CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE COMPUTACIÓN

# Security Services on Outsourced Databases

A dissertation submitted by

**Lil María Xibai Rodríguez Henríquez**

For the degree of

**Doctor of Computer Science**

Advisor
**Debrup Chakraborty, Ph.D.**

México, D. F. February, 2015

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE COMPUTACIÓN

# Servicios de Seguridad en Bases de Datos Subcontratadas

Tesis que presenta

**Lil María Xibai Rodríguez Henríquez**

para obtener el grado de

**Doctora en Ciencias en Computación**

Director de Tesis:
**Dr. Debrup Chakraborty**

México, D. F. Febrero, 2015

# Abstract

Cloud computing holds the promise of revolutionizing the manner in which enterprises manage, distribute, and share information. The data owner (client) can out-source almost all his/her information processing tasks to a "cloud". The cloud can be seen as a collection of servers, which caters the data storage, processing and maintenance needs of the client. Needless to say this new concept of computing has already brought significant savings in terms of costs for the data owner.

Among others, an important task provided by a cloud is Database as a Service (DAS). In this service the client delegates the duty of storage and maintenance of his data to a third party (an un-trusted server). The DAS model allows the client to create, modify and retrieve data from databases in a remote location. These operations are performed by the server on behalf of the client. However, delegating this duty to a third party brings in some new security challenges.

The two main security goals of cryptography are privacy and authentication. These security issues are relevant to the outsourced databases also. The client who keeps the data with an untrusted entity has two main concerns. One being that the data may be sensitive and the client may not want to reveal it to the server and the second one is the data whose storage and maintenance has been delegated would be used by the client. The typical usage of the data would be that the client should be able to query the database and the answers to the client's queries would be provided by the server. It is natural for the client to be concerned about a malicious server who does not provide correct answers to his queries. In this work we aim to devise two schemes: one in which the client can encrypt the database and still performs queries that are answered by the server and another one where the client would be able to verify whether the server is responding correctly to his queries. The first problem is called privacy in databases and the second one is known as the authenticated query processing. In this thesis we address both of them. We view the above two problems in a formal manner and provide its security definitions. Further we device some efficient schemes which solve them. Finally, we analyze the security of the proposed schemes and also provide some experimental performance data.

# Resumen

El cómputo nube promete revolucionar la forma en que las empresas manejan, distribuyen y comparten su información. El dueño de los datos (cliente) puede delegar casi todo el procesamiento de la información a la "nube". La nube puede ser vista como una colección de servidores, los cuales proveen el almacenamiento, procesamiento y mantenimiento de los datos del cliente. No hace falta decir que este nuevo concepto de cómputo ha ocasionado un ahorro considerable en términos de costos para el dueño de la información.

Entre otros, un servicio importante que provee la nube es el de bases de datos (DAS por sus siglas en inglés). En este servicio el cliente delega la tarea de almacenamiento y mantenimiento de su información a un tercero (servidor no confiable). El modelo DAS permite que el cliente cree, modifique y recuperare información de la base de datos que se encuentra en una ubicación remota. Estas operaciones son realizadas por el servidor en nombre del cliente. Sin embargo, delegar esta tarea a un tercero crea nuevos retos de seguridad.

Los dos principales objetivos de la criptografía con respecto a la seguridad son privacidad y autenticación. Estos problemas de seguridad también son relevantes para las bases de datos subcontratadas. El cliente que mantiene sus datos con una tercera entidad no confiable tiene dos preocupaciones importantes: la primera que la información puede ser sensible y por ello puede no querer revelarla al servidor y la segunda es que la información cuyo almacenamiento y mantenimiento ha sido delegado, será utilizada por él. El uso que generalmente se le da a los datos consiste en que el cliente debe poder consultar la base de datos y las respuestas a sus consultas serán generadas por el servidor. Es natural que el cliente este preocupado porque un servidor malicioso no le provea las respuestas correctas. El objetivo de este trabajo es diseñar dos esquemas: uno en el que la base de datos pueda ser cifrada y que aún así se puedan ejecutar consultas y otro en que el cliente pueda verificar si el servidor esta respondiendo de forma correcta. Estos problemas son conocidos como *privacidad en bases de datos* y *procesamiento de consultas autenticado*, respectivamente. En esta tesis se estudiaron ambos de manera formal, se plantearon sus difiniciones de seguridad, y además se diseñaron esquemas eficiententes para resolverlos. Finalmente, se analiza la seguridad de los esquemas propuestos y también se reporta su desempeño.

*A mi mamita porque te llevo conmigo y soy el reflejo de todo tu esfuerzo.*

# Agradecimientos

*Don pepe* gracias por esperar este día con tanta ilusión y entusiasmo. Por enseñarme la fortaleza de un espíritu inquebrantable.

Gracias a *la abuelita Rosita* por la huella que marcó en mi camino. No pasa un solo día sin que la recuerde.

A mis hermanos, porque cada uno de ellos ha colaborado en mi formación. Gracias *Decho* por todas esas charlas llenas de excelentes consejos. *Pancho*, por invitarme a caminar en la senda del saber. *Nena Ana*, por ser mi confidente y amiga. *Chepe*, por ayudarme a descubrir que soy fuerte.

A toda mi familia, de la cual me siento muy orgullosa, gracias por su apoyo incondicional e inspirarme a continuar en la conquista de mis sueños.

A Raymundo Domínguez Colín por ser mi compañero, amigo, y sobretodo cómplice. Gracias por ser un pilar en la búsqueda de esta meta.

A mis amigos con los que compartí largas jornadas de trabajo, sueños y tristezas: Brisbane, Sandra, Cuauhtemoc, Yee, Thomaz, Gora y Andrés. Con especial cariño a mi gran amigo *Cheche* por ser un excelente camarada. A mis amigos de vida, quienes siempre mostraron su apoyo.

Gracias al Dr. Debrup Chakraborty, quien me ha transmitido sus conocimientos y me ha forjado como investigadora.

A mis lectores, Dr. Guillermo Morales, Dr. Carlos Coello, Dr. Gerardo de la Fraga, Dr. Raúl Monroy y Dr. Carlos Coronado, por el tiempo dedicado a la lectura de este trabajo y sus valiosas aportaciones.

A Sofía Reza, Felipa Rosas y Erika Ríos; por todo su apoyo en las labores administrativas y su inigualable amabilidad.

Al CINVESTAV, por ser mi casa y permitirme adquirir nuevos conocimientos en Criptografía y Seguridad; y por el apoyo económico para la culminación de este trabajo.

Al CONACyT, por la beca que me otorgó durante estos cuatro años y al proyecto número 166763 por el soporte monetario para asistir a eventos de Criptografía.

A El Salvador, mi país, al cual aspiro representar responsablemente y aportar mi granito de arena para ayudar a resolver sus problemas.

A México, por recibirme con los brazos abiertos y compartir conmigo su cultura.

# Contents

**Abstract** **i**

**Resumen** **iii**

**1 Introduction** **1**

1.1 The Scope of the Thesis . . . . . 3

1.2 Overview of the Solutions . . . . . 5

1.2.1 Authenticated Query Processing . . . . . 5

1.2.2 Database Privacy . . . . . 7

1.3 About Rest of the Document . . . . . 8

**2 Preliminaries** **11**

2.1 General Notations . . . . . 11

2.2 Cryptography . . . . . 12

2.3 Symmetric Key Encryption . . . . . 13

2.3.1 Block Ciphers . . . . . 14

2.3.2 Pseudorandom Functions and Permutations . . . . . 14

2.3.3 Block Cipher Modes of Operations . . . . . 17

2.3.4 Message Authentication Codes . . . . . 18

2.4 Proofs by Reduction . . . . . 19

2.5 Some Other Cryptographic Primitives . . . . . 22
2.6 Relational Databases . . . . . 23
2.6.1 Relational Databases: Basic Structure and Notations . . . . . 23
2.6.2 Bitmaps and their Encodings . . . . . 26
2.6.3 Bitmap Compression . . . . . 28
2.7 Summary . . . . . 30

**3 The State of the Art 31**
3.1 Database As A Service Model . . . . . 31
3.2 Query Authentication Problem . . . . . 32
3.3 Some Existing Schemes for Authenticated Query Processing . . . . . 35
3.3.1 Approaches Based on Authenticated Data Structures . . . . . 37
3.3.2 Approaches Based on Signatures . . . . . 39
3.3.3 Discussion and Comparisons . . . . . 41
3.4 The Problem of Privacy in Outsourced Databases . . . . . 41
3.5 Order Preserving Encryption Schemes: An Overview . . . . . 43
3.6 Final Remarks . . . . . 48

**4 Relational Database Authentication Scheme 51**
4.1 RDAS: Definitions and Basic Notions . . . . . 51
4.1.1 Correctness and security . . . . . 52
4.2 RDAS1: A generic scheme for select queries involving arbitrary disjunctions . . . . . 53
4.2.1 Security of RDAS1 . . . . . 60
4.2.2 Costs and Overheads . . . . . 63
4.3 Summary . . . . . 64

**5 Extensions and Improvements of RDAS1** **65**

5.1 RDAS2: Selects Involving Arbitrary Boolean Connectives . . . . . 66

5.1.1 Security of RDAS2 . . . . . 67

5.2 The case of empty replies . . . . . 67

5.3 Using Aggregated MACs . . . . . 69

5.3.1 Security of RDAS1-agg and RDAS2-agg . . . . . 70

5.4 RDAS2-cmp: RDAS2 with Compression . . . . . 73

5.5 Dynamic databases . . . . . 74

5.6 Further Extensions and Improvements . . . . . 78

5.6.1 Including other query types . . . . . 78

5.6.2 Multiuser settings . . . . . 79

5.7 Discussions and Comparisons . . . . . 79

5.8 Summary . . . . . 82

**6 ESRQ1: A Scheme to Provide Privacy in Relational Databases** **83**

6.1 Additional Notations . . . . . 85

6.2 Encryption Scheme Supporting Range Queries (ESRQ): Definitions and Basic Notions . . . . . 86

6.2.1 Security Notion . . . . . 87

6.3 Matrix representation of bitmaps . . . . . 90

6.4 ESRQ1: An Encryption Scheme Supporting Range Queries . . . . . 95

6.4.1 Characteristics of ESRQ1 . . . . . 103

6.4.2 Security of ESRQ1 . . . . . 104

6.4.3 Proof of Theorem 6.1 . . . . . 105

6.5 Summary . . . . . 109

**7 Practical Instantiations of DEAD Schemes 111**

7.1 A Construction for Restricted Message Spaces . . . . . 112

7.2 OHCTR: A Construction for Arbitrary Message Spaces . . . . . 114

7.2.1 Security of OHCTR . . . . . 116

7.2.2 Proof of Theorem 7.2 . . . . . 116

7.3 Summary . . . . . 123

**8 Implementations 125**

8.1 System Information . . . . . 125

8.2 Results on the Query Authentication Schemes . . . . . 126

8.2.1 The Basic Building Blocks . . . . . 126

8.2.2 Experimental Settings . . . . . 127

8.2.3 Experimental Results on RDAS1 and its Variants . . . . . 129

8.2.4 Experimental results on RDAS2 and its Variants . . . . . 132

8.3 Results on ESRQ1 . . . . . 135

8.4 Final Remarks . . . . . 138

**9 Conclusions and Future Work 139**

9.1 Conclusions . . . . . 139

9.2 Future work . . . . . 141

**Appendices 143**

**A Queries 143**

**Bibliography 147**

# Notation

| | |
|---|---|
| $\perp$ | Undefined value |
| $\{0,1\}^*$ | The set of all binary strings |
| $\{0,1\}^n$ | The set of $n$ binary strings |
| $\|X\|$ | If $X$ is a string $\|X\|$ denotes its length |
| $X\|\|Y$ | Concatenation of $X$ and $Y$ |
| $bit_i(X)$ | The $i$-th bit of $X$ |
| $x \xleftarrow{\$} A$ | $x$ is an element drawn uniformly at random from the set $A$ |
| $\#A$ | Let $A$ be a set, then $\#A$ denotes its cardinality |
| $[M]$ | Let $[M]$ be a set of consecutive integers from 1 to $M$ |
| $\mathbb{F}_{2^n}$ | A finite binary field |
| $\varphi(x)$ | Irreducible Polynomial |
| $X \oplus Y$ | Addition in $\mathbb{F}_{2^n}$ |
| $XY$ | Multiplication in $\mathbb{F}_{2^n}$ |
| $\mathbf{Adv}_F^{\mathrm{prf}}(\mathcal{A})$ | Advantage of adversary $\mathcal{A}$ in breaking $F$ in sense of PRF |
| $\mathcal{A}^O \Rightarrow 1$ | An adversary interacts with an oracle $O$ and finally outputs 1 |
| $Pr[\zeta]$ | The probability of the event $\zeta$ |
| $\Xi$ | A symmetric key encryption scheme |
| $E_K()$ | Block cipher with key $K$ |
| $\mathbf{E}_K(T,M)$ | Deterministic encryption scheme with associated data |
| $MAC_K()$ | Message Authentication Code under the action of the key $K$ |
| $MAC_K^*()$ | Aggregated MAC |
| $R(A)$ | Relation over a set of attributes $A$ |
| $t_j^R$ | The $j^{th}$ tuple in the relation $R$ |
| $t_j^R[a_i]$ | the value of attribute $a_i$ in the $j^{th}$ tuple in $R$ |
| $\mathcal{R}$ | A set of relations. |
| $\mathsf{BitMap}_R(a_i, v_j^i)$ | Bitmap of attribute $a_i$ in relation $R$ and value $v_j^i$ |
| $\mathcal{L}$ | It is a list with $n$ elements |
| $\mathcal{M}$ | Matrix of bitmaps |
| $\mathsf{arr}_{\mathcal{M}}$ | The array arr represents the ordered matrix $\mathcal{M}$ |

# Abbreviations

| | |
|---|---|
| ADS | Authenticated Data Structure |
| AE | Authenticated Encryption |
| AEAD | Authenticated Encryption with Associated Data |
| AES | Advanced Encryption Standard |
| DAS | Database As a Service |
| DAE | Deterministic Authenticated Encryption |
| DAEAD | Deterministic Authenticated Encryption with Associated Data |
| ESRQ | Encryption Scheme Supporting Range Queries |
| EWAH | Enhanced Word Aligned Hybrid |
| MAC | Message Authentication Code |
| HCTR | Hash Counter Mode |
| OHCTR | Only One Hash Counter Mode |
| PRF | Pseudorandom Function |
| PRP | Pseudorandom Permutation |
| RDAS | Relational Database Authentication Scheme |
| SQL | Structure Query Lenguage |
| TES | Tweakable Enciphering Scheme |
| VO | Verification Object |

# Chapter 1

# Introduction

> With me poetry has not been a purpose, but a passion.
>
> *Edgar Allan Poe*

Now-a-days almost every company needs to handle an enormous amount of information for its proper functioning. Storage, maintenance and processing of such enormous amount of data need lots of resources like special hardware, software and human expertise. The owner of the data (the client) whose main business or expertise may not be directly related to data processing technologies is unlikely to have such special resources. Also, maintaining this infrastructure may involve high costs. Currently, this problem has taken a serious shape, as irrespective of the type of business an organization or individual is involved with, it requires to depend on various information technologies.

As a response to this challenge, the concept of *cloud computing* has emerged. *Cloud computing* is a popular computing paradigm in which computation is moved away from a single personal computer or an individual application server to a huge network of computers, now called a ‘‘cloud” [1]. Cloud computing holds the promise of revolutionizing the manner in which enterprises manage, distribute, and share information. The data owner (client) can out-source almost all its information processing tasks to a “cloud”. The cloud can be seen as a collection of servers (we shall sometimes refer to it as the server) which caters the data storage, processing and maintenance needs of the client. Needless to say this new concept of computing can bring significant savings in terms of costs for the data owner.

The cloud provides three fundamental services [2]. These services are often referred to as the “SPI Model”, where SPI stands for Software, Platform and Infrastructure. The difference between these three services lies in the control that is given to the client to use the infrastructure provided by the service provider. Software as a Service (SaaS)

is the most used, as it only provides final applications, for example, google docs. In Platform as a Service (PaaS) customers can develop their own applications, using pre-programmed blocks, for instance, google data services. Finally, regarding to Infrastructure as a Service (IaaS) the client can decide which operating system and network components he/she wants to use.

Among others, an important service that falls within the first classification (i.e., SaaS) is that of storage. In this service the client delegates the duty of storage of his/her files to a third party (an un-trusted server), and the client requires to have access to them in the future [3]. Another service that falls within the second classification (i.e., PaaS) is *Database as a Service (DAS)*. In this service the client also delegates his/her data to a third party. But, in this case the DAS model allows the client to perform operations like create, modify and retrieve from databases in a remote location [4]. These operations are performed by the server on behalf of the client.

An organization that provides some of these services can offer different tasks such as data backup, data restore, data reorganization to reclaim space, data management, etc. Moreover, the users wishing to consult the data will now access it using the hardware and software at the service provider instead of their own organization's computing infrastructure. However, delegating the duty of storage and maintenance of data to a third party brings in some new security challenges. In a typical application, some portions of the data may be sensitive and should be protected from the *adversaries*. An adversary is some individual/organization who has malicious intention and particularly the entity from whom the sensitive information needs to be kept protected. In this type of applications, the client/owner side environment is assumed to be secure and trusted therefore the main threat is from the database provider who is untrusted.

Information security had been achieved through *cryptographic techniques*. However, achieving the information security objectives in outsourced applications require a new set of these techniques, since new features are required in this model. For example, users of outsourced storage rely on their providers, which are untrusted to maintain the availability of their data. The solution to this problem is known as storage auditing: which requires to search for cryptographic systems that allow the client to verify that his/her data are still available and ready for retrieval if needed, without retrieving the files [5]. Another instance is encryption, which is a useful tool for protecting the confidentiality of sensitive data. However, when data is encrypted with classical schemes, performing queries efficiently becomes more challenging. Finally, we want to discuss authenticated query processing case. Whenever an outsourced database is queried, the corresponding query reply must be demonstrably authentic. Furthermore, a reply must include a proof of completeness to convince the querier that no data matching

the query predicate(s) has been omitted. Even though, there are well known cryptographic primitives for data authentication as message authentication codes or digital signatures, the completeness requirement makes of this scenario a challenging problem.

In this thesis we aim to study the security services on outsourced data in detail. Mainly, we will focus on two services: Privacy and Data authentication. Our study encompasses analysis of the existing schemes, we also attempt to design new protocols which can serve as a solution to these security services on outsourced databases. Finally, we are interested in their implementations and their performance.

The rest of this chapter is organized as follows. In Section 1.1 we briefly discuss the objectives of this work and provide non-technical introduction to the problems of our interest. In Section 1.2 we discuss the results that we have obtained and in the last section we discuss the structure and content of the rest of this document.

## 1.1 The Scope of the Thesis

The main goal of our work is to see the problem of security in outsourced data from a cryptographic viewpoint, and come up with realistic solutions for it. We aim to develop schemes which would be efficient and can be deployed with ease in existing systems with minimal extra overhead. Moreover we thrive for solutions which would be sound in the context of modern cryptography, i.e., we aim to see the problem in a formal cryptographic view. In modern cryptography, given a scheme one aims to define security in formal terms and prove its security in accordance with the definition. Such balanced schemes (in the two aspects as discussed above) in the area of our focus is rare.

The area of securing outsourced data is huge, and it has different directions and ramifications. In this thesis we assume a specific structure of the outsourced data, in particular we consider a relational database. We address the two most important security issues for such kind of data, and we briefly discuss them next.

1. **Authenticated Query Processing:** When a client delegates the duty of storage and maintenance of his/her data to a third party, it is important to ensure security of the information against a malicious server who does not responds correctly to the queries of the client. This problem is easy to define in the context of an outsourced relational database. Where the client stores the tables in a third party (server) and the queries are replied by it. The replies provided by the server

should be both correct and complete[1] with respect to the stored tables. The problem of authenticated query processing aims at developing schemes where the server should be able to provide "short proofs" of the fact that the replies to the specified queries are correct and complete. This problem can be seen in different dimensions, among them two important aspects of the problem are:

- **Private versus public verifiability**: In private verifiability we assume a single data owner model, where the owner and querier is a single entity who would like to verify the validity of the query response of the server. In public verifiability there may be different queriers and they may be distinct from the owner. These two scenarios require completely different treatments. Primarily, for private verifiability one can use techniques for data authentication as relevant to symmetric key cryptography whereas in case of public verifiability one needs to apply techniques from public key cryptography. In our work we focus only on private verifiability. It is worth to ensure that there are several real life scenarios where public verifiability is not required, and in general schemes with public verifiability needs to use computationally heavier machinery thus making the schemes less efficient. Thus, in scenarios where public verifiability is not required having private verifiability as the goal can lead to more efficient and usable schemes. The current literature does not duly classify these two scenarios and there are few schemes which provide only private verifiability.
- **Static versus dynamic data**: Databases can be both static and dynamic. In a static database, updates, inserts and deletions are not allowed. Ensuring authenticated query processing in static databases is comparatively easier than providing this service in dynamic databases. In this work we aim to address both scenarios.

2. **Database Privacy:** Other than authentication, the other main concern of the client about the outsourced data is privacy. However, the client uses the server to store his/her information, but the server is untrusted and the client does not want to reveal its data to the server. This problem can be solved with the help of an encryption algorithm, where the client encrypts the information in the tables before it outsources it to the server. But, traditional encryption schemes if applied for this problem would render the data unusable for query processing, as a good encryption algorithm is supposed to hide all structures present in the data, and the encrypted data would "look random". There has been numerous works to de-

[1]Correct means that records on the database have not been modified. Complete means that no answers have been omitted from the reply.

vice cryptographic schemes so that the encrypted data can be amenable to standard query processing. Again there are several dimensions of this problem, the main theme being using an encryption scheme where the ciphertext retains some properties of the plaintext. In this work we aim to address only a part of this big problem. We aim to device an order preserving encryption scheme, i.e., an encryption scheme $\mathbf{E}$, such that $\mathbf{E}(x) \geq \mathbf{E}(y)$ if and only if $x \geq y$ (monotonically non-decreasing). Such an encryption scheme helps in executing range queries on encrypted data, which is one of the main challenges in designing encryption schemes amenable to query processing.

## 1.2 Overview of the Solutions

In this section we give a short account of the solutions that we propose for the two problems stated before.

### 1.2.1 Authenticated Query Processing

Though there has been considerable amount of work on authenticated query processing on relational databases, but it has been acknowledged (for example in [6]) that the problem of query authentication largely remains open. An unified cryptographic treatment of the problem is missing in the literature. In most existing schemes cryptographic objects have been used in an ad-hoc manner, and their security guarantees are not very clear. In our work we initiate a formal cryptographic study of the problem of query authentication in a distinct direction. We propose a new scheme which does not use any specialized data structure to address the completeness problem. Our solution involves usage of bitmap indices for this purpose. Bitmap indices have gained a lot of popularity in the current days for their use in accelerated query processing [7, 8]. Many commercially available databases like Oracle, IBM DB2, Sybase IQ now implement some form of bitmap index scheme in addition to the more traditional B-tree based schemes, thus it may be easy to incorporate a bitmap based scheme in a modern database without significant extra cost. To our knowledge, bitmaps have not been used till date for a security goal.

In addition to bitmap indices we use a secure message authentication code (MAC) as the only cryptographic object. We show that by the use of these simple objects one can design a query authentication scheme which allows verification of both correctness and completeness of query results. As the basic cryptographic object is a symmetric

key primitive, thus our scheme does not provide public verifiability. Moreover, as stated before, in this work, we restrict ourselves to private verifiability only. We see private verifiability of our scheme more as a design goal than a limitation, as we believe that there exist scenarios where public verifiability may not be required, and in such scenarios it is better not to use the heavy machinery of public key signatures which uses computationally intensive number theoretic operations, whereas computational overheads of symmetric key message authentication schemes are minimal.

Next, we summarize our concrete contributions in this problem:

1. We define a generic scheme which we call as relational database authentication scheme (RDAS) which would provide the functionality of authenticated query processing. We carefully define the security goals of RDAS in line with the tradition of concrete provable security. The security definition encompasses both correctness and completeness of a query response. Such a definition is new to the literature, and we hope that this definition would help to evaluate security of existing schemes.

2. We propose several concrete instantiations of RDAS:

   - RDAS1: RDAS1 is designed using message authentication codes and bitmap indices in a novel manner. RDAS1 is capable of authenticated query processing of simple select queries and select queries involving disjunctions of equality conditions. The extra overhead for using RDAS1 both in terms of extra bandwidth and computation cost is not significant. We formally prove that RDAS1 provides authentication in accordance to our security definition. Moreover we report some performance data.
   - RDAS2: RDAS1 is efficient and secure, a serious limitation of it is that it can only authenticate a restricted class of queries. We point out various directions in which RDAS1 can be modified to incorporate other types of queries. In particular we propose a modification called RDAS2 which is capable of authenticating a large class of queries but it has more overhead than RDAS1.
   - RDAS1-agg, RDAS2-agg: These variants of RDAS1 and RDAS2 uses aggregated message authentication codes. The use of this primitive improves the bandwidth requirements of both RDAS1 and RDAS2 to a large extent. We also prove security of these variants.
   - RDAS2-cmp: This version uses the idea of compressing bitmaps with a compression scheme called Enhanced Word-Aligned Hybrid (EWAH) before they are stored, this saves storage space and reduces the bandwidth requirement compared to RDAS2. Furthermore, this compression scheme allows logical

operations on compressed bitmaps, and the performance is even faster than the scheme where uncompressed bitmaps are used.

- RDAS-dyn: The instantiations of the RDAS mentioned above work only on static scenarios, i.e., they cannot accommodate database updates. We overcome this limitation in RDAS-dyn which can work in dynamic scenarios.

3. We have done highly optimized implementations of all the above mentioned variants using some modern features of current Intel processors. With these implementations we have generated comparative performance data of these variants on a "reasonably large" real database. The performance data suggests that all these variants have negligible overhead both in terms of time and bandwidth requirement, and thus can be deployed in real life applications.

### 1.2.2 Database Privacy

As mentioned earlier, the problem of interest to us in this category is to develop an encryption scheme for relational databases which supports range queries. This has been previously addressed by encrypting each database entry with an order preserving encryption scheme. There have been some works on order preserving encryption [9, 10], where the main concern was to fix the proper security definition of such a primitive, recently there have been some advances in proposing an efficient scheme [11, 12, 13, 14]. We restrict ourselves to the scenario of encrypting numerical data in a relational database, we believe that an order preserving encryption scheme would have maximum applicability in this scenario.

As in case of authenticated query processing we propose a generic framework for encrypting databases such that range queries can be efficiently run on the encrypted tables. We call the generic scheme as ESRQ (encryption scheme supporting range queries). We also propose a novel security definition for ESRQ and finally propose a particular scheme called ESRQ1.

ESRQ1 has several interesting characteristics. In ESRQ1 a given relation is encrypted using a deterministic encryption scheme, and the order relations between different attribute values are separately stored along with the encrypted relation. With this information the server is able to respond to range queries. But the challenge in implementing this idea is to come up with a strategy to store the order information of the database value in a compact form. We use a specific type of bitmap encoding, called the range encoding for this purpose. We use a novel data structure where all order information can be compactly stored in a single array and a list. Also, this data structure allows

efficient updates and inserts in the encrypted tables. This specific data structure can be of independent interest in the context of the design of efficient index structures for databases. We adequately argue that ESRQ1 provides the required functionalities of an encrypted database, and also prove that ESRQ1 is secure according to our security definition.

The only cryptographic object that ESRQ1 requires is a deterministic encryption scheme which supports associated data and it is secure against chosen plaintext adversaries. We provide some new constructions of this cryptographic objects and prove them to be secure. Finally, we implement ESRQ1 on a real database and report some performance data for it. The overall performance data is encouraging and it suggests that ESRQ1 can be a viable option for encrypting relational databases.

## 1.3 About Rest of the Document

The rest of this document is organized as follows. In Chapter 2, we present some basic definitions of the two areas of our interest, i.e, cryptography and databases. In Chapter 3 we discuss the two main problems that we aim to study: authenticated query processing and privacy in outsourced databases, we also present a brief survey of the literature related to these two problems. Our contributions to the area are reported starting from Chapter 4.

Chapter 4 deals with the definition of a generic scheme which we call as relational database authentication scheme (RDAS). It also presents the security goals of RDAS in line with concrete provable security. Finally, in this chapter we present our basic solution for the query authentication problem called RDAS1 which allows the client to ensure both the correctness and completeness of the query results obtained from a server.

In Chapter 5, we propose multiple extensions of the basic protocol RDAS1. In particular we discuss RDAS2, which is capable of authenticating a large class of queries but has more overhead than RDAS1. We also discuss how additional features can be added to RDAS1 and RDAS2 to make them efficient in several respects. In particular we discuss, how one can use aggregate message authentication codes and bitmap compression to minimize the requirements of storage and bandwidth. Finally in this chapter we discuss the basic ideas of RDAS-dyn, which is an extension of RDAS that can be used for dynamic databases.

In Chapter 6 we initiate our study of database privacy. We discuss a basic frame-

work for encrypting relational databases so that range queries can be executed on the encrypted relation, we call this basic framework as ESRQ (to be read as encryption scheme supporting range queries). We develop a notion of security for ESRQ. Finally, we present a specific ESRQ, called as ESQR1. The novelty of ESRQ1 is in the use of a specific encoding of bitmaps, which we call as the *l*-encoding. We also device a special data structure that helps in efficient storage and manipulation of *l*-bitmaps. This data structure can have uses in other database applications also. Finally we prove that ESRQ1 is secure in our proposed security model.

In addition to *l*-bitmaps, ESRQ1 uses a deterministic encryption scheme which supports associated data (we call such schemes as DEAD). This primitive, though can be derived from other available cryptographic objects, is new. Thus, we devote Chapter 7 in the study of DEAD schemes. We report two new constructions of DEAD and prove their security.

In Chapter 8 we report the implementations of the schemes presented in this thesis. We report performance data for RDAS1, RDAS2 and ESRQ1 on a real database based on the prototypes that we implemented.

Finally in Chapter 9 we summarize our contributions and discuss some future directions of work.

# Chapter 2

# Preliminaries

> All secrets are deep. All secrets become dark. That's in the nature of secrets.
>
> *Cory Doctorow*

We focused our research on the problem of security services on outsourced databases. For easy exposition of intricacies related to the problem, an understanding of issues related to cryptographic primitives and basic concepts involving databases is necessary. Thus, in this chapter we introduce our general notation and discuss some basics of symmetric cryptographic primitives and databases.

## 2.1 General Notations

Here we note down some general notations that we would use throughout the thesis. This is not a comprehensive list of the used notations, more notations are introduced when used first time. Also for easy reference a detailed list of notations is provided at the beginning of the thesis.

The set of all binary strings is denoted by $\{0,1\}^*$. For a positive integer $n$, $\{0,1\}^n$ denotes the set of all $n$ bit strings. For $X, Y \in \{0,1\}^*$, by $X||Y$ we will mean the concatenation of $X$ and $Y$. For $X \in \{0,1\}^*$, $|X|$ will denote the length of $X$ and $\mathsf{bit}_i(X)$ will denote the $i$-th bit of $X$.

By $\mathbb{F}_q$ we shall mean a finite field with $q$ elements. For our purpose we shall be interested in the field $\mathbb{F}_{2^n}$ for some $n$. An $n$ bit string can be represented by a polynomial of degree less than $n$, whose coefficients are in $\mathbb{F}_2$. For example, if $A \in \{0,1\}^n$ such that $A = \langle a_0, a_1, \cdots, a_{n-1} \rangle$ with $a_i \in \{0,1\}$ then $A$ can be represented by the polynomial $A(x) = \sum_{i=0}^{n-1} a_i x^i$. Thus the set of all $n$ bit strings can be treated as the field $\mathbb{F}_{2^n}$ where the

addition is defined as the *xor* of the strings and multiplication is defined as the multiplication of the two polynomials corresponding to the strings modulo an irreducible polynomial $\varphi(x)$ of fixed degree $n$. Thus, we will sometimes treat the set $\{0,1\}^n$ as the field $\mathbb{F}_{2^n}$ and for $X, Y \in \{0,1\}^n$, $X \oplus Y$ and $XY$ will denote addition and multiplication in $\mathbb{F}_{2^n}$, respectively.

For a finite set $A$, $\#A$ would denote the cardinality of $A$, and $x \xleftarrow{\$} A$ would denote that $x$ is a uniform random element of $A$.

## 2.2 Cryptography

According to Menezes et. al. [15] Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, authentication, and non-repudiation. We discuss these basic information security goals in brief next:

1. **Confidentiality** is a service to keep information hidden from all, except to those authorized to have it. This goal is also sometimes called privacy or secrecy.

2. **Data integrity** is a service which avoids the unauthorized alteration of data. Where insertion, deletion and substitution may be some forms of unauthorized changes.

3. **Authentication** is a service related to identification. Generally, two different kind of authentication are important, namely, entity authentication and data origin authentication.

4. **Non- repudiation** is a service which prevents an entity from denying previous commitments or actions.

For our purpose we would focus on the first two goals, i.e., confidentiality and data integrity. The problem of data integrity can also be linked with data origin authentication. To see this consider, that a sender $A$ sends a message $m$ to $B$, which gets tampered to $m'$ in transit. Thus $B$ receives $m'$ instead of $m$. This violates the integrity of the message $m$ and also the origin of the tampered data $m$ is no more $A$. Thus the problem of data integrity and data source authentication are related and in the literature these two terms are used interchangeably. We, in this thesis, would also sometimes use authentication to mean integrity.

Cryptography can be broadly classified into two categories namely symmetric (private) key Cryptography and Asymmetric (public) key Cryptography. This classification is based on the usage of the keys. In symmetric key cryptography the sender and receiver uses the same key which is kept secret, where as in asymmetric key cryptography the sender and receiver use different keys and one of them is public. We would be using symmetric key cryptography which involves schemes where both parties share a secret information called the key. The classic private key cryptographic tools that provide the security services of confidentiality and data integrity are *encryption schemes* and *message authentication codes*, respectively. Though encryption schemes and message authentication codes cannot be used in a straightforward way to achieve our security goals in outsourced databases, but still they would form the backbone of the schemes discussed in this thesis. Next, we discuss some preliminary notions of symmetric key encryption and authentication.

## 2.3 Symmetric Key Encryption

In Symmetric Key Encryption, two parties share some secret information called a *key*, and use this key when they wish to communicate secretly with each other. A party sending a message uses the key to *encrypt* the message before it is sent, and the receiver uses the same key to *decrypt*. The message itself is commonly called the *plaintext* and the encrypted message that is transmitted by the sender to the receiver is called *ciphertext*. The shared key serves to distinguish the communicating parties from any other parties who may be eavesdropping on their communication.

**Symmetric-key encryption scheme**: A symmetric key encryption scheme $\Xi$ is comprised of three algorithms $\Xi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, the first is a procedure for generating keys, the second is a procedure for encrypting, and the third a procedure for decrypting. These algorithms have the following functionality:

1. The *key-generation algorithm* $\mathsf{Gen}$ is a probabilistic algorithm that outputs a key $K$ chosen from a pre-defined finite set $\mathcal{K}$ (called the key space) according to some distribution that is determined by the scheme. Unless mentioned otherwise, the key generation algorithm selects a key $K$ uniformly at random from $\mathcal{K}$ and outputs it.

2. The *encryption algorithm* $\mathsf{Enc}$ takes as input a key $K$ and a plaintext message $m \in \mathcal{M}$ and outputs a ciphertext $c \in \mathcal{C}$. The sets $\mathcal{M}$ and $\mathcal{C}$ are the sets of all possible messages and ciphertexts respectively and they are commonly called the message

space and cipher space. We denote by $\mathsf{Enc}_K(m)$ the encryption of the plaintext $m$ using the key $K$.

3. The *decryption algorithm* Dec takes as input a key $K \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$ and outputs a plaintext $m \in \mathcal{M}$. We denote the decryption of the ciphertext $c$ using the key $K$ by $\mathsf{Dec}_K(c)$.

The basic notion of correctness is that for all $K \in \mathcal{K}$ and $m \in \mathcal{M}$, $\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m$, i.e., irrespective of the choice of key and message the decryption always undoes the encryption. Symmetric-key encryption schemes are generally designed using cryptographic primitives like block ciphers and stream ciphers.

### 2.3.1 Block Ciphers

An $n$-bit block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, where $\mathcal{K} \neq \emptyset$ is the key space and for any $K \in \mathcal{K}$, $E(K,.)$ is a permutation on $\{0,1\}^n$. We write $E_K(\,)$ instead of $E(K,.)$. So a block cipher takes an $n$-bit input which is the plaintext and produce an $n$-bit output also called the ciphertext under the action of a $k$-bit key, the values of $n$ and $k$ vary for different block ciphers. The key is secret and in general it is selected uniformly at random from the key space $\mathcal{K}$. Some practical block ciphers are the Data Encryption Standard (DES) [16], The Advanced Encryption Standard (AES) [17], etc.

### 2.3.2 Pseudorandom Functions and Permutations

Security of a block cipher is defined using an abstract object called a pseudorandom permutation. We discuss some basic notions of pseudorandom functions and permutations next, the discussions closely follows [18].

Consider the map $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ where $\mathcal{K}, \mathcal{D}, \mathcal{R}$ (commonly called keys, domain and range respectively ) are all non-empty and $\mathcal{K}$ and $\mathcal{R}$ are finite. We view this map as representing a *family of functions* $F = \{F_K\}_{K \in \mathcal{K}}$, i.e., for each $K \in \mathcal{K}$, $F_K$ is a function from $\mathcal{D}$ to $\mathcal{R}$ defined as $F_K(X) = F(K,X)$. For every $K \in \mathcal{K}$, we call $F_K$ to be a instance of the family $F$.

Let $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a function family where $\mathcal{D} = \mathcal{R}$, and for every $K \in \mathcal{K}$, let $F_K : \mathcal{D} \to \mathcal{D}$ be a bijection, then we say that $F$ is a permutation family. So, if $F$ is a permutation family, then for every $F_K(\cdot)$, we have a $F_K^{-1}(\cdot)$, such that for all $K \in \mathcal{K}$ and all $X \in \mathcal{D}$, $F_K^{-1}(F_K(X)) = X$. Note that a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ is a permutation family.

We would be interested in probability distributions over a function family $F$, in particular we would often talk of sampling an instance at random from the family. By sampling an instance $f$ uniformly at random from $F$ we would mean $K \xleftarrow{\$} \mathcal{K}$ and $f = F_K()$, we will denote this by $f \xleftarrow{\$} F$.

Let $\mathsf{Func}(\mathcal{D}, \mathcal{R})$ be the set of all functions mapping $\mathcal{D}$ to $\mathcal{R}$, if $\mathcal{D} = \{0,1\}^m$ and $\mathcal{R} = \{0,1\}^n$ then $\mathsf{Func}(m,n)$ is the set of all functions that map from $m$ bits to $n$ bits. Note, there are exactly $2^{n2^m}$ of these functions, i.e., $\#\,\mathsf{Func}(m,n) = 2^{n2^m}$. If $\mathcal{D}$ and $\mathcal{R}$ are specified, then by a random function with domain $\mathcal{D}$ and range $\mathcal{R}$ we mean a function sampled uniformly at random from $\mathsf{Func}(\mathcal{D}, \mathcal{R})$. Hence by a random function, we are not talking of the "randomness" of a specific function but we are talking of a function sampled from a probability distribution (specifically, the uniform distribution) over the set of all possible functions with a specified domain and range.

Informally a pseudorandom function (PRF) is a family of functions whose behavior is computationally indistinguishable from a random function. Consider the function family $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$, and let $f \xleftarrow{\$} F$ and let $\eta \xleftarrow{\$} \mathsf{Func}(\mathcal{D}, \mathcal{R})$. If $F$ is a PRF family then there should be no efficient procedure to distinguish between $f$ and $\eta$. To formalize this goal of distinguishing between a random instance of $F$ and a random instance of $\mathsf{Func}(\mathcal{D}, \mathcal{R})$, we introduce an entity which we call as a PRF *adversary*. A PRF adversary is considered to be a probabilistic algorithm whose goal is to distinguish between $f$ and $\eta$, and if it can successfully do so then we say that the adversary has broken the PRF property of $F$. The adversary is not provided with the description of the functions but it has an *oracle access* to a function $g$ which is either $f$ or $\eta$ and it needs to decide whether $g = f$. By an oracle access we mean that for any $x \in \mathcal{D}$ of its choice, the adversary can obtain the value $g(x)$ by querying the oracle of $g$. The adversary has the ability to query its oracle $g$ adaptively, i.e., it may be that first it wishes to query its oracle on $x_1$ and thus obtain $g(x_1)$, seeing $g(x_1)$ it decides its next query $x_2$ and so on. The adversary can query its oracle as long as it wants and finally it outputs a bit, say it outputs a 1 if it thinks that its oracle is $f$ (a real instance from the family $F$) and a zero if it thinks its oracle is $\eta$ (a random function). An adversary $\mathcal{A}$ interacting with an oracle $O$ and outputting a 1 will be denoted by $\mathcal{A}^O \Rightarrow 1$.

The PRF advantage of an adversary $\mathcal{A}$ in distinguishing $F$ from a random function is defined as

$$\mathbf{Adv}_F^{\mathrm{prf}}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F(K,.)} \Rightarrow 1\right] - \Pr\left[\eta \xleftarrow{\$} \mathsf{Func}(\mathcal{D}, \mathcal{R}) : \mathcal{A}^{\eta(.)} \Rightarrow 1\right]. \tag{2.1}$$

Hence the PRF advantage of the adversary $\mathcal{A}$ is computed as a difference between two probabilities, the adversary $\mathcal{A}$ is required to distinguish between two situations, the first situation is where $\mathcal{A}$ is given a uniformly chosen member of the family $F$ (i.e., $\mathcal{A}$

has oracle access to the procedure $F_K$, where $K \xleftarrow{\$} \mathcal{K}$) and in the other $\mathcal{A}$ is given oracle access to a uniformly chosen element of $\mathsf{Func}(\mathcal{D},\mathcal{R})$. If no adversary can tell apart these two situations then we consider $F$ to be a pseudorandom family. In other words $F$ is considered to be pseudorandom if for all *efficient* adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A})$ is *small*.

In this definition we use *efficient* adversary with *small* advantage. We will never make this more precise, and this is standard with the paradigm of "concrete security" where a precise notion of efficiency and small advantage is never specified. What makes an adversary efficient and its advantage small is left to be interpreted with respect to the specific application where such an object would be used. We define $\mathbf{Adv}^{\mathrm{prf}}_F(q,t)$ by $\max_{\mathcal{A}}\mathbf{Adv}^{\mathrm{prf}}_F(\mathcal{A})$ where maximum is taken over all adversaries which makes at most $q$ queries and runs for time at most $t$. We consider a family $F$ to be $(\varepsilon,q,t)$ PRF, if $\mathbf{Adv}^{\mathrm{prf}}_F(q,t) \leq \varepsilon$.

Let $E : \mathcal{K} \times \mathcal{D} \to \mathcal{D}$ be a family of functions such that for every $K \in \mathcal{K}$, $E_K : \mathcal{D} \to \mathcal{D}$ is a bijection. Such a family is called a permutation family. Note that according to our definition of a block cipher, a block cipher indeed is a permutation family.

Let $\mathsf{Perm}(\mathcal{D})$ denote the set of all bijections from $\mathcal{D}$ to $\mathcal{D}$. Analogous to the definition of PRF advantage, we define the PRP advantage of an adversary in distinguishing a random instance of the family $E$ from a random permutation $\pi$ as

$$\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K()} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathsf{Perm}(\mathcal{D}) : A^{\pi()} \Rightarrow 1\right].$$

And, $E$ is considered to be a pseudorandom permutation family if for all efficient adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{prp}}_E(\mathcal{A})$ is small.
We consider a family $E$ to be $(\varepsilon,q,t)$ PRP, if $\mathbf{Adv}^{\mathrm{prp}}_E(q,t) \leq \varepsilon$.

The fact that every member of a permutation family has an inverse, allows to define a stronger notion of pseudorandomness. Here we assume that the adversary is given two oracles one of the permutation and other of its inverse and the adversary can adaptively query both oracles. As before there are two possible scenarios, in the first scenario the adversary is provided with the oracles $E_K()$ and $E_K^{-1}()$ where $K \xleftarrow{\$} \mathcal{K}$ and in the other scenario the oracles $\pi(), \pi^{-1}()$ are provided where $\pi \xleftarrow{\$} \mathsf{Perm}(\mathcal{D})$. And the goal of the adversary is to distinguish between these two scenarios. We define the advantage of an adversary $\mathcal{A}$ in distinguishing a permutation family $E$ from a random permutation in the $\pm$prp sense as

$$\mathbf{Adv}^{\pm\mathrm{prp}}_E(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(),E_K^{-1}()} \Rightarrow 1\right] - \Pr\left[\pi \xleftarrow{\$} \mathrm{Perm}(\mathcal{D}) : \mathcal{A}^{\pi(),\pi^{-1}()} \Rightarrow 1\right],$$

and if for all efficient adversaries $\mathcal{A}$, $\mathbf{Adv}^{\pm\mathrm{prp}}_E(\mathcal{A})$ is small then we say $E$ is a strong

pseudorandom permutation (SPRP) family.

**Security of Block Ciphers.** As defined in Section 2.3.1, a block cipher is a permutation family $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. Of course any such permutation family cannot be considered as a block cipher, as a block cipher should have some security properties associated with it which any permutation family will not have. Defining security of a block cipher is tricky (as is true for all cryptographic primitives), if we consider that a block cipher $E_K()$ is used to encrypt $n$ bit strings then ideally given $E_K(X)$ one should not be able to obtain any information regarding $K$ or $X$, this property can be achieved if $E_K(X)$ "looks random" to any computationally bounded adversary. In practice we consider a block cipher to be secure if it behaves like a strong pseudo-random permutation.

### 2.3.3 Block Cipher Modes of Operations

Block ciphers are themselves not encryption schemes, as their message space is restricted to only fixed length ($n$- bit) strings. Block ciphers can be used in several ways to construct encryption schemes with different security properties. A specific way of using a block cipher to achieve a security/functionality goal is called a block cipher mode of operation. We informally discuss some specific types of modes which are currently in use.

1. **Privacy Only Mode:** As the name suggests, these kinds of modes are meant to provide privacy/confidentiality. In general these modes are either stateful or randomized. The examples of such modes are the cipher block chaining (CBC), counter (CTR), cipher feedback (CFB) and output feedback (OFB).

2. **Authenticated Encryption:** Authenticated encryption (AE) provides both the services of confidentiality and data integrity. As in privacy only modes they are also either randomized or stateful. Most constructions take as input a non repeating quantity called a nonce, which helps in maintaining the state. There are several constructions of AE schemes known to date, for example [19, 20, 21].

3. **Authenticated Encryption with Associated data:** Authenticated encryption with associated data (AEAD) are a variant of AE, where the encryption algorithm takes as input a message and a message header. The goal is to both encrypt and authenticate the message, but to only authenticate the header. Most AE schemes can be converted into AEAD schemes, some existing AEAD schemes can be found in [22, 23, 24].

4. **Deterministic Authenticated Encryption:** As stated before the AE and AEAD schemes are either randomized or stateful, whereas deterministic authenticated encryption schemes (DAE) are deterministic. There are many applications where a randomized/stateful encryption scheme cannot be suitably used, as it is the case of database encryption. In such scenarios DAE schemes are useful. DAE schemes can also be modified to support associated data, such schemes are called deterministic authenticated encryption with associated data (DAEAD). Some existing DAE and DAEAD schemes can be found in [25, 26, 27].

5. **Tweakable enciphering schemes:** Tweakable enciphering schemes (TES) are a special kind of deterministic modes which are length preserving, i.e., the length of the ciphertext is strictly the same as the length of the plaintext a property which is not satisfied by the modes mentioned above. TES takes in an additional public quantity called the *tweak*, which is meant to increase the variability of the ciphertext, i.e., two same plaintexts when encrypted with two different tweaks gives rise to two different ciphertexts. TES has been extensively used to develop encryption schemes for sector oriented storage devices like hard disks, flash memories etc. Concrete constructions of secure TES can be found in [28, 29, 30, 31].

In our work we would require a specific type of mode which to our knowledge has not been explicitly studied in the existing literature. We call this mode as a deterministic encryption scheme with associated data (DEAD). DEAD is different from DAE as it does not provide authentication, and in terms of security they are weaker than both DAE and TES. But DEAD provides the exact security and functionality as required for database encryption. In Chapter 7 we discuss more about DEAD schemes.

### 2.3.4 Message Authentication Codes

Message authentication codes (MAC) provide authentication in the symmetric key setting. It is assumed that the sender and the receiver share a common secret key $K$. Given a message $x$, the sender uses $K$ to generate a footprint of the message. This footprint (commonly called a $\mathsf{tag}$) is the message authentication code (MAC) for the message $x$. The sender transmits the pair $(x, \mathsf{tag})$ to the receiver. The receiver uses $K$ to verify that $(x, \mathsf{tag})$ is a properly generated message-tag pair. Verification is generally performed by regenerating the tag on the message $x$ and comparing the generated tag with the one received. The algorithm for generating the tag is known as a $\mathsf{MAC}$. Assuming that the size of the tag is $\tau$ bits, we see the tag generation scheme as a function $\mathsf{MAC} : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^{\tau}$, where $\mathcal{K}$ and $\mathcal{M}$ are the key and message spaces respectively. In most cases we shall write $\mathsf{MAC}_K(x)$ instead of $\mathsf{MAC}(K, x)$.

An attack on a MAC scheme signifies forging a message-tag pair. The types of attacks which are important for MAC schemes can be formally described as an interaction of an adversary $\mathcal{A}$ and the procedure MAC. $\mathcal{A}$ is given an oracle access to the MAC generation procedure $\mathsf{MAC}_K(.)$, instantiated with a randomly generated key $K$, which is unknown to $\mathcal{A}$. $\mathcal{A}$ can query $\mathsf{MAC}_K(.)$ with messages of its choice, and for each query $x$ it gets $\mathsf{MAC}_K(x)$ as a response. Let us assume that $\mathcal{A}$ queries with the messages $x_1, x_2, \ldots, x_q$ and gets $y_1, \ldots, y_q$ as the responses. In the end, $\mathcal{A}$ produces a pair $(\tilde{x}, \tilde{y})$, such that $\tilde{x} \notin \{x_1, x_2, \ldots, x_q\}$. It is said that $\mathcal{A}$ had committed a successful *forgery* if $\mathsf{MAC}_K(\tilde{x}) = \tilde{y}$. We define the advantage of the adversary $\mathcal{A}$ in forging the message authentication code MAC as follows:

$$\mathbf{Adv}^{\text{auth}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr[\mathcal{A} \text{ forges }]. \tag{2.2}$$

The probability is taken over the random choice of the key $K$ and the randomness of the adversary.

In the literature there are various known construction of MACs. There are three dominant paradigms for constructing MACs which we discuss next:

1. MACs can be constructed using block ciphers. Such MACs can be seen as a block cipher mode of operation designed to provide only the security service of authentication. Two well known block cipher based MACs are OMAC [32] and PMAC [33, 24]. OMAC is based on the cipher block chaining mode of operation, and being a part of some standard is also widely used. One drawback of OMAC is that it is inherently sequential thus not very efficient in either software or hardware. Unlike OMAC, PMAC is fully parallelizable.

2. MACs can also be constructed using universal hash functions. This paradigm was first proposed by Carter and Wegman [34]. The most popular among these MACs are polynomial evaluation MACs, specific examples of this class are Poly-1305 [35], UMAC [36] etc.

3. MACs can also be constructed from collision resistant hash functions. Examples of such constructions are HMAC and NMAC [37].

## 2.4 Proofs by Reduction

Modern cryptographic schemes are generally associated with a "security proof", which provides some formal arguments regarding to its security. This security proof can be

constructed by using a well known strategy called Proofs by Reduction. The main idea behind this technique is assume that a low level problem is hard to solve, and then try to prove that the scheme that is being analyzed is secure under this assumption; i.e. the reductionist argument shows how to convert any efficient adversary $\mathcal{A}$ that succeeds in breaking the protocol $\Xi$ with non-negligible probability into an efficient algorithm $\mathcal{B}$ that succeeds in solving the problem $X$ that was assumed to be hard. This can be see it in Figure 2.1 pictorially.

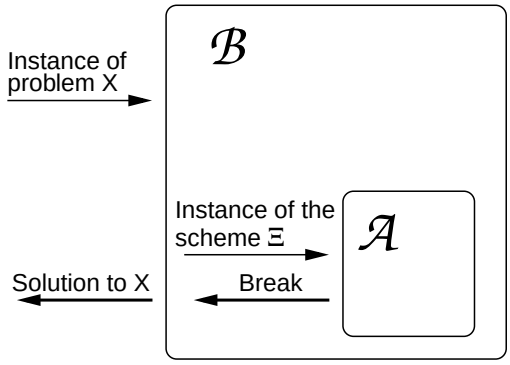



Figure 2.1: Overview of a security proof by reduction

The basic sequence of steps generally adopted to argue about the security of a scheme is discussed next.

1. Given a cryptographic scheme $\Xi$, first one needs to define the security notion of the scheme. The security notion is generally defined in terms of an interaction between the scheme $\Xi$ and an adversary $\mathcal{A}$, a specific goal for the adversary $\mathcal{A}$ is fixed and a scheme is said to be broken if an adversary $\mathcal{A}$ that achieves the goal can be constructed. This adversary is generally a computationally bounded probabilistic algorithm. For example, consider the discussions regarding MACs in Section 2.3.4. Here, the goal of the adversary is to generate a message forgery, and the goal is defined in terms of an interaction of $\mathcal{A}$ with the MAC. The advantage of an adversary (in equation (2.2)) is defined to be the probability that it achieves its goal.

2. To prove security of a scheme means to show that there exists no adversary that can achieve the goal as described in the corresponding security notion. For example, to prove that a MAC is secure one needs to show that there exists no computationally bounded adversary which has a non-negligible advantage in the sense of equation (2.2). It is to be noted, that the security is guaranteed only against

computationally bounded adversaries, such kind of security is called "computational security".

3. In computational security it is almost never possible to show that a scheme satisfies a security definition unconditionally. The security definition is satisfied by a cryptographic construction based on some assumptions. In general a scheme is constructed using some basic primitives, for example, as stated, a MAC can be constructed using block ciphers. The primitives used to build the scheme are assumed to be secure in some sense, and the security of $\Xi$ is proved based on this assumption. The proof technique involves a *reduction* which shows that if the scheme is insecure then the primitive is also insecure. Thus a security theorem is not an absolute statement and needs to be interpreted carefully[1].

We give an example to a simple reductionist security proof. Consider a MAC $f : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$, we claim (informally) that $f$ is a secure MAC if $f$ is a PRF. We can formalize this claim as below.

**Proposition 2.1.** *Let $f : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$ be a message authentication code and $\mathcal{A}$ be an arbitrary adversary attacking $f$, then there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}_f^{\text{auth}}(\mathcal{A}) = \mathbf{Adv}_f^{\text{prf}}(\mathcal{B}) + \frac{1}{2^\tau} \tag{2.3}$$

*Proof.* It is enough to construct a PRF adversary $\mathcal{B}$ such that eq. (2.3) is satisfied. We construct $\mathcal{B}$ as in Figure 2.2

Note, as $\mathcal{B}$ is a PRF adversary for $f$ hence it has as its oracle either the function $f_K()$ for some $K \xleftarrow{\$} \mathcal{K}$ or is a uniform random function in $\mathsf{Func}(\mathcal{M}, \{0,1\}^\tau)$, in Figure 2.2 the oracle of $\mathcal{B}$ is depicted as $O$. Moreover, $\mathcal{B}$ uses $\mathcal{A}$, i.e., it provides the environment required for $\mathcal{A}$ by answering its queries. As the goal of $\mathcal{A}$ is to produce a forgery, it does so in line 4. Based on this forgery, $\mathcal{B}$ decides its output.

It is easy to see from the description of $\mathcal{B}$ that if the oracle $O$ of $\mathcal{B}$ is the real function $f_K$ then the probability that $\mathcal{B}$ outputs 1 is same as the probability that $\mathcal{A}$ commits a successful forgery, i.e.,

$$\Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{f_K(\cdot)} \Rightarrow 1\right] = \mathbf{Adv}_f^{\text{auth}}(\mathcal{A}). \tag{2.4}$$

[1] In the recent years there have been some criticisms to the paradigm of provable security and there have been proposals that given the relative nature of a security theorem and its proof they should not be called as security proofs but as reductionist arguments [38, 39].

**Adversary** $\mathcal{B}^{O}$
1. $Q \leftarrow \emptyset$
2. **While** $\mathcal{A}$ queries $x$, do the following:
2.1 $y \leftarrow O(x)$
2.2 $Q \leftarrow Q \cup \{x\}$
2.3 **return** $y$
3. **Until** $\mathcal{A}$ stops querying
4. $\mathcal{A}$ returns $(\tilde{x}, \tilde{t})$
5. **if** $\tilde{x} \notin Q$ and $O(\tilde{x}) = \tilde{t}$
6. **return** 1
7. **else return** 0

Figure 2.2: The adversary $\mathcal{B}$.

On the other hand,

$$\Pr\left[\eta \xleftarrow{\$} \mathsf{Func}(\mathcal{M}, \{0,1\}^{\tau}) : \mathcal{B}^{f_K(\cdot)} \Rightarrow 1\right] = \frac{1}{2^{\tau}}. \tag{2.5}$$

As, if the oracle of $\mathcal{B}$ is a random function then the point $\tilde{x}$ can get mapped to any of the $2^{\tau}$ points in the range. Hence subtracting equation (2.5) from equation (2.4) and using the definition of PRF advantage of $\mathcal{B}$ (as in eq. 2.1), we have the proposition. □

Proposition 2.1 is a standard form of a reductionist security statement, where the security of a scheme is related to the security of something else. We formally study all the schemes that are proposed in this thesis and we state formal security theorems for our schemes. Our security theorems have the same structure as that of Proposition 2.1.

## 2.5 Some Other Cryptographic Primitives

In this section we consider some cryptographic primitives that we do not use explicitly in our constructions. However, some of the schemes that we introduce as part of the state of the art in the Chapter 3, make use of them. Therefore, we discuss briefly hash functions and signature schemes.

**Hash Function**: A cryptographic hash function is used to construct a short "fingerprint" of some message; if the message is altered, then the fingerprint will no longer be valid. For example, if the data is stored in an insecure place, its integrity can be checked from time to time by recomputing the fingerprint and verifying that the fingerprint has

not changed. Let $\mathcal{H}$ be a hash function and let $x$ be some message, where $x$ could be a binary string of arbitrary length. The corresponding fingerprint is defined as $y = \mathcal{H}(x)$ this fingerprint its know as *message digest* which could be short as 160 bits. Let say that $y$ is stored in a secure place but $x$ is not. If $x$ is changed to $x'$, then the old message digest, $y$, is not also a message digest for $x'$. Then the fact that $x$ has been altered can be detected simply by computing the message digest $y' = \mathcal{H}(x')$ and verifying that $y' \neq y$.

For the purposes of the schemes that we present in the next chapter, it is required that hash functions are *Collision resistance*. This property states that it is computationally infeasible to find two inputs, $x_1 \neq x_2$, such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$.

**Signature Scheme**: Digital signatures are the public-key counterpart of message authentication codes. The algorithm that the sender applies to a message is denoted Sign $\mathcal{S}$, and the output of this algorithm is called *signature*. The algorithm that the receiver applies to a message and a signature in order to verify legitimacy is denoted *Vf*. A signature scheme is a tuple of three polynomial-time algorithms $(Gen, \mathcal{S}, Vf)$ satisfying the following:

1. The key-generation algorithm *Gen* takes as input a security parameter $1^n$ and outputs a pair of keys $(pk, sk)$. These are called the public key and the private key, respectively.

2. The signing algorithm $\mathcal{S}$ takes as input a private key $sk$ and a message $m \in \{0,1\}^*$. It outputs a signature $\vartheta$ denoted as $\vartheta \leftarrow \mathcal{S}_{sk}(m)$.

3. The deterministic verification algorithm *Vf* takes as input a public key $pk$, a message $m$, and a signature $\vartheta$. It outputs a bit $b$, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := Vf_{pk}(m, \vartheta)$.

## 2.6 Relational Databases

In this section we describe some basic concepts of a relational database [40] and fix some notations which we would use later.

### 2.6.1 Relational Databases: Basic Structure and Notations

A relational database is a collection of data with an specific structure that stores information about the data itself and its relations. In such a database the data is repre-

sented as two-dimensional tables called *relations*. For example, consider an organization which stores details regarding its employees, such a relation is shown in Table 2.1.

Each database is a collection of related tables/relations. Each table is a physical representation of an entity or object that is in a tabular format consisting of columns and rows. Columns are the fields of a record or the *attributes* of an entity. Attributes can be seen as descriptive properties possessed by each member of an entity. Each attribute of a table has an unique name. The rows contain the values or data instances; these are also called records or tuples. For example, in the example in Table 2.1 each row (tuple) represents an employee and the columns are the specific attributes of the employee, specifically its identification number, name, gender, level in the organization and age.

We will denote by $R(A)$ a relation defined over a set of attributes $A$. If $A = \{a_1, a_2, \cdots a_n\}$, we will sometimes write $R(a_1, a_2, \cdots, a_n)$ instead of $R(A)$. Given an attribute $a$, $\mathsf{Dom}_R(a)$ represents the set of values of $A$ present in the relation $R$. We will call $\mathsf{Dom}_R(a)$ as the domain of $a$ in $R$, and when the relation $R$ is clear from the context, we will sometimes drop the subscript $R$ [2]. By our definition, for any relation $R$ and an attribute $a$, $\mathsf{Dom}_R(a)$ is always finite. If $R$, is static then $\mathsf{Dom}_R(a)$ is fixed, but when $R$ is dynamic, i.e., updates and insertions are allowed on $R$, then $\mathsf{Dom}_R(a)$ may change with each insert/update. By $\mathsf{Dom}_R(a)$ we will always mean the distinct values taken by the attribute $a$ in the current state of the relation $R$. By cardinality of an attribute we shall mean the cardinality of the domain of the attribute. We will denote the cardinality of an attribute $a$ by $\mathsf{Card}_R(a) = \#\mathsf{Dom}_R(a)$. Note, that for a dynamic relation $R$, $\mathsf{Card}_R(a)$ can also change with time.

A tuple $t$ in a relation over a set of attributes is a function that associates with each attribute a value in its specific domain. Specifically if $A = \{a_1, a_2, \cdots a_n\}$ and $R(A)$ be a relation then the $j^{th}$ tuple of relation $R(A)$ would be denoted by $t_j^R$ and for $a_i \in A$ by $t_j^R[a_i]$ we shall denote the value of attribute $a_i$ in the $j^{th}$ tuple in $R$. E.g the associated value with $t_3^R[Gender] = \ 'M'$. For $B \subseteq A$, $t_j^R[B]$ will denote the set of values of the attributes in $B$ in the $j^{th}$ tuple. We sometimes omit the subscripts and superscripts from $t_j^R$ and denote the tuple by $t$ if the concerned relation is clear from the context and the tuple number is irrelevant.

[2] In the literature the domain of an attribute is sometimes used to denote the set of permitted values of an attribute. For example, for the relation shown in Table 2.1 the domain of the attribute `Age` would be defined by the set of all integers between say 18 and 100. But, according to our definition $\mathsf{Domain}(\texttt{Age}) = \{17, 18, 33, 36, 52\}$

**Data Manipulation**

It is possible to apply different operations or transactions like *create, modify, retrieve, delete* to a database. Each of these transactions can be described using a query language. One of the most popular query languages for data manipulation is the *Structured Query Language* (SQL). We discuss the SQL syntax for some basic data manipulation below.

The *insert* operation is used to populate a table with rows. The basic syntax is:

`INSERT INTO <relation> VALUES (`$v_1$`, `$v_2$`, ⋯,`$v_l$`);`

| EmpId | Name | Gender | Level | Age |
|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 18 |
| MST | Mary | F | $L_1$ | 17 |
| JOH | John | M | $L_2$ | 52 |
| MRH | Mary | F | $L_1$ | 33 |
| ASY | Anne | F | $L_1$ | 18 |
| RZT | Rosy | F | $L_2$ | 36 |

Table 2.1: Relation Employees

The clause `<relation>` implies in which table the rows will be inserted, like `INSERT INTO Employees VALUES ('TRW','Tom','M','`$L_2$`',18)`.

The modification of data that is already in the database is referred to as updating. It is possible to update individual rows, or a subset of all rows. Each column can be updated separately, in that case the other columns are not affected. The basic syntax for this operation is:

```
UPDATE <relation> SET <attribute> = <value> WHERE <conditions>
```

It is necessary to specify the row number where the update operation should be applied, or specify which conditions a row must meet in order to be updated. For instance, consider that the client wants to update the age of Anne (tuple 5 of Table 2.1) from 18 to 19. Therefore, the SQL statement looks as follows.

```
UPDATE Employees SET age=19 WHERE EmpId='ASY'
```

We have explained how to add data to tables and how to change data. What remains is to discuss how to remove data that is no longer needed. As in an update statement, to remove rows, we need to specify the conditions that such rows have to meet. The respective syntax is:

```
DELETE FROM <relation> WHERE <conditions>
```

Going back to our example, consider that the client wants to remove the tuple corresponding to John, then the following statement has to be posed.

```
DELETE FROM Employees WHERE EmpId='JOH'
```

**Retrieving data**

The `SELECT` statement is used to retrieve data from a table. In order to explain the syntax of this operation, we use the following general query form:

```
SELECT <attributes>
FROM <relations>
WHERE <predicates>
```

The basic structure of an SQL expression consists of three clauses: `SELECT`, `FROM`, and `WHERE`. The `SELECT` clause is used to list the attributes desired in the result of a query. The `FROM` clause lists the relations to be scanned in the evaluation of the expression. The `WHERE` clause corresponds to predicates which describe relations between the above relations (*joins* [3], *outerjoins* , etc). In addition, the `WHERE` clause allows to declare restrictions, i.e. *Select all the employees names who are female*. The SQL statement for this query looks like:

```
SELECT Name
FROM Employees
WHERE Gender='F'
```

We only discussed some basic syntax of the SQL which we will use in this thesis. Needless to say, that SQL is a powerful query language which includes syntax for more complicated database operations.

### 2.6.2 Bitmaps and their Encodings

Traditionally a database has been indexed with B-trees or its variants, in the current days the Bitmap indices have gained lot of popularity for their use in accelerated query processing [7, 8], and many commercially available databases like Oracle, IBM DB2, Sybase IQ now implement some form of bitmap index scheme in addition to the more

[3]Join it is a relational database operation which selects rows from two tables such that the value in one column of the first table also appears in certain column of the second table.

traditional schemes. In this section we discuss the basic concept of bitmaps, the different encodings and how the bitmaps can be stored efficiently.

The main idea behind a bitmap index is to use a bit string to describe if the value of an attribute is equal to an specific value or not. The position of the bit denotes the tuple number in the relation. This concept is formalized next.

Consider a relation $R(a_1, \ldots, a_m)$ with $\mathsf{nT}$ many rows. Consider that for each attribute $a_i$, $\mathsf{Dom}_R(a_i) = \{v_1^i, v_2^i, \ldots v_{\lambda_i}^i\}$, thus $\mathsf{Card}_R(a_i) = \lambda_i$ for $1 \leq i \leq m$. We define the bitmap of an attribute $a_i$ corresponding to its value $v_j^i$ in the relation $R$ as $\mathsf{BitMap}_R(a_i, v_j^i) = X$, where $X$ is a binary string, such that $|X| = \mathsf{nT}$ and for $1 \leq k \leq \mathsf{nT}$,

$$\mathsf{bit}_k(X) = \begin{cases} 1 & \text{if } t_k^R[a_i] = v_j^i \\ 0 & \text{otherwise.} \end{cases}$$

The encoding for the previous definition is known as equality encoding, we shall call such bitmaps as e-encoded bitmaps. This would be more clear with an example. Consider the specific relation $R1$ on the attributes $\{\mathtt{EmpID}, \mathtt{Name}, \mathtt{Gender}, \mathtt{Level}, \mathtt{Age}\}$ as shown in Table 2.1. $\mathsf{Dom}(\mathtt{Gender}) = \{\mathrm{M}, \mathrm{F}\}$ and $\mathsf{Dom}(\mathtt{Level}) = \{L_1, L_2\}$.

From this relation we can compute the following bitmaps

$$\begin{aligned} &\mathsf{BitMap}_{R1}(\mathtt{Gender}, \mathrm{F}) = 010111 \\ &\mathsf{BitMap}_{R1}(\mathtt{Gender}, \mathrm{M}) = 101000 \\ &\mathsf{BitMap}_{R1}(\mathtt{Level}, L_1) = 010110 \\ &\mathsf{BitMap}_{R1}(\mathtt{Level}, L_2) = 101001. \end{aligned}$$

In the literature there are other kinds of bitmap encodings to allow different kinds of queries. We would be interested in a specific encoding called range encoding [41]. We define two types of range bitmaps $\mathsf{BitMap}^{<}$ and $\mathsf{BitMap}^{>}$ which we will further call as l-encoded and g-encoded bitmaps respectively. For the relation $R$ described above, we have $\mathsf{BitMap}_R^{\leq}(a_i, v_j^i) = Y$, where

$$\mathsf{bit}_k(Y) = \begin{cases} 1 & \text{if } t_k^R[a_i] < v_j^i \\ 0 & \text{otherwise.} \end{cases} .$$

From the l-encoding and e-encoding other bitmap encodings can be easily derived as follows:

$$\begin{aligned} \mathsf{BitMap}_R^{\geq}(a_i, v_j^i) &= \overline{\mathsf{BitMap}_R^{\leq}(a_i, v_j^i)} \bigoplus \mathsf{BitMap}_R(a_i, v_j^i) \\ \mathsf{BitMap}_R^{\leq}(a_i, v_j^i) &= \mathsf{BitMap}_R^{<}(a_i, v_j^i) \bigvee \mathsf{BitMap}_R(a_i, v_j^i) \\ \mathsf{BitMap}_R^{\geq}(a_i, v_j^i) &= \overline{\mathsf{BitMap}_R^{<}(a_i, v_j^i)} \end{aligned}$$

| l-encoded bitmaps | g-encoded bitmaps | e-encoded bitmaps |
|---|---|---|
| $\mathsf{BitMap}^{\leq}_{R1}(\texttt{Age}, 17) = 000000$ | $\mathsf{BitMap}^{\geq}_{R1}(\texttt{Age}, 17) = 101111$ | $\mathsf{BitMap}_{R1}(\texttt{Age}, 17) = 010000$ |
| $\mathsf{BitMap}^{\leq}_{R1}(\texttt{Age}, 18) = 010000$ | $\mathsf{BitMap}^{\geq}_{R1}(\texttt{Age}, 18) = 001101$ | $\mathsf{BitMap}_{R1}(\texttt{Age}, 18) = 100010$ |
| $\mathsf{BitMap}^{\leq}_{R1}(\texttt{Age}, 33) = 110010$ | $\mathsf{BitMap}^{\geq}_{R1}(\texttt{Age}, 33) = 001001$ | $\mathsf{BitMap}_{R1}(\texttt{Age}, 33) = 000100$ |
| $\mathsf{BitMap}^{\leq}_{R1}(\texttt{Age}, 36) = 110110$ | $\mathsf{BitMap}^{\geq}_{R1}(\texttt{Age}, 36) = 001000$ | $\mathsf{BitMap}_{R1}(\texttt{Age}, 36) = 000001$ |
| $\mathsf{BitMap}^{\leq}_{R1}(\texttt{Age}, 52) = 110111$ | $\mathsf{BitMap}^{\geq}_{R1}(\texttt{Age}, 52) = 000000$ | $\mathsf{BitMap}_{R1}(\texttt{Age}, 52) = 001000$ |

Table 2.2: Example of various bitmap encodings for the values of the attribute Age in the relation $R1$

The various encodings would be more clear with an example. Consider the specific relation $R1$ on the attributes $\{\texttt{EmpID}, \texttt{Name}, \texttt{Gender}, \texttt{Level}, \texttt{Age}\}$ as shown in Table 2.1, where $\mathsf{Dom}(\texttt{Age}) = \{18, 17, 52, 33, 36\}$. Three different bitmap encodings for all the values of the attribute Age are shown in Table 2.2.

### 2.6.3 Bitmap Compression

There had been a lot of work that has shown that bitmap based indexing works effectively in database applications. To further improve their effectiveness, compression schemes have been developed, these schemes are capable of reducing the index size without increasing query processing time. The most frequent operations over bitmaps are bitwise logical operations [42], the specific compression schemes developed for bitmaps also allows logical operations to be performed on the compressed bitmaps.

Most of the compression schemes applied to bitmaps are based on run length encoding (RLE) scheme. Basic RLE works on the basis of the following simple idea. Consecutive occurrences of identical bits are detected in a bit string, such occurrences are known as a *fill*. Each fill in a bit string can be recorded with a counter representing its length and one bit indicating the actual value. This compression scheme is lossless and very efficient.

Several variants of RLE has been developed for application to bitmap compression. In our implementations we use a specific scheme called Enhanced Word Aligned Hybrid (EWAH). EWAH was studied independently by Wu et al. [42] and Lemire et al. [43].

#### Enhanced Word Aligned Hybrid (EWAH) scheme

EWAH divides the whole bit string $X$ into 32 bit blocks and classify each block as either a *clean word* or a *dirty word*. A clean word is a 32 bit fill (either of zeros or ones), a word

which is not *clean* is called *dirty*. The basic idea is to encode in such a manner that the clean words are compressed by specifying the type of fill contained in the word and its length; and the dirty words occur verbatim in the encoded string. We explain the basic encoding procedure with an example in Figure 2.3. In the example, the original bit string is shown in the beginning followed by the compressed string. The original string is 224 bits long and is represented in hexadecimal. As we can see that in the input string the first two blocks are dirty words followed by four clean words each with a fill of zeros and the last word is a dirty word. The encoded string contains two types of 32 bit words namely *marked words* and *verbatim words*. Marked words are sort of headers which carries information regarding the length and positions of fills and verbatim words are verbatim copies of dirty words. The first bit of each marked word represents the type of clean word that is to follow, the next 16 bits of a marked word encodes the length (in words) of the clean words and the final 15 bits encodes the number of dirty words that follows the clean words.

In our example the first word of the encoded string is a marked word (this is so, for all strings) `0x00 00 00 02`, which means that the marked word is followed by no clean words but two dirty words. Next the following two dirty words are copied verbatim. Next the marked word `0x00040001` occurs, which means that this marked word would be followed by four clean words each of zero which would be followed by one dirty word. At the end of this marked word the dirty word is written verbatim. This procedure can be easily generalized.

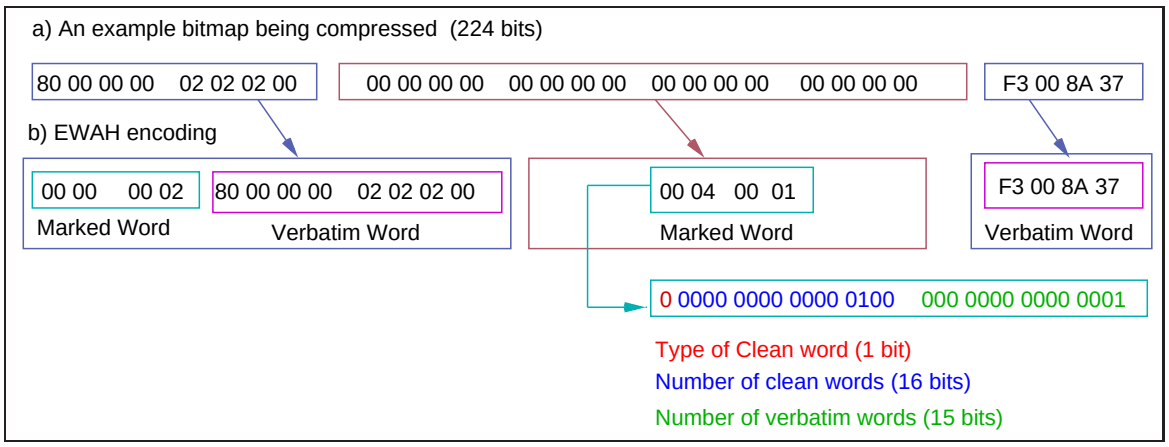



Figure 2.3: EWAH compression example.

Also, it is not difficult to see that as the EWAH works with a granularity of 32 bits, hence bitwise operations can be easily performed on the compressed words and the compressed results can be directly obtained. In [43] one can find the algorithms to

perform logical operations on RLE encoded bitmaps. These procedures can be easily extended for EWAH compression scheme.

## 2.7 Summary

In this chapter we discussed some preliminary concepts related to symmetric key cryptography and databases. Regarding cryptography, we introduced symmetric encryption schemes, block ciphers and and their modes of operations, and messages authentication codes. We also introduced the technique of proofs by reduction, that is largely used in our work to prove the security of our schemes.

About databases we described the basic structure of a database and fixed some useful notation. We also mentioned the basic operations that our schemes manage and their SQL syntax. Finally, we introduced bitmap indexes, which will form the main non cryptographic tool for the schemes that we develop later.

# Chapter 3

# The State of the Art

> Security is mostly a superstition. It does not exist in nature... Life is either a daring adventure or nothing.
>
> *Helen Keller*

Now-a-days there is a diversity of storage tools oriented to cloud computing. This variety has generated the necessity of an extended security study. Also, an intensive development of new cryptographic schemes, which provide various kind of security to these tools, has taken place. These set of schemes aim to achieve a specific security goal, for example, privacy, integrity, etc. However, most of these problems are widely open and ideal solution to them is yet to come. In this chapter we summarize the most relevant works related to privacy and integrity of outsourced databases.

## 3.1 Database As A Service Model

A DAS model is typically composed of three entities, namely, the data-owner, one or more data-users (can be same as the owner), usually called clients, and a server (see Figure 3.1). The owner stores the data on the server and the clients may query parts of this data. In DAS applications, the server is consider un-trusted. On the other hand, the client-side is considered to be secure. Therefore the main adversary is considered to be the server.

In such a scenario the data may face different threats, among others, modification, deletion or disclosure. The security model can consider two basic scenarios: a) the server is an active adversary, i.e, the server can modify or delete some of the stored information. Integrity checking becomes important in this scenario, specifically a problem of this class in databases is known as query authentication. b) the passive or curious


Figure 3.1: Database as a Service Model.

adversary, in which the server is only interested in getting information about the data that it is storing, i.e, breaking the confidentiality.

In the next sections we introduce these two problems in detail and we present how it has been addressed in the literature.

## 3.2 Query Authentication Problem

We consider the scenario where a client delegates a relational database to an un-trusted server. When the client queries its outsourced data, it expects in return a set of records (query reply) satisfying the query's predicates. As the server is not trusted, so it must be capable of proving the correctness of its responses. In other words, a malicious server may attempt to insert fake records into the database, modify existing records or simply skip some of them from the query response. Hence there must exists a mechanism, which can protect the client from such malicious server behavior. We describe the problem with the help of an example.

Consider that the relation depicted in Table 3.1 has been delegated by a client to a server, and the client poses the following query

$$\texttt{SELECT * FROM } R1 \texttt{ WHERE } Gender = \texttt{'}M\texttt{' OR } Level = \texttt{'}L_2\texttt{'}.$$

The correct response to this query is the set Res consisting of three tuples

$$\text{Res} = \{(\text{TRW, Tom, M}, L_2, 18), (\text{JOH, John, M}, L_2, 52), (\text{RZT, Rosy, F}, L_2, 36)\}.$$

In answering the query the server can act maliciously in various ways. In the context of authentication we are concerned with two properties of the response namely *correctness* and *completeness*. Correctness and completeness denote two different malicious activities of the server, we explain these notions with an example below:

1. **Incorrect result:** The server responds with three tuples, but changes the tuple (TRW, Tom, M, $L_2$, 18), to (TRW, Tom, F, $L_2$, 18). Moreover, it can be the case that the server responds with Res $\cup$ {(BRW, Bob, M, $L_2$, 26), i.e., it responds with an extra tuple which is not a part of the original relation.

2. **Incomplete result:** The server may not respond with the complete result, i.e., it can delete some valid results from the response, which means that instead of responding with Res it responds with Res $-$ {(TRW, Tom, M, $L_2$, 18)}.

| EmpId | Name | Gender | Level | Age |
|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 18 |
| MST | Mary | F | $L_1$ | 17 |
| JOH | John | M | $L_2$ | 52 |
| MRH | Mary | F | $L_1$ | 33 |
| ASY | Anne | F | $L_1$ | 18 |
| RZT | Rosy | F | $L_2$ | 36 |

Table 3.1: Relation Employees

It is to be noted that incomplete results are also incorrect, but we differentiate the two scenarios (as it has been previously done in the literature) by the fact that in an incorrect result the server inserts something which is not in the database, and in case of an incomplete answer the answer is correct but is not complete, in the sense that the server drops some valid tuples from the correct response. A client must be able to verify both correctness and completeness of a response.

The problem of correctness can be easily handled in the symmetric setting by adding a message authentication code to each tuple. In contrast, public key cryptography provides correctness by adding signatures. Both, the secure message authentication code or the secure signatures are difficult to forge, and thus this property would not allow the server to add fake entries in its response. However, digital signatures introduce significant overhead in terms of storage, bandwidth and computation. The completeness problem is more difficult and its solution is achieved through more involved schemes.

At the high level, these schemes achieve completeness by using the following strategy: the data owner not only outsources the database but also an additional data structure that contains information that helps to verify the completeness of the query results. This structure is popularly called an Authenticated Data Structure (ADS). After a client poses a query, the server looks for the results and computes the additional data necessary for proving correctness and completeness. This additional data is referred to as

the Verification Object (VO), and it is sent to the client along with the usual response. Using these information the client can verify if the response is correct and complete.

There is a number of important costs pertaining to this model, relating to the construction, and query phases. Among others, the most relevant metrics are:

1. The computation overhead for the owner, i.e, how many operations the owner has to perform in order to outsource the data.
2. The communication cost, which generally involves the size of the VO.
3. The storage overhead for the server, i.e, the amount of extra storage required in the server besides the one corresponding to the database itself.
4. The computation overhead for the server, i.e, how many operations the server has to perform to answer a query.
5. The computation cost for the client, i.e, the number of operations involved in the verification process.

The analysis of these costs helps to compare the different schemes.

In general the costs associated with the client have greater priority than the ones associated with the server. For instance, a scheme which reduces the number of operations in the verification process is preferred to a scheme that is expensive in this cost, but very cheap in the response procedure performed by the server. Also the recurring costs are more important than the one-time costs, for example the query process is more important than the initial cost for the data owner to create the data structures.

According to Bajaj et. al. [44], the existing solutions for the problem of authenticated query processing can be classified as *based on authenticated data structures (ADS)* or *based on signatures*. There are some schemes which use both. The former builds the ADS as a tree (for example, a Merkley Hash Tree, B-Tree, R-Tree etc). As part of query execution, the server traverses the tree and obtains the respective nodes to build the VO, which is sent to the client along with the query results. The client can then reconstruct the traversal path used in the query execution and verify. Signature based approaches provide a mechanism to verify the ordering between tuples. The owner builds a signature chain over the tuples. At query time, the server obtains the corresponding signatures of all the tuples that comprise the contiguous range query result (VO). Since each tuple is linked with its predecessor, sometimes also with the successor, the client can verify the completeness of the result. In the next section we explore briefly the literature in this regard.

## 3.3 Some Existing Schemes for Authenticated Query Processing

In Table 3.2 we summarize some of the schemes that aim to solve the authenticated query processing problem. In general, each of them intends to improve their predecessor in some aspect, for instance in VO size, communication cost, the types of queries that it is able to manage, or the facility to be implemented.

As we have mentioned, the main two strategies to solve authenticated query problem are: Tree approach and Signature approach. We analyze first, the works that fall in the first category. Devanbu et.al in [45, 46] applied for the first time the seminal work of Merkle in [47] to this problem. In this work the Merkle Tree is proposed as the ADS. The idea behind this approach is to have a tree whose root is signed by the data owner. In response to a query posed by the client, the server sends besides the query results the set of nodes required to reconstruct the root node. The client verifies the signature of the root, and if it verifies, then the client is satisfied that the query result is both correct and complete. This idea was extended to trees with bigger fan-out, like $B^+$-tree; the first scheme that deployed this kind of tree was by Pang et al. [48]. In this work an authenticated $B^+$-tree is proposed. In [49] another authenticated structure was proposed which also uses a $B^+$-tree but here each node of the $B^+$ tree encapsulates a Merkley tree. This structure is called embedded Merkle $B^+$-tree. This ADS provides better tradeoff between the VO size and the number of operations required to build it. Later we discuss both the Merkle tree and Merkle $B^+$-tree in a bit more detail.

Some other interesting works have been proposed for join processing [50, 51]. The straightforward manner to manage joins is to materialize the cartesian product over the relations and construct the authenticated structure over it. However, this is inefficient and costly. Pang et. al. in [50] extend their scheme for range queries to *primary key- foreign key joins*, e.g., consider a relation Employees $R_1$ (in Table 3.1) related with a second relation called Departments $R_2$. Each employee works in an specific department, e.g. informatics, human resources, etc. The data owner constructs the authenticated structure using the two tables over the attribute in common (id-department). The server answers the query with respect to the smallest relation (let us assume that is $R_2$) and completeness is checked. Then a new query is posed over the second relation to find the matching tuples in $R_1$. The first comprehensive work on authenticated queries for joins can be found in [51]. The authors propose three joins algorithms:

1. Authenticated Indexed Sort Merge Join (AISM), which utilizes a single authenticated structure on the join attribute.

2. Authenticated Index Merge Join (AIM), that requires an Authenticated structure on the join attribute for both relations, and

3. Authenticated Sort Merge Join ASM, which does not rely on any authenticated structure.

The last method is the most efficient solution proposed in the literature for authenticated join processing.

A few solutions have also studied the authentication problem for aggregated queries [52, 53], e.g (maximum MAX, minimum MIN, average AVG, etc). Pang et. al. in [52] provide a solution of aggregate range queries, in order to do so the server needs to maintain a hierarchy of partial sums over the records. Feifei et.al. in [53] suggest a new structure for aggregated queries called Authenticated Aggregation B-Tree (AABT). In a AABT each node stores the aggregated sum of its child nodes on the value of the search attribute.

| Name | Year | Operations |
|---|---|---|
| Tree Approach | | |
| MHT [45] [46] | 2003 | Range queries |
| VBT [48] | 2004 | Range queries |
| HAT [54] | 2005 | Range queries |
| EMBT [49] | 2006 | Range queries |
| Goodrich et. al. [55] | 2008 | Range queries |
| MR-tree [56] | 2008 | Range queries |
| PMD [57] | 2009 | Range quieres |
| AIM [51] | 2009 | Range queries, joins |
| AABT [53] | 2010 | Aggregation |
| Signature Approach | | |
| DSAC [58] | 2005 | Range queries |
| Pang et.al. [50] | 2005 | Range queries,joins |
| Pang et.al. [52] | 2008 | Range queries, aggregation |
| Pang et.al. [59] | 2009 | Range queries, joins, aggregation |
| Other Techniques | | |
| Query Racing [60] | 2010 | Range and logical queries |

Table 3.2: Summary of existing approaches

Now, let us discuss the schemes that fall in the second category. The first approaches that use signatures only can achieve correctness, but not completeness. The basic idea

was to sign each tuple before outsourcing the data. Narasimha et. al. in [58] provide the first scheme based only in signatures that achieves completeness for query authentication. The basic idea behind this scheme consists of computing the signature of each individual tuple along with its immediate predecessors, called chain. The client verifies that the set of tuples received in the result do form a valid chain. Independently Pang et. al. also proposed a scheme based in signature chains [50]. These schemes can use aggregated signatures, thereby resulting in a small, constant sized VO. In the next sections, we explore in more detail some of the schemes that we just mentioned.

### 3.3.1 Approaches Based on Authenticated Data Structures

**A Basic Scheme using Merkle Hash Tree:** The basic approach to accomplish completeness over a set of $n$ values $r_1, r_2, \cdots, r_n$ is the Merkle Hash Tree (MHT), see Figure 3.2. This data structure was first proposed by Merkle in [47]. In order to use it in authenticated query processing, the data owner must delegate not only the data itself, but also this tree.

A MHT is built as a binary tree, where each leaf stores the hash of a data value in the set. For example, in the tree depicted in Figure 3.2 the first leaf is associated with $r_1$ and the node contains the $\mathcal{H}(r_1)$. In contrast, each internal node contains the hash of the concatenation of the content of its two children, for example the right children of the root contains $\mathcal{H}(h_3||h_4)$. The root of the tree is authenticated with a digital signature.

This data structure can be used to verify any of the data values, all the server (prover) has to do is to provide the verifier (data-user) with the verification object (VO). The VO normally contains the data values that are going to be authenticated along with the values stored in the siblings of the path that leads from the root of the tree to the values. This information allows the verifier to re-build the root node and finally compare the hash of the computed root with the publicly authenticated value of the signature.

For example, let us consider a query whose correct response is the leaf node related with the value $r_2$, then the verification object is composed of the values $r_2$, $h_1$, $h_{3-4}$ and the signature value $\vartheta_{tree}$. Having these values the data-user computes iteratively: $h_2 = \mathcal{H}(r_2)$, $h_{1-2} = \mathcal{H}(h_1||h_2)$, and finally the root $\mathcal{H}(h_{1-2}||h_{3-4})$. Now, the data-user can check if the hash that it has computed for the root matches with the authentic published value.

Several notable works exploit this basic approach, in this regard these works varying the tree data structure (B-trees, R-trees, etc) and add some cryptographic authentication mechanism like hash functions and/or signatures schemes. Next we discuss how

to extend this ideas to a B-tree.

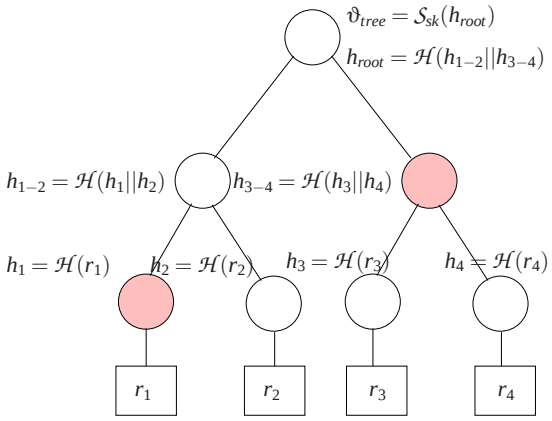



Figure 3.2: Merkeley hash tree

**The Merkle B-tree:** Devanbu, et al. [45] focus on Third-Party Publication. In this setting, the data owner produces some content that later will be published by a third party service. This work introduces the concept of how to efficiently build verification objects. Also, it presents the idea that B-trees can be used with the same techniques, which are more efficient data structures than binary trees.

The Merkle tree uses a hierarchy of hashes organized as a binary tree. Of course, this scheme can use a tree with a bigger fan-out. According to the description of the scheme in [49], a Merkle B-tree works like a $B^+-$ tree and consists of common nodes that are extended with one hash value associated with every pointer entry. The leaf nodes are associated with the hashes of the tuples. The internal nodes are related with the hashes of the concatenation of the hash values of their children. Finally, the root is publicly signed.

As before, to answer a range query the server sends the tuples that hold with the query, along with the hash values of the residual entries to the left and to the right parts of the boundary leaves. The result is also increased with one tuple to the left and one to the right of the lower-bound and upper-bound of the query result respectively, for completeness verification. Finally, the signed root of the tree is inserted as well. Consider a query which result set is composed by records $r_3$ and $r_4$, then in the VO also includes tuples $r_2$ and $r_5$, along with the $h_1, h_6$. Having this information it is possible to build the root and finally compute the signature and compare with the authentic published value.

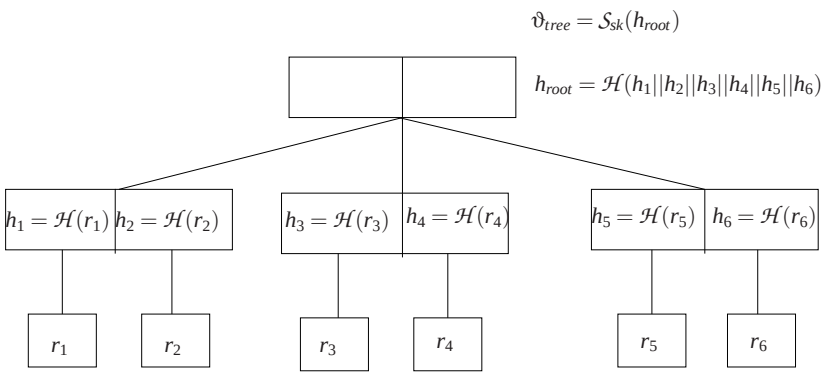



Figure 3.3: B-tree

### 3.3.2 Approaches Based on Signatures

Signature schemes have also been used in a novel manner for solution of the query authetication problem. One line of research has focussed on aggregated signatures [50, 58, 59, 61, 62]. Signature aggregation helps in reducing the communication cost to some extent and in some cases can function with constant extra communication overhead. A related line of research uses chain signatures. If one uses chain signatures as in [58], the use of specialized data structures may no longer be required. In this section we illustrate two schemes that exemplify these two lines of research.

**Aggregated Signatures with B+ trees:** This technique is used in [50]. The main idea behind this scheme is to organize the data in consecutive pairs of tuples, assuming an order on an attribute $a_i$. Then each pair is hashed and signed. Consider tuples $r_1, r_2$ in Figure 3.4, they should be hashed and signed together $\mathcal{S}_{sk}(\mathcal{H}(r_1||r_2))$. Thus the data-owner delegates the pair and also the respective signature. It is necessary to add two extra pairs composed by: an special marker with the first tuple $(\infty, r1)$ and the last tuple with the special marker $(r_n, \infty)$. A $B^+ - tree$ is constructed over attribute $a_i$. When the untrusted server answers a query, also it constructs the verification object that is sent as part of the response, this object is built as follows: a) One tuple-signature pair per query result $(r_i, \vartheta_i)$, b) the left lower-bound and the right upper bound of the result data set. Thus, the client can check completeness of the response.

This will be clearer with an example, consider a query whose result data set is composed of tuples $r_2, r_3$ then also the lower and upper bounds are part of the verification object, i.e $r_1, r_4$, along with the respective signatures pairs $(\vartheta_{1-2}, \vartheta_{2-3}, \vartheta_{3-4})$. Thus, the verifier (data-user) computes the respective hashes and verifies the respective signatures sent by the server.

It is clear from the description that the verification object contains a linear number of signatures respective to the tuples in the result set, which is a drawback. Also the verifier needs to verify the same number of signatures to check completeness. Thus, this verification become computationally expensive. Therefore a possible improvement to this scheme is to use aggregate signatures. Where the idea is to merge several signatures into a "short" string. This small change, will give us a big advantage that is the small size of the verification object. However, it still remains a computationally expensive scheme given the high initial cost of the number of signatures that needs to be computed.

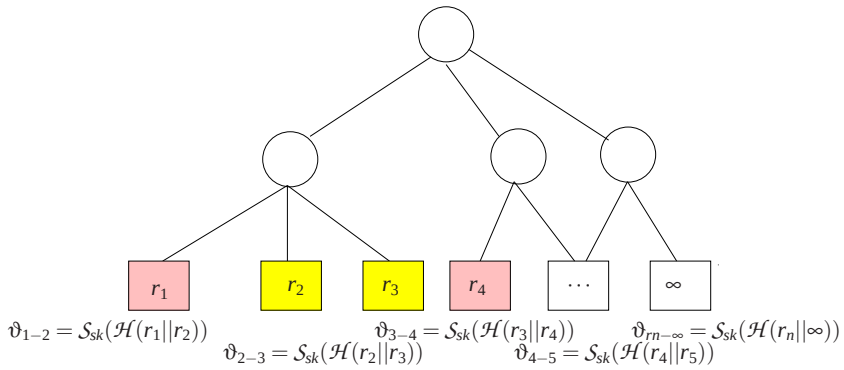



Figure 3.4: $B^+-$ tree

**Signature Aggregation and Chaining:** The main characteristic of this approach is to achieve completeness without using any specialized data structure. To achieve this goal, Narasimha et. al. in [58]; introduce the idea of *signature chain*. In order to construct the signature chains, the data-owner should order the tuples by each of the attributes that are of interest to a query. A tuple signature is computed including the hashes of all immediate predecessor tuples. Consider the example in Figure 3.5 in which there are three attributes that are going to be queried $a_1, a_2, a_3$. First, the tuples are sorted according to each attribute of interest. Figure 3.5 shows the predecessor and successor of the tuple $r_5$ based on the values of the three attributes $a_1, a_2$ and $a_3$. The tuple $r_5$ has as immediate predecessors $r_6, r_2, r_7$ based on the attributes $a_1, a_2, a_3$. Thus, the signature of $r_5$ is calculated as: $\vartheta_{r_5} = S_{sk}(\mathcal{H}(\mathcal{H}(r_5)||\mathcal{H}(r_6)||\mathcal{H}(r_2)||\mathcal{H}(r_7)))$

As in the previous scheme the untrusted server answers a range query by including all matching tuples, the lower and upper boundary tuples, along with the aggregated signature corresponding to the result set. The data-user verifies completeness by verifying the signature chain and checking that the values in the boundary are just beyond the query range.

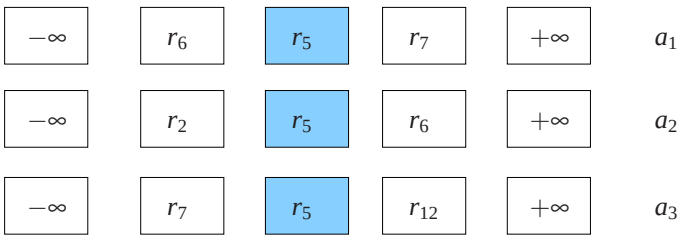



Figure 3.5: Signature chain

### 3.3.3 Discussion and Comparisons

We have presented the principal approaches to solve the authenticated query processing problem. In this section we want to point out some of the advantages and disadvantages that these schemes have.

In general, one limitation of the schemes which uses trees as authenticated data structure have, is the need to pre-compute and store a potentially large number of trees, since according to Narasimha [58] without pre-computed trees for each sort-order, it becomes impossible to prove completeness of query replies. This requirement imposes significant setup costs for the data owner, and high storage overhead for the server. Another disadvantage of trees is the difficulty to be efficiently combined to answer queries over multiple set of attributes.

On the other hand, the signature schemes also impose a large storage overhead on the servers, because in general terms signature sizes are bigger than the ones for hashes or MACs sizes. Also it has a very high initial cost, since a considerable number of signatures needs to be computed, typically the number of tuples in the relation. If a scheme that supports aggregation of signatures is used, then the obvious advantage is the small verification object sizes. However, more computations need to be performed by the server to compute the aggregate signature. The other costs remain the same.

Another common characteristic of all the schemes in the literature is that the query completeness problem is largely addressed with respect to range queries on numerical attributes.

## 3.4 The Problem of Privacy in Outsourced Databases

Another important security issue besides authentication is the *privacy* of the outsourced data. The client who owns the data may not want to reveal it to the server, this objective

can be attained by encrypting the data.

Encryption, if applied to relational databases, should be done in such a way that a large class of queries can be executed on the encrypted tables. For example, if we consider the relation $R1$ shown in Table 3.1 some representative queries on this relation can be:

1. What are the names of the employees whose age is 18 years?
2. What are the names of the employees whose age is more than 30 years?
3. What is the average age of an employee?

If the relation $R1$ is encrypted normally, i.e., encrypting separately each cell of the relation with a strong (say an IND-CPA secure) encryption, then none of the above queries can be executed on the encrypted table. Note that an IND-CPA secure encryption scheme produces ciphertexts which are indistinguishable from random strings even for plaintexts which are chosen by the adversary. Thus, an IND-CPA secure encryption produces two different (seemingly) random strings for even two equal plaintexts. Hence, to design an encryption scheme which would allow efficient query processing in an encrypted table special care is required. To do query processing on the encrypted data the ciphertext should leak some information regarding the plaintext, moreover this leaked information should not be too much that it breaches the privacy of the data. Achieving this fine tradeoff is an interesting and difficult problem.

Regarding the queries that we posed above, query (1) can be addressed if the relation is encrypted using a *deterministic encryption scheme*, such encryption preserves the equality relation of plaintexts, i.e., two equal plaintexts produce two equal ciphertexts. For query (2), it is required that the ciphertexts maintain the order of the plaintexts, this can be achieved by *order preserving encryption*. Finally, for query (3) it is required that meaningful computations can be performed on the ciphertexts, *homomorphic encryption schemes* can support such computations on encrypted data.

In general a single encryption mechanism cannot provide with all the functionalities required for query processing. Hence, it is better to treat different classes of queries separately and encrypt values in a table with multiple encryption schemes which provides different functionalities. Among others, this paradigm is followed in a recent work [63], where several encryption schemes are combined to achieve different query processing functionalities. In this work we will mainly focus on processing a class of range queries (query (2) in the example above), and we will not be concerned about other class of queries where other encryption schemes would be required.

As stated earlier, to enable range queries in an encrypted database the ciphertext values

should provide order information of the plaintexts. This can be achieved by an order preserving encryption (OPE) scheme. An OPE scheme $\mathbf{E}$ is such a scheme where $\mathbf{E}(x) \geq \mathbf{E}(y)$, iff $x \geq y$. This interesting primitive has received lot of attention in the current years.

The problem will be clear if we consider the following example over the relation in Table 3.1, the initial set of values that the client needs to encrypt for the age attribute is $\{18, 17, 52, 33, 18, 36\}$. A potential set of order-preserving ciphertexts for this set is $\{2, 1, 5, 3, 2, 4\}$. This set is ideal because it simply informs the server of the order of the values, and nothing else. If the outsourced table stores this set, then later the client can pose the following query:

$$\texttt{SELECT * FROM } R1 \texttt{ WHERE } Age >= 36$$

This query can be translated to:

$$\texttt{SELECT * FROM } R1 \texttt{ WHERE } Age >= 4$$

The server just have to perform the query in a normal fashion to recover the corresponding result: Rosy and John. However, ideally an OPE scheme should allow the client to add new encrypted values without knowing the order relation with the existent values. For example, consider that later the client wants to add a new employee which age is 25. Since all integers between 1 and 5 are assigned then there is no space for the 25. In the next section, we explore the state of the art of this problem.

## 3.5 Order Preserving Encryption Schemes: An Overview

In Table 3.3 we summarize the different approaches in literature that aim to provide an efficient encryption scheme that allows to perform range queries on encrypted data, i.e, there are order preserving encryption schemes. The first concepts appeared in the paper [64], where the main aim was to design a scheme where efficient range queries can be executed on encrypted data. The work in [64] does not delve into formal definitional and security perspectives of OPE.

The first work which formally deals with OPE is [65], where the ideal security notion for an OPE scheme, IND-OCPA, was introduced. The IND-OCPA definition specifies that the main goal for an OPE is to reveal no additional information about the plaintext values besides their order (which is the minimum requirement for the order-preserving property). The IND-OCPA definition is built over the popular IND-CPA definition. We discuss these definitions first.

**Definition 3.1.** *Let for $b \in \{0,1\}$, $\mathcal{LR}(.,.,b)$ be a function that on input $m_0, m_1$ outputs $m_b$. For a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and an adversary $\mathcal{A}$ consider the following experiment*

**Experiment** $\mathbf{Exp}^{\textit{IND-CPA-b}}(\mathcal{A})$
$K \xleftarrow{\$} \mathcal{K}$;
$d \leftarrow \mathcal{A}^{\mathcal{E}_K(\mathcal{LR}(.,.,b))}$
**return** $d$

*The experiment selects a random key from $\mathcal{K}$ and gives an oracle access of $\mathcal{E}_K$ to $\mathcal{A}$. $\mathcal{A}$ can query its oracle multiple times. Each query is the form of $(m_0, m_1)$ where $|m_0| = |m_1|$, and the oracle returns $\mathcal{E}_K(m_b)$. The task of $\mathcal{A}$ is to guess the bit b. We define the IND-CPA advantage of $\mathcal{A}$ as*

$$\mathbf{Adv}^{\text{ind}-\text{cpa}}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\textit{IND-CPA-0}}(\mathcal{A}) \Rightarrow 1] - \Pr[\mathbf{Exp}^{\textit{IND-CPA-1}}(\mathcal{A}) \Rightarrow 1].$$

*$\mathcal{SE}$ is called IND-CPA secure if for all efficient adversaries $\mathcal{A}$, $\mathbf{Adv}^{\text{ind}-\text{cpa}}(\mathcal{A})$ is small.*

IND-OCPA is a variant of the IND-CPA definition where some restrictions on the queries of $\mathcal{A}$ are imposed. Let $\mathcal{A}$ make $q$ queries like $(m_0^{(1)}, m_1^{(1)}), (m_0^{(2)}, m_1^{(2)}), \cdots, (m_0^{(q)}, m_1^{(q)})$, such that $m_0^{(i)} < m_1^{(i)}$ if and only if $m_0^{(j)} < m_1^{(j)}$, for all $1 \leq i, j \leq q$. Such an adversary $\mathcal{A}$ is called a IND-OCPA adversary for indistinguishability under ordered chosen-plaintext attack.

IND-OCPA security is a strong security notion and it is difficult to achieve. In [65] it was shown that it is infeasible to achieve IND-OCPA security with a stateless encryption scheme. As a result, they settled on a weaker security guarantee called POPF, i.e., pseudorandom order preserving function. This security notion requires that no adversary can distinguish between oracle access to the encryption algorithm of the scheme, and a corresponding random order-preserving function on the same domain and range. However, later was shown in [10] that it leaks at least half of the plaintext bits.

In [10], Boldyreva et. al. proposed a new scheme that even though is not an order preserving encryption scheme, because the ciphertext does not preserve the order, it still allows to perform range queries over encrypted data. This new scheme is called modular order-preserving encryption (MOPE), in which the scheme of Boldyreva et. al. is prepended with a random shift cipher. Later Yum et. al. [12] improve the encryption scheme proposed by Boldyreva et.al. in [65]. However, it also reveals more than the order.

Later, several other order preserving schemes were proposed [11, 13, 66], but as it is suggested in [14] none of them has achieved the ideal IND-OCPA security, they all leak more than just the order of plaintext.

Recently, in [14], the first ideal-security order-preserving encoding scheme was proposed, where the ciphertexts reveal nothing except the order of the plaintext values. The insight that allow them to avoid the infeasibility result in [65] is that the encryption protocol proposed is interactive, and a small number of ciphertexts of already encrypted values change, as new plaintext values are encrypted. A property which the authors term as "mutable".

| Name | Year | Guarantees |
|---|---|---|
| Agrawal et.al.[64] | 2004 | None |
| Boldyreva et.al. [65, 10] | 2009,2011 | POPF |
| Agrawal et.al. [9] | 2009 | None |
| Lee et.al [11] | 2009 | None |
| Xiao et. al. [13] | 2010 | IND-OLCPA |
| Yum et. al.[12] | 2012 | POPF |
| Popa et. al. [14] | 2013 | IND-OCPA |

Table 3.3: Summary of existing approaches

**The OPE Scheme in [65]** Boldyreva et. al. in [65] provide the first cryptographic study of OPE primitive and also the first order preserving encryption scheme that does not require previous knowledge of all the plaintext to encrypt. The main idea of this scheme is a relation between a random order-preserving function and the hypergeometric probability distribution.

Let $\mathcal{D}$ and $\mathcal{R}$ be finite sets such that $\#\mathcal{D} \leq \#\mathcal{R}$ and let $\mathsf{OPE}_{\mathcal{D},\mathcal{R}}$, be the set of all order preserving maps from $\mathcal{D}$ to $\mathcal{R}$. Note that a map $f : \mathcal{D} \to \mathcal{R}$ is called order preserving if for all $x, x' \in \mathcal{D}$, $f(x) \geq f(x')$ iff $x \geq x'$. In [65] it is proved that

$$\#\mathsf{OPE}_{\mathcal{D},\mathcal{R}} = \binom{\#\mathcal{R}}{\#\mathcal{D}}. \tag{3.1}$$

Also, if $[M] = \{1, 2, \ldots M\}$ and $[N] = \{1, 2, \ldots, N\}$ then, for any $x, x+1 \in [M]$, and $y \in [M]$,

$$\Pr[f \xleftarrow{\$} \mathsf{OPE}_{[M],[N]} : f(x) \leq y \leq f(x+1)] = \frac{\binom{y}{x}\binom{N-y}{M-x}}{\binom{N}{M}} \tag{3.2}$$

Now we motivate how Boldyreva et. al. uncovered the relation between order preserving functions to the hypergeometric probability distribution. Let us consider $N$

balls in a bin where $M$ of them are black and $N-M$ of them are white. At each step we draw a ball at random without replacement. Consider a random variable $X$ that denotes the number of black balls chosen after a sample size of $y$ balls are picked. $X$ has a hypergeometric distribution, and the probability $X = x$ is given by

$$\Pr[X = x] = \frac{\binom{y}{x}\binom{N-y}{M-x}}{\binom{N}{M}}.$$

It is easy to see the correspondence of this probability with Eq.(3.2). This suggests a way to construct a random order preserving function from $[M]$ to $[N]$ as follows. Consider a bin with $M$ black and $N-M$ white balls. We pick balls randomly from the bin without replacement, if the $y$-th ball which is picked is black then we map the least unmapped element in $[M]$ to $y$.

It would be easier to see the above procedure with an example. Consider that we want to construct a random function mapping $A = \{1,2,3,4\}$ to $B = \{1,2,3,4,5,6,7,8\}$. Thus, here we have a total of 8 balls, of which 4 are black and 4 white. Suppose the outcome of an experiment of picking a ball at a time without replacement be as follows:

| sample size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| color | B | W | W | W | B | W | B | B |

Which means that the first ball picked was black, the second one was a white and so on. Based on this outcome an order preserving function $f$ can be constructed as follows

$$f(1) = 1, f(2) = 5, f(3) = 7, f(4) = 8.$$

Initially all points in the domain are un-mapped. The first ball picked is black thus the least unmapped point in the domain (i.e. 1) is mapped to 1. Next the 5-th ball picked is black, hence the least unmapped point in the domain (i.e. 2) is mapped to 5, and so on.

The above procedure depicts a way to construct a random order preserving function. This procedure is too inefficient to be directly used as an order preserving encryption scheme. The formal algorithms to convert this process in an encryption scheme along with a discussion on the required parameters (e.g. range size) can be found in the original paper.

**Order Preserving Encoding:** Popa et. al. in [14] provide the first OPE with ideal security, i.e., IND-OCPA security. The scheme in [14] encrypts the data with a normal encryption scheme which is not order preserving, but also encodes the plaintext values in a certain way which reveals the order. The basic idea behind the scheme is to encode

the values to be encrypted and organize them in a binary tree, or in a more efficient structure as B-trees. Each node contains the deterministic encryption of a value, and the encrypted values are arranged in the tree according to the order of the plaintext values. Based on this tree an order preserving encoding of the plaintext is derived.

Let us consider the example of encryption of the values of the attribute `Age` in the relation shown in Table 3.1. First the client arranges the values in a binary tree and encrypt these values using a deterministic encryption scheme $E_K()$. In Figure 3.6 we show the tree.

Now based on this tree, the client constructs a binary encoding for each ciphertext according to the following rules:

1. Each edge connecting a node to its left child is labeled 0 and each edge connecting a node to its right child is labelled 1.
2. The label of a path connecting the root to any node is obtained by concatenating the labels of the edges in the path in order. Note that there is a unique path connecting the root to any node.
3. The root is encoded with the empty string.
4. Each other node is encoded with the label of the path connecting the root to that node.

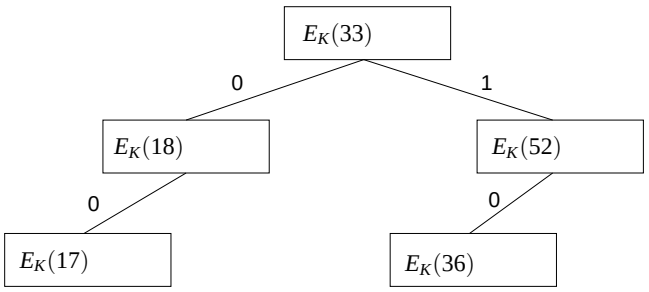
The encodings obtained in the above manner are not of equal length. An encoding length $n$ is selected, and all encodings obtained by the above steps are padded with a 1 followed by necessary number of zeros to make each encoding to be of length $n$. The ciphertexts, along with the encodings are given to the server by the client.

For the example that we consider, the encodings for the various ciphertexts are also shown in Figure 3.6. The string shown within square brackets is the code obtained before the padding. Here we consider each code to be 4 bits long. The decimal representation of the code of each cipher reveals the order of the plaintext which the cipher represents.

In [14] it is argued that this specific encoding does not reveal anything other than the order of the plaintext. Moreover, new plaintexts can be encrypted under this scheme, with an interaction between the client and server. The specific tree structure imposed on the ciphertext helps in attaining this. But, for keeping the code length restricted, at times it may be necessary to re-balance the tree. With a re-balancing, some old ciphertext values can have a changed encoding. This is the reason that this scheme is

called by its authors as mutable, in the sense that the ciphertext encodings can change with time. In the paper strategies are proposed so that ciphertext mutations and client-server interactions can be minimized.

Going back to our example, let us see how the value $E_K(36)$ is encoded, according to its position in the tree presented if Figure 3.6 the corresponding path is [10]. As we require a length of 4 bits this path is padded with the binary string 10 thus the resulting encoding is 1010, i.e 12 in decimal representation (see the last line in the table of Figure 3.6). Now consider that the client wants to add a new employee whose age is 21 to the database. Since the server does not know the underlying plaintext values, it is impossible for him to arrange the tree with the new value in the correct position. Therefore the client needs to help him to find the correct position for the new value. First, the client requires the root of the value, then the server answers with $E_K(33)$, which the client decrypts to 33. Since $21 < 33$ the client requires the left child of the root, the server answers with $E_K(18)$, and the client decrypts it to 18. The client realizes that $21 > 18$, thus it requires the right child of the node $E_K(18)$, the server answers that there does not exist such a node in the tree. Then, the client instructs the server to add the $E_K(21)$ in this position. Notice, that in this example it is not required to re-balance the tree.




| Ciphertext | OPE encoding |
|---|---|
| $E_K(33)$ | []1000 = 8 |
| $E_K(18)$ | [0]100 = 4 |
| $E_K(52)$ | [1]100 = 14 |
| $E_K(17)$ | [00]10 = 2 |
| $E_K(36)$ | [10]10 = 12 |

Figure 3.6: mOPE structure

## 3.6 Final Remarks

We discussed some main directions of previous works involving query authentication and privacy in databases. For the privacy problem we concentrated on database encryption which enables range queries to be performed efficiently. For this purpose we reviewed some previous works on order preserving encryption. In the following chapters we present our schemes for query authentication and encryption. The principal idea behind them is to use bitmaps indices. For the query authentication problem,

we use bitmaps as the authenticated data structure. This is completely novel and has several advantages over tree based ADS, as would be clear from the description of our schemes. For the privacy problem, our solution has some similarities with the scheme by Popa et. al. [14], in the sense that we also avoid using an OPE, but we encrypt the data with a deterministic encryption scheme and reveal the order with a separate encoding. The specific encoding which we use is also a type of bitmap which we call as the *l*-encoded bitmap. This specific encoding makes our scheme different from the scheme in [14] and also more amenable for implementation in real database environments.

# Chapter 4

# Relational Database Authentication Scheme

> I have never thought of writing for reputation and honor. What I have in my heart must come out; that is the reason why I compose.
>
> *Ludwig van Beethoven*

The authenticated query processing is an interesting problem of outsourced databases, we have discussed some general characteristics of this problem in Section 3.2. In this chapter we describe a generic framework called RDAS to address this issue. We also restrict our model and the construction to a static database, i.e., we do not allow inserts and updates in the database that we consider. Later in Chapter 5 we discuss some directions by which this limitation can be removed. Also it is important to note that there are practical scenarios, like applications involving data warehousing, where databases are largely static.

In what follows, in Section 4.1 we give the basic definition of our framework and we also discuss our security model. In Section 4.2 we introduce a specific protocol called RDAS1 which allows the client to authenticate disjunction queries with a reasonable overhead. We also study its security according to our model. Finally, in Section 4.2.2 we present the specific costs of RDAS1.

## 4.1 RDAS: Definitions and Basic Notions

A relational database authentication scheme (RDAS) consists of a tuple of algorithms $(\mathcal{G}, \mathcal{F}, \Phi, \Psi, \mathcal{V})$, which are described in detail in the following paragraphs.

$\mathcal{G}$ is the **key generation algorithm** and it selects one (or more) keys from a pre-specified key space and outputs them.

$\mathcal{F}$ is called the **authentication transform**, which takes in a set of relations $\mathcal{R}$ and a set of keys and outputs another set of relations $\mathcal{R}'$ along with some additional data $(M_s, M_c)$. If the set of keys is $\mathbf{K}$, we shall denote this operation as $(\mathcal{R}', M_c, M_s) \leftarrow \mathcal{F}_{\mathbf{K}}(\mathcal{R})$. A client who wants to store the set of relations $\mathcal{R}$ in an untrusted server, transforms $\mathcal{R}$ to $\mathcal{R}'$ using the authentication transform $\mathcal{F}$ and a set of keys. The transform $\mathcal{F}$ produces some additional data other than the set of relations $\mathcal{R}'$, the additional data consists of two distinct parts $M_s$ and $M_c$. The set of relations $\mathcal{R}'$ along with $M_s$ are stored in the server and the keys and the data $M_c$ are retained in the client. The key generation algorithm and the authentication transform are executed in the client side.

We call $\Phi$ as the **query translator**, it is a transformation which takes in a query for the relations in $\mathcal{R}$ and converts it into a query for relations in $\mathcal{R}'$. For ease of discussion we shall refer a query for $\mathcal{R}$ to be an $\mathcal{R}$-query and a query for $\mathcal{R}'$ to be an $\mathcal{R}'$-query. Thus, given a $\mathcal{R}$-query $q$, $\Phi(q)$ would be an $\mathcal{R}'$-query. Thus by use of the transform $\Phi$, the client would be able to translate queries meant for $\mathcal{R}$ to queries which can be executed on the transformed relations in $\mathcal{R}'$.

$\Psi$ is the **response procedure**. To execute a query $q$ on $\mathcal{R}$, the client converts the query to $\Phi(q)$ and sends it to the server. The server executes the function $\Psi$, which takes in the query $\Phi(q)$ and uses $\mathcal{R}'$ and $M_s$. The output of $\Psi$ is $\rho$, which we call as the response of the server. The server returns its response $\rho$ to the client.

The **verification procedure** is a keyed transform $\mathcal{V}_{\mathbf{K}}$ which runs in the client. It takes as input the query $q$, a response $\rho$ of the server and $M_c$ and outputs either an answer $\mathsf{ans}$ for the query $q$ or outputs a special symbol $\perp$ which signifies reject.

### 4.1.1 Correctness and security

If we fix the set of relations $\mathcal{R}$, then an $\mathcal{R}$-query $q$ when executed in $\mathcal{R}$ would have a fixed answer say $\mathsf{ans}(\mathcal{R}, \mathsf{q})$. Our goal is to transform $\mathcal{R}$ to $\mathcal{R}'$ using an RDAS in such a way that if the query $\Phi(q)$ is sent to the server, then the answer $\mathsf{ans}$ should be recoverable from the server response $\rho$ through the procedure $\mathcal{V}$, if the server follows the protocol correctly. On the other hand, if the server is malicious, i.e., it deviates from the protocol and sends a response $\rho'$ distinct from the correct response $\rho$ then the procedure $\mathcal{V}$ should reject the response by outputting $\perp$. In other words, if the answer

to a $\mathcal{R}$-query is ans, then after running the protocol, $\mathcal{V}$ will either produce ans or $\perp$, it would not produce an answer ans$'$ distinct from ans.

In the security model, we allow the adversary to choose the primary set of relations $\mathcal{R}$. Given this choice of $\mathcal{R}$, we compute $(\mathcal{R}', M_c, M_s) \leftarrow \mathcal{F}_{\mathbf{K}}(\mathcal{R})$, for a randomly selected set of keys $\mathbf{K}$ which is unknown to the adversary. We give $\mathcal{R}'$ and $M_s$ to the adversary. The adversary chooses an $\mathcal{R}$-query $q$ and the challenger provides the adversary with $\Phi(q)$, finally the adversary outputs a response $\rho$, and we say that the adversary is *successful* if $\mathcal{V}_{\mathbf{K}}(\rho, q, M_c) \notin \{\perp, \mathsf{ans}(\mathcal{R}, q)\}$.

**Definition 4.1.** *Let* $\mathsf{Succ}_{\mathcal{A}}$ *be the event that a specific adversary* $\mathcal{A}$ *is successful in the sense as described above. We say that a RDAS is* $(\varepsilon, t)$*-secure if for any adversary* $\mathcal{A}$ *which runs for time at most* $t$ $\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \varepsilon$.

Some immediate observations regarding this security definition are as follows:

1. **Encompasses both correctness and completeness**: An important thing to note is that the security definition covers both completeness and correctness, as RDAS is considered secure if the verification algorithm does not accept (except with some small probability) a wrong response.

2. **Concrete security and adversarial resources**: The definition follows the paradigm of concrete security, where we specify the running time and the probability of success of an adversary. An $(\varepsilon, t)$-secure RDAS is really secure where $t$ is "reasonable" and $\varepsilon$ is "small". These are to be interpreted in the specific context.

## 4.2 RDAS1: A generic scheme for select queries involving arbitrary disjunctions

We discuss a basic scheme for a secure RDAS which works only if the queries made are single attribute select queries or select queries involving disjunctions of an arbitrary number of equality conditions. We call this scheme as RDAS1. RDAS1 can be modified to handle certain other class of queries, but for the sake of simplicity we just concentrate on a scheme which works on disjunction queries. Later in Chapter 5 we discuss the possible extensions of RDAS1 including a particular scheme RDAS2, which is capable of handling more complex queries.

We describe the scheme assuming that the set of initial relations $\mathcal{R}$ is a singleton set consisting of a single relation $R(B)$, where $B = \{b_1, b_2, \ldots, b_{|B|}\}$ is the set of attributes,

and consider $A = \{a_1, \ldots, a_m\} \subseteq B$ to be a set of attributes on which queries are allowed, we shall call $A$ the set of allowed attributes. Note it is possible that $B = A$. The procedure $\mathcal{F}$ converts $R$ into two relations $R_\alpha$ and $R_\beta$, i.e, $\mathcal{R}' = \{R_\alpha, R_\beta\}$ and $M_s$ is empty and $M_c = \mathsf{nT}$, where $\mathsf{nT}$ is the number of tuples in $R$. The only cryptographic object used by RDAS1 is a message authentication code $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$, where $\mathcal{K}$ is the key space. Next, we discuss the details of each of the procedures involved in RDAS1. In what follows, we shall describe the procedures considering a generic relation $R(B)$, where the set of allowed attributes is $A \subseteq B$. Also for ease of exposition we shall throughout consider the relation $R1$ as depicted in Table 4.1 as a concrete example, and for simplicity, for $R1$ we shall consider the set of allowed attributes to be $\{\texttt{Gender}, \texttt{level}\}$.

| EmpId | Name | Gender | Level | Age |
|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 18 |
| MST | Mary | F | $L_1$ | 17 |
| JOH | John | M | $L_2$ | 52 |
| MRH | Mary | F | $L_1$ | 33 |
| ASY | Anne | F | $L_1$ | 18 |
| RZT | Rosy | F | $L_2$ | 36 |

Table 4.1: Relation Employees

**Key generation algorithm**

RDAS1.$\mathcal{G}$: The key space for RDAS1 is the same as the key space of the associated message authentication code MAC. The key generation algorithm selects a key $K$ uniformly at random from $\mathcal{K}$.

**Authentication transform**

RDAS1.$\mathcal{F}$: $\mathcal{F}$ produces two relations $R_\alpha$ and $R_\beta$ by the action of the key. The relation $R_\alpha$ is defined on the set of attributes $B \cup \{\texttt{Nonce}, \texttt{Tag}\}$, i.e., $R_\alpha$ has two more attributes than in $R$. If $R$ contains $\mathsf{nT}$ many tuples then $R_\alpha$ also contains the same number of tuples. The procedure for populating the tuples of $R_\alpha$ is depicted in Figure 4.1. Basically, this procedure computes a MAC for each row.

The relation $R_\beta$ contains the attributes $\{\texttt{Name}, \texttt{SearchKey}, \texttt{RowNo}, \texttt{Tag1}\}$, irrespective of the attributes in relation $R$. Where $\mathsf{Dom}(\texttt{Name}) = \{a_1, \ldots, a_m\}$, i.e., the allowed attributes in $R$. And, $\mathsf{Dom}(\texttt{SearchKey}) = \mathsf{Dom}(a_1) \cup \mathsf{Dom}(a_1) \cup \cdots \cup \mathsf{Dom}(a_m)$. Let $\Omega = \cup_{i=1}^{m} (\{a_i\} \times \mathsf{Dom}(a_i))$, note that the elements of $\Omega$ are ordered pairs of the form $(x, y)$ where $x \in \mathsf{Dom}(\texttt{Name})$ and $y \in \mathsf{Dom}(\texttt{SearchKey})$, and $\#\Omega = \sum_{i=1}^{m} \mathsf{Card}(a_i) = N$. Let $\mathcal{L}$ be a list of the elements in $\Omega$

in an arbitrary order. If $(x,y)$ be the $i$-th element in $\mathcal{L}$, then we shall denote $x$ and $y$ by $\mathcal{L}_i^1$ and $\mathcal{L}_i^2$ respectively, where $1 \leq i \leq N$ . The way the relation $R_\beta$ is populated is also shown in Figure 4.1. This procedure allows the client to store all possible pairs $\mathcal{L}_i^1, \mathcal{L}_i^2$ along with the MAC calculated over this pair concatenated with the respective bitmap and `RowNo`. Note that the bitmap is not explicitly stored in the relation $R_\beta$. The transform $\mathcal{F}$ is executed in the client side, and the resulting relations $R_\alpha$ and $R_\beta$ are stored in the server.

**Creating** $R_\alpha$
1. **for** $j = 1$ to $\mathsf{nT}$ // Number of tuples
2. **for** $i = 1$ to $\#B$ // Number of sensitive attributes
3. $t_j^{R_\alpha}[b_i] \leftarrow t_j^R[b_i]$;
4. **end for**
5. $t_j^{R_\alpha}[\texttt{Nonce}] \leftarrow j$;
6. $H \leftarrow t_j^R[b_1]||\ldots||t_j^R[b_m]||j$;
7. $t_j^{R_\alpha}[\texttt{Tag}] \leftarrow \mathsf{MAC}_K(H)$;// MAC for each row
8. **end for**

**Creating** $R_\beta$
1. **for** $j = 1$ to $N$ // Elements of $\Omega$
2. $t_j^{R_\beta}[\texttt{Name}] \leftarrow \mathcal{L}_j^1$;
3. $t_j^{R_\beta}[\texttt{SearchKey}] \leftarrow \mathcal{L}_j^2$;
4. $t_j^{R_\beta}[\texttt{RowNo}] \leftarrow \mathsf{nT} + j$;
5. $L \leftarrow \mathcal{L}_j^1||\mathcal{L}_j^2||\mathsf{BitMap}_R(\mathcal{L}_j^1, \mathcal{L}_j^2)||(\mathsf{nT} + j)$;
6. $t_j^{R_\beta}[\texttt{Tag1}] \leftarrow \mathsf{MAC}_K(L)$;// MAC for each bitmap
7. **end for**

Figure 4.1: Creating $R_\alpha$ and $R_\beta$

For a concrete example, if $\mathsf{RDAS1}.\mathcal{F}$ has as input the relation $R1$ (see Table 4.1) and the set of allowed attributes is $\{\texttt{Gender}, \texttt{level}\}$, then it would produce as output the relations $R1_\alpha$ and $R1_\beta$ as shown in Table 4.2. The relation $R1_\alpha$ is almost the same as that of $R1$, except that it has two additional attributes, `Nonce` and `Tag`. The attribute `Nonce` just contains the row numbers and is thus unique for each row. The attribute `Tag` is the message authentication code computed for a message which is produced by concatenating all the values of the attributes in that tuple.

Relation $R1_\alpha$

| EmpId | Name | Gender | Level | Nonce | Tag |
|---|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 1 | $Y_1$ |
| MST | Mary | F | $L_1$ | 2 | $Y_2$ |
| JOH | John | M | $L_2$ | 3 | $Y_3$ |
| LCT | Lucy | F | $L_1$ | 4 | $Y_4$ |
| ASY | Anne | F | $L_1$ | 5 | $Y_5$ |
| RZT | Rosy | F | $L_2$ | 6 | $Y_6$ |

Relation $R1_\beta$

| Name | SearchKey | RowNo | Tag1 |
|---|---|---|---|
| Gender | F | 7 | $Y_7'$ |
| Gender | M | 8 | $Y_8'$ |
| Level | $L_1$ | 9 | $Y_9'$ |
| Level | $L_2$ | 10 | $Y_{10}'$ |

Table 4.2: Relations $R1_\alpha$ and $R1_\beta$

The relation $R1_\beta$ contains the attributes $\{\texttt{Name}, \texttt{SearchKey}, \texttt{RowNo}, \texttt{Tag1}\}$, where in this case, $\mathsf{Dom}(\texttt{Name}) = \{\texttt{Gender}, \texttt{Level}\}$, $\mathsf{Dom}(\mathsf{SearchKey}) = \{M, F\} \cup \{L_1, L_2\}$. The tuples in

$R1_\beta$ are populated according to the procedure as shown in Figure 4.1, and the specific relation $R1_\beta$ is shown in Table 4.2.

**Query translator**

RDAS1.$\Phi$: The transform $\Phi$, transforms a query meant for the original relation $R$ to a set of queries which are meant to be executed on the relations $R_\alpha$ and $R_\beta$ which are stored in the server side. RDAS1 can authenticate only certain types of queries, the allowed queries for RDAS1 are of the following form:

$Q$: SELECT * FROM $R$ WHERE $a_1 = v_1$ OR $a_2 = v_2$ OR ..... OR $a_l = v_l$

The allowed set of queries are thus select queries on arbitrary numbers of disjunctions on different or repeated attributes[1], which includes select queries on a single attribute of the form SELECT * FROM $R$ WHERE $a_i = v$. Given as input a valid query $q$, $\Phi(q)$ outputs two queries one for the relation $R_\alpha$ (which we call $q_\alpha$) and the other for $R_\beta$ (which we call $q_\beta$). For the specific query Q, $\Phi$(Q) will output the following queries:

$Q_\alpha$: SELECT * FROM $R_\alpha$ WHERE $a_1 = v_1$ OR $a_2 = v_2$ OR ..... OR $a_l = v_l$

$Q_\beta$: SELECT * FROM $R_\beta$ WHERE (Name $= a_1$ AND SearchKey $= v_2$) OR ..... OR
( Name $= a_l$ AND SearchKey $= v_l$)

Going back to the concrete example, consider the following query $Q1$ on the relation $R1$

$Q1$: SELECT * FROM $R1$ WHERE Gender = 'M' OR $Level =$ '$L_2$'

After applying the transformation $\Phi(Q1)$ , the output queries $Q1_\alpha$ and $Q1_\beta$ would be the following:

$Q1_\alpha$: SELECT * FROM $R1_\alpha$ WHERE $Gender =$ 'M' OR $Level =$ '$L_2$'

$Q1_\beta$: SELECT * FROM $R1_\beta$ WHERE ($Name =$ 'Gender' AND $Searchkey =$'M') OR
($Name =$ 'Level' AND $Searchkey =$'$L_2$')

The reason for the specific structure of the $q_\beta$ queries would be clear from the description of the verification process and the associated example.

[1]By a query of disjunction on repeated attributes we mean a query like: SELECT * FROM $R$ WHERE $a_1 = v_1$ OR $a_1 = v_2$ OR $a_2 = v_3$. Here the attribute $a_1$ is repeated twice.

**Response procedure**

RDAS1.$\Psi$: As discussed, $\Psi$ is the transform executed in the server to generate the response for a set of queries produced by $\Phi$. In RDAS1 the response of the server is constructed just by running the queries specified by $\Phi$ on $R_\alpha$ and $R_\beta$. We denote the response by $\rho = (\rho_\alpha, \rho_\beta)$ where $\rho_\alpha$ and $\rho_\beta$ corresponds to responses of $q_\alpha$ and $q_\beta$ respectively. Thus, for the example, the server executes the queries $Q1_\alpha$ and $Q1_\beta$ on $R1_\alpha$ and $R1_\beta$ respectively and thus returns the response $\rho 1 = (\rho 1_\alpha, \rho 1_\beta)$ which is shown in Table 4.3.

Relation $\rho 1_\alpha$

| EmpId | Name | Gender | Level | Nonce | Tag |
|---|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 1 | $Y_1$ |
| JOH | John | M | $L_2$ | 3 | $Y_3$ |
| RZT | Rosy | F | $L_2$ | 6 | $Y_6$ |

Relation $\rho 1_\beta$

| Name | SearchKey | RowNo | Tag1 |
|---|---|---|---|
| Gender | M | 8 | $Y_8'$ |
| Level | $L_2$ | 10 | $Y_{10}'$ |

Table 4.3: Left side: Answer $\rho 1_\alpha$, Right side: Answer $\rho 1_\beta$

**Verification procedure**

RDAS1.$\mathcal{V}$: The verification procedure receives as input the response $\rho = (\rho_\alpha, \rho_\beta)$ from the server, the original query and the keys. The response of the server consists of two parts. We denote these two parts as two sets $\rho_\alpha$ and $\rho_\beta$ which are responses to the queries $q_\alpha$ and $q_\beta$ respectively. Thus, $\rho_\alpha$ and $\rho_\beta$ contain tuples from the relations $R_\alpha$ and $R_\beta$ respectively.

The transformed queries $q_\alpha$ and $q_\beta$ are also disjunctions of conditions, for a $q_\alpha$ query the conditions are of the form $a_i = v_i$, where $a_i$ is an attribute and $v_i$ its value, and for a $q_\beta$ query the conditions are of the form $\texttt{Name} = v \; AND \; \texttt{SearchKey} = \texttt{w}$. Thus, for the description below, we consider that $C_1^\alpha$ OR $C_2^\alpha$ OR $\ldots C_l^\alpha$ is a $\alpha$ query where each $C_i^\alpha$ is an equality condition and $C_1^\beta$ OR $C_2^\beta$ OR $\ldots C_l^\beta$ is a $\beta$ query where each $C_i^\beta$ is a conjunction of two equality conditions. Note that the number of conditions in $q_\alpha$ and $q_\beta$ would always be the same. Let SaT be a predicate which takes as input a tuple $t$ and a condition $C$ (which can also be a query $q$) and outputs a 1 if the tuple $t$ satisfies the condition $C$, otherwise outputs a zero. With these notations defined, we are ready to describe the verification algorithm. The verification algorithm consists of three procedures. We name the procedures as $\alpha$-Verify, makeBitMap and $\beta$-Verify. The procedures are shown in Figure 4.2, and they are applied sequentially in the same order as stated above.

The verification procedure checks for both the correctness and the completeness of the server response against the original query $q$. Note that the server response consists of two distinct parts $\rho_\alpha$ and $\rho_\beta$, the $\rho_\alpha$ part corresponds to the real result of the original

α-Verify

1. **for** all tuples $t \in \rho_\alpha$ //Verifying each `Tag` in $\rho_\alpha$
2. **if** $\mathsf{SaT}(t, q_\alpha) = 0$, **return** $\perp$
3. $ta \leftarrow \mathsf{MAC}_K(t[b_1]||\ldots||t[b_{|B|}]||t[\texttt{Nonce}])$;
4. **if** $ta \neq t[\texttt{Tag}]$ , **return** $\perp$;
5. **end for**

makeBitMap

6. **for** $i \leftarrow 1$ to $l$ //Creating $l$ bitmaps in zeros
7. $X_i \leftarrow 0^{\mathsf{nT}}$;
8. **end for**
9. **for** all tuples $t \in \rho_\alpha$
10. **for** $i \leftarrow 1$ to $l$
11. **if** $\mathsf{SaT}(t, C_i)$ //Set ones according with $\rho_\alpha$
12. $j \leftarrow t[\texttt{Nonce}]$;
13. $\mathsf{bit}_j(X_i) \leftarrow 1$;
14. **end if**
15. **end for**
16. **end for**

β-Verify

17. **for** $i \leftarrow 1$ to $l$
18. $T[i] \leftarrow 0$;
19. **end for**
20. **for** $i \leftarrow 1$ to $l$ //Verifying each `Tag` in $\rho_\beta$
21. **for** all tuples $t \in \rho_\beta$
22. **if** $\mathsf{SaT}(C_i^\beta, t) = 1$
23. $T[i] \leftarrow T[i] + 1$;
24. $LL \leftarrow t[\texttt{Name}]||t[\texttt{SearchKey}]||X_i||t[\texttt{RowNo}]$;
25. **if** $\mathsf{MAC}_K(LL) \neq t[\texttt{Tag1}]$ **return** $\perp$;
26. **endif**
27. **end for**
28. **end for**
29. **for** $i \leftarrow 1$ to $l$
30. **if** $T[i] \neq 1$ **return** $\perp$;
31. **end for**
32. **return** $\prod_{(b_1, b_2, \cdots, b_{|B|})} \rho_\alpha$;

Figure 4.2: The procedures involved in the verification process: we assume that the verification procedure has as input the queries $q_\alpha = C_1^\alpha$ `OR` $C_2^\alpha$ `OR` $\ldots C_l^\alpha$, $q_\beta = C_1^\beta$ `OR` $C_2^\beta$ `OR` $\ldots C_l^\beta$, and the server responses $\rho_\alpha$ and $\rho_\beta$.

query $q$ and the $\rho_\beta$ part assists the verification process to verify the completeness of the result in $\rho_\alpha$. In the part $\alpha$-Verify, the verification procedure checks for the correctness of the tuples returned by the server. As in the transformed relation $R_\alpha$ a message authentication code is associated with each tuple of the original relation, hence the $\alpha$-Verify part of the verification procedure checks whether the contents of the tuples in $\rho_\alpha$ are not modified. If any of the the tuples in $\rho_\alpha$ are modified then the computed message authentication code on the tuple will not match the attribute `Tag`. If the computed value of tag does not match with the attribute `Tag` for any tuple then the verification process rejects by returning $\perp$. Moreover in line 2 it checks whether each tuple in $\rho_\alpha$ do satisfy the specified query. If the verification process does not terminate in the $\alpha$-Verify phase then it means that the tuples in $\rho_\alpha$ are all valid tuples of the relation $R_\alpha$ and they all satisfy the specified query $q_\alpha$. The other two parts of the verification process checks the completeness of the response.

Corresponding to each condition `Name` $= v$ AND `SearchKey` $=$ `w` in the query $q_\beta$ the procedure makeBitMap constructs the corresponding bitmap $\mathsf{BitMap}_{R_\alpha}(v, w)$ using the server response $\rho_\alpha$. Note that if the server response $\rho_\alpha$ is correct then makeBitMap would be able to construct the bitmaps corresponding to each condition in $q_\beta$ correctly. This is possible due to the specific type of the allowed queries. Recall that an allowed query is formed only by the disjunctions of equality conditions. In the procedure corresponding to the $l$ conditions in $q_\beta$, $l$ bitmaps are constructed which are named $X_1, \ldots, X_l$ (See the example later for more explanation).

In the procedure $\beta$-Verify the response $\rho_\beta$ is verified using the bitmaps $X_1, \ldots, X_l$ constructed before. The procedure $\beta$-Verify first verifies whether $\rho_\beta$ contains tuples corresponding to each condition in $q_\beta$, this is done using the counter $T[i]$, where $i$ runs over the conditions in $q_\beta$. Notice, that for every condition $C_i^\beta$ the server must return only one tuple in $\rho_\beta$. The other parts of the procedure involves in verifying the tags of the tuples against the tag's of the computed bitmaps.

To make the exposition clearer let us consider the same example we have so far considered, i.e., the relation $R1$ the queries $Q1_\alpha$, $Q1_\beta$ and the corresponding server responses of $\rho 1_\alpha$ and $\rho 1_\beta$ (which are shown in Table 4.3). Given these responses the procedure $\alpha$-Verify will not terminate, as all the tuples in $\rho 1_\alpha$ do satisfy the conditions in $Q1_\alpha$ and as they are correct responses in the sense that they are just copies of the tuples present in the relation $R_\alpha$, hence the corresponding message authentication codes will match. Given the responses in $\rho 1_\alpha$, one can compute the bitmaps $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M)$ and $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2)$. To see this, see the response $\rho 1_\alpha$ in Table 4.3, where it says that the tuples satisfying the condition `Gender`$=M$ `OR` `Level`$=L_2$ are the tuples with the nonce values 1, 3 and 6. Now, as the verification procedure has as input the whole

of response $\rho 1_\alpha$, hence it can predict correctly that the rows with the nonce value 1 and 3 satisfy the condition Gender=$M$ and all the tuples in $\rho 1_\alpha$ (i.e., with nonce values 1, 3, 6) satisfy the condition Level=$L_2$. Thus, knowing that the total number of tuples in $R_\alpha$ to be 6, and assuming that server response is complete then the bitmap can be computed as $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M) = 101000$. Note that the 1st and 3rd bits of this bitmap are only one, as it corresponds to the response in $\rho 1_\alpha$. Similarly one can compute $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2) = 101001$. This is precisely what the procedure makeBitMaps would do for the example that we consider. The computation of the individual bitmaps $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M)$ and $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2)$ are possible from $\rho 1_\alpha$ as the $Q1_\alpha$ query is a disjunction of equality conditions, if in the contrary the query was a conjunction of conditions then there would be no way to compute the individual bitmaps in a straightforward way, this explains the reason for the query restriction that we impose.

Once these bitmaps are computed by using the procedure β-Verify one can verify the correctness of the response $\rho 1_\beta$. As one can concatenate corresponding the bitmaps computed by the procedure makeBitMaps with the other attributes of the tuples in $\rho_\beta$ and compute the tag using the message authentication code and thus verify if the computed tag matches the attribute Tag1.

The procedure β-Verify basically verifies the correctness of the response $\rho_\beta$, this verification is done by using the bitmaps constructed using the response $\rho_\alpha$. The correctness of the response $\rho_\beta$ implies the completeness of the response $\rho_\alpha$. We discuss about this more in the following section.

### 4.2.1 Security of RDAS1

In this section we show that the security of RDAS1 can be reduced to the security of MAC. In other words, if there exists an adversary capable of breaking RDAS1 then the MAC is not secure.

We can distinguish two possibilities for breaking RDAS: a) infringe the correctness and b) violate the completeness of the response for a fixed query. To break the correctness the opponent must make changes in one or more tuples of $\rho_\alpha$ and still pass the verification process. This implies that the adversary must forge the respective MACs. On the other hand, to violate the completeness, the adversary must change the respective bitmaps in $\rho_\beta$ which also implies forging the respective MACs. Now, we introduce this notion in a formal way.

**Theorem 4.1.** *Consider an adversary $\mathcal{A}$ attacking* RDAS1 *in the sense of definition 4.1. Let $\mathcal{A}$ choose a relation $R$ with* nT *tuples and the relation be such that the transformed relation $R_\beta$*

*contains $n'$ tuples. Then there exists an adversary $\mathcal{B}$ attacking the message authentication code* MAC *such that*

$$\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \Pr[\mathcal{B} \textit{ forges }].$$

*Also, $\mathcal{B}$ asks at most $\mathsf{n}\mathsf{T} + n'$ queries to its oracle and runs for time $t_{\mathcal{A}} + (\mathsf{n}\mathsf{T} + n')(c + t_{\mathsf{MAC}})$, where $t_{\mathcal{A}}$ is the running time of $\mathcal{A}$, $t_{\mathsf{MAC}}$ is the time for one MAC computation and $c$ is a constant.*

*Proof.* The idea of the proof is to construct an adversary $\mathcal{B}$ whose task is to forge the message authentication code MAC. $\mathcal{B}$ will use the adversary $\mathcal{A}$ by acting as its challenger. The interaction between $\mathcal{B}$ and $\mathcal{A}$ is represented in the procedure below:

**Adversary** $\mathcal{B}^{\mathsf{MAC}_K(.)}$
1. Receive the relation $R$ from $\mathcal{A}$;
2. Send $(R_\alpha, R_\beta) \leftarrow \mathsf{RDAS1}.\mathcal{F}(R)$ to $\mathcal{A}$;
   (Use the oracle $\mathsf{MAC}_K(.)$ to construct $(R_\alpha, R_\beta)$;
   Let $Q$ be the set of queries asked to $\mathsf{MAC}_K(.)$)
3. Receive a query $q$ from $\mathcal{A}$;
4. Send $(q_\alpha, q_\beta) \leftarrow \mathsf{RDAS1}.\Phi(q)$ to $\mathcal{A}$
5. Receive a response $\rho'$ from $\mathcal{A}$
6. **if** $\mathsf{RDAS1}.\mathcal{V}(\rho') \neq \perp$ AND $\mathsf{RDAS1}.\mathcal{V}(\rho') \neq \mathsf{ans}(\mathcal{R}, \mathsf{q})$;
7. $\quad (x, \mathsf{tag}) \leftarrow$ **GenerateForgery**();
8. **else** $x \xleftarrow{\$} \{0,1\}^* - Q$, $\mathsf{tag} \xleftarrow{\$} \{0,1\}^\tau$;
9. **return** $x, \mathsf{tag}$

In line 7 we mention a routine **GenerateForgery**() which will be defined later. The heart of the reduction is the following claim:

**Claim 1.** *If the condition of line 6 is satisfied, then $\mathcal{B}$ forges the MAC with probability 1.*

If the above claim is valid then we have

$$\begin{aligned}
\Pr[\mathcal{B} \text{ forges}] &= \Pr[\mathcal{B} \text{ forges} | \mathsf{Succ}_{\mathcal{A}}] \Pr[\mathsf{Succ}_{\mathcal{A}}] + \Pr[\mathcal{B} \text{ forges} | \overline{\mathsf{Succ}_{\mathcal{A}}}] \Pr[\overline{\mathsf{Succ}_{\mathcal{A}}}] \quad (4.1)\\
&= \Pr[\mathsf{Succ}_{\mathcal{A}}] + \frac{1}{2^\tau} \Pr[\overline{\mathsf{Succ}_{\mathcal{A}}}] \quad (4.2)\\
&\geq \Pr[\mathsf{Succ}_{\mathcal{A}}],
\end{aligned}$$

as desired. Eq. (4.2) follows from (4.1) as according to our claim $\Pr[\mathcal{B} \text{ forges} | \text{Succ}_{\mathcal{A}}] = 1$ and $\Pr[\mathcal{B} \text{ forges} | \overline{[\text{Succ}_{\mathcal{A}}]} = \frac{1}{2^\tau}$ as the probability that the $\tau$ bit tag of a message matches with a random $\tau$ bit string is $\frac{1}{2^\tau}$.

**Proof of Claim:** Here we will basically describe the procedure **GenerateForgery**(). Note that the response $\rho'$ of $\mathcal{A}$ consists of $(\rho'_\alpha, \rho'_\beta)$ and as the verification procedure does not output $\perp$ hence the following must be true:

1. The tuples in $\rho'_\alpha$ and $\rho'_\beta$ satisfies the conditions in the queries $q_\alpha$ and $q_\beta$ respectively, moreover there is only one tuple returned in $\rho_\beta$ corresponding to each condition in $q_\beta$ (see procedures $\alpha$-Verify and $\beta$-Verify in Figure 4.2).
2. All the tuples in $\rho'_\alpha$ and $\rho'_\beta$ are associated with their valid tags.

Moreover the condition in line 6 says that response produced by $\mathcal{V}$ is not correct. This can happen in the following two scenarios.

> **Case 1:** There is a tuple $\text{tup} \in \rho'_\alpha$ such that tup is not in $R_\alpha$. Moreover, if $\{b_1, \ldots, b_{|B|}\}$ be the original set of attributes, and $X = \text{tup}[b_1]||\ldots||\text{tup}[b_{|B|}]||\text{tup}[\texttt{Nonce}]$, then $(X, \text{tup}[\texttt{Tag}])$ is a valid message tag pair. Note, that this case signifies that the server response is incorrect.
>
> **Case 2:** There is no $\text{tup} \in \rho'_\alpha$ which is not present in $R_\alpha$. This case can only occur if the response is incomplete. This signifies that $\mathcal{A}$ has been able to forge a tag in $R_\beta$. To see this, note that the procedure $\mathcal{V}$ constructs the bitmaps corresponding to the conditions in the $q_\beta$ query based on the information in $\rho_\alpha$. If the result returned in $\rho_\alpha$ is incomplete then the bitmap corresponding to some condition would be wrongly computed by $\mathcal{V}$, but this wrong bitmap corresponds to the tag returned in $\rho'_\beta$.

In both cases above $\mathcal{B}$ can construct a forgery for the MAC in the following way:

> **GenerateForgery() for Case I:** $(X, \text{tup}[\texttt{Tag}])$ is a valid forgery, as according to the condition in line 6, $\text{tup}[\texttt{Tag}])$ is a valid tag for $X$, moreover as tup is not in $R_\alpha$ hence $\mathcal{B}$ has never asked its oracle a query of $X$.
>
> **GenerateForgery() for Case II:** Consider an arbitrary attribute $a$, and its value $v$, which is related to the query in question, such that the bitmap computed by $\mathcal{V}$ for the condition $a = v$ is $Y'$ and $Y' \neq Y$, where $Y = \text{BitMap}_v(a)$ is the real bitmap. In this case $\rho'_\beta$ would contain a tuple $(a, v, r, tag)$ where $r$ is the row number and it must be the case that $\text{MAC}_K(a||v||Y'||r) = tag$. Thus $(a||v||Y'||r, tag)$ is a valid forgery, as $\mathcal{B}$ has never asked $a||v||Y'||r$ to its MAC oracle.

□

### 4.2.2 Costs and Overheads

In Section 3.2 we discussed the different costs associated to the query authenticated problem. In this section we quantify these costs for RDAS1 scheme.

**Storage cost:** Given a relation $R(B)$ with $\mathsf{nT}$ tuples, let $\mathsf{size}(t_i[b])$ denote the size of the attribute $b$ in the tuple $t$. Then the total size of $R$ (which we also denote by $\mathsf{size}(R)$) would be given by

$$\mathsf{size}(R) = \sum_{i=1}^{\mathsf{nT}} \sum_{b \in B} \mathsf{size}(t_i[b]).$$

If this relation $R$ is converted into $(R_\alpha, R_\beta)$ with the help of the authentication transform RDAS1.$\mathcal{F}$, then we would have,

$$\mathsf{size}(R_\alpha) = \mathsf{size}(R) + \sum_{i=1}^{\mathsf{nT}} (\mathsf{size}(t_i[\texttt{Nonce}]) + \mathsf{size}(t_i[\texttt{Tag}])),$$

if we assume a tag of constant length of $\tau$ bits then we would have

$$\mathsf{size}(R_\alpha) \leq \mathsf{size}(R) + \mathsf{nT}(\lg \mathsf{nT} + \tau).$$

Again considering the set of allowed attributes of $R$ as $A = \{a_1, a_2, \ldots, a_m\}$, and $N = \sum_{i=1}^{m} \mathsf{Card}(a_i)$, we will have

$$\mathsf{size}(R_\beta) = \sum_{i=1}^{N} (\mathsf{size}(t_i[\texttt{Name}]) + t_i[\texttt{SearchKey}] + t_i[\texttt{RowNo}] + \mathsf{size}(t_i[\texttt{Tag1}])).$$

If we consider $s_{\mathsf{Name}}$ and $s_{\mathsf{sk}}$ the maximum size of the values of the attributes `Name` and `SearchKey`, then we would have

$$\mathsf{size}(R_\beta) \leq N(s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau).$$

The total cost of storage at the server side would be $\mathsf{size}(R_\alpha) + \mathsf{size}(R_\beta)$, and at the client side would be $\lg(\mathsf{nT})$ as in the client we need to store the number of tuples in the original relation.

**Communication Cost:** Consider the query `SELECT * FROM` $R_\alpha$ `WHERE` $a_1 = v_1$ `OR` $a_2 = v_2$ `OR .....` `OR` $a_l = v_l$, let the number of tuples satisfying the query be num. Let

siz be the size of the response in a normal scenario without authentication. Then the maximum size of the server response in case of RDAS1 would be

$$\mathsf{siz}_{\mathsf{RD1}} = \mathsf{siz} + \mathsf{num} \times (\lg \mathsf{nT} + \tau) + l \times (s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau), \tag{4.3}$$

where the first two terms corresponds to the $\rho_\alpha$ response and the remaining term counts for the $\rho_\beta$ response.

## 4.3 Summary

In this chapter we described RDAS, which is a basic model for authenticated query processing in outsourced databases. We also fixed a syntax for RDAS and defined its security. Moreover we presented a concrete scheme RDAS1 which uses message authentication codes and bitmap indices. There are several ways in which RDAS1 can be improved. We discuss in details some extensions over RDAS1 in the next chapter.

# Chapter 5

# Extensions and Improvements of RDAS1

> Small opportunities are often the beginning of great enterprises.
>
> *Demosthenes*

In this chapter we discuss the following extensions to our construction RDAS1:

1. In Section 5.1 we discuss RDAS2 which extends RDAS1 by allowing it to handle select queries involving all kinds of logical operators. We argue that RDAS2 enjoys same security as of RDAS1 but the response sizes of RDAS2 are greater than that of RDAS1. We also present a variant of RDAS2 called RDAS2-cmp, where we use a compression technique named Enhanced Word Aligned Hybrid (EWAH) (discussed in Section 2.6.3), in RDAS2-cmp the response sizes are significantly smaller than in RDAS2.

2. Both RDAS1 and RDAS2 suffer from a limitation that if no tuple in the outsourced table matches a posed query then there is no way to verify that the empty reply that the server responds is a correct one. We describe two ways by which the authenticity of such empty replies can be verified both in RDAS1 and RDAS2.

3. In Section 5.3 we describe two more variants called RDAS1-agg and RDAS2-agg. In these variants we improve over the basic RDAS1 and RDAS2 schemes by using aggregated message authentication schemes. These variants have significantly less communication costs compared to the basic schemes.

4. In Section 5.5 we discuss how RDAS2 can be extended to dynamic scenarios where the operations like insert, update and delete can also be applied.

## 5.1 RDAS2: Selects Involving Arbitrary Boolean Connectives

RDAS1 can be modified to support SELECT queries involving all kinds of Boolean connectives at the cost of the size of the query responses. Recall that the query restriction for RDAS1 arises from the problem of constructing the bitmaps of all the attributes involved in the query. We propose an extension of RDAS1 which can support queries of the form

$Q$: `SELECT * FROM` $R$ `WHERE` $(a_1 = v_1)\ \Delta_1\ (a_2 = v_2)\ \Delta_2\ \ldots\ldots\ \Delta_{l-1}\ (a_l = v_l)$,
where $\Delta_i$s are arbitrary Boolean connectives. An easy solution to this case would be to change RDAS1 to a new protocol RDAS2 along the following lines:

1. The relation $R_\beta$ produced by RDAS2.$\mathcal{F}$ would contain explicit bitmaps corresponding to the attributes and the values. Specifically, the attributes present in $R_\beta$ should be $\{\texttt{Name}, \texttt{SearchKey}, \texttt{RowNo}, \texttt{bitmap}, \texttt{tag1}\}$. Thus, for creating the relation $R_\beta$ we need to add a line $t_j^{R_\beta}[\texttt{bitmap}] \leftarrow \mathsf{BitMap}_R(\mathcal{L}_j^1, \mathcal{L}_j^2)$ after line 5 in the procedure **Creating** $R_\beta$ in Figure 4.1.
2. The query translation procedure and the response procedure for RDAS2 remain same as that of RDAS1.
3. The response procedure also remains the same, i.e., the server just answers the $q_\alpha$ and $q_\beta$ queries, but as the $R_\beta$ relation now explicitly contains the bitmaps, hence the bitmaps would also be a part of the query.
4. For the verification procedure in RDAS2 it is not required to create the bitmaps anymore, the client verifies the $\rho_\alpha$ response by the procedure $\alpha$-Verify in Figure 4.2, then it verifies the tags of the individual bitmaps returned in $\rho_\beta$ and finally computes the result bitmap using the returned bitmap and checks if the result bitmap matches with the result returned.

We now state the storage and communication costs for RDAS2 following the notations in Section 4.2.2. The size of $R_\alpha$ in case of RDAS2 would be the same as in RDAS1, the size of $R_\beta$ would be

$$\mathsf{size}(R_\beta) \leq N(s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau + \mathsf{nT}).$$

The size of a server response in case of RDAS2 would be

$$\mathsf{siz}_{\mathsf{RD2}} = \mathsf{siz}_{\mathsf{RD1}} + l \times \mathsf{nT} \tag{5.1}$$

where $\mathsf{siz}_{\mathsf{RD1}}$ is the size of the response of RDAS1, as given in Eq. (4.3).

In case of RDAS2, though we state that the bitmaps are to be explicitly stored in the relation $R_\beta$, but as most commercial databases use bitmaps indices for accelerating query processing, hence this may not amount to extra storage in some systems. Moreover bitmaps can be compressed, there has been substantial work on suitable encoding of bitmaps such that their sizes can be reduced and the Boolean operations be applied on the compressed bitmaps [43, 67, 68, 69, 70]. Applying proper encoding of the bitmaps can drastically reduce both storage and communication costs. This version which uses compression is discussed in Section 5.4.

### 5.1.1 Security of RDAS2

In this section we show that as in RDAS1 the security of RDAS2 depends on the strength of the MAC. Notice that the pairs $m, \mathsf{tag}$ in $\rho_\alpha, \rho_\beta$ are the same as in RDAS1. The only difference is that in RDAS2, the complete message $t[Name]||t[SearchKey]||t[Bitmap]||t[RowNo]$ is stored in $R_\beta$. Thus, the security theorem for RDAS1 applies to RDAS2 without any change.

**Theorem 5.1.** *Consider an adversary $\mathcal{A}$ attacking* RDAS2 *in the sense of definition 4.1. Let $\mathcal{A}$ choose a relation $R$ with* $\mathsf{nT}$ *tuples and the relation be such that the transformed relation $R_\beta$ contains $n'$ tuples. Then there exists an adversary $\mathcal{B}$ attacking the message authentication code* MAC *such that*

$$\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \Pr[\mathcal{B} \textit{ forges }].$$

*Also, $\mathcal{B}$ asks at most $\mathsf{nT} + n'$ queries to its oracle and runs for time $t_{\mathcal{A}} + (\mathsf{nT} + n')(c + t_{\mathsf{MAC}})$, where $t_{\mathcal{A}}$ is the running time of $\mathcal{A}$, $t_{\mathsf{MAC}}$ is the time for one MAC computation and $c$ is a constant.*

The proof follows in the same way as the proof of Theorem 4.1, hence we do not repeat it.

## 5.2 The case of empty replies

One limitation of RDAS1 and RDAS2 is that if the server replies with an empty response in $S_\beta$ corresponding to a query then there is no way to verify the validity of the response. For example, in case of the relation $R1$ (in Table 3.1), if a query

```
SELECT * FROM R1 WHERE Level = 'L3'
```

is posed, then response to this query would be empty, and there is no way to verify

whether the response is correct. It is to be noted that a server may reply an RDAS1/RDAS2 query which has a valid response with an empty response and in such scenario also RDAS1/RDAS2 provides no mechanism to detect that the response is wrong.

If we assume knowledge of the client regarding the values of the attributes present in the relation then it would be easy for it to verify whether an empty response is the correct one. We discuss two simple ways by which RDAS1 can be extended so that empty responses can be verified. The same steps are valid for RDAS2 also.

1. Recall that the authentication transform $\mathcal{F}$ returns two additional data items $M_c$ and $M_s$, which are stored in the client and server sides respectively. In RDAS1, $M_c = \mathsf{nT}$ the number of tuples in the original relation, and $M_s$ is empty. As a first solution, we suggest to augment $M_c$ with a list of all values corresponding to each sensitive attribute in the relation. With this additional information, the client can easily verify whether an empty response is correct. This functionality comes with an additional storage cost in the client side. But if we assume that the domain of the attributes has small cardinality this storage would be much less compared to the size of the whole relation. Moreover, with this small change in RDAS1 the other efficiency and security claims remain unchanged.

2. As a second solution we propose to augment $M_s$ with additional information. In particular, we suggest to build a list $\mathsf{Lst}_a$, which contains all distinct values of the attribute $a$ which occur in the original relation (encoded in an appropriate way). Corresponding to each list we compute $\mathsf{lstTag}_a = \mathsf{MAC}_{K_1}(\mathsf{Lst}_a)$, and we store $(\mathsf{Lst}_a, \mathsf{lstTag}_a)$ for each sensitive attribute $a$ in $M_s$. Whenever the server returns an empty response corresponding to a query involving a set of attributes $S$, then the server includes in its response $(\mathsf{Lst}_a, \mathsf{lstTag}_a)$ for all $a \in S$ . It is easy to see that with this information the client can verify whether the empty response is the correct one.

   This solution has no extra overhead on the client side storage, but increases the server side storage, which is generally not of much concern. Additionally, it increases the bandwidth requirement for the responses of queries with empty responses. But, for other queries the functionality of RDAS1 along with the security claims remain unchanged.

## 5.3 Using Aggregated MACs

Deterministic message authentication codes can be suitably aggregated [71]. The main motivation behind aggregated MACs is to provide authenticated communications in bandwidth constrained scenarios, say in case of a sensor network. In case of a sensor network various sensor nodes have their own secret key which they only share with the base station. Let us see an example of a sensor network consisting of $k$ nodes labeled $i_1, \ldots, i_k$. We assume a communication protocol, where first the node $i_1$ sends its authenticated data $(m_{i_1}, t_{i_1})$ ( consisting of the message $m_{i_1}$ and the authentication tag $t_{i_1}$, which is computed using $m_{i_1}$ and the key $k_{i_1}$ of the node) to the next node $i_2$. Node $i_2$ adds its own authenticated message $(m_{i_2}, t_{i_2})$ to the received message and sends $[(m_{i_1}, t_{i_1}), (m_{i_2}, t_{i_2})]$ to $i_3$. Thus, ultimately the base station receives $[(m_{i_1}, t_{i_1}), \ldots, (m_{i_k}, t_{i_k})]$ from node $i_k$. As the base station owns the key of all the nodes, hence it can verify the authenticity of all the messages received. If we assume that the authentication tags are $n$ bit long irrespective of the message lengths, then this communication protocol has an extra bandwidth overhead of $O(k^2 n)$ bits. In case of sensor networks the data sent by them are generally very short, say a temperature reading, thus the size of the data may be much smaller than the size of the authentication tag (if one uses a secure MAC to generate the tag, then $n$ would be around 128 bits). Thus a $O(k^2 n)$ bit extra bandwidth overhead may not be tolerated in a standard scenario. Aggregated MAC comes as a solution to this problem, if the MAC used for generating the authenticated tag is "aggregate-able", then node $i_1$ sends $(m_{i_1}, t_{i_1})$ to $i_2$, $i_2$ sends $[(m_{i_1}, m_{i_2}), t_{i_1} \oplus t_{i_2})]$ etc. And finally, node $i_k$ sends $[(m_{i_1}, \ldots, m_{i_k}); t_{i_1} \oplus \ldots \oplus t_{i_k})]$ to the base station. This protocol only requires an extra bandwidth overhead of $O(kn)$.

In the example above, we used $\oplus$ as the aggregation operator, but one can define it generically as it was done in [71]. Moreover, it was proved in [71] that if one has a secure deterministic message authentication code then by using $\oplus$ as the aggregation operator one can securely aggregate MACs, in the multiuser setting. The security model describes a successful adversary as one who can produce a set of messages along with the user identifiers for each message and an aggregated tag which verifies. The adversary is allowed to see the tags corresponding to messages of his/her choice, and additionally (s)he is allowed to corrupt some users and know their keys. The final message set produced by the adversary should contain at least one message user id pair $(m, id)$ such that (s)he has not seen the tag corresponding to $m$ and has not corrupted the user $id$.

Aggregated MACs can reduce communication costs both in case of RDAS1 and RDAS2. Functionally, using aggregated MACs would reduce the response sizes, as then the tags corresponding to all the tuples in $\rho_\alpha$ and $\rho_\beta$ will not be required to be sent, but an

aggregation of these would be required. But it is to be noted that our case is fundamentally different from the above example in that we do not have multiple users, the authentication tags are all generated using the same key. This has an advantage, that the security reduction that can be obtained in this scenario is tighter than that obtained in the multiuser scenario. We discuss this a bit more in Section 5.3.1.

We propose two variants of RDAS namely RDAS1-agg and RDAS2-agg which extend RDAS1 and RDAS2 by using the functionality of aggregated MACs. We describe these variants next.

To convert RDAS1 to RDAS1-agg we just need to apply little modifications in the server response procedure and the verification procedure. In the response procedure, the server sends $\rho_\alpha$ and $\rho_\beta$ as in RDAS1 but without the attributes `Tag` and `Tag1`, and it sends two additional strings $\mathsf{str}_\alpha$ and $\mathsf{str}_\beta$ which contain the xor of the tags in $\rho_\alpha$ and $\rho_\beta$ respectively. For example, consider the responses $\rho 1_\alpha$ and $\rho 1_\beta$ in Table 4.3, the response for RDAS1-agg would be projections of $\rho 1_\alpha$ and $\rho 1_\beta$ without the attributes `Tag` and `Tag1` respectively. And additionally the response procedure would return $\mathsf{str}_\alpha = Y_1 \oplus Y_3 \oplus Y_6$ and $\mathsf{str}_\beta = Y'_8 \oplus Y'_{10}$. The changes that are to be applied to RDAS2 to obtain RDAS2-agg are exactly the same.

This aggregation of tags leads to a savings in the communication cost in both RDAS1 and RDAS2. Following the notation of Section 4.2.2, the size of server response for RDAS1-agg and RDAS2-agg would be

$$\mathsf{siz}_{\mathsf{RD1\text{-}agg}} = \mathsf{siz}_{\mathsf{RD1}} - \tau(\mathsf{num} + l) \tag{5.2}$$

$$\mathsf{siz}_{\mathsf{RD2\text{-}agg}} = \mathsf{siz}_{\mathsf{RD2}} - \tau(\mathsf{num} + l). \tag{5.3}$$

### 5.3.1 Security of RDAS1-agg and RDAS2-agg

For discussion of security, first we try to formalize the security definition of the aggregated MACs. Firstly, given a message authentication code $\mathsf{MAC} : \mathcal{K} \times \mathcal{M} \to \{0,1\}^\tau$ we define the corresponding aggregated MAC $\mathsf{MAC}^*$ in the single user setting as follows. The MAC generation algorithm given as input $m \in \mathcal{M}$ and a key $K \in \mathcal{K}$ outputs $\mathsf{MAC}_K(m)$. The aggregation algorithm when given two sets of messages $M_1, M_2 \subset \mathcal{M}$ and two corresponding tags $\mathsf{tag}_1, \mathsf{tag}_2 \in \{0,1\}^\tau$ outputs $\mathsf{tag}_1 \oplus \mathsf{tag}_2$. The verification algorithm when given a key $K \in \mathcal{K}$, a set of messages $M \subset \mathcal{M}$ and a tag $\mathsf{tag} \in \{0,1\}^\tau$ computes

$$t = \bigoplus_{m \in M} \mathsf{MAC}_K(m),$$

and accepts if $\mathsf{tag} = t$ and rejects otherwise.

An adversary $\mathcal{A}$ attacking $\mathsf{MAC}^*$ is given oracle access to $\mathsf{MAC}_K()$, i.e., it can know the tags corresponding to the messages of it's choice. Let us consider that $\mathcal{A}$ asks $q$ queries $Q = \{x_1, \ldots, x_q\}$ to its oracle and gets back the corresponding tags $t_1, \ldots, t_q$. Finally $\mathcal{A}$ produces a set of messages $M_f \subset \mathcal{M}$ and a tag $\mathsf{tag}_f$. $\mathcal{A}$ is said to be successful in forging if there is at least one element $m \in M_f$ such that $m \notin Q$ and the verification algorithm of $\mathsf{MAC}^*$ on input $(K, M_f, \mathsf{tag}_f)$ accepts. We define the forging advantage of $\mathcal{A}$ as

$$\mathbf{Adv}^{\text{auth-ag}}_{\mathsf{MAC}^*}(\mathcal{A}) = \Pr[\mathcal{A} \text{ forges}].$$

If $\mathsf{MAC}_K()$ is a secure MAC then so is $\mathsf{MAC}^*_K()$. The following theorem states this more formally.

**Theorem 5.2.** *Let $\mathcal{A}$ be an arbitrary adversary attacking the aggregated MAC $\mathsf{MAC}^*$, and $\mathcal{A}$ runs for time $T$ and makes $q$ queries to its oracle. Then there exists an adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{auth}-\text{ag}}_{\mathsf{MAC}^*}(\mathcal{A}) = \mathbf{Adv}^{\text{auth}}_{\mathsf{MAC}}(\mathcal{B}).$$

*$\mathcal{B}$ in turn makes $O(q)$ queries and runs for time $O(T)$.*

*Proof.* Note $\mathcal{B}$ is attacking $\mathsf{MAC}_K$, hence it has $\mathsf{MAC}_K$ as its oracle. $\mathcal{B}$ runs $\mathcal{A}$ as follows: whenever $\mathcal{A}$ asks a query $m$, $\mathcal{B}$ returns to $\mathcal{A}$, $\mathsf{MAC}_K(m)$ through its oracle. Let $Q$ be the set of queries made by $\mathcal{A}$. Finally $\mathcal{A}$ outputs a forgery $(M_f, \mathsf{tag}_f)$. If $\mathcal{A}$ is successful in forging then $\Gamma = M_f \setminus Q \neq \emptyset$. Fix $r \in \Gamma$ and compute

$$t_{all} = \begin{cases} \bigoplus_{x \in \Gamma \setminus \{r\}} \mathsf{MAC}_K(x), & \text{if } \Gamma \setminus \{r\} \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathcal{B}$ can compute $t_{all}$ using its oracle. Finally, $\mathcal{B}$ outputs $(r, \mathsf{tag}_f \oplus t_{all})$ as its forgery. It is easy to see that if $\mathcal{A}$ successfully forges then so does $\mathcal{B}$, and the runing time of $\mathcal{B}$ and the number of queries asked by $\mathcal{B}$ are as desired. □

With this discussion we are ready to state the security of RDAS1-agg and RDAS2-agg.

**Theorem 5.3.** *Consider an adversary $\mathcal{A}$ attacking $\Upsilon \in \{\mathsf{RDAS1\text{-}agg}, \mathsf{RDAS2\text{-}agg}\}$ in the sense of definition 4.1. Let $\mathcal{A}$ choose a relation $R$ with $\mathsf{nT}$ tuples and let $R_\beta$ contains $n'$ tuples. Then there exists an adversary $\mathcal{B}$ such that*

$$\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \mathbf{Adv}^{\text{auth}}_{\mathsf{MAC}}(\mathcal{B}).$$

*Also, $\mathcal{B}$ asks at most $\mathsf{nT} + n'$ queries to its oracle and runs for time $t_{\mathcal{A}} + (\mathsf{nT} + n')(c + t_{\mathsf{MAC}})$, where $t_{\mathcal{A}}$ is the running time of $\mathcal{A}$, $t_{\mathsf{MAC}}$ is the time for one MAC computation and $c$ is a constant.*

*Proof.* The proof is similar to proof of Theorem 4.1, with some small differences. We present the reduction in two steps. First we construct an adversary $\mathcal{C}$ which attacks the aggregated MAC $\mathsf{MAC}^*$, and it runs the adversary $\mathcal{A}$, such that

$$\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \mathbf{Adv}^{\text{auth}-\text{ag}}_{\mathsf{MAC}^*}(\mathcal{C}). \tag{5.4}$$

Then using Theorem 5.2, we have that there exists an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{auth}-\text{ag}}_{\mathsf{MAC}^*}(\mathcal{B}) = \mathbf{Adv}^{\text{auth}}_{\mathsf{MAC}}(\mathcal{C}). \tag{5.5}$$

Thus, from equations (5.4) and (5.5) the Theorem follows.

To prove equation (5.5) we construct an adversary $\mathcal{C}$ which attacks the aggregate MAC $\mathsf{MAC}^*$. $\mathcal{C}$ runs $\mathcal{A}$ ($\mathcal{A}$ attacks $\Upsilon$) as follows.

**Adversary** $\mathcal{C}^{\mathsf{MAC}_K(.)}$
1. Receive the relation $R$ from $\mathcal{A}$;
2. Send $(R_\alpha, R_\beta) \leftarrow \Upsilon.\mathcal{F}(R)$ to $\mathcal{A}$;
   (Use the oracle $\mathsf{MAC}_K(.)$ to construct $(R_\alpha, R_\beta)$;
   Let $Q$ be the set of queries asked to $\mathsf{MAC}_K(.)$)
3. Receive a query $q$ from $\mathcal{A}$;
4. Send $(q_\alpha, q_\beta) \leftarrow \Upsilon.\Phi(q)$ to $\mathcal{A}$
5. Receive a response $\rho'$ from $\mathcal{A}$
6. **if** $\Upsilon.\mathcal{V}(\rho') \neq \perp$ AND $\Upsilon.\mathcal{V}(\rho') \neq \mathsf{ans}(\mathcal{R}, q)$;
7. $\quad (x, \mathsf{tag}) \leftarrow$ **AggregateForgery**();
8. **else** $x \xleftarrow{\$} \{0,1\}^* - Q$, $\mathsf{tag} \xleftarrow{\$} \{0,1\}^\tau$;
9. **return** $\{x\}, \mathsf{tag}$

Description of $\mathcal{C}$ is almost the same as the description of adversary $\mathcal{B}$ as given in the proof of Theorem 4.1. As in the proof of Theorem 4.1, we claim that if the condition in line 6 is satisfied then $\mathcal{C}$ generates a forgery for $\mathsf{MAC}^*$ with probability 1. We can see this by following the same line of arguments as in the proof of Theorem 4.1. We explain the procedure of **AggregateForgery**() briefly for $\Upsilon =$RADS2-agg, the case of $\Upsilon =$RADS1-agg is similar.

Let $B = \{b_1, b_2, \ldots, b_{|B|}\}$ be the original set of attributes and let $\rho' = (\rho'_\alpha, \rho'_\beta), \mathsf{str}_\alpha, \mathsf{str}_\beta$. Note that here $\rho'_\alpha, \rho'_\beta$ does not contain the attribute `Tag` and `Tag1` respectively. If the condition in line 6 is satisfied then either of the two cases must be satisfied:

- *Case I*: There exists at least one tuple $X$ in $\rho_\alpha$ which does not correspond to any tuple in $R_\alpha$. Construct a set $\mathbb{S}_\alpha$ as

$$\mathbb{S}_\alpha = \{t[b_1]||\cdots||t[b_{|B|}]||t[\texttt{Nonce}] : t \in \rho'_\alpha\}.$$

  Then $\mathbb{S}_\alpha, \mathsf{str}_\alpha$ constitutes a valid forgery for MAC$^*$.

- *Case II*: There exists at least one tuple $X$ in $\rho_\beta$ which does not correspond to any tuple in $R_\beta$. Construct a set $\mathbb{S}_\beta$ as

$$\mathbb{S}_\beta = \{t[\texttt{Name}]||t[\texttt{SearchKey}]||t[\texttt{bitmap}]||t[\texttt{RowNumber}] : t \in \rho'_\beta\}.$$

  Then $\mathbb{S}_\beta, \mathsf{str}_\beta$ constitutes a valid forgery for MAC$^*$.

□

## 5.4 RDAS2-cmp: RDAS2 with Compression

We discussed in Section 2.6.3, that bitmaps can be compressed in such a way that the bit operations can be applied in the compressed domain, moreover by compression the size of the bitmaps can be drastically reduced. RDAS2 and its variants require explicit storage of bitmaps and also the bitmaps need to be transmitted as the part of the query response, hence bitmap compression in this context can be very helpful. Our experiments which we present later also validate that using compressed bitmaps not only reduces storage and communication costs but it also results in considerable savings in computation time.

The changes required in RDAS2 and its variants to incorporate bitmap compression are as follows:

1. The column $R_\beta[\texttt{bitmap}]$ in the relation $R_\beta$ would be populated by the EWAH compressed bitmaps. The MAC would also be computed using the compressed bitmaps.

2. The query translation procedure and the response procedure remains same as that of RDAS2 (without compression).

3. The response procedure also remains the same, i.e., the server just answers the $q_\alpha$ and $q_\beta$ queries.

4. For the verification procedure in RDAS2 with compression it is not required to uncompress the bitmaps, the client verifies the $\rho_\alpha$ response by the procedure $\alpha$-Verify in Figure 4.2, then it verifies the tags of the individual compressed bitmaps returned in $\rho_\beta$ and finally computes the resulting bitmap using the returned compressed bitmaps, and checks if the resulting bitmap matches with the compressed bitmap obtained from the response.

The incorporation of compression in RDAS2 has no effect on the security but the storage and communication costs would decrease based on the amount of compression achieved.

The size of $R_\alpha$ in case of RDAS2 with compression would be the same as in RDAS1. Let $\mathsf{sz\_cmp}(t_N[bitmap])$ denotes the size of the compressed bitmap for a given attribute and value pair, the size of $R_\beta$ would be

$$\mathsf{size}(R_\beta) \leq N(s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{n}\mathsf{T} + N) + \tau + \mathsf{sz\_cmp}(t_N[bitmap]))$$

The size of a server response in case of RDAS2 would be

$$\mathsf{siz}_{\mathsf{RD2}} = \mathsf{siz}_{\mathsf{RD1}} + l \times \mathsf{sz\_cmp}(t_l[bitmap]), \tag{5.6}$$

where $\mathsf{siz}_{\mathsf{RD1}}$ is the size of the response of RDAS1, as given in Eq. (4.3).

Our experiments (presented later) clearly show gains in both communication and computational cost if compression is used.

## 5.5 Dynamic databases

We argued that there exist scenarios in which it may not be necessary to handle data updates, for instance, the case of data warehousing applications. Thus having a scheme which is valid only for static databases is useful. Most work in the literature on authenticated query processing focus on static databases. The proposals for dynamic scenario are quite few [6, 49, 59]. However, our proposal can be extended to dynamic scenarios also. Here we discuss the issues related to extending RDAS2 for dynamic scenarios, we shall call the new scheme as RDAS2-dyn. We equip RDAS2-dyn to handle all possible kinds of updates in a outsourced relation: insertion, deletion and updates (by an update we would mean a change in some field of an existing row).

We build RDAS2-dyn over the version of RDAS2 equipped with handling authenticity of empty replies. The basic differences of RDAS2-dyn from the baseline RDAS2 are described next.

Both $R_\alpha$ and $R_\beta$ produced by RDAS2-dyn.$\Phi$ contain a new attribute, i.e., if $B$ be the set of original attributes then for RDAS2-dyn, $R_\alpha$ will contain the attributes $B \cup \{\texttt{Nonce}, \texttt{Ucount}, \texttt{Tag}\}$ and $R_\beta$ will contain the attributes $\{\texttt{Name}, \texttt{SearchKey}, \texttt{Bitmap}, \texttt{RowNo}, \texttt{Ucount1}, \texttt{Tag1}\}$ respectively. These new attributes Ucount and Ucount1 would keep a record of the number of updates that take place in the rows of the relations $R_\alpha$ and $R_\beta$. This information would be necessary to ensure the *freshness* of the query results, a property that we have ignored for the schemes that we have so far discussed, as freshness is only meaningful in the dynamic scenario.

RDAS2-dyn.$\Phi$ populates the relations $R_\alpha$ and $R_\beta$ in almost the same way as depicted in Figure 4.1, except that it sets $t[\texttt{Ucount}]$ and $t[\texttt{Ucount1}]$ to zero for all tuples $t$ in $R_\alpha$ and $R_\beta$ respectively, and while calculating the MAC, it encodes the attributes Nonce and Ucount in $R_\alpha$ and the attributes RowNo and Ucount1 in $R_\beta$ as strings of fixed lengths, we consider for concreteness that these attributes are 32 bit strings.

In addition to $R_\alpha$ and $R_\beta$, RDAS2-dyn.$\Phi$ produces $M_s$ and $M_c$, $M_s$ consists of $(\mathsf{Lst}_b, \mathsf{lstTag}_b)$ for all attributes $b \in B$ and $M_c$ consists of nT the *current* number of tuples present in $R_\alpha$, $N$ the current number of tuples in $R_\beta$ a bit string mask of size nT, and two arrays $\mathsf{UpdateList}_\alpha$ and $\mathsf{UpdateList}_\beta$ of size nT and $N$ respectively, both initialized to zero. The value of the $i$-th element in $\mathsf{UpdateList}_\alpha$ represents the number of updates that the $i$-th tuple in $R_\alpha$ has undergone. Similar information for $R_\beta$ is kept in the list $\mathsf{UpdateList}_\beta$. Thus, these lists keep track of the number of updates that have taken place in the tuples of $R_\alpha$ and $R_\beta$. Note that nT, mask, $\mathsf{UpdateList}_\alpha$, $\mathsf{UpdateList}_\beta$ change their values with the updates; the sizes of mask, $\mathsf{UpdateList}_\alpha$, $\mathsf{UpdateList}_\beta$ are also dynamic. The bit string mask is used to handle deletions, and the structures $\mathsf{UpdateList}_\alpha$ and $\mathsf{UpdateList}_\beta$ would help in verifying freshness. This would be clear from the discussions that follow.

Next we outline the four main procedures required by RDAS2-dyn for the dynamic scenario: a) Insertion, b) Updates c) Deletion and d) Query processing.

**Insertion** We consider that the client wants to insert a new tuple $t^{new}$ in the outsourced data base. We also consider $J$ to be the set of sensitive attributes in $t^{new}$, where $J = \{J_1, J_2, \ldots, J_{|J|}\} \subseteq B$. For easy exposition let $J$=$B$. To insert $t^{new}$ in the database, the client should give proper instructions to the server to update both $R_\alpha$ and $R_\beta$. The basic steps required for the insert are noted as follows:

1. To insert a tuple corresponding to $t^{new}$ in $R_\alpha$ the client uses the same process as in RDAS2. Regarding the attributes Nonce and Ucount1, it instructs the server to set $t^{new}[\texttt{Nonce}] = \mathsf{nT} + 1$ and $t^{new}[\texttt{Ucount}] = 0$. And the attribute tag is computed using the values of all the attributes in $R_\alpha$.
2. For the insertion, the number of tuples in $R_\beta$ that need to be updated is equal to

the set of sensitive attributes $J$, as with this insertion the bitmaps corresponding to all these attributes would change. There are two distinct cases to handle, firstly it may be so that there is a specific attribute $J_i$ such that $t^{new}[J_i]$ is new, in the sense that no other existing tuple in $R_\alpha$ has this value for the attribute $J$. The second case is that there is no such new value in $t^{new}$. Note, that the client can know which case it has to handle by suitably querying the existing database, and the functionality of authenticating empty replies would help in this.

- For the first case, the client creates a new tuple for $R_\beta$ involving the new value of the attribute by creating the suitable bitmap and computing its MAC, and it sends the tuple to the server.
- For the second case the client gets all the relevant bitmaps corresponding to the values of the new tuple and updates them, computes the MACs for the updated tuples and sends them back to the server. For all the tuples of $R_\beta$ that are updated, the attribute `Ucount1` is incremented by 1.

Note that for an insert, the size of $R_\alpha$ increases by one tuple, hence the size of each bitmap should also increase by one bit. This is explicitly not done, as for all bitmaps except the bitmap for the value which is inserted the extra bit would be a one for all others it would be a zero.

3. The client updates the value of $\mathtt{nT}$ by $\mathtt{nT}+1$. It then updates $\mathsf{UpdateList}_\alpha$ by augmenting it with a new element and setting it to zero, i.e., $\mathsf{UpdateList}_\alpha = 0$. It updates $\mathsf{UpdateList}_\beta$ by incrementing its $i$-th element if the $i$-th tuple of $R_\beta$ has suffered an update. In case, that a new $R_\beta$ tuple has been generated, then the client updates the value of $N$ increasing by the number of new tuples.

**Updates:** This procedure has as main goal to update an attribute $b_i$ in tuple $t$ in the original relation from a value $a$ to $a'$. For the description we assume the following: $t^{R_\alpha}$ is the tuple in $R_\alpha$, where the update is to be made, and $t^{R_\alpha}$ occurs in the $j$-th row of $R_\alpha$. The client gets $t^{R_\alpha}$ and $j$ from the server through a suitable query. Let `tup1` and `tup2` be the answers to the following queries respectively:

SELECT * FROM $R_\beta$ WHERE `NAME`= $b_i$ AND `SearchKey` $= a$
SELECT * FROM $R_\beta$ WHERE `NAME`=$b_i$ AND `SearchKey` $= a'$

The update would be performed by following the steps below:

1. The client updates the attribute $b_i$ with the new value $a'$ in the $j$-th tuple of $R_\alpha$, increments the attribute $t^{R_\alpha}[\mathtt{Ucount}]$, and recomputes $t^{R_\alpha}[\mathtt{Tag}]$ with the up-

dated values and sends this to the server. The client also increments the value of $\mathsf{UpdateList}_\alpha[j]$ by 1.

2. As in case of inserts, for the update in $R_\beta$ there are two cases to handle. The first one being tup2 is not empty and the other one where tup2 is empty.

   - Suppose tup2 is not empty, i.e., there already exists a tuple with value $a'$ for the attribute $b_i$. In this case, the $j$-th bit of tup1[BitMap] would be 1 and the $j$-th bit of tup2[BitMap] would be 0. The client would flip the bits of these two bitmaps, increment the Ucount1 attribute of both these bitmaps and recompute the tag1 attribute and send these updated tuples to the server. Assuming tup1 and tup2 are the $k$-th and $\ell$-th rows of $R_\beta$, the client also increments the values of $\mathsf{UpdateList}_\beta[k]$ and $\mathsf{UpdateList}_\beta[\ell]$ by 1.
   - When tup2 is empty, then there is no tuple in $R_\alpha$ where the attribute $b_i$ bears the value $a'$. In this case we need to insert a new tuple tup in $R_\beta$, such that $\mathsf{tup}[\mathtt{Name}] = b_i$, $\mathsf{tup}[\mathtt{SearchKey}] = a'$, $\mathsf{tup}[\mathtt{Ucount1}] = 0$, and tup[BitMap]$= z$, such that $\mathsf{bit}_j(z) = 1$ and all other bits of $z$ are zero. Using these new values of attributes of tup, the client computes $\mathsf{tup}[\mathtt{Tag1}]$ and sends this new tuple to the server. Moreover as in the previous case the $j$-th bit of tup1[BitMap] is changed from 1 to 0. The client also increments $N$ (the number of tuples in $R_\beta$) by 1 and sets $\mathsf{UpdateList}_\beta[N] = 0$ and increments $\mathsf{UpdateList}_\beta[k]$ by 1.

**Deletion** This procedure deletes a tuple $t^{R_\alpha}$ in $R_\alpha$. We handle deletions by the use of a bit string mask. The size of mask is nT bits, and initially it is an all 1 bit string. We do not physically delete tuples from $R_\alpha$. Whenever the $j$-th tuple in $R_\alpha$ is required to be deleted then the $j$-th bit of mask is changed to zero.

**Query** The query processing in RDAS2-dyn remains the same as in case of RDAS2 except by three small changes in the Verification Process $\mathcal{V}$ at the bitmaps operation phase:

- Before verifying the $R_\alpha$ and $R_\beta$ tuples, the client checks that their Ucount and Ucount1 values matches with $\mathsf{UpdateList}_\alpha$ and $\mathsf{UpdateList}_\beta$ values respectively, which are stored in the client. By this step the freshness property is guaranteed.
- Before operating the bitmaps, the length must be updated to exactly the one specified by nT.
- An extra operation *AND* with the mask array is required. This is because in case some tuples had been deleted their values are not presented to the client.

We described the basic ideas involved in adapting RDAS2 to a dynamic scenario. We have some things which are left to do in this regard, the security of RDAS-dyn needs to be analyzed in a different model, since the security definition in 4.1 is not adequate for the dynamic scenario.

## 5.6 Further Extensions and Improvements

In this section we discuss several other ways in which the RDAS framework can be extended.

### 5.6.1 Including other query types

**Efficient range query processing:** Range queries can be handled by RDAS2 by posing a range query with several selects. But this may not be efficient. An efficient way to handle range queries would be to store range bitmaps [67, 68]. The type of bitmaps that we have used so far are called equality bitmaps, and they are good for select queries. Using range bitmaps one can encode range information of the attribute values and thus would be well suited for processing range queries. Adding other bitmap encodings in our system can be done in a straightforward way, and would increase the functionality at the cost of storage.

In the next chapter we discuss in details some novel range encoding schemes for bitmaps in the context of database privacy. These encodings can also be used in the context of authenticated query processing. But a detailed study in this regard is necessary, which we are yet to do.

**Projection and aggregation queries:** Projection and aggregation queries cannot be handled directly by our protocol. But, projections and aggregations can always be done in the client side if these functionalities are required. And this can also be added to the system, without hampering its security properties.

**Join queries:** We have not discussed the functionality of join processing for our schemes. But basic join processing (like equijoins) can again be obtained in a straightforward way by storing extra information. In particular, join bitmaps [72, 73] can be used for this purpose.

### 5.6.2 Multiuser settings

As stated earlier, we use a symmetric key primitive as the main cryptographic object, thus our scheme is restricted to a single user setting, i.e., the data owner is the one who queries the database. We can replace the message authentication scheme with a public key signature scheme to extend the scheme for multiple queriers. We see no problem in doing this, but for a multiuser setting, the security definition that we give for RDAS would no more be valid, and one needs to extend this definition. Using signatures, the computational cost would in general be more than that in using MACs. The functionality of aggregation can also be used (as we used for MACs) to reduce communication costs.

## 5.7 Discussions and Comparisons

As stated earlier there are other schemes available in the literature which achieves the functionality of authenticated query processing. The main novelty of our scheme is in the use of bitmaps and ability to work on queries other than range queries, moreover we try to analyze security in formal terms and also link the security of the scheme with the security of the message authentication code.

In this section we aim to provide a comparison of RDAS1/RDAS2 schemes with the other existing schemes, and also point out the limitations in our schemes. Providing an experimental comparison of our schemes with the existing ones is beyond the scope of this work. Some authors (for example [6]) compare the storage and communication cost of the protocols in asymptotic terms, which also do not reveal the complete picture. We would point out some of the differences of our framework in comparison to the others with respect to functionality and costs.

As stated earlier, most existing schemes use an additional authenticated data structure to ensure completeness. The additional data structure is usually a tree, in its barest form a Merkle hash tree is used [74, 75]. In some cases variants of B+ trees have also been used as the additional data structure [49, 48, 76]. The other significant direction is use of signatures. By signing the individual tuples with a secure signature scheme, as it was done in [48, 61] one can ensure correctness but not completeness. To attain completeness only with signatures a method in [58] was proposed, where chains of signatures are constructed based on a specific sort order of the attribute values. Such types of signatures allow verification of completeness for range queries, without the use of additional data structures. Signature aggregation has also been used to reduce

the size of the responses [61, 77, 58, 76]. In what follows we summarize some of the salient features of the previous works available in the literature and compare and contrast it with the RDAS framework.

**Type of queries:** In the literature, all basic schemes are designed to handle mostly range queries on numerical attributes. In most cases, to handle additional query types, drastic changes in the basic schemes are required. For most tree based schemes, even handling tuples with repeated values of attributes needs special treatment [58, 76]. In some tree based schemes join processing is achieved in rather a straight forward way [76, 78, 49, 59], which leads to significant increase in the size of the query responses and also the stored data in the server side. In [52] a special tree structure called authenticated aggregation B tree was specifically designed to handle aggregation queries, other tree based schemes cannot handle aggregation queries as a part of the scheme.

The signature based schemes like in [58, 76] which uses chain signatures are mainly designed for range queries and are not efficient for other types of queries. It has been claimed in [76] that joins can be handled, but here too the computational costs and the size of the responses would be prohibitively large.

RDAS2 can handle selects involving arbitrary Boolean operations. These query types can be modified to handle range queries, without any extra overhead. In the recent years other efficient bitmap encodings, like the range encodings [67, 68] have been proposed. Such bitmap encoding can provide functionality of range queries in a more efficient way. Such extensions are straightforward and can be easily implemented. Projections and aggregations cannot be handled directly by the protocol, but they can be implemented in the client side without any difficulties. Moreover, simple join queries can also be accommodated if some additional information is stored through join bitmaps [72, 73]. Hence the spectrum of query types that can be handled by the RDAS framework is comparable to the existing schemes.

In the following discussion we will denote the number of rows in a database by $n$ and the number of attributes by $m$. By $n_q$ we shall mean the number of tuples included in a correct and complete response of a query $q$, and $l_q$ will denote the number of bitmaps involved in a query $q$.

**Storage Costs:** Tree based schemes need to store a tree for each attribute. Asymptotically, a tree based scheme (such as the one described in [49, 76]) requires $O(mn)$ extra storage. Whereas in case of RDAS1 we require $O(n+N)$ storage, where $N$ is the cardinality of all the attributes present. We know that $N \leq mn$, but in most scenarios $N$ is smaller than $mn$, and in certain scenarios $N << mn$. In case of the basic RDAS2, we require $O(n+Nn)$ storage as we require to store the bitmaps also corresponding to each

attribute and its value. Assuming $N = O(mn)$, we have the asymptotic storage requirement for RDAS2 as $O(mn^2)$, but the constants involved in this are much smaller than in the case of tree based solutions, and this is an extreme overestimate for databases with low cardinality attributes. Moreover, in case of RDAS2 the bitmaps can be compressed. It is difficult to give a proper estimate of the amount of compression that can be achieved in general. But, it is clear that when the attribute cardinalities increases, the bitmaps become more sparse (in the sense that they would have lower Hamming weights), which would allow better compression. In case $N = mn$, each bitmap would have only one 1 and rest zeros, and can be encoded in constant length irrespective of the number of rows in the database. Thus, it is expected that with increase in $N$ the encoded size of the bitmaps decreases. In general terms, it has been said (for example in [79]) that the sizes of the compressed bitmap indices are relatively small compared with the typical B-tree indices. This is true even for attributes with very high cardinalities.

Let us see a simple comparison between B-tree and equal bitmap encoding, a bitmap index on an attribute $a$ of a table $R$ requires in space $\frac{nN}{8}$ bytes. On the other hand, building a B-tree on the same attribute requires about $\frac{1.44 \times n}{M} \times p$ bytes, where $p$ is the page size, and $M$ is the degree of the B-tree [80, 81]. Now let us assume that $p = 4K$ and $M = 512$ and the cardinality of $a$ is smaller than 93 then building a equal encoding bitmap is cheaper in size that building a B-tree. The schemes based on signature chains or aggregate signatures like [58, 59] also use $= O(mn)$ storage.

**Query execution costs:** The tree based schemes (as [49]) do not have any extra computational overhead in query execution. This is also true for the basic RDAS1 and RDAS2. In case of RDAS2 $-$ agg, for query execution the tags should be aggregated, thus, requires a number of xor operations which grows linearly with the response size. But in tree based schemes, answering a query requires traversing the tree to find the relevant node, and for multiple attributes this has to be done in multiple trees, and there exists no trivial way to combine the trees corresponding to different attributes. In RDAS, query execution is simpler, moreover the bitmaps stored in case of RDAS2 can even act as indexes and thus make query execution further efficient. In case of schemes using signature chains [58, 59], the signatures are also pre-computed, hence additional computation in query execution is not required. But schemes which use aggregated signatures are required to be aggregated for responding queries. In such a scenario, the server needs to aggregate $O(n_q)$ signatures. It is to be noted that aggregating signatures is much more costly than aggregating MACs. For example if RSA signatures are used then to aggregate two signatures one requires a modular multiplication modulo the RSA modulus (which should be at least 1024 bits long).

**Query Verification costs:** The schemes based on trees structures [49] require to compute $O(mn_q)$ hashes plus one signature to verify a query. In the case of [76], $O(mn_q)$ signatures are required to be computed, which is very expensive. On the other hand, RDAS1 and RDAS2 requires the computation of $O(n_q + l_q \cdot n)$ tags, where $l_q$ is the number of bitmaps involved in the query. RDAS2 only differs from RDAS1 in that it is necessary to build the involved bitmaps. In case of schemes using signature chains ([58, 59]) the cost is similar to the tree approach.

**Communication costs:** In RDAS1 and RDAS2, the extra communication cost is $O(n_q + n \cdot l_q)$. Once more the difference between RDAS1 and RDAS2 is that in RDAS2 the communication costs grows because bitmaps are also sent as part of the response. The extra cost can be reduced by compression as in RDAS2-cmp. In case of RDAS1-agg the extra overhead is constant, since just two aggregated tags are sent irrespective of the response size. Finally in RDAS2-agg the object verification size has a bound of $O(n \cdot l_q)$ where $l_q$ is the number of bitmaps involved in the query. The schemes based on trees [49] have a communication cost of $O(\log n)$ hashes that need to be sent. The scheme in [76] has a constant communication cost, as only an aggregate signature is sent here irrespective of the response size. The schemes described in [59, 58] need to send $O(mn_q)$ hashes plus one aggregated signature.

## 5.8 Summary

In this chapter we presented some improvements to our basic construction RDAS1, and provided a detailed comparison of our framework with the other existing schemes in the literature. In this chapter we end our study on authenticated query processing, and in the next chapter we begin our study of privacy in databases.

# Chapter 6

# ESRQ1: A Scheme to Provide Privacy in Relational Databases

> Every child is an artist. The problem is how to remain an artist once he grows up.
>
> *Pablo Picasso*

Till now we have concentrated on the query authentication problem in outsourced databases. In this chapter we deal with privacy. In Section 3.4 some of the intricacies involving privacy in databases were discussed. It was also stated that we only attempt to address the problem of running range queries in an encrypted database. In the existing literature, it has been proposed that this problem could be handled if the database entries are encrypted using order preserving encryption schemes. Several order preserving encryption schemes are available in the existing literature [9, 10, 11, 12, 13, 64, 65], but as Popa et.al. [14] recently suggest, none of them has achieved the ideal IND-OCPA security, they either leak more than just the order of plaintext values or propose other security notions weaker than the IND-OCPA notion. Also, there does not exist any comprehensive security analysis of the scenario when the above mentioned schemes are applied in a real database.

We see the problem of executing range queries in an encrypted database from a different direction. The main component of our solution is not to use any special type of encryption scheme, i.e., the encryption scheme we use is not an OPE scheme. We encrypt the tables with a deterministic encryption scheme, additionally we maintain some extra data in the server side which contains just the order information of the various attribute values. Both the encrypted table and the additional data are kept with the server, and the server can respond to range queries without the knowledge of the real attribute values in the table. Our solution has some relationship with the philosophy adopted in [14], where encryption was done using a deterministic encryption scheme

and in addition to the ciphertext a special order preserving encoding of the plaintext was used to reveal the order. But the details of our solution are completely different from [14]. A novel component of our proposal is that it uses bitmap indices to encode the order information of the attributes, instead of using a tree based data structure as Popa et al. in [14] did (see section 3.5). The specific form of encoding that we use is more suitable for use in the context of relational databases. Moreover, our encoding enables more efficient query processing. Also as will be evident from the following discussions, we use a cryptographic primitive that provide the exact security required for encrypting databases. The specific primitive we use allow us to have a light weight protocol.

As we did in the case of the problem of authenticated query processing, here we propose a generic framework for a database encryption scheme supporting range queries (ESRQ), and define its syntax and security notion. We also provide a complete scheme called ESRQ1.

The main cryptographic component required in our scheme is a deterministic encryption scheme **E**. **E** has some curious requirements, firstly it should support associated data, i.e., it receives as input not only a plaintext but a public associated data (also called tweak). Moreover we require **E** to be secure against deterministic chosen plaintext adversaries (CPA secure). Though deterministic encryption schemes which provide stronger security against chosen ciphertext adversaries (CCA secure) exist [82, 83, 84], but the possibility of a more efficient deterministic scheme which is only CPA secure has not been adequately explored in the literature. Though, it has been acknowledged [14, 63] that CPA secure deterministic schemes are required for database encryption, but still it suggests the use of schemes which provide stronger security. We fix some efficient schemes that provide the required security. In particular we propose a class of constructions called OHCTR which are deterministic CPA secure and are more efficient than CCA secure deterministic schemes. These schemes can be of independent interest, and are described separately in Chapter 7.

One interesting property of ESRQ1 is that it supports updates and inserts in the database. To enable this functionality within ESRQ1, we required to device a specific data structure to store bitmaps. This data structure allows efficient updates and also results in a short representation of bitmaps. This specific data structure can also be of independent interest.

In what follows, in Section 6.1, we introduce some additional notations and also briefly discuss the syntax and security of deterministic encryption schemes supporting associated data (DEAD). In Section 6.2 we discuss the syntax of a ESRQ scheme along with the security notion. In Section 6.3 we introduce our proposed structure to store bitmaps

and its properties. Finally, in Section 6.4 we present our specific construction ESRQ1 that allows to encrypt a database in such a way that range queries can be performed.

## 6.1 Additional Notations

**Lists and arrays**. We will require dynamic lists for our descriptions. For our description we will see the lists as arrays, though other implementation options are possible. In most cases the lists that we will use would be sorted in the ascending order, i.e., if $\mathcal{L}$ is a list with $n$ elements then

$$\mathcal{L}[1] \leq \mathcal{L}[2] \leq \cdots \leq \mathcal{L}[n].$$

Given a sorted list $\mathcal{L}$ we will frequently use some operations on $\mathcal{L}$. We describe this operations next, with the help of an example list $\mathcal{L}_1 = \langle 3, 9, 19, 26, 52, 76, 111 \rangle$.

> $\mathsf{len}(\mathcal{L})$, will denote the number of elements in $\mathcal{L}$, for example, $\mathsf{len}(\mathcal{L}_1) = 7$.
>
> $\mathsf{exists}(\mathcal{L}, \mathsf{val})$ returns 1 if val exists in the list $\mathcal{L}$ and returns a 0 otherwise. For example, $\mathsf{exists}(\mathcal{L}_1, 26) = 1$, and $\mathsf{exists}(\mathcal{L}_1, 25) = 0$.
>
> $\mathsf{position}(\mathcal{L}, \mathsf{val})$ returns the position of val in $\mathcal{L}$. Note, if val is not present in $\mathcal{L}$, then also it returns a value. For example, $\mathsf{position}(\mathcal{L}, 19)$ returns 3, and $\mathsf{position}(\mathcal{L}, 10)$ also returns 3 signifying that though 10 does not exists in $\mathcal{L}_1$, but if it is inserted in $\mathcal{L}_1$ it will have the position 3.
>
> $\mathsf{Insert}(\mathcal{L}, i, \mathsf{val})$, inserts val in the $i^{th}$ position of $\mathcal{L}$. This operation increases the number of elements in $\mathcal{L}$ by 1. For example, $\mathsf{Insert}(\mathcal{L}, 2, 5)$ changes the original list $\mathcal{L}_1$ to $\langle 3, 5, 9, 19, 26, 52, 76, 111 \rangle$.

**Deterministic Encryption Schemes:** A deterministic encryption scheme with associated data (DEAD) is a deterministic function $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{C}$, where $\mathcal{K}$, $\mathcal{T}$, $\mathcal{M}$ and $\mathcal{C}$ are the key space, tweak (associated data) space, message space and cipher space, respectively. Thus, $\mathbf{E}$ receives as input a key $K \in \mathcal{K}$, a tweak $T \in \mathcal{T}$ and a message $M \in \mathcal{M}$ and produces as output a cipher $C \in \mathcal{C}$. We shall often write $\mathbf{E}_K(T, M)$ instead of $\mathbf{E}(K, T, M)$. $\mathbf{E}$ also has an inverse function $\mathbf{D} : \mathcal{K} \times \mathcal{T} \times \mathcal{C} \to \mathcal{M}$, such that for every $K \in \mathcal{K}$, every $T \in \mathcal{T}$ and every $M \in \mathcal{M}$, $\mathbf{D}_K(T, \mathbf{E}_K(T, M)) = M$. This implies that for every $T \in \mathcal{T}$ and every $K \in \mathcal{K}$, $\mathbf{E}_K(T, \cdot) : \mathcal{M} \to \mathcal{C}$ is an injective function.

To define security of $\mathbf{E}$ we consider an adversary $\mathcal{A}$ which is given an oracle $O$. The oracle $O$ can either be $\mathbf{E}_K(\cdot, \cdot)$, i.e., the encryption scheme which is instantiated with a

uniform random key $K$ selected from $\mathcal{K}$, or it can be $\$(\cdot,\cdot)$, i.e., it is an oracle which on input $(T,M) \in \mathcal{T} \times \mathcal{M}$ outputs a random string of size $\mathbf{E}_K(T,M)$. $\mathcal{A}$ is allowed to query its oracle and decide with whom it is interacting. One important restriction of $\mathcal{A}$ is that is not allowed to repeat any query. We formally define the advantage of $\mathcal{A}$ as

$$\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{A}) \;=\; \left|\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1]\right|. \tag{6.1}$$

We define $\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}{}_{(q,\sigma,t)}$ by $\max_A \mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}{}_{(A)}$ where maximum is taken over all adversaries that make at most $q$ queries having at most $\sigma$ many blocks and run for time at most $t$. We consider an encryption scheme to be $(\varepsilon,q,t)$ det-cpa secure if $\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}{}_{(q,\sigma,t)} \leq \varepsilon$.

This syntax and the security notion of DEAD would be enough for us to describe the basic ESRQ scheme in this chapter. In the next chapter we discuss more details of DEAD, including some practical instantiations.

## 6.2 Encryption Scheme Supporting Range Queries (ESRQ): Definitions and Basic Notions

An encryption scheme supporting range queries (ESRQ) consists of a tuple of algorithms $(\mathcal{G}, \mathsf{Enc}, \Phi, \Psi, I, \mathcal{U}, \mathsf{Dec})$, which work as follows.

The **key generation algorithm** $\mathcal{G}$ runs at the client side and outputs a set of keys that are randomly selected from a pre-specified key space. We will denote the key space by $\mathcal{K}$.

The **privacy transform** $\mathsf{Enc}$ receives as an input a relation $R$ and the set of keys generated by $\mathcal{G}$. This transform generates a new relation called $R'$ along with some additional data $(M_s, M_c)$ . We denote this operation as $(R', M_s, M_c) \leftarrow \mathsf{Enc}_K(R)$. This transform is executed in the client side. The resulting relation $R'$ is an encrypted table.

The **query translator** $\Phi$ is an interactive procedure that runs between the client and the server. It takes in the key $K$ and a query $q$ meant for the relation $R$ and converts it to a query $q'$ which can be executed in $R'$. For convenience, we shall sometimes refer to a query meant for $R$ to be an $R$-query and a query meant for $R'$ as $R'$-query. Thus $\Phi$ converts an $R$-query to an $R'$-query with the help of a key.

The **response procedure** $\Psi$ runs at the server side. $\Psi$ allows the server to answer a query $\Phi(q)$ on $R'$ . The output of $\Psi$ for a query is the response of the server,

called ρ. ρ contains the encrypted answer that should be decrypted at the client side. The server returns this reply to the client.

The **Insert procedure** $I$ is an interactive procedure which inserts a tuple $t$ to the relation $R$. The procedure takes in a tuple $t$ and $I(t)$ does the necessary updates in $R'$ and $M_s, M_c$.

The **Update procedure** $\mathcal{U}$ is an interactive procedure which updates a tuple $t$ already present in $R$. The procedure takes in the original tuple $t_o$ and the new tuple $t_n$, and $\mathcal{U}(t_o, t_n)$ does the necessary updates in $R'$ and $M_s, M_c$.

The **decryption procedure** is a keyed transform $\mathsf{Dec}_K$ which runs in the client. It receives as input the server response ρ and outputs the answer ans for the query $q$.

A client who wants to store a relation $R$ in an un-trusted server, executes $\mathsf{Enc}_K(R)$ using $K \in \mathcal{K}$. The transform Enc produces a relation $R'$ along with $(M_s, M_c)$. The relation $R'$ and $M_s$ is stored in the server and $M_c$ and the keys are retained in the client side. The client can pose an $R$-query, the query is transformed to a query $R'$-query using $\Phi$, and it is sent to the server. Then the server answers the $R'$-query through the $\psi$ procedure. Finally, the answer ρ is sent to the client to be decrypted. The client performs the decryption procedure $\mathsf{Dec}_K$ and recovers the answer ans for the $R$-query. The client is also allowed to update the database by inserting more tuples in it or by changing some existing tuples.

### 6.2.1 Security Notion

To define the security notion, we first define some important concepts.

**Definition 6.1.** *Given a relation R over a set of attributes A, we classify the attributes in two different classes: the ones where a range query is valid and the ones where such queries are not valid. Define* $\mathsf{ty} : A \to \{0, 1\}$, *where for any* $a \in A$, $\mathsf{ty}(a) = 0$, *if range queries on the attribute a is not applicable and* $\mathsf{ty}(a) = 1$, *otherwise. An attribute* $a \in A$, *such that* $\mathsf{ty}(a) = 1$ *is called a range attribute.*

**Definition 6.2.** *Two relations R and S are said to be equivalent (denoted by* $R \approx S$*) if the following conditions hold.*

1. *R and S are defined over the same set of attributes A.*
2. *R and S contains the same number of tuples* nT.

3. *For every* $i, j \in \{1, 2, \ldots, \mathsf{nT}\}$, *and every* $a \in A$, *such that* $\mathsf{ty}(a) = 1$,

$$t_i^R[a] \geq t_j^R[a] \iff t_i^S[a] \geq t_j^S[a].$$

4. *For every* $i \in \{1, 2, \ldots, \mathsf{nT}\}$, *and every* $a \in A$, $|t_i^R[a]| = |t_i^S[a]|$.

**Game** $\mathrm{ESRQ}_{\Upsilon}^{\mathcal{A}}$

1. The challenger selects $K \xleftarrow{\$} \mathcal{K}$
2. $\mathcal{A}$ selects two relations $R_0, R_1$
   such that $R_0 \approx R_1$, and gives them to the challenger.
3. The challenger selects a bit $b \xleftarrow{\$} \{0, 1\}$
4. The challenger computes $(R', M_s, M_c) \leftarrow \Upsilon.\mathsf{Enc}_K(R_b)$, and publishes $(R', M_s)$ .

Now the adversary $\mathcal{A}$ can perform inserts and updates as follows:

5. For an insert $\mathcal{A}$ submits two tuples $(t_*^0, t_*^1)$,
   such that $R_0 \sqcup t_*^0 \approx R_1 \sqcup t_*^1$. The challenger executes $\mathsf{Ins}(t_b)$
6. For an update, $\mathcal{A}$ selects a $1 \leq i \leq \mathsf{nT}$, and two tuples $(t_*^0, t_*^1)$,
   such that $R_0 \uplus (t_i^{R_0}, t_*^0) \approx R_1 \uplus (t_i^{R_1}, t_*^1)$. The challenger executes $\mathcal{U}(t_i^{R_b}, t_*^b)$
7. $\mathcal{A}$ outputs a bit $b'$.
8. **if** $b = b'$ **output** 1
9. **else output** 0

Figure 6.1: Game used to define security of ESRQ

A tuple can be considered to be a relation with only one row. Thus we will sometimes talk of the equivalence of two tuples in the above sense.

Moreover given a relation $R$, we will denote by $R \sqcup t$ the new relation obtained by inserting $t$ to $R$. And, $R \uplus (t_o, t_n)$ will denote the new relation obtained by changing the tuple $t_o$ in $R$ to $t_n$.

The basic goal of ESRQ is to transform $R$ to $R'$ in such a way that $R'$ should not contain any information beyond the order relation between the attribute values. We formally define security of ESRQ $\Upsilon$ as a game between an adversary and a challenger, the interaction between these two entities is described in Figure 6.1.

In the game described in Figure 6.1, the challenger first selects the keys that would be necessary for ESRQ $\Upsilon$. Next, the adversary chooses two equivalent relations $R_0$ and $R_1$, such that $R_0 \approx R_1$, and submits these relations to the challenger. The challenger

selects a bit $b$ uniformly at random and applies $\Upsilon.\mathsf{Enc}_K(R_b)$ to obtain $(R', M_s, M_c)$ and reveals $(R', M_s)$ to $\mathcal{A}$. The task of $\mathcal{A}$ is to guess the bit $b$ chosen by the challenger, i.e., to guess which of the relations $R_0$ or $R_1$ was encrypted by the challenger. The adversary can additionally seek updates or inserts to the original relation. For seeking inserts, it needs to submit two tuples $(t_*^0, t_*^1)$ where $t_*^0$ and $t_*^1$ are meant to be inserted in $R_0$ and $R_1$ respectively. Further it is required that if these tuples are inserted in $R_0$ and $R_1$ then also the relations remain equivalent. Similarly, the adversary can ask for updates where he chooses a tuple to change and supplies the new tuple for both the relations. It is again required that after the update both relations $R_0$ and $R_1$ should be equivalent. Note that with inserts and updates being allowed, the relation $R'$ and $M_s$ are dynamic and they change with each update sought by the adversary, and the adversary is allowed to seek multiple inserts/updates. Finally, the adversary outputs a bit $b'$ and the game returns a 1 if $b = b'$ and returns a 0 otherwise.

**Definition 6.3.** *The advantage of an adversary $\mathcal{A}$ in attacking an* ESRQ $\Upsilon$ *is defined as*

$$\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(\mathcal{A}) = \left| \Pr[\text{ESRQ}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2} \right|.$$

The relevant resources of an adversary attacking an ESRQ scheme is its running time, cell complexity and query complexity, which we define next:

1. The running time of the adversary is the time spent by $\mathcal{A}$ in the $\text{ESRQ}^{\mathcal{A}}$ game, before it stops.

2. The cell complexity of $\mathcal{A}$ is computed in the following manner. Suppose the initially chosen relations $R_0$ and $R_1$ has $n_1$ attributes and $n_2$ tuples. Further the adversary asks for $n_i$ many inserts and $n_u$ many updates. Then the cell complexity $q$ of $\mathcal{A}$ would be $q = n_1(n_2 + n_i + n_u)$.

3. The query complexity is the total number of $n$ block queries that the adversary makes, where $n$ is a previously chosen parameter. Note that the cell complexity of $\mathcal{A}$ is the total number of cells queried by it. If the $i^{th}$ cell is of size $s_i$, then the query complexity of $\mathcal{A}$ is computed as

$$\sigma_n = \sum_{i}^{q} \lceil s_i/n \rceil.$$

We define $\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(q, \sigma, t) = \max \mathbf{Adv}_{\Upsilon}^{\text{esrq}}(\mathcal{A})$, where the maximum is taken over all adversaries $\mathcal{A}$ which run for time at most $t$ and have cell complexity and query complexity of at most $q$ and $\sigma_n$, respectively. Moreover, we say that an ESRQ scheme $\Upsilon$ is $(\varepsilon, q, \sigma_n, t)$ secure, if for all adversaries which run for time at most $t$, $\mathbf{Adv}_{\Upsilon}^{\text{esrq}}(q, \sigma_n, t) \leq \varepsilon$.

This definition bears lot of similarity with the IND-OCPA security definition for encryption schemes. The definition implicitly allows the encryption procedure to leak the order of the plaintext. This is done through the restriction that the initial relations $R_0$ and $R_1$ chosen by the adversary are such that $R_0 \approx R_1$. As per Definition 6.2, if two relations are equivalent then the values in the tables bear the same order relation (see points 3 and 4 in the definition). Thus, even if the encryption procedure leaks the order of the plaintexts then also the adversary would not be able to distinguish between the two encrypted relations. The concept of equivalence that we use here is an adaptation of the IND-OCPA definition [65] for relational databases.

## 6.3 Matrix representation of bitmaps

In this section we discuss some specific representations and operations on bitmaps which would help us in the description of a specific encryption protocol ESRQ1.

**Column Ordered Matrices.** Let $\mathcal{M}$ be a $m \times n$ matrix. We will denote the entry in the $i^{th}$ row and $j^{th}$ column by $\mathcal{M}(i, j)$. For $1 \leq j \leq m$ and $1 \leq i \leq n$, $\mathsf{Col}_{\mathcal{M}}(j)$ and $\mathsf{Row}_{\mathcal{M}}(i)$ will denote the $j^{th}$ column and $i^{th}$ row of $\mathcal{M}$, respectively.

**Definition 6.4.** *Let $\mathcal{M}$ be a $m \times n$ bit matrix. A column $j$ $(1 \leq j \leq n)$ of $\mathcal{M}$ is said to be* ordered *if*

$$\mathcal{M}(1, j) \leq \mathcal{M}(2, j) \leq \cdots \leq \mathcal{M}(m, j).$$

**Definition 6.5.** *A matrix $\mathcal{M}$ is said to be* column ordered *if all its columns are ordered.*

If $\mathcal{M}$ is a column ordered matrix of bits, then each column would contain a block of zeros followed possibly by a block of ones, where either block can be empty. For example, the matrix:

$$\mathcal{M}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{6.2}$$

is a column ordered bit matrix.

**Definition 6.6.** *Let $\mathcal{M}$ be a $m \times n$ column ordered bit matrix. For $1 \leq j \leq n$, define $\mathsf{one}(\mathcal{M}, j) = i$, where $i \leq m$ is the smallest positive integer such that $\mathcal{M}(i, j) = 1$. If no such $i$ exists, then we define $\mathsf{one}(\mathcal{M}, j) = 0$.*

For example, we have $\mathsf{one}(\mathcal{M}_1, 1) = 3$, $\mathsf{one}(\mathcal{M}_1, 3) = 0$, etc.

A $m \times n$ column ordered bit matrix $\mathcal{M}$ can be represented by an array $\mathsf{arr}_{\mathcal{M}}$ of size $n$, where

$$\mathsf{arr}_{\mathcal{M}}[i] = \mathsf{one}(\mathcal{M}, i), \text{ for } 1 \leq i \leq n.$$

The array representation of the matrix $\mathcal{M}_1$ in Eq. (6.2) would be $\mathsf{arr}_{\mathcal{M}_1} = \langle 3, 2, 0, 4, 3, 5 \rangle$.

Note that each entry of $\mathsf{arr}_{\mathcal{M}}$ is a non negative integer less than $m+1$. Thus, this representation would occupy $n \lg m$ bits contrary to the usual bit representation which occupies $mn$ bits.

For our schemes described later, we would require to store bitmaps of attributes and operate on them efficiently. We discuss some of these issues next.

**Definition 6.7.** *Given a relation $R(A)$ and an attribute $a \in A$. Let $\mathcal{L}$ be a sorted list (in increasing order) of the values in $\mathsf{Dom}_R(a)$. We call $\mathcal{L}$ as the ordered list of $a$ for $R$. We denote the length of $\mathcal{L}$ as $\mathsf{len}(\mathcal{L})$. Hence, $\mathsf{len}(\mathcal{L}) = \#\mathsf{Dom}(a)$.*

**Definition 6.8.** *Let $R(A)$ be a relation with $\mathsf{nT}$ rows. Let $a \in A$ be an attribute and $\mathcal{L}$ be the ordered list of $a$ in $R$. Let $\mathcal{M}$ be a $\#\mathsf{Dom}(a) \times \mathsf{nT}$ bit-matrix such that for $1 \leq i \leq \mathsf{len}(\mathcal{L})$, $\mathsf{Row}_{\mathcal{M}}(i) = \mathsf{BitMap}^{<}(a, \mathcal{L}[i])$. We call $\mathcal{M}$ as the l-bitmap matrix of $a$ for $R$.*

Let us review with an example the above two definitions. Consider the relation $R1$ in Table 6.1, now if we want to construct the ordered list of the attribute age for the relation $R1$, we have to list all the possible values of age in increasing order. Thus the list looks as follows:

$$\mathcal{L}_1 = \langle 17, 18, 33, 36, 52 \rangle$$

The $l$-bitmap matrix of the attribute age for the relation $R1$ is shown in Equation 6.2, to build this matrix we have to include each of the l-bitmaps for each of the values in the ordered list. Thus, the first row of $\mathcal{M}_1$ is the $l$-encoded bitmap of 17, the second row is the $l$-encoded bitmap of 18, etc.

| EmpId | Name | Gender | Level | Age |
|---|---|---|---|---|
| TRW | Tom | M | $L_2$ | 18 |
| MST | Mary | F | $L_1$ | 17 |
| JOH | John | M | $L_2$ | 52 |
| MRH | Mary | F | $L_1$ | 33 |
| ASY | Anne | F | $L_1$ | 18 |
| RZT | Rosy | F | $L_2$ | 36 |

Table 6.1: Relation Employees

**Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$
1. Find $i$ such that $\mathcal{L}[i] = v$
2. **for** $j = 1$ to $\mathsf{nT}$
3. **if** $(\mathsf{arr}[j] > i$ *OR* $\mathsf{arr}[j] == 0)$
4. $b[j] \leftarrow 0$
5. **else** $b[j] \leftarrow 1$
6. **end for**
7. **return** $b$

**Equal**$(\mathcal{L}, \mathsf{arr}, a, v)$
1. Find $i$, such that $\mathcal{L}[i] = v$
2. $b \leftarrow$ **Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$
3. **if** $i = \#\mathsf{Dom}_R(a)$
4. **return** $\overline{b}$
5. **else**
6. $b' \leftarrow$ **Lesser**$(\mathcal{L}, \mathsf{arr}, a, \mathcal{L}[i+1])$
7. **return** $b \oplus b'$
8. **end if**

**Greater**$(\mathcal{L}, \mathsf{arr}, a, v)$
1. $b \leftarrow$ **Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$
2. $b' \leftarrow$ **Equal**$(\mathcal{L}, \mathsf{arr}, a, v)$
3. **return** $b \oplus b'$

Figure 6.2: The procedures to obtain different bitmap encodings for an attribute $a$ and value $v$, given the ordered list $\mathcal{L}$ and an array representation $\mathsf{arr}$ of the $l$-bitmap matrix of the attribute $a$ in relation $R$. **Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$ computes $\mathsf{BitMap}^{<}(a, v)$, **Equal**$(\mathcal{L}, \mathsf{arr}, a, v)$ computes $\mathsf{BitMap}(a, v)$ and **Greater**$(\mathcal{L}, \mathsf{arr}, a, v)$ computes $\mathsf{BitMap}^{>}(a, v)$.

It is easy to verify the following proposition.

**Proposition 6.1.** *Let $R(A)$ be a relation, and $a \in A$ be an arbitrary attribute. If $\mathcal{M}$ be the $l$-bitmap matrix of $a$, then $\mathcal{M}$ is column ordered.*

Hence, the $l$-bitmap matrix of an attribute can be represented with an array as discussed above. Moreover, given $\mathsf{arr}$ (the array representation of the $l$-bitmap matrix) and $\mathcal{L}$ (the ordered list) of any attribute $a \in A$, all bitmap encodings for $a$ corresponding to all values in $\mathsf{Dom}(a)$ can be efficiently computed. Assuming $\mathsf{arr}$, and $\mathcal{L}$ are given, we describe the procedures to obtain the bitmaps $\mathsf{BitMap}^{<}_R(a, v)$, $\mathsf{BitMap}_R(a, v)$ and $\mathsf{BitMap}^{>}_R(a, v)$ in the algorithms **Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$, **Equal**$(\mathcal{L}, \mathsf{arr}, a, v)$ and **Greater**$(\mathcal{L}, \mathsf{arr}, a, v)$ in Figure 6.2.

**Proposition 6.2.** *The procedures in Figure 6.2 are correct. In particular*

1. **Lesser**$(\mathcal{L}, \mathsf{arr}, a, v)$ *computes* $\mathsf{BitMap}^{<}(a, v)$.
2. **Equal**$(\mathcal{L}, \mathsf{arr}, a, v)$ *computes* $\mathsf{BitMap}(a, v)$.
3. **Greater**$(\mathcal{L}, \mathsf{arr}, a, v)$ *computes* $\mathsf{BitMap}^{>}(a, v)$.

*Proof.* In what follows we assume that $\mathsf{arr}$ and $\mathcal{L}$ represents the matrix $\mathcal{M}$, which is the $l$-bitmap matrix for $a$ in $R$.

1. It is easy to see that the procedure **Lesser** just reconstructs the $i^{th}$ row of $\mathcal{M}$ and the $i^{th}$ row of $\mathcal{M}$ is the $l$-encoded bitmap corresponding to the attribute $a$ and value $v$.

2. There are two cases to handle

   (a) If $i = \#\mathsf{Dom}(a)$, then $\mathcal{L}[i]$ is the the largest element in $\mathsf{Dom}(a)$, i.e., $\mathcal{L}[i]$ is the largest value for the attribute $a$ in the relation $R$. Thus, the $i^{th}$ row of $\mathcal{M}$ contains a zero in only those positions where the relation contains the value $\mathcal{L}[i]$, in other positions it contains a 1. Thus the bit-wise complement of the $i^{th}$ row gives the equality bitmap.

   (b) If $i < \#\mathsf{Dom}(a)$, then the $i^{th}$ and the $(i+1)^{th}$ rows of $\mathcal{M}$ differs only in those positions, where the relation contains the value $\mathcal{L}[i]$, thus the bitwise xor of the $i^{th}$ and $(i+1)^{th}$ rows gives the equality bitmap.

3. This follows directly from the fact that the bitwise complement of $\mathsf{BitMap}^{\leq}_{R}(a, v)$, gives $\mathsf{BitMap}^{\geq}_{R}(a, v)$.

□

**Updating *l*-bitmap matrices.** The discussion above shows that if the *l*-bitmap indices of a relation $R$ are stored then it would be enough to answer a variety of range queries on $R$. Here we discuss some issues about updating *l*-bitmap indices when updates or inserts are applied to the original relation $R$.

Consider a relation $R(A)$, and let $a$ be the only range attribute in $A$. Let $\mathcal{L}_a$ and $\mathcal{M}_a$ be the ordered list and *l*-bitmap matrix of $a$ in $R$, respectively. Let $\mathsf{arr}_a$ be the array representation of $\mathcal{M}_a$. With insertions and updates, new values for the attribute $a$ may be introduced which would change the *l*-bitmap matrix for $a$. We are interested in fixing procedures which can update the matrix $\mathcal{M}_a$ incrementally. Consider, that we need to insert a new tuple $t$ in $R$. With this new tuple inserted into $R$, the matrix $\mathcal{M}_a$ changes. The procedure that is required to update the representation $\mathsf{arr}_a$ of the matrix $\mathcal{M}_a$ for this insertion is depicted in Figure 6.3(a). We show an example for the running of these procedures next.

Consider the relation $R1$ in Table 6.1. The attribute age of this relation generates the *l*-bitmap matrix, the array representation arr and the ordered list $\mathcal{L}_1$ depicted in the left side of Table 6.2. Now, suppose that the client wants to insert a new tuple in relation $R1$ where age is equal to 25. According to the procedure in Figure 6.3(a) $\mathsf{position}(\mathcal{L}_1, 25) = 3$. The returned position is equal to 3 because as we explain in the definition of this function (see Section 6.1) even though 25 is a quantity that is not part of the original list, this function outputs the position in which it should be inserted. Since the function $\mathsf{exists}(\mathcal{L}_1, 25)$ returns 0, the l-bitmap matrix $\mathcal{M}_1$ suffers changes in the two dimensions, first a new row is inserted $\mathcal{M}_1(i)(3)$ and also a new column $\mathcal{M}_1(7)(j)$ (see Table 6.2 right side colored in red). Therefore the array representation also changes. The procedure

**Matrix_Update_For_Insert**$(\mathcal{L}_a, \text{arr}_a, t[a])$
1. $p \leftarrow \text{position}(\mathcal{L}_a, t[a])$ //looking for the position
2. **if** $\text{exists}(\mathcal{L}_a, t[a]) = 0$ //if the value not exists
3. **for** $j = 1$ to nT
4. **if** $\text{arr}[j] > p$,
5. $\text{arr}_a[j] \leftarrow \text{arr}_a[j] + 1$
6. **end if**
7. **end for**
8. $\text{Insert}(\mathcal{L}_a, p, t[a])$//inserting the new value
9. **end if**
10. $\text{nT} \leftarrow \text{nT} + 1$ //updating the number of tuples
11. $\text{arr}_a[\text{nT}] = p + 1$
12. **if** $\text{arr}_a[\text{nT}] > \text{len}(\mathcal{L}_a)$
13. $\text{arr}_a[\text{nT}] \leftarrow 0$
14. **end if**

(a)

**Matrix_Update_For_Update**$(\mathcal{L}_a, \text{arr}_a, j, v_N)$
1. $p \leftarrow \text{position}(\mathcal{L}_a, v_N)$ //looking for the position
2. **if** $\text{exists}(\mathcal{L}_a, t[a]) = 0$ //if the value not exists
3. **for** $i = 1$ to nT
4. **if** $\text{arr}[j] > p$,
5. $\text{arr}_a[i] \leftarrow \text{arr}_a[i] + 1$
6. **end if**
7. **end for**
8. $\text{Insert}(\mathcal{L}_a, p, t[a])$;//inserting the new value
9. **end if**
10. $\text{arr}_a[j] \leftarrow p + 1$ //updating the array
11. **if** $\text{arr}_a[j] > \text{len}(\mathcal{L}_a)$
12. $\text{arr}_a[j] \leftarrow 0$
13. **end if**

(b)

Figure 6.3: Insert and update procedures

**Matrix_Update_For_Insert** in lines 4 to 6 suggests a comparison between each of the entries in the $\text{arr}_{\mathcal{M}_1}$ and the position $p$. The ones that are greater than the position are updated. For example, consider the first entry $\text{arr}_{\mathcal{M}_1}[1]$ whose original value is 3, since it is not greater than the position $p$, then the entry $\text{arr}_{\mathcal{M}_1[1]}$ does not change. On the other hand if we consider the entry $\text{arr}_{\mathcal{M}_1}[6]$ whose original value is 5 then an update is required and now this entry will be equal to 6. Also, the new column requires a new entry in the array representation that is equal to $\text{arr}_{\mathcal{M}_1}[7] = p+1$, i.e. 4. Finally, the order list for attribute age is updated and it is shown in right side of the Table 6.2. Notice, that the value 25 was inserted in the third position.

$\mathcal{M}$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

arr

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2 | 0 | 4 | 3 | 5 |

$\mathcal{L}_1 = \langle 17, 18, 33, 36, 52 \rangle$.

$\mathcal{M}$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |

arr

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 2 | 0 | 5 | 3 | 6 | 4 |

$\mathcal{L}_1 = \langle 17, 18, 25, 33, 36, 52 \rangle$.

Table 6.2: Insert example

Another operation that we need to consider is updating an already existing tuple. We consider the update of the $t_j[a]$ in $R$ from a value $v_O$ to $v_N$. The corresponding procedure to update the matrix $\mathsf{arr}_a$ and the list $\mathcal{L}$ is depicted in Figure 6.3(b). This procedure resembles to the insert procedure, the only difference is that instead of creating a new column in the matrix $\mathcal{M}_a$ an existing column will change. First, this procedure determines the correct position $p$ of the new value $v_N$ in the list $\mathcal{L}$. As in the insert procedure we will have two possible cases: the one where the new value $v_N$ exists in the list and the one where $v_N$ corresponds to a value that no other tuple has, thus it does not exists in the list. In the first case, there is nothing to do except for modify the $\mathsf{arr}_a[j]$ by adding 1 to the present value. In the other case, the insertion of the new value in the list $\mathcal{L}$ is required.

This will be clear with an example, consider again our original scenario of relation $R1$ in Table 6.1. In Table 6.3 we show in the right side, the l-bitmap matrix, the array representation, and the ordered list of the age attribute. Now suppose that the client wants to update the age of Anne (tuple 5 in Table 6.1) from 18 to 33. Since the value 33 already exists in the list $\mathcal{L}_1$ the only change that the matrix will suffer is the column $\mathcal{M}_a(i)(5)$, i.e., the colored blue column in Table 6.3. Notice the change in the red column by contrasting it with the blue column of the matrix after the update is performed. Therefore, the entry $\mathsf{arr}_a[5]$ will change by adding 1 to the current value that was 3, see the blue entry in Table 6.3.

$\mathcal{M}$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

arr

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2 | 0 | 4 | 3 | 5 |

$\mathcal{L}_1 = \langle 17, 18, 33, 36, 52 \rangle$.

$\mathcal{M}$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

arr

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2 | 0 | 4 | 4 | 5 |

$\mathcal{L}_1 = \langle 17, 18, 33, 36, 52 \rangle$.

Table 6.3: Update example

## 6.4 ESRQ1: An Encryption Scheme Supporting Range Queries

Here we discuss a scheme ESRQ for encrypting relations such that simple select and range queries can be executed in the encrypted relations. Consider a relation $R(A)$ where $A = \{a_1, a_2, \ldots a_{|A|}\}$, and the function $\mathsf{ty} : A \to \{0, 1\}$ defined on $A$ (see Defini-

tion 6.1 for a definition of ty). We consider that a client wants to outsource this generic relation $R(A)$ to a server. To ensure privacy, the client, encrypts the relation $R(A)$ using ESRQ1 and delegates this outsourced relation instead of the original one. The client will pose queries to the server and expects the server to execute these queries on his/her behalf without knowing the real contents of the relation $R(A)$.

In what follows, we present a generic description of the scheme, also throughout we discuss a specific example based on the relation shown in Table 6.1. The only cryptographic object used by ESRQ1 is a deterministic encryption scheme $\mathbf{E}$ which is secure in the sense of Definition 6.1. We assume that $\mathbf{E} : \mathcal{K} \times \{0,1\}^{\tau} \times \{0,1\}^{*} \rightarrow \{0,1\}^{*}$, i.e., $\mathbf{E}$ has as its key space $\mathcal{K}$, the space of associated data are binary strings of length $\tau$, and the message/cipher space contains arbitrary length binary strings. The specific instantiation of $\mathbf{E}$ in practical scenarios is discussed later in Sections 7.1 and 7.2. Other than the deterministic encryption scheme $\mathbf{E}$, ESRQ1 uses $l$-encoded bitmap indices, which are stored as $l$- bitmap matrices.

The various algorithms involved in ESRQ1 are discussed next in order.

ESRQ1.$\mathcal{G}$: This procedure selects a key $K$ uniformly at random from $\mathcal{K}$. Where $\mathcal{K}$ is the key space of the deterministic encryption scheme $\mathbf{E}$ involved.

ESRQ1.Enc: Given $R(A)$ and the key $K$ as input, ESRQ1.Enc outputs a relation $R_{\alpha}$ along with additional server side data $M_s$ and client side data $M_c$. We assume that $R(A)$ contains nT many tuples and $A = \{a_1, a_2, \ldots, a_{\#A}\}$. To each attribute $a_i \in A$ we associate an unique identifier $\mathsf{id}_i \in \{0,1\}^{\tau}$. Among other possibilities, this identifier can be the (appropriately coded) name of the attribute or a counter. Given an attribute $a \in A$, we shall sometime refer to its unique identifier as $\mathsf{id}(a)$.

$R_{\alpha}$ contains nT tuples and it is defined over the attributes $B = \{\mathtt{RowNo}\} \cup \{b_1, b_2, \ldots, b_{\#A}\}$. Where $b_i = \mathbf{E}_K(\mathsf{id}^*, a_i)$ for some $\mathsf{id}^* \in \{0,1\}^{\tau}$ such that $\mathsf{id}^* \notin \{\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_{\#A}\}$. Hence, $R_{\alpha}$ has one attribute more than in $R$, the extra attribute is RowNo, the other attributes of $R_{\alpha}$ are the encryption of the attribute names in $A$. The specific way in which $R_{\alpha}$ is created from $R$ is shown in Figure 6.4, which shows that $R_{\alpha}$ contains the encryption of the values present in $R$. Note that, while encryption each attribute a different associated data id is used. This creates a separation between values taken by different attributes, i.e., two different attributes occur with the same value in $R$, then their encryptions in $R_{\alpha}$ would be different. This property would be required for our scheme to satisfy the security definition.

Other than $R_{\alpha}$, ESRQ1.Enc creates the server side and client side additional data $M_s$ and $M_c$. We discuss $M_s$ first.

**Creating** $R_\alpha$
1. **for** $j = 1$ to $\#A$, // Encrypting the attributes names
2. $\quad b_j \leftarrow \mathbf{E}_K(\mathsf{id}^*, a_j)$;
3. **end for**
4. **for** $j = 1$ to $\mathsf{nT}$,// Encrypting the attributes cells
5. $\quad t_j^{R_\alpha}[\mathtt{RowNo}] \leftarrow j$;
6. $\quad$ **for** $i = 1$ to $\#A$,
7. $\quad\quad t_j^{R_\alpha}[b_i] \leftarrow \mathbf{E}_K(\mathsf{id}_i, t_j^R[a_i])$;
8. $\quad$ **end for**
9. **end for**

Figure 6.4: Creating $R_\alpha$.

Let $\Lambda = \{a \in A : \mathsf{ty}(a) = 1\}$, i.e., $\Lambda$ contains the range attributes in $A$. For each $\lambda \in \Lambda$ let $\mathsf{Lst}_\lambda$ be the ordered list of $\lambda$ in $R$ and let $\mathsf{arr}_{\mathbf{E}_K(\mathsf{id}^*,\lambda)}$ be (the an array representation of) the $l$-bitmap matrix of $\lambda$ in $R$. From each list $\mathsf{Lst}_\lambda$, we create a new list $\mathcal{L}_{\mathbf{E}_K(\mathsf{id}^*,\lambda)}$, such that $\mathsf{len}(\mathcal{L}_{\mathbf{E}_K(\mathsf{id}^*\lambda)}) = \mathsf{len}(\mathsf{Lst}_\lambda)$, and $\mathcal{L}_{\mathbf{E}_K(\mathsf{id}^*,\lambda)}[i] = \mathbf{E}_K(\mathsf{Lst}_\lambda[i])$. For each $\lambda \in \Lambda$, $\mathsf{arr}_\lambda$ and $\mathcal{L}_\lambda$ are created by the client and then sent to the server and this constitutes the server side additional data $M_s$. Note, we index the list $\mathcal{L}$ and the array arr using $\mathbf{E}_K(\mathsf{id}^*, \lambda)$ instead of $\lambda$ to stress the fact that it is not required even to reveal the attribute names to the server.

The client side data $M_c$ consists of the attribute names $a_1, a_2, \ldots, a_{|A|}$, the corresponding identifiers $\mathsf{id}_1, \mathsf{id}_2, \ldots, \mathsf{id}_{\#A}$ and the identifier $\mathsf{id}^*$.

For a concrete example, consider that ESRQ1.Enc has as input the relation $R1$ as shown in Table 6.1. The only attribute in $R1$, where range queries are meaningful is the attribute Age. Then ESRQ1.Enc($R1$) would produce as output the relations $R1_\alpha$ and the server side data as shown in Table 6.4. The relation $R1_\alpha$ is the same as $R1$ except that each value is separately encrypted, and contains an additional attribute RowNo, which is stored in clear. Moreover, while applying encryption, the unique identifier of each column is used as the associated data. The attribute names of the original relation $R1$ also occur in $R1_\alpha$ in the encrypted form.

Relation $R1_\alpha$

| RowNo | $\mathbf{E}_K(\mathsf{id}^*, \mathtt{EmpId})$ | $\mathbf{E}_K(\mathsf{id}^*, \mathtt{Name})$ | $\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})$ |
|---|---|---|---|
| 1 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{TRW})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{Tom})$ | $\mathbf{E}_K(\mathsf{id}_3, 18)$ |
| 2 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{MST})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{Mary})$ | $\mathbf{E}_K(\mathsf{id}_3, 17)$ |
| 3 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{JOH})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{John})$ | $\mathbf{E}_K(\mathsf{id}_3, 52)$ |
| 4 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{MRH})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{Mary})$ | $\mathbf{E}_K(\mathsf{id}_3, 33)$ |
| 5 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{ASY})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{Anne})$ | $\mathbf{E}_K(\mathsf{id}_3, 18)$ |
| 6 | $\mathbf{E}_K(\mathsf{id}_1, \mathrm{RZT})$ | $\mathbf{E}_K(\mathsf{id}_2, \mathrm{Rosy})$ | $\mathbf{E}_K(\mathsf{id}_3, 36)$ |

$\mathcal{M}$

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

arr

| | | | | | |
|---|---|---|---|---|---|
| 3 | 2 | 0 | 4 | 3 | 5 |

$\mathcal{L}_{\mathtt{Age}}$

| |
|---|
| $\mathbf{E}_K(\mathsf{id}_3, 17)$ |
| $\mathbf{E}_K(\mathsf{id}_3, 18)$ |
| $\mathbf{E}_K(\mathsf{id}_3, 33)$ |
| $\mathbf{E}_K(\mathsf{id}_3, 36)$ |
| $\mathbf{E}_K(\mathsf{id}_3, 52)$ |

Table 6.4: The data stored at server side after encrypting $R1$ with ESRQ1: Relation $R1_\alpha$, the $l$-bitmap matrix $\mathcal{M}$ and the ordered list $\mathcal{L}_{\mathtt{Age}}$. Note that instead of the matrix $\mathcal{M}$ its array representation arr is stored.

The server side data $M_s$ consists of the list $\mathcal{L}_{\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})}$, which contains the ciphertexts of the values of the attribute Age sorted according to the real values. And the matrix $\mathcal{M}_{\mathtt{Age}}$, whose $i^{th}$ row corresponds to the $l$-encoded bitmap of the value $\mathbf{E}_K^{-1}(\mathsf{id}^*, \mathcal{L}_{\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})}[i])$. The array representation of $\mathcal{M}_{\mathtt{Age}}$ is shown as $\mathsf{arr}_{\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})}$. The array $\mathsf{arr}_{\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})}$ is stored in the server instead of the matrix $\mathcal{M}_{\mathtt{Age}}$. It would be clear from the discussions presented later that this information would be enough for the server to respond to any range query of the client.

ESRQ1.$\Phi$: The transform $\Phi$ receives as input a query meant for $R$ and converts it to a query which can be executed in $(R_\alpha)$. The allowed set of queries are simple select queries and range queries. The generic format of an allowed query is

$$Q\text{: SELECT * FROM } R \text{ WHERE } (a_1 \lhd_1 v_1) \circledcirc_1 (a_1 \lhd_2 v_2) \circledcirc_2 \ \ldots\ldots \ \circledcirc_{\ell-1} (a_\ell \lhd_1 v_\ell),$$

where $a_i$ represent an attribute name and $v_i$ a value of the attribute. $\lhd_i \in \{=, >, <, \leq, \geq\}$, and $\circledcirc_i$ can be an arbitrary Boolean connective, say $\vee$, $\wedge$ etc.

The query translation takes place in two phases. Given the query $Q$, in the first phase the values $v_1, v_2, \ldots, v_\ell$ are re-written to $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell$, such that the response of the re-written query is the same as the query $Q$ when executed in the original relation $R$, and the values $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell$ are present in the original relation $R$ in the respective columns. For example, if we take the relation $R1$, the query SELECT * FROM R1 WHERE AGE$\geq 19$ AND AGE$\leq 40$ will be re-written as SELECT * FROM R1 WHERE AGE$> 18$ AND AGE$< 52$. Note that the response of these two queries would be the same in the relation $R1$ and the values 18 and 52 occur in $R1$ in the attribute Age, whereas the values 19 and 40 not occur.

Generally speaking, let $v$ be any possible value of an attribute $a$ and let $v^{\mathrm{lo}}$ and $v^{\mathrm{hi}}$ be respectively, the largest and the smallest values in $\mathsf{Dom}(a)$ such that $v^{\mathrm{lo}} \leq v \leq v^{\mathrm{hi}}$. If

such a $v^{\text{lo}}$ or $v^{\text{hi}}$ does not exist we set their values to $\infty$. If $v \in \text{Dom}_R(a)$, then $v^{\text{lo}} = v^{\text{hi}} = v$. Now, we claim that given an attribute $a \in A$ and a value $v$ it is possible for the client to determine $v^{\text{lo}}$ and $v^{\text{hi}}$.

Let $\text{findLo}_a(v)$ and $\text{findHi}_a(v)$ be the procedures which return $v^{\text{lo}}$ and $v^{\text{hi}}$, respectively. These procedures can be realized by an interactive binary search between the client and the server. The lists stored as part of $M_s$ in the server would be used for this purpose. For a given attribute $a$ and value $v$, the client initiates the procedure by sending $c = \mathbf{E}_k(\text{id}^*, a)$ to the server. The server returns the middle element of the list $\mathcal{L}_c$, call this as $\alpha$. The client decrypts $\alpha$ using its key and the associated identifier of the attribute $a$, if the decrypted value (call it $\beta$) is equal to $v$ then it stops and sets $\text{findLo}_a(v) = \text{findHi}_a(v) = v$. If $\beta > v$ then it asks the server to return the middle element of the lower half of the list and if $\beta < v$ then the server returns the middle element in the upper half of the list $\mathcal{L}_c$. This interaction continues as in a binary search. It is easy to see that on termination of this binary search the server would be able to determine the values of $\text{findLo}_a(v)$ and $\text{findHi}_a(v)$.

In the query re-writing phase each condition $(a_i \triangleleft_i v_i)$ of the given query $Q$ is converted into an equivalent condition following the procedure described in Figure 6.5. Thus, after the query $Q$ undergoes re-writing it is converted into a new query $\tilde{Q}$ which has the same structure as of $Q$.

$$\tilde{Q}\text{: SELECT * FROM } R \text{ WHERE } (a_1 \triangleleft_1 \tilde{v}_1) \circledcirc_1 (a_1 \triangleleft_2 \tilde{v}_2) \circledcirc_2 \ \ldots\ldots \ \circledcirc_{\ell-1} (a_\ell \triangleleft_1 \tilde{v}_\ell),$$

In the second phase, the attribute names and the values in $\tilde{Q}$ are encrypted to produce the final output $Q'$. As the input query has a specific structure, hence the final translated query $\Phi(Q)$ is just:

$$Q'\text{: } (c_1 \triangleleft_1 c'_1) \circledcirc_1 (c_2 \triangleleft_2 c'_2) \circledcirc_2 \ldots\ldots \circledcirc_{\ell-1} (c_\ell \triangleleft_1 c'_\ell),$$

where $c_i = \mathbf{E}_K(\text{id}^*, a_i)$ and $c'_i = \mathbf{E}_K(\text{id}_i, \tilde{v}_i)$.

Going back to the concrete example, consider the following query $Q1$ for the relation $R1$

$$Q1\text{: SELECT * FROM } R \text{ WHERE Age} \geq 19 \text{ AND Age=} \leq 36$$

After re-writing, this query becomes:

$$\widetilde{Q1}\text{: SELECT * FROM } R \text{ WHERE Age} \geq 33 \text{ AND Age=} \leq 36$$

reWrite($a \lhd v$)

1. $v^{\text{lo}} \leftarrow \mathsf{findLo}_a(v)$ //Finding the smallest value in $Dom(a)$ s.t. $v^{\text{lo}} \leq v \leq v^{\text{hi}}$
2. $v^{\text{hi}} \leftarrow \mathsf{findHi}_a(v)$ //Finding the largest value in $Dom(a)$ s.t. $v^{\text{lo}} \leq v \leq v^{\text{hi}}$
3. **if** $v^{\text{lo}} = v^{\text{hi}}$,
4. **return** $a \lhd v$;

5. **if** $\lhd$ is '$=$', //Equality case
6. **return** $a = v$;

7. **if** $\lhd$ is '$<$' OR '$\leq$', //Less and less equal case
8. **if** $v^{\text{lo}} \neq \infty$, **return** $(a \leq v^{\text{lo}})$;
9. **else return** $(a < v^{\text{hi}})$;

10. **if** $\lhd$ is '$>$' OR '$\geq$', //Greater and greater equal case
11. **if** $v^{\text{hi}} \neq \infty$, **return** $(a \geq v^{\text{hi}})$;
12. **else return** $(a > v^{\text{lo}})$ ;

Figure 6.5: The query re-writing procedure

And, the final output of $\Phi(Q1)$, will be:

$$Q1'\text{: } (\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 33)) \bigwedge (\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \leq \mathbf{E}_K(\mathsf{id}_3, 36))$$

This transformed query is sent to the server.

ESRQ1.$\Psi$: As discussed, $\Psi$ is the transform that runs in the server to execute the client instructions. The instruction from the client comes to the server encoded as $Q'$, and it executes the procedure $\Psi(Q')$ as described in Figure 6.6. $Q'$ contains a collection of equality/inequality conditions aggregated by some Boolean connectives. $\Psi$ treats these equality/inequality conditions separately, and constructs a bitmap for each of these conditions. Based on the type of condition, the procedure uses the functions **Equal**, **Lesser** and **Greater**. These functions are also described in Figure 6.2.

Once the bitmaps for the individual conditions are constructed, it aggregates the bitmaps using the given Boolean connectives. Note that in line 13, the Boolean operations are applied bit-wise. The final bitmap $B$ constructed in line 13 contains the information regarding the tuples in $R_\alpha$, which satisfies the client's query. Thus, the server returns these valid tuples to the client.

**Response Procedure** $\Psi(Q')$
(Assume $Q' \equiv (c_1 \lhd_1 c_1') \circledcirc_1 (c_2 \lhd_2 c_2') \circledcirc_2 \ldots\ldots \circledcirc_{\ell-1} (c_\ell \lhd_1 c_\ell'))$
1. **for** $i \leftarrow 1$ to $\ell$ //Bitmaps generation
2. $\mathcal{L} \leftarrow \mathcal{L}_{c_i}$; arr $\leftarrow$ arr$_{c_i}$;
2. **if** $\lhd_i$ is $'='$,
3. $B_i \leftarrow$ **Equal**$(\mathcal{L}, \mathsf{arr}, c_i, c_i')$
4. **if** $\lhd_i$ is $'<'$,
5. $B_i \leftarrow$ **Lesser**$(\mathcal{L}, \mathsf{arr}, c_i, c_i')$
6. **if** $\lhd_i$ is $'>'$,
7. $B_i \leftarrow$ **Greater**$(\mathcal{L}, \mathsf{arr}, c_i, c_i')$
8. **if** $\lhd_i$ is $'\geq'$,
9. $B_i \leftarrow \overline{\textbf{Lesser}(\mathcal{L}, \mathsf{arr}, c_i, c_i')}$
10. **if** $\lhd_i$ is $'\leq'$,
11. $B_i \leftarrow$ **Lesser**$(\mathcal{L}, \mathsf{arr}, c_i, c_i') \vee$ **Equal**$(\mathcal{L}, \mathsf{arr}, c_i, c_i')$
12. **end for**

13. $B \leftarrow B_1 \circledcirc_1 B_2 \circledcirc_2 \cdots \circledcirc_{\ell-1} B_\ell$ //Operating Bitmaps
14. $\Delta \leftarrow \mathsf{MakeSet}(B, \mathsf{nT})$ //Extracting indices
( Let $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_m\}$)
15. Run the query:
`SELECT * FROM` $R_\alpha$ `WHERE`
`RowNo IN` $(\Delta_1, \Delta_2, \ldots, \Delta_m)$
16. Let ρ be the response to the above query;
17. **return** ρ;

Figure 6.6: The response procedure $\Psi$. The procedures **Lesser**$(\cdot,\cdot,\cdot,\cdot)$, **Greater**$(\cdot,\cdot,\cdot,\cdot)$ and **Equal**$(\cdot,\cdot,\cdot,\cdot)$ are described in Figure 6.2.

Going back to our example, the query $Q1'$ consists of two inequality conditions. The first one is $(\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 33))$, the bitmap for this condition is constructed by checking the array arr and applying the algorithm of Figure 6.2, the bitmap corresponding to $\mathtt{Name} = \mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})$ AND $\mathtt{SearchKey} = \mathbf{E}_K(\mathsf{id}_3, 33)$ is 110010. But recall that the bitmap is the l-encoded bitmap. Hence, the bitmap for the condition $(\mathbf{E}_K(\mathsf{id}^*, \mathtt{Age}) \geq \mathbf{E}_K(\mathsf{id}_3, 33))$ would be the complement of the stored bitmap, i.e., 001101. The bitmap for the second condition $(\mathbf{E}_k(\mathsf{id}^*, \mathtt{Age}) \leq \mathbf{E}_K(\mathsf{id}_3, 36))$ is computed by retrieving the l-encoded bitmap for $\mathtt{Name} = \mathbf{E}_K(\mathsf{id}^*, \mathtt{Age})$ AND $\mathtt{SearchKey} = \mathbf{E}_K(\mathsf{id}_3, 36)$ that is equal to 110110. Then the e-encoded bitmap is computed from the arr using again the algorithm of Figure 6.2 , i.e., 000001. Finally these two bitmaps are operated by an OR bitwise, giving as result 110111. As in the query the two conditions are connected by an AND, hence the bitmap for the query is $001101 \wedge 110111 = 000101$. This bitmap corresponds to the rows 4 and 6 of the table $R_\alpha$. Hence the server sends the response ρ as:

Response ρ

| RowNo | $\mathbf{E}_K(\mathsf{id}^*, \mathtt{EmpId})$ | $\mathbf{E}_K(\mathsf{id}^*, \mathtt{Name})$ | $\mathbf{E}_K(\mathsf{id}^* \mathtt{Age})$ |
|---|---|---|---|
| 4 | $\mathbf{E}_K(\mathsf{id}_1, MRH)$ | $\mathbf{E}_K(\mathsf{id}_2, Mary)$ | $\mathbf{E}_K(\mathsf{id}_3, 33)$ |
| 6 | $\mathbf{E}_K(\mathsf{id}_1, RZT)$ | $\mathbf{E}_K(\mathsf{id}_2, Rosy)$ | $\mathbf{E}_K(\mathsf{id}_3, 36)$ |

**Update Procedure** ESRQ1.$\mathcal{U}$: Let us assume that the client wants to update the value of $t_j^R[a_i]$ from old to new. First, note that it is possible for the client to know the RowNo of the corresponding tuple in $R_\alpha$ by posing a suitable query, and hence, direct the server to change the corresponding value in $R_\alpha$ with $\mathbf{E}_K(\mathsf{id}_i, \mathsf{new})$. To update the ordered list and the respective matrices we are going to use the algorithm **Matrix_Update_For_Update** described in Figure 6.3 and all previous functions as exists$(\mathcal{L}, \mathsf{val})$ or position$(\mathcal{L}, \mathsf{val})$. However, regarding to position function we have to consider that the elements in the ordered list are encrypted, thus if val is not present in $\mathcal{L}$, this function is realized by the server with an interactive binary search assisted by the client, as in the procedures findHi and findLo described in Section 6.4. We describe the update procedure in Figure 6.7. The first step is to encrypt the attribute name, and also the new value, with this information is possible to build an update query that instructs the server to change the old value of tuple $j$ in attribute $\mathbf{E}_K(\mathsf{id}^*, a_i)$ with the value $\mathbf{E}_K(\mathsf{id}_i, \mathsf{new})$ (see line 3). After, this update is performed, the next step is to update the respective matrix and ordered list, by calling the **Matrix_Update_For_Update** algorithm.

**Insert Procedure** ESRQ1.$I$: Let tup be a tuple for the original relation $R$. In Figure 6.7 we show the steps required to insert the tuple tup. The description assumes $\Lambda$ to be the set of indices of the attributes which are range attributes. First, for all attributes in the relation $R$ their respective values in the tuple tup are encrypted and a query

**Procedure** Update$(j, a_i, \mathsf{new})$
1. $\mathsf{val}_n \leftarrow \mathbf{E}_K(\mathsf{id}_i, \mathsf{new})$;
2. $c \leftarrow \mathbf{E}_K(\mathsf{id}^*, a_i)$;
3. Execute the following query:
   `UPDATE` $R_\alpha$ `SET` $c = \mathsf{val}_n$ `WHERE ROWNO=` $j$;
4. **Matrix_Update_For_Update**$(\mathcal{L}_c, \mathsf{arr}_c, j, \mathsf{val}_n)$ ;

**Procedure** Insert(tup)
1. **for all** $a_i \in A$,
2. $\quad t_{\mathsf{nT}}^{R_\alpha}[a_i] \leftarrow \mathbf{E}_K(\mathsf{id}_i, \mathsf{tup}[a_i])$;
3. **end for**
4. Execute the following query:
   `INSERT INTO` $R_\alpha$ `VALUES` $(t_{\mathsf{nT}}^{R_\alpha}[a_1], t_{\mathsf{nT}}^{R_\alpha}[a_2], \cdots t_{\mathsf{nT}}^{R_\alpha}[a_n])$;
5. **for all** $i \in \Lambda$;
6. $\quad c \leftarrow \mathbf{E}_K(\mathsf{id}^*, a_i)$;
7. $\quad \mathsf{val} \leftarrow \mathbf{E}_K(\mathsf{id}^i, \mathsf{tup}[a_i])$;
8. $\quad$ **Matrix_Update_For_Insert**$(\mathcal{L}_c, \mathsf{arr}_c, val)$
9. **end for**

Figure 6.7: The update and insert procedures

for insertion is sent to the server (see line 4). Finally the respective updates to the matrices and ordered lists of the range attributes are performed by calling the **Matrix_Update_For_Insert** described in Figure 6.3.

**Decryption Procedure** ESRQ1.Dec: The decryption procedure receives as input the response $\rho$ from the server and the keys. This procedure uses the inverse of the encryption $\mathbf{E}_K^{-1}$ to decrypt the server response.

### 6.4.1 Characteristics of ESRQ1

We list some of the important characteristics of ESRQ1.

1. **Efficiency:** It is clear from the description that to encrypt a relation with $\mathsf{nT}$ tuples and $m$ attributes, at most $m(\mathsf{nT}+1)$ many encryption calls are required, which form the bulk of the computational overhead. Note, that the encryptions for the attributes lists are not required to be done again, as these values are already included in $R_\alpha$. For translating a query involving $\ell$ conditions $2\ell$ encryptions are required. The query response procedure does not require any encryption/decryption. For decryption $m(s+1)$ invocations of $\mathbf{E}^{-1}$ are required if the response contains $s$ tuples.

2. **Storage:** The storage requirement depends on the nature of $\mathbf{E}$. Let $\mathsf{str}_{\mathbf{E}}(x) : \mathcal{M} \rightarrow \mathbb{N}$, define the stretch of the encryption scheme $\mathbf{E}$, i.e., for any $K \in \mathcal{K}$, $\mathsf{str}_{\mathbf{E}}(x) = |\mathbf{E}_K(x)| - |x|$. If $\mathbf{E}$ is length preserving, then for every $x \in \mathcal{M}$, $\mathsf{str}_{\mathbf{E}}(x) = 0$. For other practical schemes the stretch would be generally a constant for every message.

   If $R$ has $\mathsf{nT}$ tuples and $m$ attributes $A = \{a_1, \ldots, a_m\}$, and if $\mathsf{siz}(R)$ denotes the size

of the relation $R$ in bits, then we would have

$$\mathsf{siz}(R_\alpha) \leq \mathsf{siz}(R) + \sum_{j=1}^{\mathsf{nT}} \sum_{i=1}^{m} \mathsf{str}_{\mathbf{E}}(t_j^R[a_i]) + \mathsf{nT}\lg(\mathsf{nT}).$$

Let $B \subseteq A$ be the set of range attributes in $A$, then we have

$$\mathsf{siz}(M_s) = \sum_{b \in B} \sum_{x \in \mathsf{Dom}_R(b)} (\mathsf{str}_{\mathbf{E}}(b) + \mathsf{str}_{\mathbf{E}}(x) + \mathsf{nT}).$$

3. **Role of the associated data:** We have used a different identifier as an associated data for encrypting values in each column. This plays an important role. As the encryption algorithm to be used is deterministic, hence same plaintext values yield the same ciphertexts. It is important that for a specific attribute the equality relations in different tuples is maintained in the encrypted relation. This property is important for query processing and it does not amount to insecurity in our proposed model. But, it can be the case that in a relation two different attributes take the same value. We do not want that this type of equality is revealed in the encrypted tables, as even in our security model this kind of leakage may allow the adversary to be successful in distinguishing in the ESRQ game. To prevent such leakages we use a distinct associated data to encrypt values of different attributes.

4. **Security:** The security of ESRQ1 depends on the encryption scheme **E**. In Section 6.4.2, we provide a theorem, which relates the security of ESRQ1 with the encryption scheme **E**.

5. **Other query types:** For ease of discussion we restricted ourselves to a specific type of query which involves equality/inequality conditions aggregated by Boolean operators. But, it is easy to see that a few more types of queries like `COUNT`, `GROUP BY`, `DISTINCT`, `ORDER BY`, `MIN`, `MAX` etc. can also be answered using the information in $R_\alpha$ and $M_s$. The discussion that we presented assumed that the client always delegates a single relation. The scheme can be extended in a natural way to accommodate multiple relations, where for each relation $R$ the pair $(R_\alpha)$ and the respective lists and arrays would be created and some additional information in the form of join bitmaps [85] would be stored. In such situations queries involving equality joins can also be answered.

### 6.4.2 Security of ESRQ1

The following Theorem specifies the security of $\psi = \mathsf{ESRQ1}[\mathbf{E}]$.

**Theorem 6.1.** *Fix natural numbers $q, \sigma_n, t$, and a deterministic encryption scheme $\mathbf{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{C}$, such that the smallest ciphertext in $\mathcal{C}$ has size $s$. Let $\Psi = \mathsf{ESRQ1}[\mathbf{E}]$. Then*

$$\mathbf{Adv}^{\text{esrq}}_{\Psi}(q, \sigma_n, t) \leq \mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(q', \sigma'_n, t') + \frac{q^2}{2^{s+1}}, \tag{6.3}$$

*where $q' \leq q$, $\sigma'_n \leq \sigma_n$ and $t' = O(t)$.*

The above Theorem relates the ESRQ security of $\mathsf{ESRQ1}[\mathbf{E}]$, with the det-cpa security of $\mathbf{E}$, and it implies that the ESRQ1 scheme provides almost the same security as that of $\mathbf{E}$ with a small degradation. The degradation depends on the size of the relation being encrypted and surprisingly on the size of the smallest ciphertext. The bound becomes stronger if the size of the smallest ciphertext is bigger. We provide the complete proof in the next section.

### 6.4.3 Proof of Theorem 6.1

First, it is easy to see that because of the specific restrictions that we put on the adversary, the server side data $M_s$ for both the relations $R_0$ and $R_1$ (required for the ESRQ security game, see Figure 6.1) would always be the same, thus this data would be of no help to the adversary in guessing the bit $b$. The only way that $\mathcal{A}$ can guess the bit is by using the information in the encrypted relation.

We construct a det-cpa adversary $\mathcal{B}$ with oracle $O$, which acts as a challenger for $\mathcal{A}$ in the ESRQ security game.

For acting as the challenger, $\mathcal{B}$ follows the procedure described in Figures 6.4 and 6.6, and whenever a call to $\mathbf{E}(\cdot,\cdot)$ is required it uses its oracle $O$, moreover $\mathcal{B}$ never repeats a query to $O(\cdot,\cdot)$. $\mathcal{B}$ achieves this by keeping record of the oracle outputs for each of its queries. Hence, if the cell complexity of $\mathcal{A}$ is $q$ , and the query complexity of $\mathcal{A}$ is $\sigma_n$. Then the number of queries asked by $\mathcal{B}$ is at most $q$, and the query complexity is at most $\sigma_n$.

From the description of $\mathcal{B}$ and the ESRQ game in Figure 6.1, it is easy to see that

$$\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{E}_K(\cdot,\cdot)} \Rightarrow 1] = \Pr[\text{ESRQ}^{\mathcal{A}} \Rightarrow 1]. \tag{6.4}$$

We want to bound the det-cpa advantage of $\mathcal{B}$, for doing that we would require a bound on $\Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1]$. To make the proof cleaner, we do this in two steps, for which we would require some additional technicalities. Let the message space $\mathcal{M}$ and tweak space $\mathcal{T}$ and cipher space $\mathcal{C}$ be all non-empty subsets of $\{0,1\}^*$. Let $e : \mathcal{T} \times \mathcal{M} \to \mathbb{N}$ be

such that $e(T,X) = |\mathbf{E}_K(T,X)|$ for any $K \in \mathcal{K}$. The function $e$ defines the length of the ciphertext for the encryption scheme $\mathbf{E}$, and it is dependent only on $|X|$ and $|T|$. Also,

$$\mathcal{C} \subseteq \bigcup_{t \in \mathcal{T}, X \in \mathcal{M}} \{0,1\}^{e(T,M)}.$$

Let $\mathsf{Inj}_e(\mathcal{M},\mathcal{T})$ be the set of all injective functions from $\mathcal{T} \times \mathcal{M}$ to $\mathcal{C}$, such that for every $f \in \mathsf{Inj}_e(\mathcal{T},\mathcal{M})$ and every $(T,X) \in \mathcal{T} \times \mathcal{M}$, $|f(T,X)| = e(T,X)$. Now, we make the following claims.

**Claim 6.1.** *For any arbitrary adversary $\mathcal{B}$ which never repeats a query and asks at most $q$ oracle queries*

$$\left|\Pr[f \xleftarrow{\$} \mathsf{Inj}_e(\mathcal{T},\mathcal{M}) : \mathcal{B}^{f(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1]\right| \leq \frac{q^2}{2^{s-1}}, \tag{6.5}$$

*where $\$(T,X)$ returns a random string from $\{0,1\}^{e(T,X)}$, and $s$ is the size of the smallest string in $\mathcal{C}$.*

*Proof.* A more general version of this claim is proved in [82], for completeness we again provide a proof here. We use the sequence of games. Consider the procedures $\mathsf{F0}$ and $\mathsf{F1}$ described in Figure 6.8. It is clear from the description in Figure 6.8 that the procedure $\mathsf{F0}$ mimics the behaviour of a uniform random function $f \in \mathsf{Inj}_e(\mathcal{T},\mathcal{M})$. On the other hand, $\mathsf{F1}$ on each query $(\mathsf{id},X)$ returns a uniform random string in $\{0,1\}^{e(\mathsf{id},X)}$. Hence, for an arbitrary adversary $\mathcal{B}$ we have

$$\Pr[f \xleftarrow{\$} \mathsf{Inj}_e(\mathcal{T},\mathcal{M}) : \mathcal{B}^{f(\cdot,\cdot)} \Rightarrow 1] = \Pr[\mathcal{B}^{\mathsf{F0}} \Rightarrow 1], \tag{6.6}$$

and

$$\Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1] = \Pr[\mathcal{B}^{\mathsf{F1}} \Rightarrow 1]. \tag{6.7}$$

Moreover, the procedures $\mathsf{F0}$ and $\mathsf{F1}$ runs in exactly the same way unless the $\mathsf{bad}$ flag is set to $\mathtt{true}$. Hence, by the fundamental lemma of game playing

$$\left|\Pr[\mathcal{B}^{\mathsf{F0}} \Rightarrow 1] - \Pr[\mathcal{B}^{\mathsf{F1}} \Rightarrow 1]\right| \leq \Pr[\mathcal{B}^{\mathsf{F1}} \text{ sets bad}]. \tag{6.8}$$

If $\mathsf{BAD}$ be the event that $\mathcal{B}^{\mathsf{F1}}$ sets the flag $\mathsf{bad}$ to $\mathtt{true}$, then using equations (6.6), (6.7) and (6.8), we have

$$\left|\Pr[f \xleftarrow{\$} \mathsf{Inj}_e(\mathcal{M},\mathcal{T}) : \mathcal{B}^{f(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1]\right| \leq \Pr[\mathsf{BAD}]. \tag{6.9}$$

Thus, to complete the proof we need an upper bound for $\Pr[\mathsf{BAD}]$.

Procedures F0 and F1

Initialization

20. bad $\leftarrow$ true;
21. **for all** id $\in \mathcal{T}$
22. **for all** $X \in \mathcal{M}$
23. Range(id, $X$) $\leftarrow$ undefined
24. **end for**
25. **end for**

On receiving a query (id, $X$), do the following:

201 $c \leftarrow e(\mathsf{id}, X)$;
202. $Y \xleftarrow{\$} \{0,1\}^c$;
203. **if** $Y \in$ Range(id, $\cdot$);
204. bad $\leftarrow$ true;
205. $\boxed{Y \leftarrow \{0,1\}^c \setminus \mathsf{Range}(\mathsf{id}, \cdot)}$;
206. Range(id, $X$) $\leftarrow Y$;
207. **end if**
208. **return** $Y$;

Figure 6.8: The procedures F0 and F1. F0 is the complete code; in F1 the boxed statement is omitted.

$\mathcal{B}$ asks $q$ distinct queries. After $\mathcal{B}$ stops querying, there would be $q$ entries in the table Range, which are different from undefined. We collect these $q$ values in a multiset Ran. If COLL be the event that two elements in Ran are equal, then $\Pr[\mathsf{BAD}] = \Pr[\mathsf{COLL}]$. Now, if two elements in Ran are of different length then they cannot be equal and if they are of the same length $c$, then they are uniform random elements in $\{0,1\}^c$, thus the probability of them being equal is $1/2^c$. As $s$ is the size of the smallest element in $\mathcal{C}$, hence by the union bound we have,

$$\Pr[\mathsf{BAD}] = \Pr[\mathsf{COLL}] \leq \binom{q}{2}/2^s < \frac{q^2}{2^{s+1}}. \tag{6.10}$$

Now, using Eqs. (6.9) and (6.10) we have the claim. □

**Claim 6.2.** *For the adversary $\mathcal{B}$, described in Figure 6.1,*

$$\Pr[f \xleftarrow{\$} \mathsf{Inj}_e(\mathcal{M},\mathcal{T}) : \mathcal{B}^{f(\cdot,\cdot)} \Rightarrow 1] \leq \frac{1}{2}. \tag{6.11}$$

*Proof.* When the oracle of $\mathcal{B}$ is a function $f$ which is random element in $\mathsf{Inj}_e(\mathcal{T},\mathcal{M})$, then it provides a perfect environment for $\mathcal{A}$. Note that $\mathcal{A}$ expects that the encryptions in $(R_\alpha, M_s)$ are done by a deterministic encryption scheme $\mathbf{E}_K : \mathcal{T} \times \mathcal{M} \to \mathcal{C}$ which is injective. Moreover, a random element $f$ of $\mathsf{Inj}_e(\mathcal{T},\mathcal{M})$ maps each distinct element in $\mathcal{T} \times \mathcal{M}$ to a uniform random element in $\{0,1\}^{e(X,T)}$, maintaining injectivity. It may be convenient to see $f$ as the procedure F0 described in Figure 6.8.

As $R_0, R_1$ which is chosen by $\mathcal{A}$ are equivalent, i.e., $R_0 \approx R_1$, hence if $\mathcal{B}$'s oracle is $f \xleftarrow{\$} \mathsf{Inj}_e(\mathcal{T},\mathcal{M})$, then the $(R_\alpha, R_\beta)$ that it gets from $\mathcal{B}$ is independent of $R_0, R_1$. Thus, the only way $\mathcal{A}$ can determine whether $R_0$ or $R_1$ was encrypted is by guessing randomly, and this is what we claim. □

Finally, from the claims 6.1 and 6.2, we have

$$\left|\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{\mathbf{E}(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{B}^{\$(\cdot,\cdot)} \Rightarrow 1]\right| \geq \left|\Pr[\mathrm{ESRQ}^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right| - \frac{q^2}{2^{s+1}}.$$

Now, using Eq. (6.1) and Definition 6.3 we have

$$\mathbf{Adv}^{\mathrm{esrq}}_{\mathsf{ESRQ1}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{det-cpa}}_{\mathbf{E}}(\mathcal{B}) + \frac{q^2}{2^{s+1}}.$$

As $\mathcal{A}$ is an arbitrary adversary with cell complexity $q$ and query complexity $\sigma_n$, the Theorem follows. ■

## 6.5 Summary

In this chapter we described ESRQ, which is a basic model for providing privacy in outsourced databases. We introduced a novel structure to store bitmaps that uses an ordered matrix and its array representation. We also fixed a syntax for ESRQ and defined its security. Moreover we presented a concrete scheme ESQR1 which uses deterministic encryption scheme with associated data (DEAD) and bitmap indices. Finally we proved security of ESRQ1 according our security definition. In the next chapter we discuss some new constructions for DEAD.

# Chapter 7

# Practical Instantiations of DEAD Schemes

> Doubt is not a pleasant condition, but certainty is absurd.
>
> *Voltaire*

We described and proved the security of ESRQ using an encryption scheme **E**. To prove security we required **E** to be a det-cpa secure deterministic encryption scheme with associated data (DEAD). Moreover, **E** should be chosen in such a way that it has favourable efficiency and usability properties. The type of security and functionality required by **E** can be immediately obtained by two existing and well studied cryptographic objects, namely, deterministic authenticated encryption with associated data (DAEAD) [82], and tweakable enciphering schemes (TES) [83, 84, 86].

We first briefly discuss the suitability of these objects next.

- DAEAD schemes are deterministic encryption schemes which provide both privacy and authenticity, they provide security against (deterministic) chosen cipher text adversaries. Thus, these schemes provide more security than required for this application. This enhanced security comes at an extra cost, the significant cost being that of the ciphertext expansion, i.e., the ciphertext is considerably longer than the plaintext. This expansion takes place due to the fact that the ciphertext includes an authentication tag, for practical secure schemes this extra tag length would be around 128 bits. This length expansion may not be tolerable in certain applications, say where the database contains only “small" numerical values.
- TES are length preserving schemes which also provide security against (deterministic) chosen ciphertext adversaries. Being length preserving, TES not have

the same disadvantage as that of DAEAD schemes stated above. But the known constructions of TES are computationally heavy compared to only CPA secure encryption schemes like CBC, counter etc.

Thus, from the discussion above it is clear, that there is a scope to come up with some new encryption schemes which are strictly det-cpa secure and are more efficient than TES or DAEAD schemes. In this chapter we explore several such constructions. In Section 7.1 we provide a construction which has some restriction on the message space, and further in Section 7.2 we propose another class of constructions which we call as OHCTR where the restriction on the message space is removed. We analyze all the proposed constructions formally and we argue regarding their security.

## 7.1 A Construction for Restricted Message Spaces

We give here a block cipher based solution for a det-cpa scheme assuming that the message space is restricted. We fix $n$ as the block length of the block cipher. Additionally, we assume that the tweak space is $\{0,1\}^{\lambda}$, and the message space contains strings at most $\ell$-bits long. The necessary restriction is $\ell + \lceil \lg \ell \rceil + \lambda \leq n$.

Let $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Using only $E$, we construct $\mathbf{E}$ as shown in Figure 7.1. In the description we assume that $\tau = n - (\ell + \lambda)$, $\mathsf{pad}_n(x) = x||0^{n-|x|}$, $\mathsf{lsb}_\lambda(x)$ and $\mathsf{msb}_\lambda(x)$ outputs the $\lambda$ least and most significant bits of $x$, respectively. For an integer $i \leq 2^n - 1$, $\mathsf{bin}_n(i)$ is the $n$-bit binary representation of $i$ and $\mathsf{toInt}(x)$ returns the positive integer represented by the binary string $x$.

| $\mathbf{E}_K(\mathsf{id}, x)$ | $\mathbf{E}_K^{-1}(\mathsf{id}, z)$ |
|---|---|
| 1. $z_1 \leftarrow \mathsf{pad}_\ell(x)$ ; | 1. $y \leftarrow E_K^{-1}(z)$ ; |
| 2. $z_2 \leftarrow \mathsf{bin}_\tau(\lvert x \rvert)$ ; | 2. $z_2 \leftarrow \mathsf{toInt}(\mathsf{lsb}_\lambda(y))$ ; |
| 3. $z \leftarrow E_K(z_1 \lvert\lvert \mathsf{id} \lvert\lvert z_2)$; | 3. $x \leftarrow \mathsf{msb}_{z_2}(y)$; |
| 4. **return** $z$ | 4. **return** $x$ |

Figure 7.1: A block cipher based construction of $\mathbf{E}$ for restricted message spaces.

The construction requires just one block cipher call for one encryption and the ciphertext is always $n$-bits long. Also if the block cipher $E$ is a secure pseudorandom permutation then $\mathbf{E}$ is det-cpa secure. We state this formally in the next Theorem.

**Theorem 7.1.** *Let $\mathcal{A}$ be an arbitrary det-cpa adversary attacking* **E** *who asks at most q queries and runs for time at most t. Then there exists a prp adversary $\mathcal{B}$ attacking E, such that*

$$\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{A}) = \mathbf{Adv}_{E}^{\text{prp}}(\mathcal{B}).$$

*Moreover $\mathcal{B}$ runs for the same time as of $\mathcal{A}$ and asks the same number of queries as $\mathcal{A}$.*

*Proof.* The idea of the proof is to construct an adversary $\mathcal{B}$ whose task is to break the pseudorandomness of $E$. $\mathcal{B}$ will use the adversary $\mathcal{A}$ by acting as its challenger. The interaction between $\mathcal{B}$ and $\mathcal{A}$ is represented in the procedure below:

**Adversary** $\mathcal{B}^{O}$
1. Whenever $\mathcal{A}$ queries its encryption oracle on a $(m, id)$ pair:
2. $c \leftarrow E(pad_{\ell}(m)||\mathsf{id}||bin_{\tau}(|m|))$;
   $\mathcal{B}$ uses its oracle to compute $c$
3. Eventually, $\mathcal{A}$ outputs a bit $b'$
4. **return** $b'$

From the procedure above we can argue the following:

1. When the oracle of $\mathcal{B}$ is $E_K$, then the adversary $\mathcal{A}$ is basically interacting with the scheme $\mathbf{E}_K$, thus

$$\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{B}^{E_K()} \Rightarrow 1] = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathbf{E}_K(.,.)} \Rightarrow 1] \tag{7.1}$$

2. When the oracle of $\mathcal{B}$ is a $\pi$ selected uniformly at random from $\text{Perm}(n)$, then $\mathcal{A}$ in response to its queries gets uniform random strings in $\{0,1\}^n$. This is because, $\mathcal{A}$ never repeats a query, and in consequence $\pi$ is evaluated only on distinct inputs. As $\pi$ is an uniform random permutation, thus its outputs on distinct inputs are uniformly distributed in $\{0,1\}^n$. Thus we have

$$\Pr[\pi \xleftarrow{\$} \text{Perm}(n) : \mathcal{B}^{\pi(.)} \Rightarrow 1] = \Pr[\mathcal{A}^{\$(.,.)} \Rightarrow 1] \tag{7.2}$$

Thus subtracting equation (7.2) from (7.1) and using the definitions of the det-cpa advantage of $\mathcal{A}$ and the prp advantage of $\mathcal{B}$, we can conclude

$$\mathbf{Adv}_{\mathbf{E}}^{\text{det-cpa}}(\mathcal{A}) = \mathbf{Adv}_{E}^{\text{prp}}(\mathcal{B}).$$

□

## 7.2 OHCTR: A Construction for Arbitrary Message Spaces

The OHCTR construction is motivated by the HCTR construction which is a tweakable enciphering scheme. HCTR stands for Hash counter, it uses two universal hash functions and a counter mode of operation to obtain a tweakable enciphering scheme. OHCTR stands for *only one hash counter*, thus in the name we want to stress that unlike HCTR this construction uses only one universal hash, thus it is more efficient than HCTR, but of course it does not provide the strong security of HCTR. OHCTR is det-cpa secure, thus it is strictly weaker than HCTR in terms of security.

We propose two versions of OHCTR namely OHCTRa and OHCTRb. Both are constructed using a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. For OHCTRa we assume that the message space contains strings which are at least $n$-bits long. The other construction does not require this restriction.

Before we discuss the construction we fix some components which would be used in the construction. For $X \in \{0,1\}^*$, $\mathsf{FormatA}_n(X)$ outputs $X_1||X_2||\ldots||X_m$ where $m = \lceil |X|/n \rceil$ and for $1 \leq i \leq m-1$, $|X_i| = m$ and $|X_m| \leq n$. Hence, $\mathsf{Format}_n(X)$ divides $X$ into $m$ blocks where the first $m-1$ blocks are $n$ bits long the last block can be less than $n$ bits. $\mathsf{pad01}(X)$ returns $X||10^*$, i.e., it pads a 1 to the input string and then pads zeros until the length of the resulting string becomes a multiple of $n$.

The constructions uses a counter mode of operation CTR which is defined as follows. Let $K \in \mathcal{K}$ and $S \in \{0,1\}^n$, $X \in \{0,1\}^*$, and $X_1||X_2||\cdots||X_m \leftarrow \mathsf{Format}_n(X)$, then

$$\mathsf{CTR}_K^S(X) = E_K\left(S \oplus \mathsf{bin}_n(1)\right) \oplus X_1 || E_K(S \oplus \mathsf{bin}_n(2)) \oplus X_2 || \cdots || \mathsf{lsb}_{|X_m|}(E_K(S \oplus \mathsf{bin}_n(m))) \oplus X_m.$$

In addition to the counter mode we would require a polynomial hash defined as

$$H_h(X) = X_1 h^{m+1} \oplus X_2 h^m \oplus \cdots \oplus \mathsf{pad}_n(X_m) h^2 \oplus \mathsf{bin}_n(|X|) h,$$

where $h \in \{0,1\}^n$, $X_1||X_2||\ldots||X_m = \mathsf{Format}_n(X)$, and recall that $\mathsf{pad}_n(Y) = Y||0^{n-|Y|}$.

The encryption and decryption algorithms for OHCTR*a* is described in Figure 7.2.

OHCTRb, shown in Figure 7.3, is same as OHCTRa, the only difference being that, in the encryption procedure the input message is padded with $10^*$, and in the decryption process the postfix $10^*$ is removed after decryption.

We discuss a few properties of the constructions OHCTRa and OHCTRb next.

1. **Message/Cipher Lengths**: OHCTRa has the restriction that the messages should be at least $n$-bits long, but OHCTRb does not have any such restriction. OHCTRa

| **Algorithm** $\mathsf{OHCTRa}.\mathbf{E}_{K,h}(\mathsf{id},P)$ | **Algorithm** $\mathsf{OHCTRa}.\mathbf{D}_{K,h}(\mathsf{id},C)$ |
|---|---|
| 1. $P_1 \| \cdots \| P_m \leftarrow \mathsf{Format}_n(P)$; | 1. $C_1 \| \cdots \| C_m \leftarrow \mathsf{Format}_n(P)$; |
| 1. $MM \leftarrow P_1 \oplus H_h(P_2 \| \ldots \| P_m \| \mathsf{id})$; | 1. $CC \leftarrow C_1 \oplus H_h(C_2 \| C_3 \| \ldots \| C_m \| \mathsf{id})$; |
| 2. $CC \leftarrow E_K(MM)$; | 2. $MM \leftarrow E_K^{-1}(CC)$; |
| 3. $S \leftarrow MM \oplus CC$; | 3. $S \leftarrow MM \oplus CC$; |
| 4. $(C_2 \| \ldots \| C_{m-1}, C_m)$ $\leftarrow \mathsf{CTR}_K^S(P_2 \| \ldots \| P_m)$; | 4. $(P_2 \| \cdots \| P_{m-1} \| P_m)$ $\leftarrow \mathsf{CTR}_K^S(C_2 \| \cdots \| C_m)$; |
| 5. $C_1 \leftarrow CC$; | 5. $P_1 \leftarrow MM$; |
| 6. return $(C_1 \| \cdots \| C_m)$; | 6. return $(P_1 \| \cdots \| P_m)$; |

Figure 7.2: Encryption using OHCTR. $K$ is the block-cipher key, $h$ the hash key and id the associated data. The message space is $\mathcal{M} = \cup_{i \geq n} \{0,1\}^i$.

| **Algorithm** $\mathsf{OHCTRb}.\mathbf{E}_{K,h}(\mathsf{id},P)$ | **Algorithm** $\mathsf{OHCTRb}.\mathbf{D}_{K,h}(\mathsf{id},C)$ |
|---|---|
| 1. $Y \leftarrow \mathsf{pad10}(P)$; | 1. $Z \leftarrow \mathsf{OHCTRa}.\mathbf{D}_{K,h}(\mathsf{id},C)$; |
| 2. $Z \leftarrow \mathsf{OHCTRa}.\mathbf{E}_{K,h}(\mathsf{id},Y)$ ; | 2. $X \leftarrow \mathsf{extract}(Z)$; |
| 3. **return** $Z$; | 3. **return** $X$; |

Figure 7.3: Encryption and decryption using OHCTRb. $K$ is the block-cipher key, $h$ the hash key and id the associated data. The message space is $\{0,1\}^*$. The procedure $\mathsf{extract}(X)$ removes the postfix $10^*$ from $X$ and returns the remaining string.

is length preserving, i.e., the ciphertext always has the same length as that of the plaintext. In OHCTRb the ciphertext is always longer than the plaintext as an injective padding pad10 is applied to the plaintext before encryption. The ciphertext expansion in OHCTRb is at most $n$ bits.

2. **Efficiency:** To encrypt a $m$ block message, OHCTRa requires $m$ block cipher calls and $m+1$ finite field multiplications. For OHCTRb it would require at most $m+1$ block cipher calls and $m+2$ finite field multiplications. If the underlying block cipher is an AES then both the block cipher calls and the finite field multiplications can be implemented very efficiently using the dedicated AES-NI instructions in any modern Intel processor. As discussed before, the OHCTR construction is much more efficient than the original HCTR construction.

3. **Security:** Both the constructions are "provably secure" in the det-cpa sense (we formally state the security theorems in Section 7.2.1), and thus they provide the necessary security required for the application.

### 7.2.1 Security of OHCTR

Let us denote OHCTRa instantiated with a block cipher $E$ by OHCTRa$[E]$, similar notation is used for OHCTRb. The following Theorem specifies the security of OHCTR.

**Theorem 7.2.** *Fix $n, \sigma$ to be positive integers and an $n$-bit block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. Let $\Upsilon \in \{\text{OHCTR}a[E], \text{OHCTR}b[E]\}$. Then:*

$$\mathbf{Adv}^{\text{det-cpa}}_{\Upsilon}(\sigma, t) \leq \frac{3\sigma^2}{2^n} + \mathbf{Adv}^{\text{prp}}_{E}(\sigma, t') \tag{7.3}$$

*where $t' = t + O(\sigma)$.*

### 7.2.2 Proof of Theorem 7.2

The proof is a standard reduction which resembles closely the structure of the security proof in [87]. In this proof we will show that for an arbitrary adversary $\mathcal{A}$ who asks $q$ queries and has a query complexity $\sigma$,

$$\mathbf{Adv}^{\text{det-cpa}}_{\text{OHCTRa}\,[\text{Perm}(n)]}(\mathcal{A}) \leq \frac{3\sigma^2}{2^n}, \tag{7.4}$$

where OHCTRa [Perm($n$)] denotes OHCTRa instantiated by a permutation $\pi$ selected uniformly at random from Perm($n$). The bound in Eq. (7.4) is called the information theoretic bound, the corresponding complexity theoretic bound is stated in Theorem 7.2, which can be derived by a standard way from the bound in Eq. (7.4) (see [86] for details).

Also the following proof is for the construction OHCTRa, the proof for OHCTRb involves the same arguments, and can be easily derived from the proof that follows.

We use the technique of sequence of games to model the interaction of an arbitrary adversary $\mathcal{A}$ with OHCTRa. In what follows, we will denote OHCTRa instantiated with $\pi \xleftarrow{\$}$Perm($n$) by $\boldsymbol{\Upsilon}_\pi$. We briefly discuss the sequence of the games next.

**Game $\mathsf{G}_0$:** In $\mathsf{G}_0$, the adversary interacts with $\boldsymbol{\Upsilon}_\pi$ where $\pi$ is a randomly chosen permutation from Perm($n$). Instead of initially choosing $\pi$, we build up $\pi$ using the function ch-$\pi$ in the following manner. Initially $\pi$ is assumed to be undefined everywhere. When $\pi(X)$ is needed, but the value of $\pi$ is not yet defined at $X$, then a random value is chosen among the available range values. The domain and range of $\pi$ are maintained in two sets *Domain* and *Range*, and $\overline{\textit{Domain}}$ and $\overline{\textit{Range}}$ are the complements of *Domain* and *Range* relative to $\{0,1\}^n$. The game $\mathsf{G}_0$ is shown in Figure 7.4. The figure shows the subroutine Ch-$\pi$, the initialization steps and how the game responds to an encryption query of the adversary.

The game $\mathsf{G}_0$ accurately represents the attack scenario, and by our choice of notation, we can write

$$\Pr[\mathcal{A}^{\boldsymbol{\Upsilon}_\pi} \Rightarrow 1] = \Pr[\mathcal{A}^{\mathsf{G}_0} \Rightarrow 1]. \tag{7.5}$$

**Game $\mathsf{G}_1$:** We modify $\mathsf{G}_0$ by deleting the boxed entries in $\mathsf{G}_0$ and call the modified game as $\mathsf{G}_1$. By deleting the boxed entries it cannot be guaranteed that $\pi$ is a permutation as though we do the consistency checks but we do not reset the value of $Y$ in Ch-$\pi$. The games $\mathsf{G}_0$ and $\mathsf{G}_1$ are identical apart from what happens when the bad flag is set. By using the fundamental lemma of game playing [88], we obtain

$$|\Pr[\mathcal{A}^{\mathsf{G}_0} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1]| \leq \Pr[\mathcal{A}^{\mathsf{G}_1} \text{ sets bad}]. \tag{7.6}$$

Another important thing to note is that in $\mathsf{G}_1$ in line 102, for an encryption query, $CC^s$ gets set to a random $n$ bit string. Similarly in lines 105 and 108 $Z_i^s$ gets set to random values. Thus, in the game $\mathsf{G}_1$ the adversary gets random strings in response to his

Subroutine Ch-$\pi(X)$

11. $Y \xleftarrow{\$} \{0,1\}^n$; **if** $Y \in Range$ **then** bad $\leftarrow$ true; $Y \xleftarrow{\$} \overline{Range}$; **endif**;
12. **if** $X \in Domain$ **then** bad $\leftarrow$ true; $Y \leftarrow \pi(X)$; **endif**
13. $\pi(X) \leftarrow Y$; $Domain \leftarrow Domain \cup \{X\}$; $Range \leftarrow Range \cup \{Y\}$; **return**$(Y)$;

Initialization:

17. **for** all $X \in \{0,1\}^n$ $\pi(X) =$ undef **endfor**
18. bad $\leftarrow$ false

Respond to the $s^{th}$ query as follows: (Assume $l^s = n(m^s - 1) + r^s$, with $0 \leq r^s < n$.)

Encipher query: $\mathsf{Enc}(T^s; P_1^s, P_2^s, \ldots P_{m^s}^s)$

101. $MM^s \leftarrow P_1^s \oplus H_h(P_2^s || \ldots || P_m^s || id^s)$;
102. $CC^s \leftarrow$ Ch-$\pi(MM^s)$;
103. $S^s \leftarrow MM^s \oplus CC^s$;
104. **for** $i = 1$ to $m^s - 2$,
105. $Z_i^s \leftarrow$ Ch-$\pi(S^s \oplus \mathsf{bin}_n(i))$;
106. $C_{i+1}^s \leftarrow P_{i+1}^s \oplus Z_i^s$;
107. **end for**
108. $C_1^s \leftarrow CC^s$;
109. **return** $C_1^s || C_2^s || \ldots || C_{m^s}^s$

Figure 7.4: Games $\mathsf{G}_0$ and $\mathsf{G}_1$

Respond to the $s^{th}$ adversary query as follows:

ENCIPHER QUERY $\mathsf{Enc}(T^s; P_1^s, P_2^s, \ldots, P_{m^s}^s)$
$ty^s = \mathsf{Enc}$; $C_1^s||C_2^s||\ldots||C_{m^s-1}^s||D_{m^s}^s \xleftarrow{\$} \{0,1\}^{nm^s}$;
$C_{m^s}^s \leftarrow \mathsf{drop}_{n-r^s}(D_{m^s})$ **return** $C_1^s||C_2^s||\ldots||C_{m^s}^s$;

**Finalization:**

Case $ty^s = \mathsf{Enc}$:

$MM^s \leftarrow P_1^s \oplus H_h(P_2^s||\ldots||P_m^s||id^s)$;
$CC^s \leftarrow C_1^s \oplus H_h(C_2^s||\ldots||C_m^s||id^s)$;
$S^s \leftarrow MM^s \oplus CC^s$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{MM^s\}$;
$\mathcal{R} \leftarrow \mathcal{R} \cup \{CC^s\}$;
**for** $i = 2$ to $m^s - 1$,
$Y_i^s \leftarrow C_i^s \oplus P_i^s$;
$\mathcal{D} \leftarrow \mathcal{D} \cup \{S^s \oplus \mathsf{bin}_n(i-1)\}$;
$\mathcal{R} \leftarrow \mathcal{R} \cup \{Y_i^s\}$;
**end for**

SECOND PHASE
$\mathsf{bad} = \mathsf{false}$;
**if** (some value occurs more than once in $\mathcal{D}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**;
**if** (some value occurs more than once in $\mathcal{R}$) **then** $\mathsf{bad} = \mathsf{true}$ **endif**.

Figure 7.5: Game $\mathsf{G}_2$

encryption queries. Hence,

$$\Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1] = \Pr[A^{\$(.,.)} \Rightarrow 1] \tag{7.7}$$

So using Equations (6.1), (7.6) and (7.7) we get

$$\begin{aligned} \mathbf{Adv}^{\text{det-cpa}}_{\mathsf{OHCTRa}\,[\mathrm{Perm}(n)]}(\mathcal{A}) &= |\Pr[\mathcal{A}^{\mathbf{r}_\pi} \Rightarrow 1] - \Pr[A^{\$(.,.)} \Rightarrow 1]| && (7.8)\\ &= |\Pr[\mathcal{A}^{\mathsf{G}_0} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{G}_1} \Rightarrow 1]| \\ &\leq \Pr[A^{\mathsf{G}_1} \text{ sets } \mathsf{bad}] && (7.9) \end{aligned}$$

**Game $\mathsf{G}_2$:** Now we make some subtle changes in the game $\mathsf{G}_1$ to get a new game $\mathsf{G}_2$ which is described in Figure 7.5. In game $\mathsf{G}_1$ the permutation was not maintained and a call to the permutation was responded by returning random strings, so in Game $\mathsf{G}_2$ we no more use the subroutine Ch-$\pi$. Here we immediately return random strings to the adversary in response to his encryption queries. Later, in the finalization step we adjust variables and maintain multisets $\mathcal{D}$ and $\mathcal{R}$ where we list the elements that were supposed to be inputs and outputs of the permutation. In the second phase of the finalization step, we check for collisions in the sets $\mathcal{D}$ and $\mathcal{R}$, and in the event of a collision we set the bad flag to true.

Game $\mathsf{G}_1$ and Game $\mathsf{G}_2$ are indistinguishable to the adversary, as in both cases it gets random strings in response to his queries. Also, the probability with which $\mathsf{G}_1$ sets bad is same as the probability with which $\mathsf{G}_2$ sets bad. Thus we get:

$$\Pr[\mathcal{A}^{\mathsf{G}_1} \text{ sets bad}] = \Pr[\mathcal{A}^{\mathsf{G}_2} \text{ sets bad}] \tag{7.10}$$

Thus from Equations (7.9) and (7.10) we obtain

$$\mathbf{Adv}^{\text{det-cpa}}_{\text{OHCTRa}[\text{Perm}(n)]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{\mathsf{G}_2} \text{ sets bad}] \tag{7.11}$$

Now our goal would be to bound $\Pr[A^{\mathsf{G}_2} \text{ sets bad}]$. We notice that in Game $\mathsf{G}_2$ the bad flag is set when there is a collision in either of the sets $\mathcal{D}$ or $\mathcal{R}$. So if COLLD and COLLR denote the events of a collision in $\mathcal{D}$ and $\mathcal{R}$ respectively then we have

$$\Pr[A^{\mathsf{G}_2} \text{ sets bad}] \leq \Pr[\mathsf{COLLR}] + \Pr[\mathsf{COLLD}].$$

The rest of the proof is devoted to bound $\Pr[\mathsf{COLLR}]$ and $\Pr[\mathsf{COLLD}]$. In the analysis we consider the sets $\mathcal{D}$ and $\mathcal{R}$ to consist of the formal variables instead of their values. For example, whenever we set $\mathcal{D} \leftarrow \mathcal{D} \cup \{X\}$ for some variable $X$ we think of it as setting $\mathcal{D} \leftarrow \mathcal{D} \cup \{\text{“}X\text{”}\}$ where “$X$” is the name of that formal variable. Our goal is to bound the probability that two formal variables in the sets $\mathcal{D}$ and $\mathcal{R}$ take the same value. After $q$ queries of the adversary where the $s^{th}$ query has $m^s$ blocks of plaintext and $t$ blocks of tweak, then the sets $\mathcal{D}$ and $\mathcal{R}$ can be written as follows:

$$\begin{aligned}
\mathcal{D} &= \{MM^s : 1 \leq s \leq q\} \bigcup \{S^s_j : 1 \leq s \leq q; 1 \leq i \leq m^s - 1\}, \\
\mathcal{R} &= \{CC^s : 1 \leq s \leq q\} \bigcup \{Y^s_i : 2 \leq i \leq m^s; 1 \leq s \leq q\}.
\end{aligned}$$

where

$$\begin{aligned} MM^s &= P_1^s \oplus H_h(P_2^s \parallel \cdots \parallel P_{m^s}^s \parallel \mathsf{id}^s), \\ S_j^s &= S^s \oplus \mathsf{bin}_n(j) = (P_1^s \oplus C_1^s) \oplus (H_h(P_2^s \parallel \cdots \parallel P_{m^s}^s \parallel \mathsf{id}^s) \oplus \mathsf{bin}_n(j)), \\ CC^s &= C_1^s, \\ Y_i^s &= C_i^s \oplus P_i^s. \end{aligned}$$

Before we compute the collision probabilities in the sets $\mathcal{D}$ and $\mathcal{R}$, it is important to note the following points.

1. The ciphertext blocks $C_i$ are all generated uniformly at random from $\{0,1\}^n$.
2. The hash key $h$ is generated uniformly at random from $\{0,1\}^n$ and is independent of the $C_i$s, and $P_i$s. Note that $P_i$s are supplied by the adversary, the $i^{th}$ query supplied by the adversary may depend on the previous outputs obtained by the adversary, but as the output of game $G_2$ is not dependent in any way on the hash key $h$ thus the queries supplied by the adversary are independent of $h$.
3. We consider $\mathsf{id}^s$ as $t$ $n$-bit blocks. Thus, for any $s$, $H_h(P_2^s||\cdots||P_{m^s}^s||id^s)$ has degree at most $m^s+t$. We denote $\sigma = qt + \sum_s m^s$. We denote $\ell^{s,s'} = \max\{m^s, m^{s'}\}+t$. Since $\ell^{s,s'} \leq m^s + m^{s'} + t$, we have the following inequality

$$\begin{aligned} \sum_{1\leq s<s'\leq q} \ell^{s,s'} &\leq \binom{q}{2} t + \sum_{1\leq s<s'\leq q} (m^s + m^{s'}) \\ &\leq \binom{q}{2} t + (q-1)(\sigma - qt) \\ &\leq (q-1)\sigma + \frac{qt(q-1)}{2} - qt(q-1) \\ &\leq (q-1)\sigma. \end{aligned} \tag{7.12}$$

Now we make the following claims:

**Claim 7.1.** *For any $(s,s')$ such that $s \neq s'$, $\Pr[MM^s = MM^{s'}] = \ell^{s,s'}/2^n$.*

*Proof.* Let, $\Xi^s = P_2^s \parallel \cdots \parallel P_{m^s}^s \parallel T^s$, hence

$$MM^s \oplus MM^{s'} = (P_1^s \oplus P_1^{s'}) \oplus H_h(\Xi^s) \oplus H_h(\Xi^{s'}).$$

*Case 1:* $\Xi^s = \Xi^{s'}$. Then $P_1^s \neq P_1^{s'}$ as $\mathcal{A}$ does not repeat any query. Thus in this case $\Pr[MM^s = MM^{s'}] = 0$
*Case 2:*$\Xi^s \neq \Xi^{s'}$. In this case $MM^s \oplus MM^{s'}$ is a non zero polynomial of degree $\ell^{s,s'}$ on $h$. As $h$ is a uniform random string in $\{0,1\}^n$, hence $\Pr[MM^s \oplus MM^{s'} = 0] \leq \ell^{s,s'}/2^n$. □

**Claim 7.2.** *For* $(s,i) \neq (s',j)$, $\Pr[S_i^s = S_j^{s'}] \leq 1/2^n$.

*Proof.* Note, that $S_j^s = S^s \oplus \mathsf{bin}_n(j) = (P_1^s \oplus C_1^s) \oplus (H_h(P_2^s \parallel \cdots \parallel P_{m^s}^s \parallel \mathsf{id}^s) \oplus \mathsf{bin}_n(j))$. If $s = s'$, then $S_i^s \oplus S_j^s = \mathsf{bin}_n(i) \oplus \mathsf{bin}_n(j) \neq 0$, as $i \neq j$, thus making $\Pr[S_i^s = S_j^{s'}] = 0$. If $s \neq s'$ then $C_1^s$ and $C_1^{s'}$ are two uniform and independent strings in $\{0,1\}^n$, thus making $\Pr[S_i^s = S_j^{s'}] \leq 1/2^n$. □

**Claim 7.3.** *For* $1 \leq s,s' \leq q$ *and* $1 \leq j \leq m^s - 1$, $\Pr[S_j^s = MM^{s'}] = 1/2^n$.

*Proof.* As $S_j^s$ contains the term $C_1^s$, which is a uniform random element in $\{0,1\}^n$. □

To compute the collision probability in $\mathcal{D}$, we need to consider the collision probability of each pairs of variables in $\mathcal{D}$, it is easy to see that $\Pr[\mathsf{COLLD}] = p_1 + p_2 + p_3$, where

$$\begin{aligned} p_1 &= \Pr[MM^s = MM^{s'} : \text{for some } 1 \leq s < s' \leq q], \\ p_2 &= \Pr[S_i^s = S_j^{s'} : 1 \leq s,s' \leq q; 1 \leq i \leq m^s - 1; 1 \leq j \leq m^{s'} - 1; (s,i) \neq (s',j)], \\ p_3 &= \Pr[S_j^s = MM^{s'} : 1 \leq s,s' \leq q; 1 \leq j \leq m^s - 1]. \end{aligned}$$

From claims 7.1, 7.2 and 7.3 we have

$$\begin{aligned} p_1 &\leq \sum_{1 \leq s < s' \leq q} \ell^{s,s'} \leq \frac{q(\sigma-1)}{2^n}, \\ p_2 &\leq \binom{\sigma - q - qt}{2} / 2^n, \\ p_3 &\leq \frac{q\sigma}{2^n}. \end{aligned}$$

Which gives us,

$$\Pr[\mathsf{COLLD}] \leq \frac{5\sigma^2}{2^{n+1}}.$$

The variables present in $\mathcal{R}$ are $CC^s$ and $Y_i^s$ and both are uniform random elements in $\{0,1\}^n$. Hence for variables $X, Y \in \mathcal{R}$ such that $X \neq Y$, the probability that they take the same value is at most $1/2^n$. Now, as there are $q + \sum_{s=1}^{q}(m_s - 1) = \sigma$ many elements in $\mathcal{R}$, hence we have

$$\Pr[\mathsf{COLLR}] \leq \binom{\sigma}{2} / 2^n \leq \frac{\sigma^2}{2^{n+1}}$$

Hence putting all together, we have from Eq. (7.11)

$$\mathbf{Adv}^{\text{det-cpa}}_{\text{OHCTRa}[\text{Perm}(n)]}(\mathcal{A}) \leq \frac{3\sigma^2}{2^n},$$

as desired. ■

## 7.3 Summary

In this chapter we described some practical constructions of DEAD. We also proved its security. In the next chapter we discuss the details of our implementations and we report some performance data.

# Chapter 8

# Implementations

> One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.
>
> *Ada Lovelace*

In this chapter we discuss in details the experimental results obtained from our implementations of authentication (RDAS) and privacy (ESRQ) schemes. In Section 8.1 we discuss the system information and then in Sections 8.2 and 8.3 we discuss the details of the experiments and we report performance data for our different versions of RDAS and ESRQ1 respectively.

## 8.1 System Information

The results of all experiments were obtained by testing the implementations in a machine with the following specifications:

- **CPU:** Four-core i5-2400 Intel processor (3.1GHz).
- **OS:** Ubuntu 12.04.4 LTS.
- **DataBase:** PostgreSQL 9.2.6
- **Compiler:** gcc 4.7.3

We use Census-Income data set [89] to test performance of our relational database authentication schemes. This data contains weighted census data extracted from the 1994

and 1995 population surveys conducted by the U.S. Census Bureau. The number of tuples in the data set is 199523. The data contains 42 demographic and employment related attributes, the sum of the cardinalities of all the attributes is 103419, and the total size of the dataset is 99.1 MB. To test the performance of ESQR1, we create a new data based on the Census-Income. Out of the total 42 attributes, we chose 5 range attributes and 2 alphanumerical attributes only for the experiments. In these attributes, all values has a length less than 128 bits.

As explained, our schemes work in an environment where one needs to perform computations in both the client and the server side. In the authentications schemes all server-side computations are implemented in the PostgreSQL database using the PostgreSQL tools. We implemented the client in C, wherever possible we used the Intel SIMD instructions using Intel intrinsics. We designed the server side code in such a way that all computations can be handled by the default PostgreSQL tools. In case of our ESQR1 we used C for some server side computations also.

This specific implementation choice makes our client much more powerful than the server, and also it leaves space for a much more optimized implementation. Such an optimized implementation would require the development of all database engine functionalities, which we think is beyond the scope of this work. But, we would like to mention that this specific design choice also gives us the opportunity to see how good one can do by adding the functionalities to an already existing database system.

## 8.2 Results on the Query Authentication Schemes

In this thesis we have proposed two schemes for authenticated query processing: RDAS1 and RDAS2. The former allows to authenticate queries over an arbitrary number of disjunctions. The later allows to authenticate queries that involves an arbitrary logical operations. In Section 8.2.1 we describe the implementational issues of the basic building blocks used for the RDAS constructions. In Sections 8.2.3 and 8.2.4 we give the detailed performance results in the different versions of RDAS.

### 8.2.1 The Basic Building Blocks

Both RDAS1 and RDAS2 can be implemented with any secure MAC, we chose two MACs for our implementations (a) PMAC instantiated with an AES with 128 bit key (in particular we use the description in [90]) and (b) Polynomial evaluation Mac (which

we will further call as PolyMac).

PMAC is a block cipher based MAC where the main operations involved are block cipher calls. The way we implement PolyMac is as follows. Let $X_1||X_2||\ldots||X_m = \mathsf{Format}_n(X)$, and let $\mu \in \{0,1\}^n$, we define

$$\mathsf{PolyMac}_{h,k}(X,\mu) = (X_1 h \oplus X_2 h^2 \oplus \mathsf{cpad}_n(X_m)h^m \oplus |X|h^{m+1}) \oplus E_k(\mu), \tag{8.1}$$

where $X$ is the message and $\mu$ a non-repeating quantity associated with each message, $E_k()$ is a block cipher and $\mathsf{cpad}_n(X)$ pads the necessary number of zeros to the end of $X$, to make it $n$-bit long. The additions and multiplications in Eq. (8.1) are in the field $GF(2^n)$. Such polynomial evaluation MACs are known to be secure when the quantity $\mu$ is non-repeating. For our implementations we take $n = 128$, and we consider the attribute `Nonce` in $R_\alpha$ and `RowNo` in $R_\beta$ as the quantity $\mu$. For PolyMac also we choose the block cipher as AES with 128 bit key.

One thing to notice is that PolyMac is not a deterministic MAC. It is a stateful MAC, the quantity $\mu$ is a state of the algorithm and repetition of $\mu$ completely breaks down its security. As we stated in Section 5.3.1 only deterministic MACs can be aggregated, thus, Theorem 5.2 for aggregated MACs does not hold for PolyMac. Thus PolyMac cannot be used in RDAS1-agg and RDAS2-agg.

For implementation of the block cipher in both MACs we use the new Intel dedicated instructions for AES. Finite field multiplications required for the PolyMac were implemented using the `PCLMULQDQ` instruction, which can perform carry-less multiplication of two 64 bit strings. These 64 bit multiplications were combined using the Karatsuba technique to obtain multiplication of two 128 bit strings, the final reduction was performed using a technique described in [91].

### 8.2.2 Experimental Settings

The experiments were performed using the set of queries presented in Table 8.1 (the specific queries can be seen in Appendix A). Table 8.1 shows the characteristics of the queries in terms of the number of restrictions, the query type and the size of the query response. The restrictions are all equality conditions aggregated by some Boolean operators. Query Q1-Q5 are disjunctions of equality conditions, whereas the rest of the queries have additional Boolean operators like AND and NOT. The last column shows the percentage of the response size in terms of the whole database size. Note that the number of restrictions corresponds to the number of tuples which would be included in a correct and complete $\rho_\beta$ response and the response size would be same as the number of tuples in the $\rho_\alpha$ result.

| Query Id | Number of Restrictions | Query type | Response Size (tuples) | Database Percentage |
|---|---|---|---|---|
| Q1 | 10 | OR | 20115 | 10 |
| Q2 | 20 | OR | 35452 | 18 |
| Q3 | 30 | OR | 92791 | 46 |
| Q4 | 40 | OR | 106065 | 53 |
| Q5 | 50 | OR | 198869 | 99 |
| Q6 | 3 | OR, AND | 4016 | 2 |
| Q7 | 3 | OR, AND, NOT | 10354 | 5 |
| Q8 | 4 | OR, AND | 24722 | 12 |
| Q9 | 3 | OR, AND | 64028 | 32 |

Table 8.1: Summary of the different queries used for performance testing

| Query Id | Number of Attributes | Updated Bitmaps | New Bitmaps |
|---|---|---|---|
| I1 | 42 | 42 | 0 |
| I2 | 42 | 30 | 12 |
| U1 | 1 | 2 | 0 |
| U2 | 1 | 1 | 1 |

Table 8.2: Summary of the different insertions, updates used for performance testing

The dynamic databases experiments were performed using the set of queries presented in Table 8.2. Table 8.2 shows the characteristics of the queries in terms of the number of attributes affected, the number of bitmaps that need to be updated and number of bitmaps that need to be created. The first two lines correspond to an insertion queries, and the last two lines correspond to an update queries.

We present results for the scenarios presented in Table 8.3. Table 8.3 shows the queries that can be managed in each scenario.

RDAS1 in all its variants only can manage the queries Q1-Q5, because these are queries in which the Boolean connectives are disjunctions. On the other hand RDAS2 can handle all the queries Q1-Q9, because it is designed to work with queries involving all kinds of Boolean connectives.

RDAS1-agg and RDAS2-agg are implemented only using PMAC as PolyMac cannot be used as an aggregate MAC. As stated before, we implemented the aggregation at the server side with PostgreSQL XOR function and at the client side with the Intel SIMD instruction for xor, this of course has performance implications that we discuss later.

RDAS2 is only implemented with compression, we name the variant as RDAS2-cmp. As in RDAS1 explicit bitmaps are neither stored nor transmitted, hence the compressed bitmap version is not applicable in case of RDAS1. For compressing the bitmaps and

| Scenario | Queries |
|---|---|
| RDAS1 PolyMac | Q1-Q5 |
| RDAS1 PMAC | Q1-Q5 |
| RDAS1-agg | Q1-Q5 |
| RDAS2 PolyMac | Q1-Q9 |
| RDAS2 PMAC | Q1-Q9 |
| RDAS2-agg | Q1-Q9 |
| RDAS2-cmp PolyMac | Q1-Q8 |
| RDAS2-cmp PMAC | Q1-Q8 |
| RDAS2-dyn PolyMac | I1,I2-U1,U2 |
| RDAS2-dyn PMAC | I1,I2-U1,U2 |

Table 8.3: Summary of the different scenarios used for performance testing

applying logic operations on them we used the Lemire library [92]. This library only implements OR, AND, XOR, operations over compressed bitmaps. This is the reason why we report only results for queries Q1-Q9 except Q7 for RDAS2-cmp. Though using these basic operations as provided by the library one can implement other logic operations, but we have not done this, as we feel that for a proof of concept the query classes that we handle would be enough.

RDAS2 is implemented with the new characteristics required to manage insertions, updates and deletions (See Section 5.5). This new variant is called RDAS2-dyn. This version was implemented using PolyMac and also PMAC. RDAS2-dyn was tested for its new funtionalities using the queries in Table 8.2.

### 8.2.3 Experimental Results on RDAS1 and its Variants

In Table 8.4 we report the time required for executing the set of queries (from Q1 to Q5). We report the normal time (i.e., the time for execution without any authentication) along with the times required for RDAS1 with both PolyMac and PMAC, and for RDAS1-agg. All reported times are in milliseconds and it is the average of 250 executions of the same query. In Table 8.4 we also report the extra overhead of each of our schemes over the normal scheme without authentication. The data presented in Table 8.4 is also presented pictorially in Figure 8.1

The results of Table 8.4 show that in general PMAC is marginally faster than PolyMac. Even though the aggregated scheme is implemented with PMAC, the results show that this scheme is very expensive, in general it takes a bit more than the double of the normal time (time to respond the query without authentication). This can be explained by the fact that in case of RDAS1-agg the server and the client need to calculate the aggregated tags, this is done by XORing all the involved tags. We will see in Table 8.5

| Query | Normal time | RDAS1 [PolyMac] | | RDAS1 [PMAC] | | RDAS1-agg [PMAC] | |
|---|---|---|---|---|---|---|---|
| Id | - | Avg time | Extra Overhead(%) | Avg time | Extra Overhead(%) | Avg time | Extra Overhead(%) |
| Q1 | 437.05 | 525.74 | 20.29 | 522.01 | 19.44 | 879.69 | 101.28 |
| Q2 | 747.58 | 1062.52 | 42.13 | 1048.32 | 40.23 | 1743.47 | 133.22 |
| Q3 | 1708.08 | 2668.86 | 56.25 | 2652.40 | 55.29 | 4025.26 | 135.66 |
| Q4 | 1944.71 | 2895.41 | 48.88 | 2877.48 | 47.97 | 4357.16 | 124.05 |
| Q5 | 3739.53 | 7568.17 | 102.38 | 7564.11 | 102.27 | 10357.86 | 176.98 |

Table 8.4: Execution times for OR queries with RDAS1 (milliseconds).

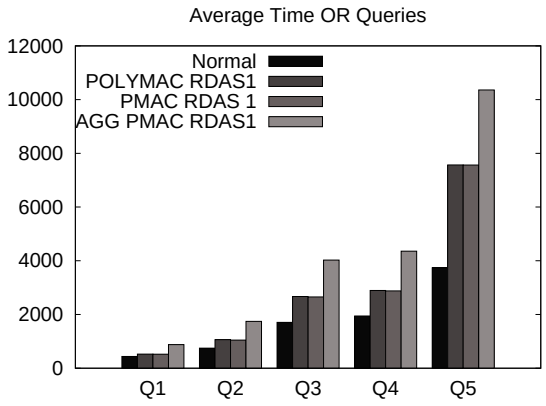



Figure 8.1: RDAS1 OR queries times (milliseconds).

that the time for responding the query increases significantly in RDAS1-agg compared to RDAS1.

In Table 8.5 we present the following times:

1. Time required for query translation (the $\Phi$ function).
2. Time required by the server to respond to the query (the $\Psi$ function).
3. The time required for verification (the $\mathcal{V}$ function).

Note that out of the above functions only the response procedure ($\Psi$) is executed in the server, and the other two procedures run in the client. Also the time for execution of $\mathcal{V}$ is only dependent on the type of message authentication code used.

In Table 8.5 we report the average time for each of these functions. We show the verification time for both the MACs in case of RDAS1. Table 8.5 clearly shows that the time required for query translation is negligible, the most of the time is spent in the server response and the verification procedures.

The time reported for the server response procedure in the case of RDAS1-agg is significantly more than that of RDAS1. This is because that in case of RDAS1-agg, the server needs to aggregate the MACs before sending them. Thus, the server needs to compute xors equal to the total number of tuples in $\rho_\alpha$ and $\rho_\beta$. And these operations are not required to be performed by the server in case of RDAS1. As the server side is implemented using the PostgreSQL, these XORs are expensive. But the verification procedure in case of RDAS1-agg is marginally faster than in RDAS1. Note that the number of MAC computations in the verification process of both RDAS1 and RDAS1-agg are the same. But, in case of RDAS1 the verification process requires to compare each computed tag with the tag received. But in case of RDAS1-agg, individual tag verification is not required, here the computed tags are xor-ed and the final value is compared with the aggregated tag, which received as a part of the query response. As stated, the verification process is implemented using SIMD instructions, hence the aggregation does not take as much time as individual comparison of the tags.

The communication overhead of RDAS1 and RDAS1-agg is discussed in Sections 4.2.2 and 5.3 respectively. Specifically Eq. (4.3) gives an upper bound on the response size for RDAS1 and Eq. (5.2) gives the same for RDAS1-agg. In Table 8.6 we give the numerical values of the response size for the specific queries used in our implementation. In Table 8.6, columns 2 and 3 represents the size in tuples for $\rho_\alpha$ and $\rho_\beta$ respectively. The values in columns labeled $\mathsf{siz}_{\mathsf{RD1}} - \mathsf{siz}$ and $\mathsf{siz}_{\mathsf{AGG-RD1}} - \mathsf{siz}$ represent the extra size of the response (in bytes) in case of RDAS1 and RDAS1-agg respectively. These were calculated using equations( 4.3) and (5.2) respectively. For these calculations we assume the tag size to be 16 bytes, the size for the `Nonce` and `RowNo` as 4 bytes, and $s_{\mathsf{Name}} + s_{\mathsf{sk}} = 200$ bytes. Table 8.6 clearly shows that RDAS1-agg has significantly lower communication cost compared to RDAS1.

| Query | RDAS1 | | | | RDAS1-agg | | |
|---|---|---|---|---|---|---|---|
| Id | $\phi$ | $\psi$ | $\mathcal{V}$[PolyMac] | $\mathcal{V}$[PMAC] | $\phi$ | $\psi$ | $\mathcal{V}$ [PMAC] |
| Q1 | .0127 | 442.78 | 83.26 | 77.86 | .0170 | 811.60 | 76.56 |
| Q2 | .0260 | 843.92 | 216.33 | 202.67 | .0339 | 11558.59 | 201.84 |
| Q3 | .0262 | 1806.30 | 862.74 | 844.75 | .0422 | 3209.64 | 827.17 |
| Q4 | .0262 | 1980.46 | 922.22 | 901.82 | .0382 | 3489.79 | 884.24 |
| Q5 | .0419 | 3983.31 | 3638.79 | 3560.35 | .0677 | 6868.57 | 3532.19 |

Table 8.5: Execution times for primitives with RDAS1-OR queries (milliseconds).

| Query Id | Size $\rho_\alpha$ | Size $\rho_\beta$ | $siz_{RD1} - siz$ | $siz_{AGG-RD1} - siz$ |
|---|---|---|---|---|
| Q1 | 20115 | 10 | 404500 | 82532 |
| Q2 | 35452 | 20 | 713440 | 145920 |
| Q3 | 92791 | 30 | 1862420 | 377316 |
| Q4 | 106065 | 40 | 2130100 | 432452 |
| Q5 | 198869 | 50 | 3988380 | 805708 |

Table 8.6: Object Verification Extra-Size for RDAS1 in bytes.

### 8.2.4 Experimental results on RDAS2 and its Variants

In this section we present the results of the variants of RDAS2 in the same way as we did in the previous section for RDAS1. In Table 8.7 we report the time required for executing the set of queries (Q1-Q9) with RDAS2, RDSAS2-agg and RDAS2-cmp. In Table 8.8 we report the time taken for various sub-processes involved in the query execution. In Figures 8.2, 8.3 we present the data of Table 8.7 pictorically. In Table 8.9 we report data corresponding to the size of the response.

| Query | Normal | RDAS2 [PolyMac] | | RDAS2 [PMAC] | | RDAS2-agg [PMAC] | | RDAS2-cmp [PolyMac] | | RDAS2-cmp [PMAC] | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Id | time | Avg time | Over-head(%) | Avg time | Over-head(%) | Avg time | Over-head(%) | Avg time | Over-head(%) | Avg time | Over-head(%) |
| Q1 | 437.05 | 515.06 | 17.85 | 508.88 | 16.43 | 870.48 | 99.17 | 496.03 | 13.49 | 496.82 | 13.67 |
| Q2 | 747.58 | 971.57 | 29.96 | 969.90 | 29.74 | 1671.36 | 123.57 | 963.94 | 28.94 | 959.39 | 28.33 |
| Q3 | 1708.08 | 2316.89 | 35.64 | 2311.90 | 35.35 | 3719.12 | 117.74 | 2302.87 | 34.82 | 2178.18 | 27.52 |
| Q4 | 1944.71 | 2504.30 | 28.78 | 2502.96 | 28.68 | 3988.83 | 105.11 | 2454.10 | 26.19 | 2284.95 | 17.49 |
| Q5 | 3739.53 | 6331.86 | 69.32 | 6326.28 | 69.17 | 9130.61 | 144.16 | 6298.63 | 68.43 | 6266.40 | 67.57 |
| Q6 | 108.64 | 184.93 | 70.22 | 175.72 | 61.75 | 322.71 | 197.04 | 175.67 | 61.70 | 166.92 | 53.64 |
| Q7 | 182.37 | 286.44 | 57.06 | 276.79 | 51.77 | 488.53 | 167.87 | - | - | - | - |
| Q8 | 374.05 | 572.92 | 53.17 | 558.98 | 49.44 | 954.73 | 155.24 | 570.75 | 52.59 | 545.83 | 45.93 |
| Q9 | 784.72 | 1179.85 | 50.35 | 1162.56 | 48.15 | 1874.71 | 138.90 | 1166.25 | 48.62 | 1142.06 | 45.54 |

Table 8.7: Execution times with RDAS2 (milliseconds).

| Query | RDAS2 | | | | RDAS2-agg[PMAC] | | | RDAS2-cmp | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Id | $\phi$ | $\psi$ | $\mathcal{V}$ [PolyMac] | $\mathcal{V}$ [PMAC] | $\phi$ | $\psi$ | $\mathcal{V}$ | $\phi$ | $\psi$ | $\mathcal{V}$ [PolyMac] | $\mathcal{V}$[PMAC] |
| Q1 | 0.0117 | 457.20 | 55.62 | 55.09 | 0.2367 | 806.88 | 60.51 | 0.0128 | 451.98 | 54.87 | 54.79 |
| Q2 | 0.0227 | 857.55 | 109.95 | 108.99 | 0.2033 | 1534.68 | 120.56 | 0.0255 | 841.26 | 114.26 | 115.11 |
| Q3 | 0.0280 | 1815.84 | 500.97 | 498.63 | 0.1583 | 3177.27 | 513.94 | 0.0304 | 1805.03 | 498.38 | 499.64 |
| Q4 | 0.0254 | 1995.49 | 514.39 | 513.00 | 0.1618 | 3451.78 | 523.04 | 0.0291 | 1949.91 | 512.31 | 511.50 |
| Q5 | 0.0413 | 3962.19 | 2368.64 | 2355.91 | 0.2354 | 6875.67 | 2400.21 | 0.0442 | 3970.33 | 2369.98 | 2362.77 |
| Q6 | .0048 | 163.83 | 20.19 | 14.90 | 0.1663 | 301.95 | 20.07 | 0.0052 | 152.19 | 14.37 | 13.59 |
| Q7 | .0057 | 252.35 | 33.49 | 27.88 | 0.1662 | 451.71 | 33.90 | - | - | - | - |
| Q8 | .0085 | 497.18 | 70.75 | 61.29 | 0.2183 | 878.69 | 69.25 | 0.0097 | 490.65 | 60.42 | 60.34 |
| Q9 | .0061 | 1026.55 | 146.92 | 136.15 | 0.1671 | 1725.26 | 145.86 | 0.0065 | 1008.72 | 134.41 | 135.49 |

Table 8.8: Execution times for primitives with RDAS2 (milliseconds).

| Query Id | Size $\rho_\alpha$ | Size $\rho_\beta$ | $siz_{RD2} - siz$ | $siz_{AGG-RD2} - siz$ | $siz_{CMP-RD2} - siz$ |
|---|---|---|---|---|---|
| Q1 | 20115 | 10 | 653910 | 331942 | 524520 |
| Q2 | 35452 | 20 | 1212260 | 644740 | 796649 |
| Q3 | 92791 | 30 | 2610650 | 1125546 | 1933237 |
| Q4 | 106065 | 40 | 3127740 | 1430092 | 2187503 |
| Q5 | 198869 | 50 | 5235430 | 2052758 | 4086144 |
| Q6 | 4016 | 3 | 155803 | 91531 | 129932 |
| Q7 | 10354 | 3 | 282563 | 116883 | 277908 |
| Q8 | 24722 | 4 | 569923 | 199500 | 562899 |
| Q9 | 64028 | 3 | 1356043 | 331579 | 1338472 |

Table 8.9: Object Verification Extra-Size in bytes.

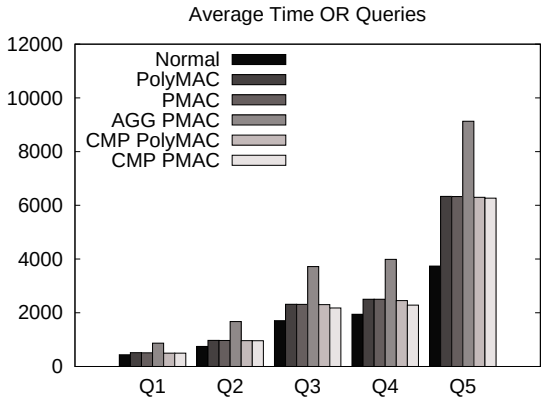



Figure 8.2: RDAS2 OR queries times (milliseconds).

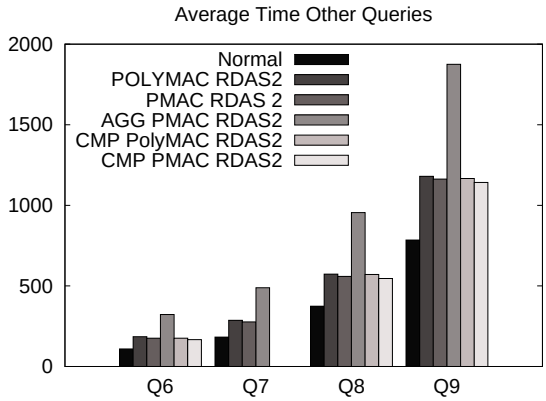



Figure 8.3: RDAS2 queries times (milliseconds).

From the Tables we can infer the following:

1. RDAS2-cmp has the best performance in terms of time for all queries, and RDAS2-agg has the worst performance. The reason for good performance of RDAS2-cmp is probably due to the fact that as compression reduces the size of the bitmaps, hence operations on them can be performed much more efficiently. The performance of RDAS2-agg can be further improved by optimizing the server. Table 8.8 clearly shows that for RDAS2-agg most of the time is consumed by the query response procedure ($\Psi$).

2. All versions of RDAS2 performs better than the corresponding versions of RDAS1. This is because in all versions of RDAS1 the verification process has to create the bitmaps, from the responses. This can take up some time. But in case of RDAS2, the bitmaps are already there as part of the response.

3. From Table 8.9, we can see that RDAS2-agg have the best performance in terms of communication cost, followed by RDAS2-cmp and RDAS2. But all versions of RDAS2 have more communication cost compared to the corresponding versions of RDAS1.

**RDAS2-dyn Version**

The experiments of this version are based on the queries described in Table 8.2. To interpret this table it is necessary to take a look to the description in Section 5.5, when a tuple is inserted or it is updated, two cases are considered: a) we are in presence of a new value, or b) the value exists in other tuple. In the former case a new bitmap needs to be inserted. In the last case the corresponding bitmap needs to be updated. Let us analyze the Table 8.2 which is composed by four columns, column 2 shows the number of attributes affected by the query. Column 3 shows the number of updated bitmaps, and column 4 shows the number of new bitmaps. All these insertions and updates are performed by the protocol, plus the original operation. For instance row 1 corresponds to query I1 that affects the 42 attributes, it updates 42 bitmaps and it does not insert any bitmap. In other words, this query inserts a tuple only with values that previously exist in the database.

The tuples for insertion in I1 were constructed by selecting a random value among the existing values for each attribute. On the other hand, the tuples for insertion in I2 were constructed with the same procedure except for the numeric attributes, for them new random values were included. The updates queries in U1 were constructed by randomly select a tuple and an attribute to be updated. The update value was choosen by randomly selecting a value in the domain of the attribute. The updates queries in U2 were constructed with the same procedure that in I2.

We report in Table 8.10 the time required for executing an insertion and an update in a plaintext database, along with the times required for RDAS2-dyn with both PolyMac and PMAC. All reported times are in milliseconds. For the insertion $I1$ experiment we report the average time over the minimums obtained in 10 executions for each tuple, and the average of 100 executions in any other case. The reported time involves all steps in the insertion and update procedures respectively, i.e., time for query translation, time required for the server to respond and the time for update the aditional information in $R_\beta, M_c, M_s$.

This version imposes a considerable time to be executed compared with the plaintext database. However, this can be explained by the fact that for each insertion or update query, several other update or insertion operations needs to be performed to maintain the consistency of the $R_\alpha$ and $R_\beta$ and to provide the authentication service. For instance

consider the case of query $I2$, while in the plaintext case just 1 insertion is done, RDAS2-dyn has to apply 30 updates and 12 insertions besides the original insertion to $R_\alpha$. The PMAC variant is a little bit faster than the PolyMac case.

| Query Id | Normal time | RDAS2-dyn [PolyMac] | RDAS2-dyn [PMAC] |
|---|---|---|---|
| I1 | 12 | 2500.36 | 2485.44 |
| I2 | 12 | 6930.64 | 6902.35 |
| U1 | 140.56 | 551.21 | 534.22 |
| U2 | 140.56 | 877.945 | 875.57 |

Table 8.10: Execution times for RDAS2-dyn (milliseconds).

## 8.3 Results on ESRQ1

In this section we discuss the performance of our implementation of ESQR1. For this implementation, we used the deterministic encryption scheme $\mathbf{E}_K$ as described in Figure 7.1 instantiated with an AES 128. As noted, $\mathbf{E}$ has a restriction on the message space, hence we chose only those attributes from the Census-data which satisfy this message length restriction. This would not have been required if we chose OHCTR as our deterministic encryption scheme. But, the basic performance in terms of query processing is expected to remain same with OHCTR, but the encryption decryption times may vary. In the implementations we use the array encoding of the $l$-bitmap matrix to represent the bitmaps.

The experiments were performed using the set of queries presented in Table 8.11 (the specific queries can be seen in Appendix A). Table 8.11 shows the characteristics of the queries in terms of the number of restrictions, the logical operators, the range operators $(<, >, \leq, \geq)$, the size of the query response and the number of bitmaps that are required to be computed to respond to the query. The restrictions are conditions aggregated by some Boolean operators. Note that the number of restrictions are equal to the number of bitmaps that will be aggregated to find out which tuples of $R_\alpha$ satisfy the query. The response size would be same as the number of the set bits in the final bitmap. The last column of the table shows the number of bitmaps that are to be created from the array encoding of the $l$-bitmap matrix to solve the query. For instance, in query Q2 just one bitmap is calculated because the required bitmap is the one that has $l$-encoding. Thus no further calculation is required. On the contrary, the other queries require creation of more bitmaps. For instance to solve query Q10, one $l$-bitamp, one $g$-bitmap and two equality bitmaps are required to be created.

| Query Id | Number of Restrictions | Logical operator | Range operators | Response Size (tuples) | No. of bitmaps to be computed |
|---|---|---|---|---|---|
| Q10 | 2 | AND | $\geq, \leq$ | 9483 | 4 |
| Q11 | 1 | - | $<$ | 2839 | 1 |
| Q12 | 1 | - | $>$ | 4648 | 3 |
| Q13 | 1 | - | $=$ | 2995 | 2 |
| Q14 | 2 | OR | $\geq, =$ | 8972 | 3 |

Table 8.11: Summary of the different queries used for performance testing of ESRQ1

In Table 8.12 we report the time required for executing the set of queries (Q10-Q14). We report the normal time (i.e., the time for execution in a plaintext database) along with the times required for ESQR1. All reported times are in milliseconds and is the average of 100 executions of the same query. The reported time involves the following steps: time for query translation (excluding the rewriting phase[1]), time required for the server to calculate the involved bitmaps and operate them, and the time to solve the last query, the one constructed with the row number RowNo of the tuples obtained from the resulting bitmap. In Table 8.12 we also report the extra overhead of each query on encrypted data against with the run on a plaintext database.

It is important to notice that queries Q11 and Q13 have an negative extra overhead, this make sense because in the array enconding of the *I*-bitmap matrix we have the pre-computed index to solve this type of queries, the normal execution instead has to transverse the index to find out which tuples are part of the result set. The query Q11 has better performance than query Q13, because as we have mentioned the $<$ query does not require any computation except recovery the bitmap. On the other hand Q13 involves an equality condition, thus two bitmaps have to be recovered and it has to be xored.

| Query | Normal time | ESRQ1 | |
|---|---|---|---|
| Id | - | Avg time | Extra Overhead(%) |
| Q10 | 33.12 | 38.38 | 15.89 |
| Q11 | 22.13 | 10.08 | −54.43 |
| Q12 | 22.52 | 27.48 | 22.03 |
| Q13 | 21.02 | 17.69 | −15.86 |
| Q14 | 35.03 | 40.38 | 15.28 |

Table 8.12: Execution times for queries with ESRQ1 (milliseconds).

In Table 8.13 we show the breakdown of ESRQ1 times presented in Table 8.12. To

[1]The queries that we select not require re-writing. As re-writing involves a client server interaction, hence measuring time with re-writing involves many complexities, and we do not consider them in our implementation.

interpret these results, it would be necessary to take a close look at the response procedure. The response procedure which runs at the server can be broken down into two phases, the first one involves creating the required bitmaps based on the conditions of the query and operating the bitmaps based on the Boolean connectives present in the query. As a final result of this phase a single bitmap is obtained and the bits that are set in this final bitmap corresponds to the row numbers of the tuples which would form the result. The second phase consists of collecting the tuples corresponding to these row numbers and sending them to the client.

Table 8.13 consists of four columns, the second column gives the total response time for the queries in a normal (un-encrypted ) database and the other two columns give the time for creating and operating on the bitmaps (i.e., the first phase) and the time required to obtain the tuples based on the evaluated bitmap. The query time for the separate phases were measured by the EXPLAIN ANALYZE tool of PostgreSQL. This tool devises a query plan and also performs it (analyze option), which allow us to see execution time without considering the time spent transmitting result rows to the client.

Table 8.13 clearly shows that the time spent in the first phase is always less than the time spent in the second phase. The cost of the bitmap operations depends on the required number of bitmap computations, which is the minimum for Q11 and maximum for Q14.

| Query Id | Normal Query | Phase 1 (Bitmap Operations) | Phase 2 (Query evaluation) |
|---|---|---|---|
| Q10 | 33.12 | 7.06 | 31.32 |
| Q11 | 22.13 | 1.52 | 8.57 |
| Q12 | 22.52 | 10.33 | 17.15 |
| Q13 | 21.02 | 7.93 | 9.75 |
| Q14 | 35.03 | 10.61 | 29.77 |

Table 8.13: Breakdown execution times for queries with ESRQ1 (milliseconds).

**Insertions and Updates:** In table 8.14 we report the time required for the operations insert and update in ESRQ1. For insertion we constructed a new tuple by selecting a random value among the existing values for each attribute. We created 100 such tuples and inserted them. The average time in milliseconds for insert both for a normal data base and ESRQ1 are shown in the second row of the table. For updates, we selected randomly a tuple, and an attribute and updated the value of the selected attribute with a randomly selected value in the domain of that attribute. Again we updated 100 tuples, and report the time in the third row of the table.

| Query Id | Normal Query | ESQR1 | Overhead |
|---|---|---|---|
| I0 | 13.08 | 20.74 | 58.55 |
| U0 | 12.41 | 20.65 | 66.42 |

Table 8.14: Insertions and updates times with ESRQ1 (milliseconds).

## 8.4 Final Remarks

We presented the experimental protocols and performance results of all the schemes. The performance results are in general encouraging, and it seems that the procedures when applied to databases will not give rise to significant extra overhead.

# Chapter 9

# Conclusions and Future Work

> One never notices what has been done; one can only see what remains to be done.
>
> *Marie Curie*

So far we have discussed several issues involving security of outsourced databases. Starting from Chapter 4 to Chapter 8 we presented several aspects of some new constructions that aim to solve the problems of authenticated query processing and privacy in relational databases. In this chapter we summarize the main contributions of our work and finally we note down some immediate thoughts to improve and extend this research.

## 9.1 Conclusions

In this thesis we have studied the problems of query authentication and privacy in outsourced databases. Our study includes three directions:

1. We studied the problems from a formal cryptographic viewpoint, which includes formulating appropriate security definitions for them.
2. Designing cryptographic schemes which provide solution to the problem. We focussed on proposing solutions which are both efficient and secure. We also proved security of our protocols in line with the proposed security definitions.
3. Finally we implemented these schemes and generated performance data in a realistic setting.

Next, we summarize our specific contributions in the above three directions.

For the problem of **authenticated query processing** we achieved the following:

1. **Formal Study**: The problem of authenticated query processing has not been studied before from a formal cryptographic perspective. In Chapter 4 on page 51 we initiate such a study, where we introduced a general cryptographic framework called RDAS. We specify a generic syntax for RDAS and formulate a novel security notion in line with the paradigm of concrete provable security. We think that this framework can accommodate other schemes proposed in the literature, and this formal model can facilitate the security analysis of some existing schemes.

2. **New Constructions**: In Chapter 4 on page 51, we proposed the basic scheme RDAS1 which allows to authenticate queries with arbitrary number of disjunction conditions. The main innovative idea of this scheme is that it uses a simple structure to provide *completeness*, named bitmaps, and it uses a lightweight cryptographic primitive called MAC. In Chapter 5 on page 65 we proposed several improvements over the RDAS1 scheme. We call the main improved scheme as RDAS2, that allows authentication of queries which involves an arbitrary number of conditions with any logical operator. We also develop other variants of RDAS1 and RDAS2, namely, RDAS1-agg, RDAS2-agg, that use aggregated message authentication codes and thus helps in reducing communication costs. We also propose RDAS2-cmp which uses compression bitmaps and thus helps in reducing both serverside storage and communication costs. Finally we propose RDAS2-dyn, which is suitable for dynamic databases. We provide a formal security analysis of all the proposed schemes and prove them to be secure in the proposed security model.

3. **Implementations**: We implemented all our proposed schemes, and we tested them in a real dataset, in Chapter 8 on page 125 we presented our experimental results over these implementations. The results are encouraging, and they suggest that they are practical and thus can be included in practical database systems without significant additional overhead.

We dealt with a small part of the problem of **privacy in outsourced databases**. We concentrated on database encryption schemes where range queries can be efficiently executed. In this problem we have the following contributions:

1. **Theoretical study**: The literature suggests that an order preserving encryption scheme should be used to encrypt databases so that range queries can be run on it. Recently, there have been ample theoretical studies involving order preserving encryption schemes and some security definitions for such schemes have

also been proposed. These studies only deal with the specific encryption primitive, and they not handle the scenario when such a primitive would be used in a real database environment. We provide a theoretical model for the problem of database encryption where range queries are possible. Our security definition is motivated by the IND-OCPA security definition of OPE schemes, but it is more suitable for application in a real database secenario.

2. **New Constructions**: In Chapter 6 on page 83, we proposed a new scheme for encrypted databases called ESRQ1. The main novelty of ESRQ1 lies in the fact that it uses a specific type of encoding of range bitmaps which we call the array encoding. The array encoding of bitmaps are used to reveal the order of the plaintexts. Other than this ESRQ1 uses a new cryptographic primitive called deterministic encryption with associated data (DEAD). DEAD is related to an existing primitive called deterministic authenticated encryption (DAE), but DEAD is less secure than DAE and it provides the exact security that is required for databases. In Chapter 7 on page 111 we also provide some practical constructions of this primitive.

3. **Implementations**: We implemented the ESQR1 scheme, and tested its performance on a real dataset. In Chapter 8 on page 125 we presented our experimental results on this implementation.

## 9.2 Future work

We note down some issues of immediate interest which were not treated in this thesis but we wish to take up in the near future:

1. **Security analysis of other schemes:** We have proposed a general framework for security of authenticated query processing problem. It will be interesting to perform a study of the security of other schemes proposed in the literature in our framework.

2. **Extend the type of queries.** The type of queries that our schemes RDAS1, RDAS2 can authenticate are still restricted. We wish to explore the possibility of extending the schemes so that other query types can be handle. Among others, the study of join bitmaps indices, and their applicability in processing join queries is of our immediate interest.

3. **Improvements over** RDAS-dyn**.** We have presented our RDAS-dyn as a scheme that can authenticate queries in the presence of insertions, updates and deletions.

It seems that the specific structure of *l*-bitmap matrices that we use in Chapter 6 will be better suited to design authentication schemes for dynamic databases. We plan to analyze this option.

4. ***l*-Bitmap Matrix.** We have proposed this new structure to allow efficient storing of bitmaps in the context of ESRQ1. We think that *l*-bitmap matrices and their array representations can be of independent interest and can be used as an index for ordinary databases also. A proper experimental evaluation of *l*-bitmap indices and a comparative analysis with other existing index structures is of immediate interest to us.

5. **A combined scheme for authentication and privacy**. Since our proposals for the query authentication problem and the privacy problem work based on bitmaps, we consider that the design of a single scheme that provides these two security services is possible.

6. **Cloud deployments.** Until now all our implementations are prototypical and does not run in the cloud, we have only simulated the client-server environment. Thus, it seems appealing to deploy our schemes in a real cloud scenario. This can generate some problems that we did not consider in this thesis.

7. **Cryptographic database engine.** A very challenging project is to develop a new database engine that facilitates the addition of cryptographic schemes as a natural part of the database operations.

# Appendix A

# Queries

Q1
```
SELECT * FROM data_alpha WHERE age='28' OR age='85' OR age='9'
OR age='80' OR age='26' OR age='3' OR age='33'
OR age='39' OR age='88' ORDER BY nonce
```

Q2
```
SELECT * FROM data_alpha WHERE occupation='23' OR occupation='7' OR age='66'
OR education='Bachelorsdegree(BAABBS)' OR wage='2208' OR wage='385'
OR wage='1121' OR wage='275' OR wage='2480' OR wage='1910' OR wage='690'
OR wage='468' OR wage='1471' OR wage='1385' OR wage='2800' OR wage='853'
OR wage='455' OR wage='2160' OR mstatus='Married-AFspousepresent'
OR moccup='Precisionproductioncraft&repair' ORDER BY nonce
```

Q3
```
SELECT * FROM data_alpha WHERE occupation='20' OR occupation='10' OR age='56'
OR wage='2138' OR wage='839' OR wage='936' OR wage='890' OR wage='1866'
OR wage='1315' OR wage='805' OR wage='466' OR wage='1314' OR wage='1195'
OR wage='1450' OR wage='691' OR wage='535' OR wage='735' OR wage='1119'
OR wage='3190' OR wage='1592' OR wage='1633' OR wage='5250'
OR mstatus='Nevermarried' OR indcode='Socialservices' OR losses='2457'
OR dividends='833' ORDER BY nonce
```

Q4
```
SELECT * FROM data_alpha WHERE age='57' OR wage='2198' OR wage='1971'
OR wage='1269' OR wage='709' OR wage='1311' OR wage='822' OR wage='671'
OR wage='1152' OR wage='1130' OR wage='1834' OR wage='879' OR wage='2988'
OR wage='758' OR wage='1730' OR wage='1042' OR wage='602' OR wage='1850'
OR wage='1107' OR wage='1035' OR indcode='Hospitalservices' OR sex='Female'
OR gains='14344' OR losses='1887' OR losses='1504' OR dividends='1059'
```

```
        OR dividends='157' OR dividends='2143' OR dividends='1240'
        OR dividends='762' order by nonce

Q5      SELECT * FROM data_alpha WHERE msa='972.45' OR msa='974.37'
        OR msa='975.77' OR msa='976.67' OR msa='976.83' OR msa='977.06'
        OR msa='977.59' OR msa='978.89' OR msa='979.62' OR msa='980.61'
        OR msa='981.86' OR msa='982.11' OR msa='982.38' OR msa='982.54'
        OR msa='983.6' OR msa='984.26'OR msa='984.64' OR msa='985.58'
        OR msa='985.73' OR msa='985.75' OR msa='988.21' OR msa='988.73'
        OR msa='989.41' OR msa='989.42' OR msa='989.76' OR msa='98.98'
        OR msa='990.74' OR msa='992.42' OR msa='992.62' OR msa='993.1'
        OR msa='993.39' OR msa='993.45' OR msa='996.35' OR msa='998.11'
        OR msa='998.16' OR msa='998.18' OR msa='998.63' OR msa='999.22'
        OR msa='999.43' OR msa='999.56' OR live='Nonmover'
        OR live='DifferentstateinMidwest' OR self='Puerto-Rico'
        OR self='Nicaragua' OR self='India' OR self='Taiwan'
        OR citizen='Taiwan' OR business='Native-BornintheUnitedStates'
        OR year='43' OR band='-50000.'  order by nonce

Q6      Select * FROM data WHERE age='45' OR age='50' AND sex='Female'

Q7      SELECT * FROM data WHERE enroll='Highschool'
        OR enroll='Collegeoruniversity' AND NOT sunbelt='Yes'
Q8      Select * FROM data WHERE mstatus='Divorced' OR mstatus='Widowed'
        OR househ='Householder' AND age='35'

Q9      SELECT * FROM data WHERE class='Private' AND race='White'
        OR race='AmerIndianAleutorEskimo'

Q10     Select * FROM smalldata_num WHERE age ≥ 8 AND age ≤ 10

Q11     Select * FROM smalldata_num WHERE age < 1

Q12     Select * FROM smalldata_num WHERE age > 80

Q13     Select * FROM smalldata_num WHERE age = 42

Q14     Select * FROM smalldata_num WHERE age ≤ 1 OR age = 42
```

# Publications of the Author Related to the Thesis


1. Lil María Rodríguez-Henríquez and Debrup Chakraborty. RDAS: A symmetric key scheme for authenticated query processing in outsourced databases. In Rafael Accorsi and Silvio Ranise, editors, *Security and Trust Management-9th International Workshop, STM 2013, Egham, UK, September 12-13, 2013. Proceedings, volume 8203 of Lecture Notes in Computer Science*, pages 115-130. Springer, 2013.

2. Lil María Rodríguez-Henríquez and Debrup Chakraborty. RDAS: A symmetric key scheme for authenticated query processing in outsourced databases. *IACR Cryptology ePrint Archive,* 2013:814, 2013.

3. Lil María Rodríguez-Henríquez and Debrup Chakraborty. Using bitmaps for executing range queries in encrypted databases. In Mohammad S. Obaidat, Andreas Holzinger, and Pierangela Samarati, editors, *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014*, pages 432-438. SciTePress, 2014.

# Bibliography


[1] Wenchao Zhou, William R. Marczak, Tao Tao, Zhuoyao Zhang, Micah Sherr, Boon Thau Loo, and Insup Lee. Towards secure cloud data management. url: http://netdb.cis.upenn.edu/papers/ds2_socc10.pdf, 2010.

[2] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. url: https://cloudsecurityalliance.org/csaguide.pdf, 2009.

[3] Hovav Shacham and Brent Waters. Compact proofs of retrievability. *J. Cryptology*, 26(3):442–483, 2013.

[4] Hakan Hacigümüs, Sharad Mehrotra, and Balakrishna R. Iyer. Providing database as a service. In Rakesh Agrawal and Klaus R. Dittrich, editors, *ICDE*, pages 29–38. IEEE Computer Society, 2002.

[5] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In Galen C. Hunt, editor, *HotOS*. USENIX Association, 2007.

[6] Qingji Zheng, Shouhuai Xu, and Giuseppe Ateniese. Efficient query integrity for outsourced dynamic databases. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 71–82, New York, NY, USA, 2012. ACM.

[7] Ladjel Bellatreche, Rokia Missaoui, Hamid Necir, and Habiba Drias. Selection and pruning algorithms for bitmap index selection problem using data mining. In Il Yeal Song, Johann Eder, and Tho Manh Nguyen, editors, *DaWaK*, volume 4654 of *Lecture Notes in Computer Science*, pages 221–230. Springer, 2007.

[8] R. Wrembel and C. Koncilia. *Data warehouses and OLAP: concepts, architectures, and solutions*. Gale virtual reference library. IRM Press, 2007.

[9] Divyakant Agrawal, Amr El Abbadi, Fatih Emekçi, and Ahmed Metwally. Database management as a service: Challenges and opportunities. In Yannis E.

Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 1709–1716. IEEE, 2009.

[10] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 578–595. Springer, 2011.

[11] Seungmin Lee, Tae-Jun Park, Donghyeok Lee, Taekyong Nam, and Sehun Kim. Chaotic order preserving encryption for efficient and secure queries on databases. *IEICE Transactions*, 92-D(11):2207–2217, 2009.

[12] DaeHyun Yum, DukSoo Kim, JinSeok Kim, PilJoong Lee, and SungJe Hong. Order-preserving encryption for non-uniformly distributed plaintexts. In Souhwan Jung and Moti Yung, editors, *Information Security Applications*, volume 7115 of *Lecture Notes in Computer Science*, pages 84–97. Springer Berlin Heidelberg, 2012.

[13] Liangliang Xiao and I-Ling Yen. A note for the ideal order-preserving encryption object and generalized order-preserving encryption. *IACR Cryptology ePrint Archive*, 2012:350, 2012.

[14] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy*, pages 463–477. IEEE Computer Society, 2013.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, USA, 1996.

[16] National Institute of Standards and Technology. *FIPS PUB 46-3: Data Encryption Standard (DES)*. pub-NIST, pub-NIST:adr, oct 1999. supersedes FIPS 46-2.

[17] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[18] Cuauhtemoc Mancillas López. *Studies on Disk Encryption*. PhD thesis, Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional, 2013.

[19] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[20] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer, 2004.

[21] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.

[22] Debrup Chakraborty and Palash Sarkar. On modes of operations of a block cipher for authentication and authenticated encryption. Cryptology ePrint Archive, Report 2014/627, 2014. http://eprint.iacr.org/.

[23] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.

[24] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Lee [93], pages 16–31.

[25] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006.

[26] Tetsu Iwata and Kan Yasuda. BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2009.

[27] Tetsu Iwata and Kan Yasuda. HBS: A single-key mode of operation for deterministic authenticated encryption. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009.

[28] Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.

[29] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Boneh [94], pages 482–499.

[30] David A. McGrew and Scott R. Fluhrer. The security of the extended codebook (XCB) mode of operation. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[31] Debrup Chakraborty, Cuauhtemoc Mancillas-Lopez, and Palash Sarkar. STES: A stream cipher based low cost scheme for securing stored data. Cryptology ePrint Archive, Report 2013/347, 2013. http://eprint.iacr.org/.

[32] Tetsu Iwata and Kaoru Kurosawa. OMAC: One-Key CBC MAC. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 129–153. Springer, 2003.

[33] John Black and Phillip Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *EUROCRYPT*, pages 384–397, 2002.

[34] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[35] Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[36] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.

[37] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1996.

[38] Neal Koblitz and Alfred Menezes. Another look at "provable security". *J. Cryptology*, 20(1):3–37, 2007.

[39] Neal Koblitz and Alfred Menezes. Another look at "provable security". ii. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 148–175. Springer, 2006.

[40] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[41] Kesheng Wu, Arie Shoshani, and Kurt Stockinger. Analyses of multi-level and multi-component compressed bitmap indexes. *ACM Trans. Database Syst.*, 35(1), 2010.

[42] Arie Shoshani Kesheng Wu, Ekow Otoo and H. Nordberg. Notes on design and implementation of compressed bit vectors. Technical Report LBNL/PUB-3161, Lawrence Berkeley National Laboratory, Berkeley, CA, 2001.

[43] Daniel Lemire, Owen Kaser, and Kamel Aouiche. Sorting improves word-aligned bitmap indexes. *CoRR*, abs/0901.3751, 2009.

[44] Sumeet Bajaj and Radu Sion. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Trans. Knowl. Data Eng.*, 26(3):752–765, 2014.

[45] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In Bhavani M. Thuraisingham, Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari, editors, *Data and Application Security, Development and Directions, IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security, Schoorl, The Netherlands, August 21-23, 2000*, volume 201 of *IFIP Conference Proceedings*, pages 101–112. Kluwer, 2000.

[46] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11(3):291–314, 2003.

[47] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[48] HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 560–571. IEEE Computer Society, 2004.

[49] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In Chaudhuri et al. [95], pages 121–132.

[50] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In Özcan [96], pages 407–418.

[51] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving queries on encrypted data. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 479–495. Springer, 2006.

[52] HweeHwa Pang and Kian-Lee Tan. Verifying completeness of relational query answers from online servers. *ACM Trans. Inf. Syst. Secur.*, 11(2), 2008.

[53] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for aggregation queries. *ACM Trans. Inf. Syst. Secur.*, 13(4):32, 2010.

[54] Glen Nuckolls. Verified query results from hybrid authentication trees. In Sushil Jajodia and Duminda Wijesekera, editors, *DBSec*, volume 3654 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2005.

[55] Michael T. Goodrich, Roberto Tamassia, and Nikos Triandopoulos. Super-efficient verification of dynamic outsourced databases. In Malkin [97], pages 407–424.

[56] Yin Yang, Stavros Papadopoulos, Dimitris Papadias, and George Kollios. Spatial outsourcing for location-based services. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *ICDE*, pages 1082–1091. IEEE, 2008.

[57] Kyriakos Mouratidis, Dimitris Sacharidis, and HweeHwa Pang. Partially materialized digest scheme: an efficient verification method for outsourced databases. *VLDB J.*, 18(1):363–381, 2009.

[58] Maithili Narasimha and Gene Tsudik. DSAC: integrity for outsourced databases with signature aggregation and chaining. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *CIKM*, pages 235–236. ACM, 2005.

[59] HweeHwa Pang, Jilian Zhang, and Kyriakos Mouratidis. Scalable verification for outsourced dynamic databases. *PVLDB*, 2(1):802–813, 2009.

[60] Bernardo Palazzi, Maurizio Pizzonia, and Stefano Pucacco. Query racing: Fast completeness certification of query results. In Sara Foresti and Sushil Jajodia, editors, *DBSec*, volume 6166 of *Lecture Notes in Computer Science*, pages 177–192. Springer, 2010.

[61] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *TOS*, 2(2):107–138, 2006.

[62] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In Mong-Li Lee, Kian-Lee Tan, and Vilas Wuwongse, editors, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 420–436. Springer, 2006.

[63] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In Ted Wobber and Peter Druschel, editors, *SOSP*, pages 85–100. ACM, 2011.

[64] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *SIGMOD Conference*, pages 563–574, 2004.

[65] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2009.

[66] Vladimir Kolesnikov and Abdullatif Shikfa. On the limits of privacy provided by order-preserving encryption. *Bell Labs Technical Journal*, 17(3):135–146, 2012.

[67] Chee Yong Chan and Yannis E. Ioannidis. Bitmap index design and evaluation. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD Conference*, pages 355–366. ACM Press, 1998.

[68] Chee Yong Chan and Yannis E. Ioannidis. An efficient bitmap encoding scheme for selection queries. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD Conference*, pages 215–226. ACM Press, 1999.

[69] Kurt Stockinger. Bitmap indices for speeding up high-dimensional data analysis. In Abdelkader Hameurlain, Rosine Cicchetti, and Roland Traunmüller, editors, *DEXA*, volume 2453 of *Lecture Notes in Computer Science*, pages 881–890. Springer, 2002.

[70] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, 31(1):1–38, 2006.

[71] Jonathan Katz and Andrew Y. Lindell. Aggregate message authentication codes. In Malkin [97], pages 155–169.

[72] Patrick E. O'Neil and Goetz Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24(3):8–11, 1995.

[73] Ladjel Bellatreche, Rokia Missaoui, Hamid Necir, and Habiba Drias. A data mining approach for selecting bitmap join indices. *JCSE*, 1(2):177–194, 2007.

[74] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic data publication over the internet. *Journal of Computer Security*, 11(3):291–314, 2003.

[75] Charles U. Martel, Glen Nuckolls, Premkumar T. Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.

[76] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In Özcan [96], pages 407–418.

[77] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In *ESORICS*, pages 160–176, 2004.

[78] Yin Yang, Dimitris Papadias, Stavros Papadopoulos, and Panos Kalnis. Authenticated join processing in outsourced databases. In *SIGMOD Conference*, pages 5–18, 2009.

[79] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. On the performance of bitmap indices for high cardinality attributes. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 24–35. Morgan Kaufmann, 2004.

[80] Jiang-Hsing Chu and Gary D. Knott. An analysis of B-trees and their variants. *Inf. Syst.*, 14(5):359–370, 1989.

[81] Douglas Comer. The ubiquitous B-Tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[82] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. *IACR Cryptology ePrint Archive*, 2006:221, 2006.

[83] P. Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *Information Theory, IEEE Transactions on*, 55(10):4749–4760, 2009.

[84] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[85] Ladjel Bellatreche and Kamel Boukhalfa. Yet another algorithms for selecting bitmap join indexes. In *Data Warehousing and Knowledge Discovery*, pages 105–116. Springer Berlin Heidelberg, 2010.

[86] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Boneh [94], pages 482–499.

[87] Debrup Chakraborty and Mridul Nandi. An improved security bound for HCTR. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 2008.

[88] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. `http://eprint.iacr.org/`.

[89] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[90] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Lee [93], pages 16–31.

[91] Shay Gueron and Michael E. Kounavis. Efficient implementation of the galois counter mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.*, 110(14-15):549–553, 2010.

[92] Daniel Lemire. lemurbitmaindex. url:http://code.google.com/p/lemurbitmapindex/, 2013.

[93] Pil Joong Lee, editor. *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*. Springer, 2004.

[94] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*. Springer, 2003.

[95] Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*. ACM, 2006.

[96] Fatma Özcan, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. ACM, 2005.

[97] Tal Malkin, editor. *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, volume 4964 of *Lecture Notes in Computer Science*. Springer, 2008.