 3.2: Database server with built-in privacy protocol.

3.1.1 Utilizing Intermediate Application

Although in the first model which is shown in Figure 3.1, there is no need to change the database system, it is hard to devise an intermediate application to interact with both users and database systems since, there are wide variety of different database systems. Also each specific application has its own communication and data structure policies. Nevertheless, it is very challenging to design such a generic application which can operate with different kind of applications. Furthermore, any intermediate application needs to access database system internal management layers or administrator privilege access to the data stored in the database, while no database systems allow any third

party applications to access their internal management layers because of data integrity, correctness, and security. In addition, benefiting an intermediate application is costly in term of communication and processing complexities. Due to the fact that, any data transaction between users and server needs to be sent and processed in that intermediate application before and after transaction process in the server side, it will have more processing load as well as communication cost. Finally, designing and implementation of this protocol is also challenging, because we need to decide where and how this intermediate application should be implemented and deployed. Ian Goldberg et al [30] proposed a method called TransPIR which extends Private Information Retrieval (PIR) systems such a way that SQL queries can be privately evaluated over rational database systems. Figure 3.3 shows functionality of their proposed method.

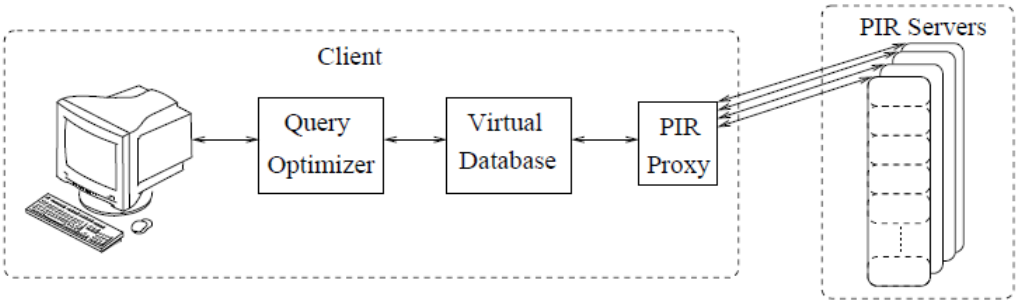

Figure 3.3: TransPIR, an intermediate application proposed by [30] to convert PIR to rational database.

In their proposed model an intermediate application (TransPIR) lies between users and database system. However, as it is shown in the Figure 3.3, query processor, virtual database, and PIR proxy part of TransPIR protocol

are located on the client side. As a result, the client side needs to have powerful communication and computation power in order to perform all those operations, which is not practically feasible for all applications.

3.1.2 Generating New Database Model

In contrast, second transaction model which is shown in Figure 3.2, needs to be designed and implemented independently by using cryptographic protocols. However, this model needs to design all essential functionality of database system as well. Additionally, abstraction is another important issue which is very important to be considered in the designation of this protocol. Abstraction means that, designation, implementation, and deployment of the database system should be completely separated from each other and should be hidden from end user of the system. Therefore, end-users of the system no need to have any detail information of implementation of the system. In other word, end-users of the system can send their SQL command to the system and obtain their requested data without any knowledge about the detail of system implementation.

3.2 Privacy Protocols Against Untrustworthy Database Servers

We evaluate three most-addressed cryptographic "privacy-providing" protocols which can be used against untrustworthy database servers. The protocols are 1) Private Information Retrieval (PIR), 2) Oblivious Transfer (OT), and 3) Oblivious Random Access Memory (ORAM). We have studied most of previous works and evaluate some of them to evaluate their efficiency in a real-world application. Hence, following sections are the detailed evaluation of each aforementioned protocols.

3.3 Private Information Retrieval (PIR)

The goal of PIR is to provide requested data access to the users, without revealing which data is accessed to the database administrator. Since untrustworthy database server can follow user's queries and infer what data the user wishing to access. PIR is a one-way cryptographic primitive which means that it just provides users' privacy against database server but the inverse is not provided by this protocol. In other word, user has a privilege of accessing to all information in the database. In some application which is being involved with public databases such as hotel and flight booking services, it is not considered as a problem if users can access to partial or even all information in the database since they are stored in public form.

Very simple and trivial way to achieve PIR is to download entire contents of database from the server therefore server could not infer which data has been retrieved by the user. Although this method has a high theoretic privacy level and also it is considered as the ideal way to achieve PIR, it is not feasible to utilize it in most of practical applications. Due to the fact that, most of the databases in an actual applications are very huge in size, besides it is not applicable to transfer that amount of information to the user in every transaction. In fact, there are two main mechanisms to address PIR protocol: first one, is to make database server computationally bounded and second one, is to use multiple non-cooperating database servers such a way that each of them have a similar copy of data.

3.3.1 Single-Database, Computationally Bounded PIR

According to Claude Shannon [33] theory, a cryptographic protocol is secure if it cannot be broken by even an unlimited computing power. The first single-database computationally private information retrieval (CPIR) scheme has been introduced by Kushilevitz and Ostrovsky [17] in 1997. They achieved communication complexity of $O(n^{\epsilon})$ for any $\epsilon > 0$ in their works where n is the number of stored bits in the database. In their scheme, database is an $n - bit$ string x, which user can send index i to obtain the bit x_i while hiding the index i from the database server. Their proposed scheme security is based on the quadratic residuosity problem. Later, in reference [3] Christian Cachin, et al. presented a new CPIR scheme based on poly-logarithmic communication complexity with intractability assump-

tion called $\Phi - HidingAssumption$ (ΦHA). In reference [19] Helger Lipmaa proposed a new scheme which reduced the communication complexity to $O(l \log n + k \log^2 n)$, where l is the length of the strings and k is the security parameter. This protocol could be used as a CPIR as well due to its lower communication complexity, although, it is an oblivious transfer protocol. In reference [11] Craig Gentry, et al. proposed a novel scheme with communication complexity of $O(k + d)$ where $k \geq \log n$. n and d are the database size and the bit-length of the retrieved database block respectively. The significance of their work was that, users requested data size was independent from data retrieving operations. The security of their scheme was based on simple variant of $\Phi - HidingAssumption$ (ΦHA) was introduced by Cachin, et al. [4]. As indicated by Ostrovsky, et al. [27] the proposed schemes by Kushilevitz, et al. [17] and Lipmaa [19] use the same ideas based on homomorphic encryption. However, the Kushilevitz and Ostrovsky protocols are based on Goldwasser–Micali cryptosystem while the protocol by Lipmaa is based on the Damgård–Jurik cryptosystem.

3.3.2 Multiple-Database, Information Theoretic PIR

Information-theoretic PIR protocol, is based on multiple non-cooperating database servers, each having a similar copy of the same database. Chor et al [5] introduced the method in the 1995 for the first time. In their scheme user can access to k databases ($k \geq 2$) then he/she can privately retrieve data stored in the database. In other word each individual database server which is holding a replicated copy of the same database infer no information

related to the identity of the item retrieved by the user. In their works, they presented two-server scheme and and achieved communication complexity $O(n^{1/3})$.

3.3.3 Overview of PIR

There are also many other PIR schemes of both single-server and multiple-server models have been addressed by other researches. Some of new discovered protocols achieved better communication complexity. For instance, single-server (Computationally private) PIR can be accomplished with constant communication and k-database (information theoretic) PIR can be achieved with $n^{\frac{\log\log k}{k\log k}}$ communication. However, beside all those improvements in communication and computation complexities of PIR protocols there are still major problems remains. Most of the protocols which have been proposed in academia are not practically implemented and not proper to be used in real applications. For instance, we evaluated one of recent works by Carlos Aguilar-melcho and Philippe Gaborit [20]. Their works was a single-server lattice-based PIR scheme which uses matrices. We implemented the protocol and evaluated it with various number of data. however communication complexity in the scheme was not acceptable for practical applications.

3.4 Oblivious Transfer Protocol (OT)

Oblivious Transfer (OT) is an cryptographic primitive which can be considered as a two-way version of PIR protocol which means that none of the sender and receiver can infer any information related the data except the one that they have privilege access them. However, it cannot be considered as an replacement for PIR because oblivious transfer imposes an additional privacy requirement for the database, on the other hand, PIR requires communication sub-linear in n, while oblivious transfer has no such necessity. In this protocol, sender sends more than one encrypted data items to the receiver and receiver are able to decrypt only authorized data which is belong to him/her. Therefore, sender could not figure out which data item has been accessed by the user also receiver could not decrypt other data except the one belong to him/her. Oblivious transfer protocol can be thought as an stronger version of PIR because it not only provides users' privacy against database but also users cannot access other data except their own data. However, depends on the application, we can decide whether we need two-way privacy or not. In this thesis, database privacy is not considered as a crucial problem because the data stored in the database are public and everybody have a privilege to access them. However, still it is feasible to use this protocol instead of PIR protocol.

Oblivious Transfer (OT) Protocol was firstly introduced in 1981 by Michael O. Rabin [29]. In his proposed model which was based on PSA public key cryptosystem, the sender sends two messages to the receiver with probability of $\frac{1}{2}$. Therefore both sender and receiver are remain oblivious

to each other. The common and more useful oblivious transfer protocol called 1-2 Oblivious Transfer or "one out of two oblivious transfer" (OT_1^2). It is introduced by S. Evan et al [8] in order to builds protocols for secure multiparty computation. OT_1^2 protocol can be generalized to one-out-of-n oblivious transfer protocol (OT_1^N) such a way that user can only obtain one of there elements while database server are not able to infer which data element accessed by the user. Claude Crépeau [7] presented a proof those two ("one-out-of-two") notions are computationally equivalent. $1-n$ oblivious transfer protocols were propsed by Moni Naor and Benny Pinkas [22], William Aiello et al [1], Sven Laur and Helger Liomaa [18]. Later, Brassard, Crépeau and Robert have generalized this notion to k-n oblivious transfer protocol OT_k^N [2] which means that k-out-of-n data items in database which have been sent to the user are accessible by user. The protocol with k-out-of-n scheme can be defined either non-adaptive or consecutively. Moni Naor and Benny Pinkas [23] proposed a new method with $O(N)$ computation complexity in preprocessing stage and fixed computation (independent of k) complexity for each new value the receiver obtains. Furthermore, OT_k^N protocol is a specified version of Generalized oblivious transfer, which firstly was introduced by Ishai and Kushilevitz [14]. Their works was based on parallel invocations of OT_1^2 while making use of special model of private protocols. Afterwards, other protocols have been proposed based on secret sharing. There are some remarkable OT protocols which are working based on secret sharing were published by Bhavani Shankar et al [32] and later Tamir Tassa [37] proposed another scheme of the protocol.

3.5 Oblivious Random Access Memory (ORAM)

Oblivious Random Access Memory (Oblivious RAM or ORAM) is third protocol of cryptographic primitive series to hide users' access pattern or requested data from curious database server. The concept introduced by Oded Goldreich [12] and Rafail Ostrovsky [26], which enables a user, that can store constant amount of data locally, and store the rest remotely on the server, so that user can access the data item in the server while hiding the identities of the items which are being accessed. In fact, ORAM protocol is mostly devised to be used in a resource-restricted devices such as mobile phones. However it can be also used for other purposes such as searching data on encrypted data for preventing cache attacks or protection against any other kind of privacy violation. Goldreich [12] investigated the ORAM protocol to utilize it in software protection problem. The goal of his works was to hide the access pattern of a software to main memory that is because of preventing reverse engineering of the software. Later, Goldreich and Ostrovsky generated new model of ORAM for software protection [13] with the best result of their works. In particular, each ORAM scheme should hide the following information from the server to provide users' privacy against eavesdropper who might be the server itself: (1) the location of the accessed data item in the memory, (2) the order of data requests, and (3) the number of requests to the same location. Furthermore, any possible types of access such as get-value, set value, insert-new-item, delete-item, etc. must also be indistinguishable to the server. Most of the presented ORAM schemes have provided acceptable level of privacy for users, However, almost all of them are unfeasible to

implement them in the real-world applications. For instance, the cost of the best protocol of Goldreich-Ostrovsky [13] efficient but clearly unfeasible for any reasonable application. Because for storing n data item there should be $O(n \log n)$ memory available; furthermore, each access to a data item was replaced by $O(\log^3 n)$ data requests to the stored data. Nonetheless, due to the overwhelming overhead of the oblivious RAM protocol, it was often cited as a theoretical solution which could solve many privacy related problem, however it is apparently impractical to be used in real-world applications. Therefore, some new works are there to make the protocol much practical and feasible in order to make use of them in real-world applications. Benny Pinkas and Tzachy Reinman [28], re-investigated the Oblivious RAM protocol which is introduced by Goldreich and Ostrovsky [13]. In their scheme, they improved previous protocol to make it feasible in practice, also they described a new construction with a considerably improved overhead therefore it requires the client to store only $O(n)$ items, and replace each data request with $O(\log^2 n)$ access to the data stored in the database. Although Pinkas and Reinman improved ORAM protocol's efficiency to make them more practical, still the ORAM protocol was very costly and highly complicated to implement it in practice. Nonetheless, there are other subbranches of ORAM protocol have been introduced by other researchers. One of very appropriate protocols of ORAM protocol family is Path Oblivious RAM protocol. Due to the importance of this protocol, we will discuss it thoroughly in the next section.

3.6 Path Oblivious RAM

Path Oblivious RAM (Path ORAM) is introduced in 2012 by Emil Stefanov et. al. [35] and it improved by them later in 2013 [36]. The advantages of the protocol are that is highly simple to implement because, it can be described by a few lines of pseudo-code also this protocol is very close to the goal of ORAM protocol which was designed to be used in a resource-restricted devices such as smart cards or smart phones. Namely, it is very practical to be used in a devices with a small amount of storage capacity. They proved that the protocol requires $O(\log^2 N/\log x)$ bandwidth overhead for block size $B = x \log N$. Also Their protocol works better than best known ORAM scheme for the block sizes bigger than $\omega(\log^2 N)$. As we mentioned earlier, we need a cryptographic scheme in order to use it in outsourced storage application (in our works we are aiming to outsource identification and authentication process of customers in the bank). However, in the outsourced storage, due to the fact that data stored are publicly accessible, clients access can leak a significant amount of sensitive information about the data through statistical inference. Islam et. al. [15] proved that an adversary can infer as much as 80% of the search queries of the users' accesses to the encrypted email repository. Stefanov el. al. [36] works proposes a new approach to the ORAM scheme, which makes the following contributions:

3.6.1 Simplicity and practical efficiency

Although there is no formal way of measuring its simplicity, it is apparently obvious that their protocol is very simple with respect to any other previous proposed ORAM protocols. The reason is that, we can describe the core of the protocol in just 16 lines of pseudo-code (see Algorithm 1). Furthermore, there is no need to perform an extra and sophisticated operations such as oblivious sorting and oblivious cuckoo hash table construction like many existing ORAM. In fact, each access to the server database in Path ORAM scheme, can be expressed as simply fetching, shuffling, and storing a single path in a tree stored remotely on the server. In addition, simple nature of Path ORAM makes it more practical than any existing ORAM construction with small local storage which can be constant or poly-logarithmic depends on the applications being used.

3.6.2 Asymptotic efficiency

They also prove that, their proposed Path ORAM protocol can achieve asymptotic bandwidth cost of $O(\log^2 N \,/\, \log x)$ blocks and consumes $O(\log^2 N / \log x)\omega(1)$ blocks in client-side storage, by taking to account that the size of blocks are reasonably large ($B = x \log N$ bits where N in the total number of blocks) by using recursion, where recursion is proposed in Shi et. al. [34]. Also they claim that the above result achieves a failure probability of $N^{-\omega(1)}$ which is negligible with respect to the size of N. Emil Stefanov et. al. [35], compared Most of well-known ORAM schemes in their work. They compared Asymp-

totic bandwidth cost of their works with other well-known ORAM schemes in Table 3.1.

ORAM Scheme	Client Storage Read & Write Bandwidth (# blocks of size B)	Read & Write Bandwidth (# blocks of size B)
Kushilevitz et. al. [16] $B = \Omega(\log N)$	$O(1)$	$O(\log^2 N / \log\log N)$
Gentry et. al. [10] $B = \Omega(\log N)$	$O(\log^2 N).\omega(1)$	$O(\log^3 N / \log\log N).\omega(1)$
Chung et. al. [6] $B = \Omega(\log N)$	$O(\log^{2+\epsilon}(N))$	$O(\log^2 N \ . \ \log\log N).\omega(1)$
Recursive Path ORAM for small blocks. [35] $B = \Omega(\log N)$	$O(\log N).\omega(1)$	$O(\log^2 N)$
Recursive Path ORAM for moderately sized blocks [36], (block size of $B = \Theta(\log N)$)	$O(\log N).\omega(1)$	$O(\log N)$

Table 3.1: Comparison of Emil Stefanov et. al. [35] proposed method with other known ORAM schemes.

3.6.3 Practical and theoretic impact of Path ORAM

As we pointed out earlier, first version of Path ORAM scheme introduced by E. Stefanov et. al. in 2012, has made both a practical and a theoretic impact in the users' privacy and outsourcing topics. Due to the conceptual simplicity it is more suitable to be implemented in resource-restricted devices as well as hardware. An example simulation has been built by Ren et al. [31] for secure processor architecture based on path ORAM algorithm and the Ascend processor architecture introduced int 2012 by Fletcher [9] as a primitive. Furthermore, there are several theoretic works adoptd the same idea of eviction in their ORAM constructions. Gentry et al. [10] and Chung et al. [6] tried to improve ORAM bounds based on the binary tree construction by Shi et al. [34]

Path ORAM protocol is based upon the binary-tree ORAM framework propsed by Shi et al. [34]. the protocol is designed to prevent untrustworthy database server from accessing users' access pattern or queries and the data they requests which are very sensitive information. In this protocol, assumption is that, a client has restricted processing as well as storage resources. Therefore, our assumption is that the server is untrustworthy and the client is trusted, including the client's processor, memory, and disk. In the following, we will explain Path ORAM protocol functionality in detail.

3.6.4 Notation and Definition

We assume that the client fetches/stores data on the server in atomic units, referred to *blocks*, of size B bits each. Also, throughout the document, we will use N as the number of distinct data blocks that are stored in Path ORAM. Table 3.2 indicates all notations required in Path ORAM protocol.

Notation	Description
N	Total number of blocks outsourced to server
L	Height of binary tree
B	Block size (in bits)
Z	Capacity of each bucket (in blocks)
R	Capacity of each block (in records)
$P(x)$	Path from leaf node x to the root
$P(x,\ l)$	The bucket at level l along the path $P(x)$
S	Client's local stash memory
$Position$	Client's local position map
$x := Position[a]$	Block a is currently associated with leaf node x, i.e., block a resides somewhere along $P(x)$ or in the stash.

Table 3.2: Path ORAM Notations

3.6.5 Path ORAM Protocol Description

In Path ORAM scheme tree structure has been used to store data in server side. Although, for the sake of simplicity, we use binary tree in our descrip-

tion, it can be a tree with more than two children per node. However, our implementation is fully parametric and it is possible to set the number of children per node as well as other settings. For binary tree with the height L there are 2^L leaves in the tree. The levels of the tree are numbered 0 to L where level 0 denotes the root of the tree and level L denotes the leaves. It is possible to implement tree structure using either array or pointer (fixed size v.s. dynamic size).

Each node in the tree is called a *Bucket*. Each bucket has a capacity of Z real blocks. However, if there are less than Z blocks to store inside block, it is padded with dummy blocks to always be of size Z. There is no such a rule for size of bucket z or number of children per node, although it is better to choose small size for bucket (e.g. $Z = 4$).

We have developed two separate program: 1) in-RAM and 2) mapped to database. In the first implementation (in-RAM), each *block* contains R number of *records* and each record contains users' information fields such as user ID, name, family, public key, etc. However, atomic access to the database in both implementations is block which is similar to the proposed protocol by Stefanov et. al. [36].

According to the authors of the Path ORAM protocol [36], $x \in \{0,\ 1,\ ...,\ 2^L - 1\}$ denotes the x-th leaf node in the tree. $P(x)$ is a function which is being used to denotes the all buckets along the path from leaf x to the root node of the tree. In addition, $P(x,\ l)$ is a function which denotes the bucket in $P(x)$ at level l in the tree. In other word, using function $P(x)$, we can obtain all buckets starting from leaf node x along the root, while using $P(x,\ l)$ we refer

to a certain node (bucket) of tree which is resides in level l and on the path of leaf node x to the root. Furthermore, the size of the database (tree) depends on the parameters: 1) tree height, number of children per node, number of blocks per bucket, and number of records per block. Since there are about N buckets (nodes) in the tree and each bucket contains Z blocks, also each block contains R records, the total server storage is about $Z.N$ blocks or $R.Z.N$ records.

3.6.6 Client Storage and Bandwidth

Client side in Path ORAM protocol consists of two separate data structure, a stash memory and a position map. Followings sections describe these structures functionality in detail.

3.6.6.1 Stash Memory

Stash is a local memory of client to keep retrieved blocks from the server temporarily. Furthermore, it can be used to store some blocks if there will be no free space in the server (overflow in the buckets of tree). Thus, stash S is used as a temporary memory space to keep blocks. They proved that stash has a worst-case size of $O(\log N).\omega(1)$ blocks with high probability. Also they showed that the stash is usually empty after each Path ORAM read/write operation complete.

3.6.6.2 Position Map

The client uses position map to get leaf node number of the tree in such a way that it can access to the block a which is stored in a bucket (node) along the way from leaf node x to the root and we indicate it as $x := Position[a]$ also if is not in the tree i is stored in the stash memory temporarily. Therefore, it is recommended to check stash before searching data on the tree or somehow indicate it in the position map so that user can know whether block is in stash or tree. In fact, position map is an array which maps block number to the leaf node number of tree.

3.6.7 Path ORAM Initialization and Data I/O

Initially, database tree has been generated in server side with encrypted random dummy blocks. Respectively, In client side, Stash is empty and position map is set with independent random numbers between 0 and $2^L - 1$. Basically, all data transaction operation such as read, write, update, and delete is done via single protocol called "Access" which is described in Algorithm 1. In fact there are to separate operations: Read and Write. to read block a, client calls $data \leftarrow Acess(read, a, None)$ and to write $data^*$ to block a the client performs $Access(write, a, Data^*)$. Emil Stefanov et. al. [36] summarized Path ORAM scheme in 4 main steps as follow.

1. **Remap Block** (Lines 1 to 2): Randomly remap the position of block a to a new random position. Let x denote the block's old position.

2. **Read Whole Path** (Lines 3 to 5): Read the path $P(x)$ containing block a.
3. **Update Block** (Lines 6 to 9): If access is write (new record, edit record, and delete record), update the data stored for block a.
4. **Write Path** (Lines 10 to 15): Write the path back and possibly include some additional blocks from the stash if there is any vacant place in the database tree.

Algorithm 1: Path Oblivious RAM Algorithm

Function ***Access*** $(op, a, data^*)$

- $x \leftarrow$ position $[a]$
- position $[a] \leftarrow$ `uniformRandom`$(0...2^L - 1)$
- **for** i $\leftarrow 0, 1, ..., L$ **do**
  - S $\leftarrow$ S $\cup$ `ReadBucket(P(`x, l`))`
- data $\leftarrow$ $ReadBlock$ a $from$ S
- **if** op $=$ $write$ **then**
  - S $\leftarrow$ (S $-\{(a, data)\}) \cup \{(a, data^*)\}$
- **for** $l \leftarrow L$ downto 0 **do**
  - $S' \leftarrow \{(a', data') \in S$: `P(`x, l`)` $=$ `P(`position $[a'], l$`)`$\}$
  - $S' \leftarrow$ Select `Min(`$|S'|, Z$`)` Blocks $from$ S'
  - $S \leftarrow S - S'$
  - `WriteBucket(P(`x, l`),` S'`)`
- **return** data

Chapter 4

Proposed Study: Path Oblivious-RAM Based Public Key Repository

We use Path ORAM protocol to achieve the goal of hiding users' requests against untrustworthy outsourced database server. However, due to the fact that information, which is being stored at server side is in relational database structure, clients should send their requests in a form of SQL command to the server. Therefore, we should either change request commands at client side to Path ORAM protocol data request format, or somehow convert users' requests to path ORAM protocol request. In the first approach, performing significant modifications at client side software and data structure are inevitable. Nevertheless, such modifications are costly and require consider-

able configuration changes. As a result, it is essential to find another solution in order to tackle this issue.

In this study, we developed a model in which the client queries are obtained in regular SQL format and then parsed so that it can be processed via Path ORAM protocol. In our model, a position map is kept for each search key. After SQL parsing, we decide on which buckets are to be retrieved starting which leave nodes. Moreover, in our model we developed an adaptation layer between Path ORAM's tree structure and a relational database so that the buckets are stored and retrieved from relational database. Now, we are going to explain our model in more detail.

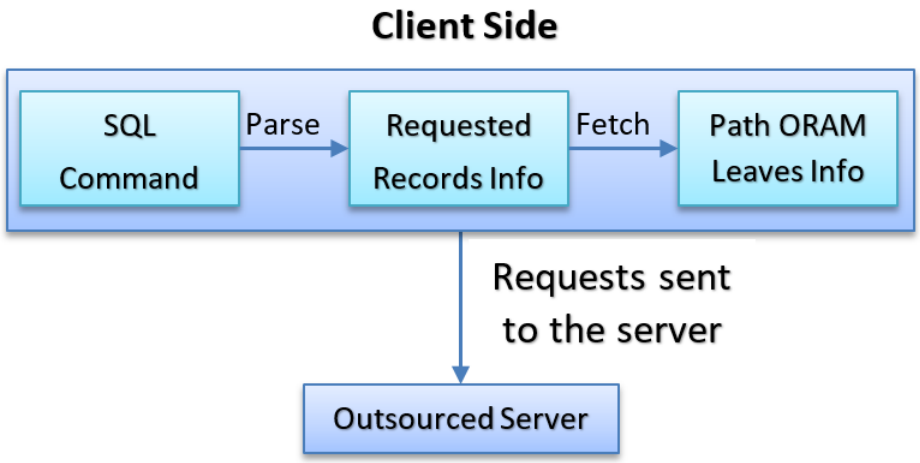

Figure 4.1: First Step of our proposed method (Client Request).

Client side operations are shown in Figure 4.1. Client first passes SQL command to the parser. SQL command parser is a function that takes a SQL command as an input and after analyzing returns a list of records that client requested. Then this information is sent to the position map to get

leaf nodes indices of all requested records. Finally, these leaf nodes indices are sent to the server with Path ORAM data structure to fetch records form data server.

A requirement of aforementioned approach is that, each data transformation between client and server changes the leaf node index of requested record. Therefore, it is essential to use a flag for fetched records then it is possible to infer whether the record has already been fetched or not.

Next phase of data fetch operation is being run at the server side. After receiving requested data from a client, server should operate the request. However, storing data structure of Path ORAM at the server side has its own challenges. Owing to the fact that Path ORAM uses tree data structure to store data records, we need to generate tree structure in the server side. Although, generation and management of the tree structure using an object oriented languages is fast and not considered a hard task, enterprises do not prefer to change their institutional database infrastructures to have extra security and privacy services. In this respect, Path ORAM's nonstandard tree data structure is a disadvantage in its entirety. In our system, we address this problem by adapting Path ORAM's tree data structure into a relational database.

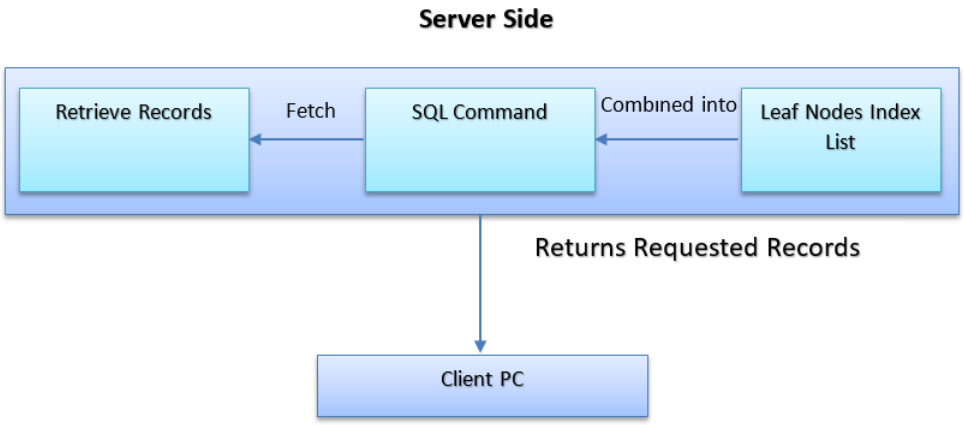

Figure 4.2: Second Step of our proposed method (Server Response).

Figure 4.2 gives an overview of our proposed method of storing and retrieving data by the use of relational database management system. As represented in Figure 4.2, as soon as server receives the requests from client side (leaf nodes indexes), it maps the requested data into a corresponding SQL command. Then, this generated SQL command is sent to the relational database system in order to fetch requested records. Note that, generated SQL command at the server side is not the same with the one provided by the client in the first step of the operation. SQL command of the first step is what the user aimed to retrieve from the data server; however, the second one is the SQL command which is generated by the mapper function in order to fetch those information from Path ORAM tree, which is mapped into a relational database management system. Finally, all requested data (a path starting from a leaf node through root node) are sent back to the client as a result of its request.

4.1 Parsing SQL Command into a Path ORAM Data Request

In this section, we discuss client side parser process in detail. As we discussed in the previous section of this chapter, clients uses relational database data request command (SQL Command) in order to send it to the server. This SQL command, in turn, needs to be transformed into a Path ORAM data request. For this purpose, we developed a flexible parser so it can parse any complex queries.

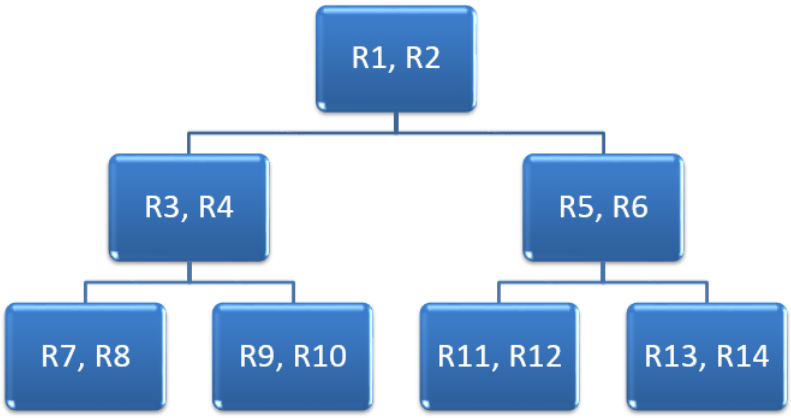

Figure 4.3: Sample tree data structure with 7 records.

Algorithm 2: SQL query to Path ORAM data request parser.

```
H   Function Parser (SqlExp)
    pSqlExp ← parenthesize(SqlExp)
    Stack ← toPostfix(pSqlExp)
    LeaveNodesList ← Empty
    Condition ← pop(Stack)
    LeaveNodesList ← readPosMapList(Condition)
    while Stack ≠ Empty do
        Condition ← pop(Stack)
        lst ← readPosMapList(Condition)
        Operator ← pop(Stack)
        if Operator = AND then
            LeaveNodesList ← LeaveNodesList ∩ lst
        if Operator = OR then
            LeaveNodesList ← LeaveNodesList ∪ lst
    Return LeaveNodesList
```

Table 4.1: Records information stored in the tree.

ID	Name	Family	Leaf index	ID	Name	Family	Leaf index
R1	N1	F1	3	R8	N2	F3	1
R2	N2	F3	1	R9	N3	F3	2
R3	N3	F1	1	R10	N1	F1	2
R4	N1	F2	2	R11	N2	F3	3
R5	N2	F2	4	R12	N3	F1	3
R6	N3	F2	3	R13	N1	F2	4
R7	N1	F3	1	R14	N2	F3	4

Algorithm 2 describes steps of operations to parse SQL command into a leave nodes lists. In the first step of the algorithm, SQL expression is parenthesized according to prioritized order of evaluation and then is converted into postfix notation to be stored in a stack. Note that, priority of the operations are controlled by postfix expression evaluation using stack. After that, we made a loop over stack data and in each iteration of the loop, condition and operator are popped from the stack. For each condition, we retrieve a list of corresponding leave nodes (*lst*) by looking up from the position tables. If the operator is AND, current leave nodes list is the intersection of the current one and *lst*. If the operator is OR, then the current leave nodes list is the union of current one and *lst*. These steps apply to all elements inside stack and after loop finishes, the list called *LeaveNodesList* contains the leave nodes indices which meets all conditions in the SQL query.

Following is a sample run of a parser method in client side applied on the tree given in Figure 4.3. The database field values and the leaf node indices are given in the table 4.1.

We exemplify using two queries given below.

1. SELECT * FROM tbl WHERE ID = $R6$
2. SELECT * FROM tbl WHERE (Name = $N2$) AND (Family = $F3$)

Query 1 is a single query that returns a simple record. For such single queries, we just need to find it in position map and get leaf node index of that record. Query 1 resulted in a single record with $ID = R6$ and the leaf node index to access this record is 3. On the other hand, query 2 returns more than one record. According to our parser, we should fetch leave nodes indices for the first condition $Name = N2$ which are $1, 3, 4$. In the second step of the algorithm, we obtain the leaf node indices of the second condition of the query which are $1, 2, 3, 4$. Finally, since AND operator is used in this complex query, we should obtain a list of leave nodes indices by applying intersection on those two lists, so the resulted list becomes $1, 3, 4$.

4.2 Mapping Path ORAM data structure into relational database

In this section, we explain how Path ORAM tree data structure is mapped into a relational database system. In order to map the data structure of Path ORAM (tree) into a relational database system, we use some relational database tables and SQL queries.

4.2.1 User Data Table

This table contains records of users. Maximum number of records that we can store in this table is the capacity of records in the corresponding Path ORAM tree. In order to associate the user records to Path ORAM's structure, we use *BlockID* as the primary key of the table. This field is also the foreign key of the Path ORAM mapping table, which will be explained in the next subsection.

Other fields of this table are the actual user database's fields, which vary application-wise.

4.2.2 Path ORAM Mapping Table

In order to map a tree data structure which is used by Path ORAM protocol into a relational database system, we need to devise a mechanism using tables

and relations between tables. We generated two tables, one for storing user information, which is explained in the previous subsection, and the other one to keep tree structure information such as Block ID, bucket ID (tree node ID) and leaf node index. Theses two tables has a one to one relation using *BlockID* as a primary key in the user information table and foreign key in mapping table. After generating inner-joined table, fetching all records which belong to a specific leaf node is reduced to a simple SQL query. The following SQL commands are the inner-join and a sample information fetch commands, respectively.

- Inner-join SQL command:

SELECT tblMapping.*, tblPathOram.* **FROM** tblMapping
INNER JOIN tblPathOram
ON tblMapping.BlockID = tblPathOram.BlockID

- Sample SQL command to fetch a path with leaf node index i:

SELECT * **FROM** tblInnerJoined **WHERE** LeafNodeIndex = i

Chapter 5

Performance Evaluation

In this chapter, we evaluate efficiency of our work in order to use it as a public database but preserving users' privacy. Users send their SQL-Like queries to the remote database server and server returns corresponding results for each query of each user. We used Path ORAM protocol in order to provide users' privacy in the database, but database design and implementation details are hidden from users. Users just send out normal SQL-like queries. The queries first parsed into leaf node indices of Path ORAM and they are sent to the database server. Then, database server fetches them from the mapped relational database.

Our server specifications and the details of development environment are as follows.

- **CPU:** Intel ®, Core™, i7-2600 @ 3.40 GHz, 64-Bit
- **RAM:** 8 GB DDR3
- **Ethernet:** 100 Mbps
- **H.D.D.:** 700 GB
- **OS:** MS Windows 10 Professional, 64-Bit
- **Programming Language:** MS Visual C#.NET 2015
- **Database System:** MS SQL Server 2014

5.1 Communication Models

In our experiments, we used three different communication models between clients and server.

- **Database in the RAM:** In this model, entire database has been generated and stored locally in the RAM. There is no network since client and server are running in the same machine. The goal of this setup is to evaluate performance of our system for the cases where all overheads are reduced. In other words, this model gives the pure computational cost of our proposed system. Moreover, in this model database is stored using native Path ORAM tree structure.
- **Crossover Link:** In this model, client and server machines are separated and connected directly via an ethernet cable. In other words, database is stored in the server and the client is connected to send

queries over this crossover cable. In this model, network delays are almost zero. This model is useful to apply our proposed adaptation layers to Path ORAM, but the communication overhead is minimized to see the real computational performance. In this model, SQL parsing and relational database adaptation layers are fully integrated.

- **LAN Connection:** In this communication model, the server and client machines are separated, as in crossover cable model. Moreover, all adaptation layers are integrated as well. The only difference between this setup and the crossover connection setup is that, clients and server are connected over LAN instead of direct connection in order to take into account possible network traffic delays.

5.2 Databases Used in Tests

The databases that we use in our performance tests are synthetically generated. We use a sample database structure with six fields. Table 5.1 shows the fields and data type of each fields of the database.

Table 5.1: Fields of database table used in our experiments.

Field Name	Data Type	Primary Key	Is Unique
ID	String	Yes	Yes
Public Key	BigInteger	No	Yes
Name	String	No	No
Family	String	No	No
Email	String	No	Yes
City	String	No	No

Because of memory restriction in the RAM, we are enforced to use relatively less amount of records for "Database in RAM" communication model as compared to other communication models, in which database is stored on disk as a relational database.

We have generated five different databases using the same table format, but with different number of records by applying different settings to *Path ORAM* tree. In all generated databases for the network based models, there are two children per node and 25 blocks per bucket. However, the height of tree varies to generate various number of records. Table 5.2 indicates various tree heights and respective numbers of generated records for network based communication models. Here, the database is mapped into MS-SQL Server 2014 database system.

Table 5.2: Database sizes used (for network based models).

Tree Height	Total # of Records Generated
11	102,375
12	204,775
13	409,575
14	819,175
15	1,638,375

We also generate databases to be used in RAM. This one is a pure Path ORAM tree data structure and not mapped to relational database model. However, because of the memory restrictions of RAM, the number of generated records is smaller than the other models. All *in-RAM* databases have two children per node and 20 blocks per bucket, but with different tree heights in order to generate various numbers of records. Table 5.3 shows different settings of generated *in-RAM* databases with respective number of generated records for each tree height.

Table 5.3: Database sizes used (for in-RAM database).

Tree Height	Total # of Records Generated
9	20,460
10	40,940
11	81,900
12	163,820
13	327,660

5.3 Query types tested

For each setup, we executed four different queries as follows:

- **Query 1:** *SELECT * FROM db.tbl WHERE ID = '12'.* This query has a result of single record, because the ID is unique and is the primary key of the table. This query is used to evaluate single record fetch time. Moreover, this is the most common query type in the transaction signing scenario, which is the main motivation of the project.
- **Query 2:** *SELECT * FROM db.tbl WHERE Email = 'EAAAAB@US.COM'.* This query's result is also one record, because email address is also unique for each user among table records.
- **Query 3:** *SELECT * FROM db.tbl WHERE (Name = 'NAAAAB' AND Family = 'FAAAAC').* This query results in more than one record.

The number of fetched records depends on the repetitions of field values in the database. This type of complex query is not a typical one for transaction signing application; it can only be used for system administrations for some advanced offline tests and applications.

- **Query 4:** *SELECT * FROM db.tbl WHERE (Name = 'NAAAAB' AND City = 'CAAAAB') OR (Family = 'FAAAAB' AND City = 'CAAAAC')*. This query also returns more than one record and it is more complicated than *Query 3* in term of conditions because there are combinations of *AND* & *OR* operators. As *Query 3*, this one is also not typical one, but can be needed for some advanced offline tests.

5.4 Record Fetch Timing Analyses

Figure 5.1 shows Query 1 fetch timings for both crossover and LAN connection models. As shown in this figure, for a database with $100,000$ records, end to end latency of sending the query and getting the response takes less than one second for crossover connection; 1.6 seconds in LAN connection model. As the number of records increases, latency also increases linearly. It takes 3.1 and 4.0 seconds for crossover and LAN connection, respectively, in a database with $1,600,000$ records.

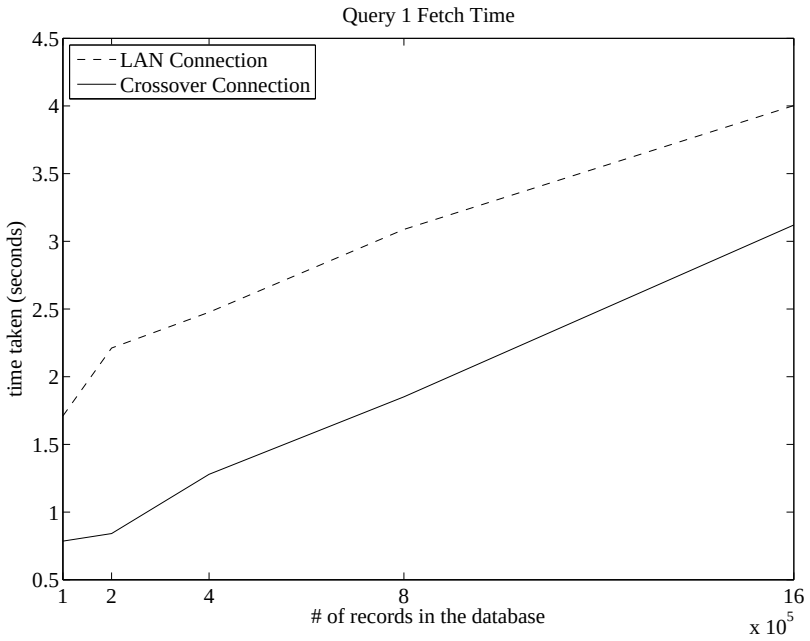

Figure 5.1: Record fetch timings of Query 1 (crossover and LAN connection modes)

Figure 5.2 depicts Query 2 fetch timings in both crossover and LAN connection models. As seen in this figure, record fetch timings are almost similar with the ones of Query 1. The reason for this is that both queries result in single record. Owing to the fact that, we need to fetch public key of the users which is also a unique among all stored records in transaction signing application, this kind of single record fetch queries is of utmost importance regarding the timings.

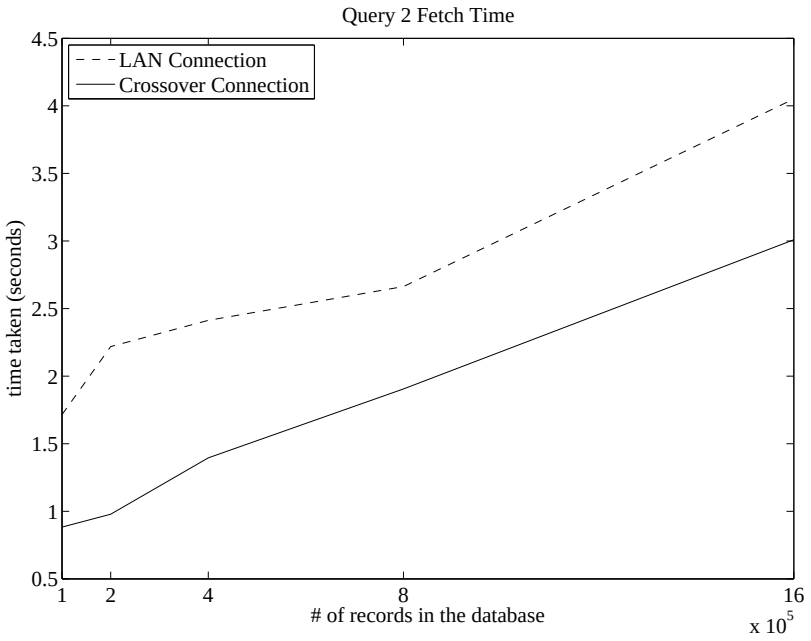

Figure 5.2: Record fetch timings of Query 2 (crossover and LAN connection modes)

Table 5.4: Number of fetched records in the on-disk databases.

	# of Records in Database				
Query No.	102,375	204,775	409,575	819,175	1,638,375
3	2	4	8	20	31
4	6	9	27	51	81

Figure 5.3 shows Query 3 fetch timings for both crossover and LAN connection models. This query is more complex than Queries 1 and 2 and returns more than one record. Consequently, it takes greater time than the

previous two queries. As the number of records increases in the database, the number of fetched records also increases (the numbers of fetched records for Queries 3 and 4 are shown in Table 5.4). This, in turn, causes several database accesses for various paths. For instance, record fetch time for the database with $100,000$ records takes less than 5 seconds in crossover connection, while it takes 47 seconds in crossover connection and 100 seconds in LAN connection for a database with $1,600,000$ records.

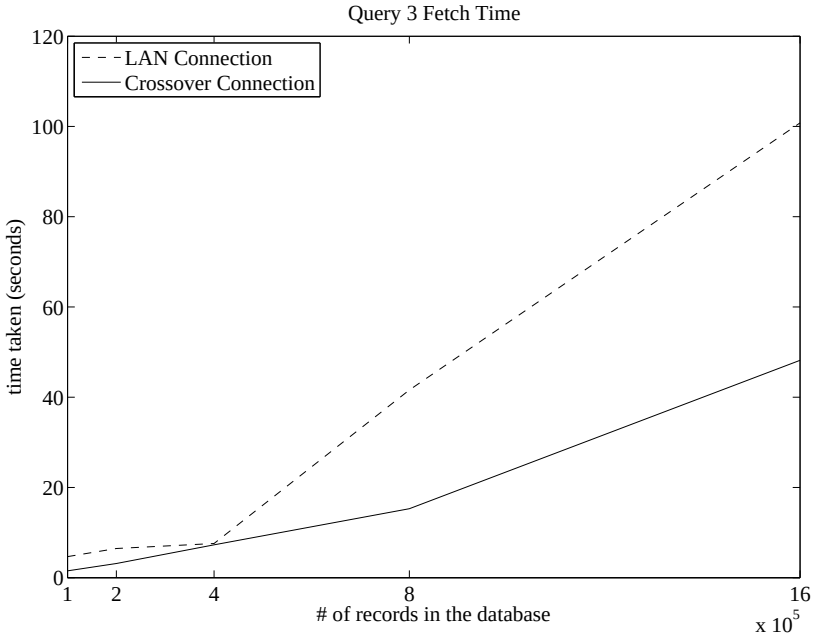

Figure 5.3: Record fetch timings of Query 3 (crossover and LAN connection modes)

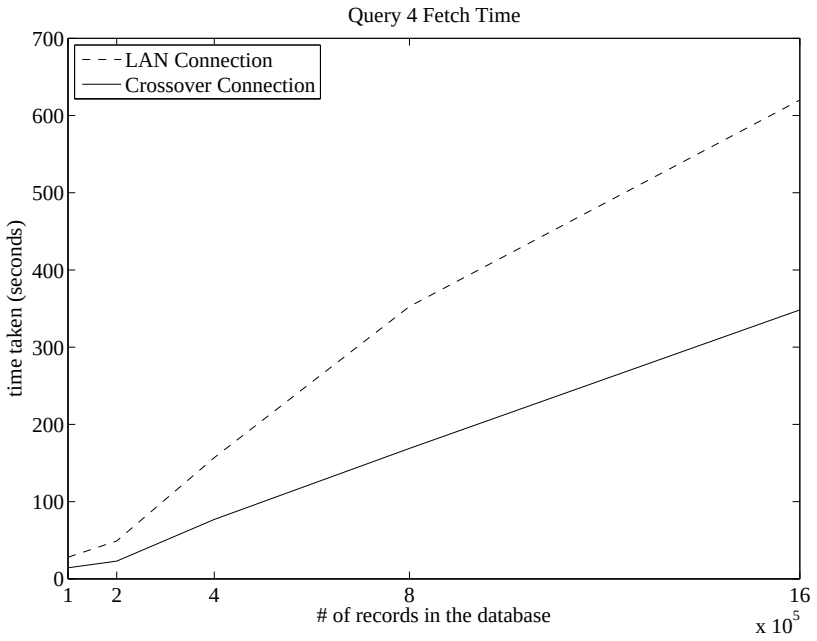

Figure 5.4: Record fetch timings of Query 4 (crossover and LAN connection modes)

Query 4 timings are shown in Figure 5.4. This query is the most complex one among all other queries; consequently, it returns greatest number of records as compared to other tested queries. As result, this query takes higher fetch time than the other queries. Although these timings are high, since this type of complex queries are not directly used in transaction signing applications, the total duration of transaction signing is not affected directly. This type of complex queries would be used for some offline analyses only.

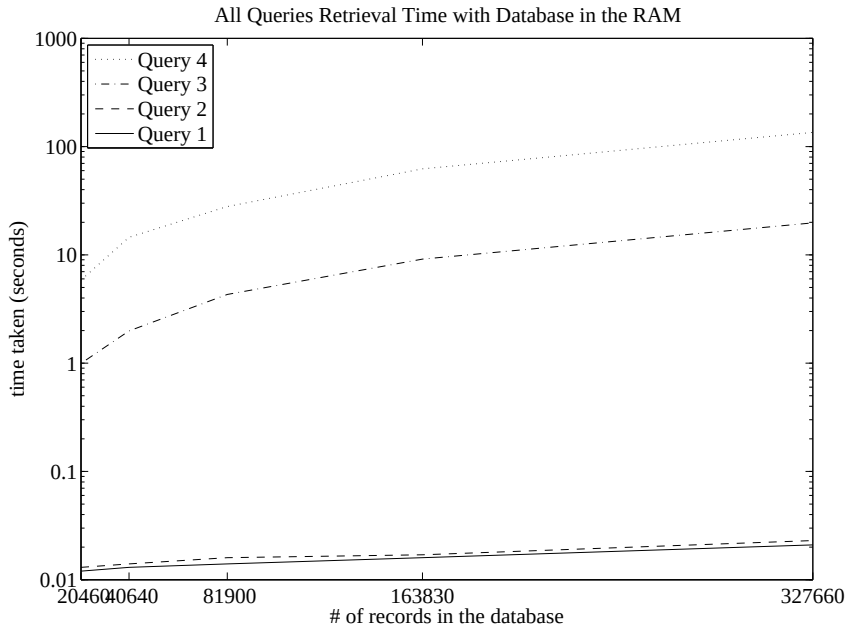

Figure 5.5: Record fetch timings for in-RAM database model - all queries

Figure 5.5 shows all queries' timing result for the model in which the database is stored in RAM as a tree. Because of the storage capacity restriction of main memory, we generated databases up to almost $327,000$ records. The goal of this test is to obtain computational cost for the case where the costs of communications and relational database/SQL adaptation are reduced. Queries that return single record take between 10 and 20 milliseconds, while more complex queries take up to 100 seconds. As discussed earlier, complex queries are not commonly used ones in real world applications. Since the flow of transaction signing requires single record queries and the computational cost of these queries, with the costs of communication and adaptation reduced, is below 100 milliseconds even for a database with one

million records (due to linear increase in Figure 5.5), we can conclude that the additional cost of Path ORAM-based privacy enhancing techniques is at marginal level.

Figure 5.6 and 5.7 give the timings of 4 queries altogether for crossover and LAN connection models.

We also evaluate response time of the four separate queries with almost fixed number of records (around $440,000$), but with different Path ORAM tree database settings such as tree height, number of children per node, and number of blocks (*Record*) per bucket (*tree node*). The results are given in Table 5.5. As shown in the this table, the time needed to fetch a single record from a database with almost $440,000$ records is approximately one second (Queries 1 and 2). However, it is also clear that the height of tree has an effect on the data fetch time; the greater the tree height, the grater the fetch time for both Queries 1 and 2.

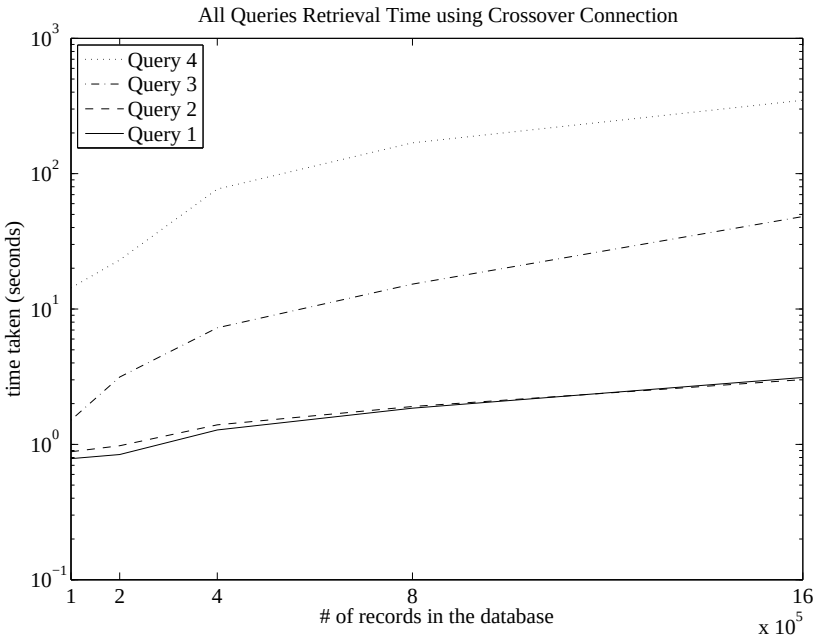

Figure 5.6: Record fetch timings for crossover connection model - all queries

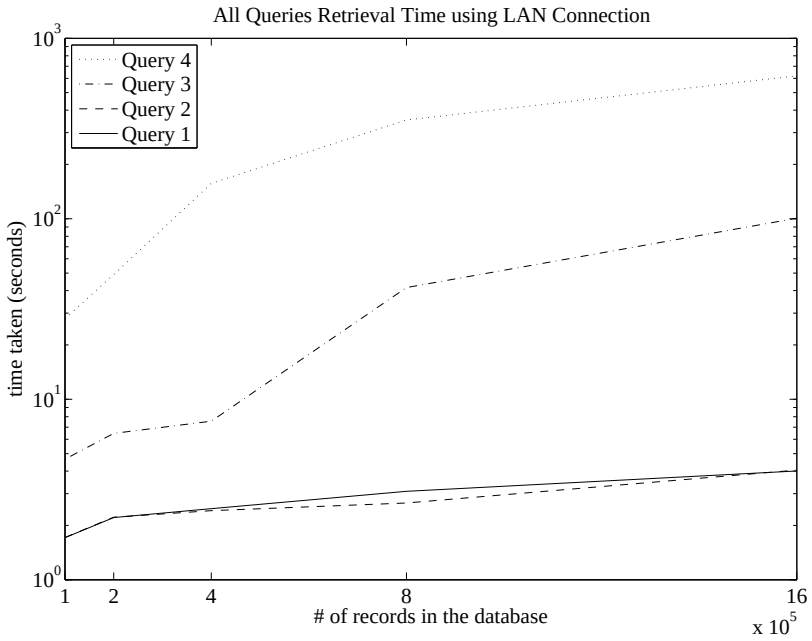

Figure 5.7: Record fetch timings for LAN connection model - all queries

Table 5.5: Effect of tree parameters

Tree Height	# of Child	Blocks / Bucket	# of Records	Fetch time (for all records returned via crossover connection) (Sec.)	Query No.	# of records returned
13	2	25	409600	1.279	1	1
9	3	14	413336	0.889	1	1
8	4	5	436905	0.81	1	1
7	5	5	488282	0.724	1	1
13	2	25	409600	1.395	2	1
9	3	14	413336	0.922	2	1
8	4	5	436905	0.856	2	1
7	5	5	488282	0.881	2	1
13	2	25	409600	7.268	3	8
9	3	14	413336	5.502	3	9
8	4	5	436905	3.067	3	7
7	5	5	488282	3.564	3	8
13	2	25	409600	68.154	4	27
9	3	14	413336	37.796	4	24
8	4	5	436905	18.340	4	21
7	5	5	488282	27.391	4	31

As seen in Table 5.5, the fetch times of Queries 3 and 4 are relatively higher than other queries since these queries return several records and each record fetch needs a separate access to the database in Path ORAM. Moreover, the determining effect at the tree height is also seen for Queries 3 and 4.

Chapter 6

Conclusions and Future Work

In this thesis, we designed and developed a proof of concept solution for the problem of customer database outsourcing for a bank to be used in transaction verification problem. In this model, the bank needs to query the customers' public keys in a secure and privacy preserving manner. Our solution is purely based on an existing privacy preserving data access model, namely Path ORAM [35, 36]; however, we developed end user/system adaptation layers in order to hide the non-standard features of Path ORAM from the client (the bank in our use case) and the server (the cloud provider). Such an abstraction is, actually, a must for a practical system since the enterprises do not prefer to change their workflows and infrastructures for extra security and privacy.

In our model, we get regular SQL queries from the client side and all user data is kept in a relational database model at the server side. The

SQL queries are first parsed according to Path ORAM requirements and the records are fetched from the relational database via its natural language. Thus, the end systems do not see any details of Path ORAM and its non-standard features.

We also performed extensive performance analyses in order to understand the feasibility of such an approach. We have seen that the computational overhead of end-to-end query processing, when used for transaction verification to return single record, is around 2 seconds per million records in the database. One should remark that this timing is with a regular computer used as a server; with high capacity server systems, this latency could easily be reduced.

Thus, we conclude that the idea of using Path ORAM to keep user data in outsourced databases in a privacy preserving manner is viable for financial transaction verification applications.

As future work, the adaptation layers that we developed for SQL and relational databases can be redeveloped for NoSQL databases.

References

[1] Aiello, W., Ishai, Y., and Reingold, O. Priced oblivious transfer: How to sell digital goods. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology* (London, UK, UK, 2001), EUROCRYPT '01, Springer-Verlag, pp. 119–135.

[2] Brassard, G., Crépeau, C., and Robert, J.-M. All-or-nothing disclosure of secrets. In *Proceedings on Advances in cryptology—CRYPTO '86* (London, UK, 1987), Springer, pp. 234–238.

[3] Cachin, C., Micali, S., and Stadler, M. *Computationally Private Information Retrieval with Polylogarithmic Communication.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 402–414.

[4] Cachin, C., Micali, S., and Stadler, M. Computationally private information retrieval with polylogarithmic communication. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques* (Berlin, Heidelberg, 1999), EUROCRYPT'99, Springer-Verlag, pp. 402–414.

[5] Chor, B., Kushilevitz, E., Goldreich, O., and Sudan, M. Private information retrieval. *J. ACM 45*, 6 (Nov. 1998), 965–981.

[6] Chung, K., Liu, Z., and Pass, R. Statistically-secure ORAM with $\tilde{o}(\log^2 n)$ overhead. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II* (2014), pp. 62–81.

[7] Crépeau, C. *Equivalence Between Two Flavours of Oblivious Transfers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 350–354.

[8] Even, S., Goldreich, O., and Lempel, A. A randomized protocol for signing contracts. *Commun. ACM 28*, 6 (June 1985), 637–647.

[9] Fletcher, C. W., Dijk, M. v., and Devadas, S. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing* (New York, NY, USA, 2012), STC '12, ACM, pp. 3–8.

[10] Gentry, C., Goldman, K. A., Halevi, S., Julta, C., Raykova, M., and Wichs, D. *Optimizing ORAM and Using It Efficiently for Secure Computation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 1–18.

[11] Gentry, C., and Ramzan, Z. *Single-Database Private Information Retrieval with Constant Communication Rate*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 803–815.

[12] Goldreich, O. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1987), STOC '87, ACM, pp. 182–194.

[13] Goldreich, O., and Ostrovsky, R. Software protection and simulation on oblivious rams. *J. ACM 43*, 3 (May 1996), 431–473.

[14] Ishai, Y., and Kushilevitz, E. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings* (1997), pp. 174–184.

[15] Islam, M. S., Kuzu, M., and Kantarcioglu, M. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *in Network and Distributed System Security Symposium (NDSS)* (2012).

[16] Kushilevitz, E., Lu, S., and Ostrovsky, R. On the (in)security of hash-based oblivious ram and a new balancing scheme. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2012), SODA '12, Society for Industrial and Applied Mathematics, pp. 143–156.

[17] Kushilevitz, E., and Ostrovsky, R. Replication is not needed: single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on* (Oct 1997), pp. 364–373.

[18] Laur, S., and Lipmaa, H. *A New Protocol for Conditional Disclosure of Secrets and Its Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 207–225.

[19] Lipmaa, H. *An Oblivious Transfer Protocol with Log-Squared Communication*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 314–328.

[20] MELCHOR, C. A., and GABORIT, P. A lattice-based computationally-efficient private information retrieval protocol, 2007. Short version presented in WEWORC, in July 2007, Bochum, Germany carlos.aguilar@unilim.fr 13844 received 27 Nov 2007, last revised 27 Nov 2007.

[21] Microsoft Corp. MS SQL Server, 2016. `https://www.microsoft.com/en-us/cloud-platform/sql-server` - Last retrived on 2016-09-14.

[22] Naor, M., and Pinkas, B. Oblivious transfer and polynomial evaluation. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1999), STOC '99, ACM, pp. 245–254.

[23] Naor, M., and Pinkas, B. *Oblivious Transfer with Adaptive Queries*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 573–590.

[24] Open Source. SQLite, 2016. `https://www.sqlite.org/about.html` - Last retrieved on 2016-09-21.

[25] Oracle Corp. mySql, 2016. `https://www.mysql.com/` - Last retrieved on 2016-09-23.

[26] Ostrovsky, R. Efficient computation on oblivious rams. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1990), STOC '90, ACM, pp. 514–523.

[27] Ostrovsky, R., and Skeith, W. E. *A Survey of Single-Database Private Information Retrieval: Techniques and Applications.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 393–411.

[28] Pinkas, B., and Reinman, T. *Oblivious RAM Revisited.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 502–519.

[29] Rabin, M. O. How to exchange secrets with oblivious transfer, 2005. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005.

[30] Reardon, J., Pound, J., and Goldberg, I. Relational-complete private information retrieval. Tech. rep., 2007.

[31] Ren, L., Yu, X., Fletcher, C. W., van Dijk, M., and Devadas, S. Design space exploration and optimization of path oblivious ram in secure processors. *SIGARCH Comput. Archit. News 41*, 3 (June 2013), 571–582.

[32] Shankar, B., Srinathan, K., and Rangan, C. P. *Alternative Protocols for Generalized Oblivious Transfer.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 304–309.

[33] Shannon, C. E. Communication theory of secrecy systems. *The Bell System Technical Journal 28*, 4 (Oct 1949), 656–715.

[34] Shi, E., Chan, T. H. H., Stefanov, E., and Li, M. *Oblivious RAM with O((logN)3) Worst-Case Cost*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 197–214.

[35] Stefanov, E., and Shi, E. Path O-RAM: an extremely simple oblivious RAM protocol. *CoRR abs/1202.5150* (2012).

[36] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., and Devadas, S. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 299–310.

[37] Tassa, T. Generalized oblivious transfer by secret sharing. *Designs, Codes and Cryptography 58*, 1 (2011), 11–21.